

QuantLib

An open source library for quantitative finance

Version 0.8.1

Generated by Doxygen 1.5.2

1 Jun 2007

Contents

1	Getting started	1
1.1	Introduction	1
1.2	Project overview	2
1.3	Where to get QuantLib	10
1.4	Installation	11
1.5	User configuration	12
1.6	Usage	14
1.7	Version history	15
1.8	Additional resources	39
1.9	The QuantLib Group	40
1.10	QuantLib License	41
2	QuantLib Module Index	43
2.1	QuantLib Modules	43
3	QuantLib Namespace Index	45
3.1	QuantLib Namespace List	45
4	QuantLib Hierarchical Index	47
4.1	QuantLib Class Hierarchy	47
5	QuantLib Class Index	69
5.1	QuantLib Class List	69
6	QuantLib File Index	87
6.1	QuantLib File List	87
7	QuantLib Module Documentation	101
7.1	Numeric types	101
7.2	Currencies and FX rates	104

7.3	Date and time calculations	108
7.4	Calendars	111
7.5	Day counters	114
7.6	Pricing engines	115
7.7	Asian option engines	116
7.8	Barrier option engines	117
7.9	Basket option engines	118
7.10	Cap/floor engines	119
7.11	Cliquet option engines	120
7.12	Forward option engines	121
7.13	Quanto option engines	122
7.14	Swaption engines	123
7.15	Vanilla option engines	124
7.16	Finite-differences framework	128
7.17	Short-rate modelling framework	130
7.18	Financial instruments	133
7.19	Lattice methods	137
7.20	Math tools	140
7.21	Monte Carlo framework	142
7.22	Design patterns	143
7.23	Stochastic processes	144
7.24	Term structures	146
7.25	Utilities	148
7.26	QuantLib macros	150
7.27	Generic macros	151
7.28	Numeric limits	152
7.29	Template capabilities	153
7.30	Iterator support	154
7.31	Output manipulators	155
7.32	Debugging macros	157
8	QuantLib Namespace Documentation	161
8.1	std Namespace Reference	161
9	QuantLib Class Documentation	163
9.1	Abcd Class Reference	163
9.2	AbcdFunction Struct Reference	166

9.3	AbcdVol Class Reference	168
9.4	AccountingEngine Class Reference	169
9.5	Actual360 Class Reference	170
9.6	Actual365Fixed Class Reference	171
9.7	ActualActual Class Reference	172
9.8	AcyclicVisitor Class Reference	173
9.9	AdditiveEQPBinomialTree Class Reference	174
9.10	AffineModel Class Reference	175
9.11	AmericanCondition Class Reference	176
9.12	AmericanExercise Class Reference	177
9.13	AmericanPayoffAtExpiry Class Reference	178
9.14	AmericanPayoffAtHit Class Reference	179
9.15	AnalyticBarrierEngine Class Reference	180
9.16	AnalyticCapFloorEngine Class Reference	181
9.17	AnalyticCliquetEngine Class Reference	182
9.18	AnalyticContinuousFixedLookbackEngine Class Reference	183
9.19	AnalyticContinuousFloatingLookbackEngine Class Reference	184
9.20	AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference	185
9.21	AnalyticDigitalAmericanEngine Class Reference	186
9.22	AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference	187
9.23	AnalyticDividendEuropeanEngine Class Reference	188
9.24	AnalyticEuropeanEngine Class Reference	189
9.25	AnalyticHestonEngine Class Reference	190
9.26	AnalyticPerformanceEngine Class Reference	191
9.27	Argentina Class Reference	192
9.28	ArmijoLineSearch Class Reference	194
9.29	Array Class Reference	195
9.30	ARSCurrency Class Reference	199
9.31	AssetOrNothingPayoff Class Reference	200
9.32	AssetSwap Class Reference	201
9.33	AssetSwap::arguments Class Reference	203
9.34	AssetSwap::results Class Reference	204
9.35	ATSCurrency Class Reference	205
9.36	AUDCurrency Class Reference	206
9.37	AUDLibor Class Reference	207
9.38	Australia Class Reference	208

9.39	Average Struct Reference	209
9.40	BackwardFlat Class Reference	210
9.41	BackwardFlatInterpolation Class Reference	211
9.42	BaroneAdesiWhaleyApproximationEngine Class Reference	212
9.43	Barrier Struct Reference	213
9.44	BarrierOption Class Reference	214
9.45	BarrierOption::arguments Class Reference	216
9.46	BarrierOption::engine Class Reference	217
9.47	BasketOption Class Reference	218
9.48	BasketOption::arguments Class Reference	219
9.49	BasketOption::engine Class Reference	220
9.50	BatesEngine Class Reference	221
9.51	BatesModel Class Reference	223
9.52	BDTCurrency Class Reference	224
9.53	BEFCurrency Class Reference	225
9.54	BermudanExercise Class Reference	226
9.55	BGLCurrency Class Reference	227
9.56	Bicubic Class Reference	228
9.57	BicubicSpline Class Reference	229
9.58	Bilinear Class Reference	230
9.59	BilinearInterpolation Class Reference	231
9.60	BinomialConvertibleEngine Class Template Reference	232
9.61	BinomialDistribution Class Reference	233
9.62	BinomialTree Class Template Reference	234
9.63	BinomialVanillaEngine Class Template Reference	235
9.64	Bisection Class Reference	236
9.65	BivariateCumulativeNormalDistributionDr78 Class Reference	237
9.66	BivariateCumulativeNormalDistributionWe04DP Class Reference	238
9.67	Bjerk SundStenslandApproximationEngine Class Reference	239
9.68	BlackCalculator Class Reference	240
9.69	BlackCapFloorEngine Class Reference	243
9.70	BlackConstantVol Class Reference	244
9.71	BlackIborCouponPricer Class Reference	246
9.72	BlackKarasinski Class Reference	247
9.73	BlackKarasinski::Dynamics Class Reference	248
9.74	BlackProcess Class Reference	249

9.75	BlackScholesCalculator Class Reference	250
9.76	BlackScholesLattice Class Template Reference	252
9.77	BlackScholesMertonProcess Class Reference	253
9.78	BlackScholesProcess Class Reference	254
9.79	BlackSwaptionEngine Class Reference	255
9.80	BlackVarianceCurve Class Reference	256
9.81	BlackVarianceSurface Class Reference	258
9.82	BlackVarianceTermStructure Class Reference	260
9.83	BlackVolatilityTermStructure Class Reference	262
9.84	BlackVolTermStructure Class Reference	264
9.85	Bond Class Reference	267
9.86	BoundaryCondition Class Template Reference	271
9.87	BoundaryConstraint Class Reference	273
9.88	BoxMullerGaussianRng Class Template Reference	274
9.89	Brazil Class Reference	275
9.90	Brent Class Reference	277
9.91	BRLCurrency Class Reference	278
9.92	BrownianBridge Class Reference	279
9.93	BSMOperator Class Reference	280
9.94	Business252 Class Reference	281
9.95	BYRCurrency Class Reference	282
9.96	CADCurrency Class Reference	283
9.97	CADLibor Class Reference	284
9.98	Calendar Class Reference	285
9.99	Calendar::Impl Class Reference	289
9.100	Calendar::OrthodoxImpl Class Reference	290
9.101	Calendar::WesternImpl Class Reference	291
9.102	CalibratedModel Class Reference	292
9.103	CalibrationHelper Class Reference	294
9.104	Callability Class Reference	296
9.105	Callability::Price Class Reference	297
9.106	Canada Class Reference	298
9.107	Cap Class Reference	299
9.108	CapFloor Class Reference	300
9.109	CapFloor::arguments Class Reference	302
9.110	CapFloor::engine Class Reference	303

9.111	CapHelper Class Reference	304
9.112	CapletConstantVolatility Class Reference	305
9.113	CapletVolatilityStructure Class Reference	307
9.114	CappedFlooredCoupon Class Reference	309
9.115	CapVolatilityStructure Class Reference	311
9.116	CapVolatilityVector Class Reference	313
9.117	CashFlow Class Reference	315
9.118	CashFlows Class Reference	317
9.119	CashOrNothingPayoff Class Reference	320
9.120	Cdor Class Reference	321
9.121	CeilingTruncation Class Reference	322
9.122	CHFCurrency Class Reference	323
9.123	CHFLibor Class Reference	324
9.124	China Class Reference	325
9.125	CLGaussianRng Class Template Reference	326
9.126	CliquetOption Class Reference	327
9.127	CliquetOption::arguments Class Reference	329
9.128	CliquetOption::engine Class Reference	330
9.129	Clone Class Template Reference	331
9.130	ClosestRounding Class Reference	332
9.131	CLPCurrency Class Reference	333
9.132	CmsCoupon Class Reference	334
9.133	CmsCouponPricer Class Reference	335
9.134	CmsMarket Class Reference	336
9.135	CMSMMDriftCalculator Class Reference	337
9.136	CmsRateBond Class Reference	338
9.137	CMSwapCurveState Class Reference	339
9.138	CNYCurrency Class Reference	340
9.139	Collar Class Reference	341
9.140	Composite Class Template Reference	342
9.141	CompositeConstraint Class Reference	343
9.142	CompositeInstrument Class Reference	344
9.143	CompositeQuote Class Template Reference	346
9.144	CompoundForward Class Reference	347
9.145	ConjugateGradient Class Reference	349
9.146	ConstantEstimator Class Reference	350

9.147	ConstantParameter Class Reference	351
9.148	ConstrainedEvolver Class Reference	352
9.149	Constraint Class Reference	353
9.150	Constraint::Impl Class Reference	354
9.151	ContinuousAveragingAsianOption Class Reference	355
9.152	ContinuousAveragingAsianOption::arguments Class Reference	357
9.153	ContinuousAveragingAsianOption::engine Class Reference	358
9.154	ContinuousFixedLookbackOption Class Reference	359
9.155	ContinuousFixedLookbackOption::arguments Class Reference	360
9.156	ContinuousFixedLookbackOption::engine Class Reference	361
9.157	ContinuousFloatingLookbackOption Class Reference	362
9.158	ContinuousFloatingLookbackOption::arguments Class Reference	363
9.159	ContinuousFloatingLookbackOption::engine Class Reference	364
9.160	ConundrumPricer Class Reference	365
9.161	ConundrumPricerByBlack Class Reference	367
9.162	ConundrumPricerByNumericalIntegration Class Reference	368
9.163	ConvergenceStatistics Class Template Reference	369
9.164	ConvertibleBond Class Reference	370
9.165	ConvertibleFixedCouponBond Class Reference	372
9.166	ConvertibleFloatingRateBond Class Reference	373
9.167	ConvertibleZeroCouponBond Class Reference	374
9.168	COPCurrency Class Reference	375
9.169	CostFunction Class Reference	376
9.170	CoterminalSwapCurveState Class Reference	377
9.171	Coupon Class Reference	378
9.172	CovarianceDecomposition Class Reference	380
9.173	CoxIngersollRoss Class Reference	381
9.174	CoxIngersollRoss::Dynamics Class Reference	383
9.175	CoxRossRubinstein Class Reference	384
9.176	CrankNicolson Class Template Reference	385
9.177	Cubic Class Reference	387
9.178	CubicSpline Class Reference	388
9.179	CumulativeBinomialDistribution Class Reference	390
9.180	CumulativeNormalDistribution Class Reference	391
9.181	CumulativePoissonDistribution Class Reference	392
9.182	CuriouslyRecurringTemplate Class Template Reference	393

9.183	Currency Class Reference	394
9.184	Curve Class Reference	396
9.185	CurveState Class Reference	397
9.186	CYPCurrency Class Reference	398
9.187	CzechRepublic Class Reference	399
9.188	CZKCurrency Class Reference	401
9.189	Date Class Reference	402
9.190	DayCounter Class Reference	406
9.191	DayCounter::Impl Class Reference	408
9.192	DecInterpCapletVolStructure Class Reference	409
9.193	DEMCurrency Class Reference	410
9.194	Denmark Class Reference	411
9.195	DepositRateHelper Class Reference	412
9.196	DerivedQuote Class Template Reference	414
9.197	DirichletBC Class Reference	415
9.198	Discount Struct Reference	417
9.199	DiscrepancyStatistics Class Reference	418
9.200	DiscreteAveragingAsianOption Class Reference	419
9.201	DiscreteAveragingAsianOption::arguments Class Reference	421
9.202	DiscreteAveragingAsianOption::engine Class Reference	422
9.203	DiscreteGeometricASO Class Reference	423
9.204	DiscretizedAsset Class Reference	424
9.205	DiscretizedDiscountBond Class Reference	427
9.206	DiscretizedOption Class Reference	428
9.207	Disposable Class Template Reference	430
9.208	Dividend Class Reference	431
9.209	DividendVanillaOption Class Reference	433
9.210	DividendVanillaOption::arguments Class Reference	435
9.211	DividendVanillaOption::engine Class Reference	436
9.212	DKKCurrency Class Reference	437
9.213	DKKLibor Class Reference	438
9.214	DMinus Class Reference	439
9.215	Domain Class Reference	440
9.216	DoubleStickyRatchetPayoff Class Reference	441
9.217	DownRounding Class Reference	443
9.218	DPlus Class Reference	444

9.219	DPlusDMinus Class Reference	445
9.220	DriftTermStructure Class Reference	446
9.221	Duration Struct Reference	448
9.222	DZero Class Reference	449
9.223	EarlyExercise Class Reference	450
9.224	EarlyExercisePathPricer Class Template Reference	451
9.225	EEKCurrency Class Reference	452
9.226	EndCriteria Class Reference	453
9.227	EqualJumpsBinomialTree Class Template Reference	455
9.228	EqualProbabilitiesBinomialTree Class Template Reference	456
9.229	Error Class Reference	457
9.230	ErrorFunction Class Reference	458
9.231	ESPCurrency Class Reference	459
9.232	EulerDiscretization Class Reference	460
9.233	EURCurrency Class Reference	462
9.234	Euribor Class Reference	463
9.235	Euribor10M Class Reference	464
9.236	Euribor11M Class Reference	465
9.237	Euribor1M Class Reference	466
9.238	Euribor1Y Class Reference	467
9.239	Euribor2M Class Reference	468
9.240	Euribor2W Class Reference	469
9.241	Euribor365 Class Reference	470
9.242	Euribor365_10M Class Reference	471
9.243	Euribor365_11M Class Reference	472
9.244	Euribor365_1M Class Reference	473
9.245	Euribor365_1Y Class Reference	474
9.246	Euribor365_2M Class Reference	475
9.247	Euribor365_2W Class Reference	476
9.248	Euribor365_3M Class Reference	477
9.249	Euribor365_3W Class Reference	478
9.250	Euribor365_4M Class Reference	479
9.251	Euribor365_5M Class Reference	480
9.252	Euribor365_6M Class Reference	481
9.253	Euribor365_7M Class Reference	482
9.254	Euribor365_8M Class Reference	483

9.255	Euribor365_9M Class Reference	484
9.256	Euribor365_SW Class Reference	485
9.257	Euribor3M Class Reference	486
9.258	Euribor3W Class Reference	487
9.259	Euribor4M Class Reference	488
9.260	Euribor5M Class Reference	489
9.261	Euribor6M Class Reference	490
9.262	Euribor7M Class Reference	491
9.263	Euribor8M Class Reference	492
9.264	Euribor9M Class Reference	493
9.265	EuriborSW Class Reference	494
9.266	EuriborSwapFixA10Y Class Reference	495
9.267	EuriborSwapFixA12Y Class Reference	496
9.268	EuriborSwapFixA15Y Class Reference	497
9.269	EuriborSwapFixA1Y Class Reference	498
9.270	EuriborSwapFixA20Y Class Reference	499
9.271	EuriborSwapFixA25Y Class Reference	500
9.272	EuriborSwapFixA2Y Class Reference	501
9.273	EuriborSwapFixA30Y Class Reference	502
9.274	EuriborSwapFixA3Y Class Reference	503
9.275	EuriborSwapFixA4Y Class Reference	504
9.276	EuriborSwapFixA5Y Class Reference	505
9.277	EuriborSwapFixA6Y Class Reference	506
9.278	EuriborSwapFixA7Y Class Reference	507
9.279	EuriborSwapFixA8Y Class Reference	508
9.280	EuriborSwapFixA9Y Class Reference	509
9.281	EuriborSwapFixAvs3M Class Reference	510
9.282	EuriborSwapFixAvs6M Class Reference	511
9.283	EuriborSwapFixB10Y Class Reference	512
9.284	EuriborSwapFixB12Y Class Reference	513
9.285	EuriborSwapFixB15Y Class Reference	514
9.286	EuriborSwapFixB1Y Class Reference	515
9.287	EuriborSwapFixB20Y Class Reference	516
9.288	EuriborSwapFixB25Y Class Reference	517
9.289	EuriborSwapFixB2Y Class Reference	518
9.290	EuriborSwapFixB30Y Class Reference	519

9.291	EuriborSwapFixB3Y Class Reference	520
9.292	EuriborSwapFixB4Y Class Reference	521
9.293	EuriborSwapFixB5Y Class Reference	522
9.294	EuriborSwapFixB6Y Class Reference	523
9.295	EuriborSwapFixB7Y Class Reference	524
9.296	EuriborSwapFixB8Y Class Reference	525
9.297	EuriborSwapFixB9Y Class Reference	526
9.298	EuriborSwapFixBvs3M Class Reference	527
9.299	EuriborSwapFixBvs6M Class Reference	528
9.300	EuriborSwapFixIFR10Y Class Reference	529
9.301	EuriborSwapFixIFR12Y Class Reference	530
9.302	EuriborSwapFixIFR15Y Class Reference	531
9.303	EuriborSwapFixIFR1Y Class Reference	532
9.304	EuriborSwapFixIFR20Y Class Reference	533
9.305	EuriborSwapFixIFR25Y Class Reference	534
9.306	EuriborSwapFixIFR2Y Class Reference	535
9.307	EuriborSwapFixIFR30Y Class Reference	536
9.308	EuriborSwapFixIFR3Y Class Reference	537
9.309	EuriborSwapFixIFR4Y Class Reference	538
9.310	EuriborSwapFixIFR5Y Class Reference	539
9.311	EuriborSwapFixIFR6Y Class Reference	540
9.312	EuriborSwapFixIFR7Y Class Reference	541
9.313	EuriborSwapFixIFR8Y Class Reference	542
9.314	EuriborSwapFixIFR9Y Class Reference	543
9.315	EuriborSwapFixIFRvs3M Class Reference	544
9.316	EuriborSwapFixIFRvs6M Class Reference	545
9.317	EURLibor Class Reference	546
9.318	EURLibor10M Class Reference	547
9.319	EURLibor11M Class Reference	548
9.320	EURLibor1M Class Reference	549
9.321	EURLibor1Y Class Reference	550
9.322	EURLibor2M Class Reference	551
9.323	EURLibor2W Class Reference	552
9.324	EURLibor3M Class Reference	553
9.325	EURLibor4M Class Reference	554
9.326	EURLibor5M Class Reference	555

9.327	EURLibor6M Class Reference	556
9.328	EURLibor7M Class Reference	557
9.329	EURLibor8M Class Reference	558
9.330	EURLibor9M Class Reference	559
9.331	EURLiborSW Class Reference	560
9.332	EurliborSwapFixA10Y Class Reference	561
9.333	EurliborSwapFixA12Y Class Reference	562
9.334	EurliborSwapFixA15Y Class Reference	563
9.335	EurliborSwapFixA1Y Class Reference	564
9.336	EurliborSwapFixA20Y Class Reference	565
9.337	EurliborSwapFixA25Y Class Reference	566
9.338	EurliborSwapFixA2Y Class Reference	567
9.339	EurliborSwapFixA30Y Class Reference	568
9.340	EurliborSwapFixA3Y Class Reference	569
9.341	EurliborSwapFixA4Y Class Reference	570
9.342	EurliborSwapFixA5Y Class Reference	571
9.343	EurliborSwapFixA6Y Class Reference	572
9.344	EurliborSwapFixA7Y Class Reference	573
9.345	EurliborSwapFixA8Y Class Reference	574
9.346	EurliborSwapFixA9Y Class Reference	575
9.347	EurliborSwapFixAvs3M Class Reference	576
9.348	EurliborSwapFixAvs6M Class Reference	577
9.349	EurliborSwapFixB10Y Class Reference	578
9.350	EurliborSwapFixB12Y Class Reference	579
9.351	EurliborSwapFixB15Y Class Reference	580
9.352	EurliborSwapFixB1Y Class Reference	581
9.353	EurliborSwapFixB20Y Class Reference	582
9.354	EurliborSwapFixB25Y Class Reference	583
9.355	EurliborSwapFixB2Y Class Reference	584
9.356	EurliborSwapFixB30Y Class Reference	585
9.357	EurliborSwapFixB3Y Class Reference	586
9.358	EurliborSwapFixB4Y Class Reference	587
9.359	EurliborSwapFixB5Y Class Reference	588
9.360	EurliborSwapFixB6Y Class Reference	589
9.361	EurliborSwapFixB7Y Class Reference	590
9.362	EurliborSwapFixB8Y Class Reference	591

9.363	EurliborSwapFixB9Y Class Reference	592
9.364	EurliborSwapFixBvs3M Class Reference	593
9.365	EurliborSwapFixBvs6M Class Reference	594
9.366	EurliborSwapFixIFR10Y Class Reference	595
9.367	EurliborSwapFixIFR12Y Class Reference	596
9.368	EurliborSwapFixIFR15Y Class Reference	597
9.369	EurliborSwapFixIFR1Y Class Reference	598
9.370	EurliborSwapFixIFR20Y Class Reference	599
9.371	EurliborSwapFixIFR25Y Class Reference	600
9.372	EurliborSwapFixIFR2Y Class Reference	601
9.373	EurliborSwapFixIFR30Y Class Reference	602
9.374	EurliborSwapFixIFR3Y Class Reference	603
9.375	EurliborSwapFixIFR4Y Class Reference	604
9.376	EurliborSwapFixIFR5Y Class Reference	605
9.377	EurliborSwapFixIFR6Y Class Reference	606
9.378	EurliborSwapFixIFR7Y Class Reference	607
9.379	EurliborSwapFixIFR8Y Class Reference	608
9.380	EurliborSwapFixIFR9Y Class Reference	609
9.381	EurliborSwapFixIFRvs3M Class Reference	610
9.382	EurliborSwapFixIFRvs6M Class Reference	611
9.383	EurodollarFuturesImpliedStdDevQuote Class Reference	612
9.384	EuropeanExercise Class Reference	613
9.385	EuropeanOption Class Reference	614
9.386	Event Class Reference	615
9.387	EvolutionDescription Class Reference	617
9.388	ExchangeRate Class Reference	618
9.389	ExchangeRateManager Class Reference	620
9.390	Exercise Class Reference	622
9.391	ExplicitEuler Class Template Reference	623
9.392	ExtendedCoxIngersollRoss Class Reference	625
9.393	ExtendedCoxIngersollRoss::Dynamics Class Reference	627
9.394	ExtendedCoxIngersollRoss::FittingParameter Class Reference	628
9.395	ExtendedDiscountCurve Class Reference	629
9.396	Extrapolator Class Reference	631
9.397	Factorial Class Reference	632
9.398	FalsePosition Class Reference	633

9.399	FaureRsg Class Reference	634
9.400	FDBermudanEngine Class Reference	635
9.401	FDDividendEngineMerton73 Class Reference	636
9.402	FDDividendEngineShiftScale Class Reference	637
9.403	FDEuropeanEngine Class Reference	638
9.404	FDStepConditionEngine Class Reference	639
9.405	FIMCurrency Class Reference	640
9.406	FiniteDifferenceModel Class Template Reference	641
9.407	Finland Class Reference	642
9.408	FixedCouponBondHelper Class Reference	643
9.409	FixedDividend Class Reference	645
9.410	FixedRateBond Class Reference	647
9.411	FixedRateBondForward Class Reference	648
9.412	FixedRateCoupon Class Reference	651
9.413	FlatForward Class Reference	653
9.414	FloatingRateBond Class Reference	655
9.415	FloatingRateCoupon Class Reference	656
9.416	FloatingRateCouponPricer Class Reference	659
9.417	FloatingTypePayoff Class Reference	660
9.418	Floor Class Reference	661
9.419	FloorTruncation Class Reference	662
9.420	Forward Class Reference	663
9.421	ForwardEngine Class Template Reference	666
9.422	ForwardFlat Class Reference	667
9.423	ForwardFlatInterpolation Class Reference	668
9.424	ForwardMeasureProcess Class Reference	669
9.425	ForwardMeasureProcess1D Class Reference	670
9.426	ForwardOptionArguments Class Template Reference	671
9.427	ForwardPerformanceEngine Class Template Reference	672
9.428	ForwardRate Struct Reference	673
9.429	ForwardRateAgreement Class Reference	674
9.430	ForwardRateStructure Class Reference	677
9.431	ForwardSpreadedTermStructure Class Reference	679
9.432	ForwardTypePayoff Class Reference	681
9.433	ForwardValueQuote Class Reference	682
9.434	ForwardVanillaOption Class Reference	683

9.435	FractionalDividend Class Reference	685
9.436	FraRateHelper Class Reference	687
9.437	FRFCurrency Class Reference	689
9.438	FuturesConvAdjustmentQuote Class Reference	690
9.439	FuturesRateHelper Class Reference	692
9.440	G2 Class Reference	693
9.441	G2::FittingParameter Class Reference	695
9.442	G2ForwardProcess Class Reference	696
9.443	G2Process Class Reference	698
9.444	G2SwaptionEngine Class Reference	700
9.445	GammaFunction Class Reference	701
9.446	GapPayoff Class Reference	702
9.447	Garch11 Class Reference	704
9.448	GarmanKlassAbstract Class Reference	705
9.449	GarmanKohlagenProcess Class Reference	706
9.450	GaussChebyshev2thIntegration Class Reference	707
9.451	GaussChebyshev2thPolynomial Class Reference	708
9.452	GaussChebyshevIntegration Class Reference	709
9.453	GaussChebyshevPolynomial Class Reference	710
9.454	GaussGegenbauerIntegration Class Reference	711
9.455	GaussGegenbauerPolynomial Class Reference	712
9.456	GaussHermiteIntegration Class Reference	713
9.457	GaussHermitePolynomial Class Reference	714
9.458	GaussHyperbolicIntegration Class Reference	715
9.459	GaussHyperbolicPolynomial Class Reference	716
9.460	GaussianOrthogonalPolynomial Class Reference	717
9.461	GaussianQuadrature Class Reference	718
9.462	GaussJacobiIntegration Class Reference	719
9.463	GaussJacobiPolynomial Class Reference	720
9.464	GaussKronrodAdaptive Class Reference	721
9.465	GaussKronrodNonAdaptive Class Reference	722
9.466	GaussLaguerreIntegration Class Reference	723
9.467	GaussLaguerrePolynomial Class Reference	724
9.468	GaussLegendreIntegration Class Reference	725
9.469	GaussLegendrePolynomial Class Reference	726
9.470	GBPCurrency Class Reference	727

9.471	GBPLibor Class Reference	728
9.472	GeneralizedBlackScholesProcess Class Reference	729
9.473	GeneralStatistics Class Reference	731
9.474	GenericEngine Class Template Reference	735
9.475	GenericGaussianStatistics Class Template Reference	736
9.476	GenericModelEngine Class Template Reference	739
9.477	GenericRiskStatistics Class Template Reference	740
9.478	GenericSequenceStatistics Class Template Reference	743
9.479	GeometricBrownianMotionProcess Class Reference	745
9.480	Germany Class Reference	746
9.481	GRDCurrency Class Reference	749
9.482	Greeks Class Reference	750
9.483	HaltonRsg Class Reference	751
9.484	Handle Class Template Reference	752
9.485	HestonModel Class Reference	754
9.486	HestonModelHelper Class Reference	755
9.487	HestonProcess Class Reference	756
9.488	HKDCurrency Class Reference	758
9.489	HongKong Class Reference	759
9.490	HUFCurrency Class Reference	761
9.491	HullWhite Class Reference	762
9.492	HullWhite::Dynamics Class Reference	764
9.493	HullWhite::FittingParameter Class Reference	765
9.494	HullWhiteForwardProcess Class Reference	766
9.495	HullWhiteProcess Class Reference	768
9.496	Hungary Class Reference	770
9.497	IborCoupon Class Reference	771
9.498	IborCouponPricer Class Reference	772
9.499	IborIndex Class Reference	773
9.500	Iceland Class Reference	774
9.501	IEPCurrency Class Reference	776
9.502	ILSCurrency Class Reference	777
9.503	IMM Struct Reference	778
9.504	ImplicitEuler Class Template Reference	780
9.505	ImpliedStdDevQuote Class Reference	781
9.506	ImpliedTermStructure Class Reference	782

9.507	ImpliedVolTermStructure Class Reference	784
9.508	IncrementalStatistics Class Reference	786
9.509	Index Class Reference	789
9.510	IndexManager Class Reference	791
9.511	India Class Reference	792
9.512	Indonesia Class Reference	794
9.513	INRCurrency Class Reference	796
9.514	Instrument Class Reference	797
9.515	IntegralEngine Class Reference	800
9.516	InterestRate Class Reference	801
9.517	InterestRateIndex Class Reference	804
9.518	InterpolatedDiscountCurve Class Template Reference	806
9.519	InterpolatedForwardCurve Class Template Reference	808
9.520	InterpolatedZeroCurve Class Template Reference	810
9.521	Interpolation Class Reference	812
9.522	Interpolation2D Class Reference	814
9.523	Interpolation2D::Impl Class Reference	816
9.524	Interpolation2D::templateImpl Class Template Reference	817
9.525	Interpolation::Impl Class Reference	818
9.526	Interpolation::templateImpl Class Template Reference	819
9.527	IntervalPrice Class Reference	820
9.528	InverseCumulativeNormal Class Reference	821
9.529	InverseCumulativePoisson Class Reference	822
9.530	InverseCumulativeRng Class Template Reference	823
9.531	InverseCumulativeRsg Class Template Reference	824
9.532	IQDCurrency Class Reference	825
9.533	IRRCurrency Class Reference	826
9.534	ISKCurrency Class Reference	827
9.535	Italy Class Reference	828
9.536	ITLCurrency Class Reference	830
9.537	JamshidianSwaptionEngine Class Reference	831
9.538	Japan Class Reference	832
9.539	JarrowRudd Class Reference	833
9.540	Jibar Class Reference	834
9.541	JointCalendar Class Reference	835
9.542	JPYCurrency Class Reference	836

9.543	JPYLibor Class Reference	837
9.544	JumpDiffusionEngine Class Reference	838
9.545	JuQuadraticApproximationEngine Class Reference	839
9.546	KnuthUniformRng Class Reference	840
9.547	KRWCurrency Class Reference	841
9.548	KWDCurrency Class Reference	842
9.549	Lattice Class Reference	843
9.550	LatticeShortRateModelEngine Class Template Reference	845
9.551	LazyObject Class Reference	846
9.552	LeastSquareFunction Class Reference	848
9.553	LeastSquareProblem Class Reference	849
9.554	LecuyerUniformRng Class Reference	850
9.555	LeisenReimer Class Reference	851
9.556	LevenbergMarquardt Class Reference	852
9.557	LexicographicalView Class Template Reference	853
9.558	LfmCovarianceParameterization Class Reference	855
9.559	LfmCovarianceProxy Class Reference	856
9.560	LfmHullWhiteParameterization Class Reference	857
9.561	LfmSwaptionEngine Class Reference	858
9.562	Libor Class Reference	859
9.563	LiborForwardModel Class Reference	860
9.564	LiborForwardModelProcess Class Reference	862
9.565	Linear Class Reference	864
9.566	LinearInterpolation Class Reference	865
9.567	LinearLeastSquaresRegression Class Template Reference	866
9.568	LineSearch Class Reference	867
9.569	LmConstWrapperVolatilityModel Class Reference	869
9.570	LmCorrelationModel Class Reference	870
9.571	LmExponentialCorrelationModel Class Reference	871
9.572	LmExtLinearExponentialVolModel Class Reference	872
9.573	LmLinearExponentialCorrelationModel Class Reference	873
9.574	LmLinearExponentialVolatilityModel Class Reference	874
9.575	LMMCurveState Class Reference	875
9.576	LMMDriftCalculator Class Reference	876
9.577	LMMNormalDriftCalculator Class Reference	877
9.578	LmVolatilityModel Class Reference	878

9.579	LocalConstantVol Class Reference	879
9.580	LocalVolCurve Class Reference	880
9.581	LocalVolSurface Class Reference	882
9.582	LocalVolTermStructure Class Reference	884
9.583	LogLinear Class Reference	886
9.584	LogLinearInterpolation Class Reference	887
9.585	LogNormalCmSwapRatePc Class Reference	888
9.586	LogNormalCotSwapRatePc Class Reference	889
9.587	LogNormalFwdRateEuler Class Reference	890
9.588	LogNormalFwdRateEulerConstrained Class Reference	891
9.589	LogNormalFwdRateIpc Class Reference	892
9.590	LogNormalFwdRatePc Class Reference	893
9.591	LongstaffSchwartzPathPricer Class Template Reference	894
9.592	LTLCurrency Class Reference	895
9.593	LUFCurrency Class Reference	896
9.594	LVLCurrency Class Reference	897
9.595	MakeCapFloor Class Reference	898
9.596	MakeCms Class Reference	899
9.597	MakeMCAmericanEngine Class Template Reference	900
9.598	MakeMCDigitalEngine Class Template Reference	901
9.599	MakeMCEuropeanEngine Class Template Reference	902
9.600	MakeMCEuropeanHestonEngine Class Template Reference	903
9.601	MakeMCHullWhiteCapFloorEngine Class Template Reference	904
9.602	MakeMCVarianceSwapEngine Class Template Reference	905
9.603	MakeSchedule Class Reference	906
9.604	MakeVanillaSwap Class Reference	907
9.605	MarketModel Class Reference	908
9.606	MarketModelCapFloorEngine Class Reference	909
9.607	MarketModelComposite Class Reference	910
9.608	MarketModelEvolver Class Reference	912
9.609	MarketModelFactory Class Reference	913
9.610	MarketModelMultiProduct Class Reference	914
9.611	Matrix Class Reference	915
9.612	MCAmericanBasketEngine Class Template Reference	920
9.613	MCAmericanEngine Class Template Reference	921
9.614	MCBarrierEngine Class Template Reference	922

9.615	MCBasketEngine Class Template Reference	924
9.616	McCliquetOption Class Reference	926
9.617	MCDigitalEngine Class Template Reference	927
9.618	MCDiscreteArithmeticAPEngine Class Template Reference	928
9.619	McDiscreteArithmeticASO Class Reference	930
9.620	MCDiscreteAveragingAsianEngine Class Template Reference	931
9.621	MCDiscreteGeometricAPEngine Class Template Reference	933
9.622	MCEuropeanEngine Class Template Reference	934
9.623	MCEuropeanHestonEngine Class Template Reference	935
9.624	McEverest Class Reference	936
9.625	McHimalaya Class Reference	937
9.626	MCHullWhiteCapFloorEngine Class Template Reference	938
9.627	MCLongstaffSchwartzEngine Class Template Reference	940
9.628	McMaxBasket Class Reference	942
9.629	McPagoda Class Reference	943
9.630	McPerformanceOption Class Reference	944
9.631	McPricer Class Template Reference	945
9.632	McSimulation Class Template Reference	946
9.633	MCVanillaEngine Class Template Reference	948
9.634	MCVarianceSwapEngine Class Template Reference	949
9.635	MersenneTwisterUniformRng Class Reference	951
9.636	Merton76Process Class Reference	952
9.637	Mexico Class Reference	954
9.638	MixedScheme Class Template Reference	955
9.639	Money Class Reference	957
9.640	MonotonicCubicSpline Class Reference	960
9.641	MonteCarloModel Class Template Reference	961
9.642	MoreGreeks Class Reference	962
9.643	MoroInverseCumulativeNormal Class Reference	963
9.644	MTBrownianGenerator Class Reference	964
9.645	MTLCurrency Class Reference	965
9.646	MultiAssetOption Class Reference	966
9.647	MultiAssetOption::arguments Class Reference	968
9.648	MultiAssetOption::results Class Reference	969
9.649	MultiCubicSpline Class Template Reference	970
9.650	MultiPath Class Reference	971

9.651	MultiPathGenerator Class Template Reference	972
9.652	MultiProductComposite Class Reference	973
9.653	MultiProductMultiStep Class Reference	974
9.654	MultiProductOneStep Class Reference	975
9.655	MultiVariate Struct Template Reference	976
9.656	MXNCurrency Class Reference	977
9.657	NaturalCubicSpline Class Reference	978
9.658	NaturalMonotonicCubicSpline Class Reference	979
9.659	NeumannBC Class Reference	980
9.660	Newton Class Reference	982
9.661	NewtonSafe Class Reference	983
9.662	NewZealand Class Reference	984
9.663	NLGCurrency Class Reference	985
9.664	NoConstraint Class Reference	986
9.665	NOKCurrency Class Reference	987
9.666	NonLinearLeastSquare Class Reference	988
9.667	NormalDistribution Class Reference	989
9.668	NormalFwdRatePc Class Reference	990
9.669	Norway Class Reference	991
9.670	NPRCurrency Class Reference	992
9.671	Null Class Template Reference	993
9.672	NullCalendar Class Reference	994
9.673	NullCondition Class Template Reference	995
9.674	NullParameter Class Reference	996
9.675	NZDCurrency Class Reference	997
9.676	NZDLibor Class Reference	998
9.677	Observable Class Reference	999
9.678	ObservableValue Class Template Reference	1000
9.679	Observer Class Reference	1001
9.680	OneAssetOption Class Reference	1002
9.681	OneAssetOption::arguments Class Reference	1005
9.682	OneAssetOption::results Class Reference	1006
9.683	OneAssetStrikedOption Class Reference	1007
9.684	OneDayCounter Class Reference	1009
9.685	OneFactorAffineModel Class Reference	1010
9.686	OneFactorModel Class Reference	1011

9.687	OneFactorModel::ShortRateDynamics Class Reference	1012
9.688	OneFactorModel::ShortRateTree Class Reference	1013
9.689	OperatorFactory Class Reference	1014
9.690	OptimizationMethod Class Reference	1015
9.691	Option Class Reference	1016
9.692	Option::arguments Class Reference	1018
9.693	OrnsteinUhlenbeckProcess Class Reference	1019
9.694	Parameter Class Reference	1021
9.695	Parameter::Impl Class Reference	1022
9.696	Path Class Reference	1023
9.697	PathGenerator Class Template Reference	1024
9.698	PathPricer Class Template Reference	1025
9.699	Payoff Class Reference	1026
9.700	PercentageStrikePayoff Class Reference	1027
9.701	Period Class Reference	1028
9.702	PiecewiseConstantParameter Class Reference	1030
9.703	PiecewiseYieldCurve Class Template Reference	1031
9.704	PiecewiseZeroSpreadedTermStructure Class Reference	1033
9.705	PKRCurrency Class Reference	1035
9.706	PlainVanillaPayoff Class Reference	1036
9.707	PLNCurrency Class Reference	1037
9.708	PoissonDistribution Class Reference	1038
9.709	Poland Class Reference	1039
9.710	PositiveConstraint Class Reference	1040
9.711	PricingEngine Class Reference	1041
9.712	PrimeNumbers Class Reference	1042
9.713	Problem Class Reference	1043
9.714	ProjectedCostFunction Class Reference	1045
9.715	PTECurrency Class Reference	1046
9.716	QuantoEngine Class Template Reference	1047
9.717	QuantoForwardVanillaOption Class Reference	1049
9.718	QuantoOptionArguments Class Template Reference	1050
9.719	QuantoOptionResults Class Template Reference	1051
9.720	QuantoTermStructure Class Reference	1052
9.721	QuantoVanillaOption Class Reference	1054
9.722	Quote Class Reference	1056

9.723	RandomizedLDS Class Template Reference	1057
9.724	RandomSequenceGenerator Class Template Reference	1059
9.725	RatchetMaxPayoff Class Reference	1060
9.726	RatchetMinPayoff Class Reference	1061
9.727	RatchetPayoff Class Reference	1062
9.728	RateHelper Class Reference	1063
9.729	RelativeDateRateHelper Class Reference	1065
9.730	RelinkableHandle Class Template Reference	1066
9.731	ReplicatingVarianceSwapEngine Class Reference	1067
9.732	Ridder Class Reference	1068
9.733	ROLCurrency Class Reference	1069
9.734	RONCurrency Class Reference	1070
9.735	Rounding Class Reference	1071
9.736	SABR Class Reference	1073
9.737	SABRInterpolation Class Reference	1074
9.738	SalvagingAlgorithm Struct Reference	1075
9.739	Sample Struct Template Reference	1076
9.740	SampledCurve Class Reference	1077
9.741	SARCurrency Class Reference	1079
9.742	SaudiArabia Class Reference	1080
9.743	Schedule Class Reference	1081
9.744	Secant Class Reference	1082
9.745	SeedGenerator Class Reference	1083
9.746	SegmentIntegral Class Reference	1084
9.747	SEKCurrency Class Reference	1085
9.748	Settings Class Reference	1086
9.749	Settlement Struct Reference	1088
9.750	SGDCurrency Class Reference	1089
9.751	ShortRateModel Class Reference	1090
9.752	ShoutCondition Class Reference	1091
9.753	SimpleCashFlow Class Reference	1092
9.754	SimpleDayCounter Class Reference	1093
9.755	SimpleLocalEstimator Class Reference	1094
9.756	SimpleQuote Class Reference	1095
9.757	Simplex Class Reference	1096
9.758	SimpsonIntegral Class Reference	1097

9.759	Singapore Class Reference	1098
9.760	SingleAssetOption Class Reference	1100
9.761	SingleProductComposite Class Reference	1102
9.762	Singleton Class Template Reference	1103
9.763	SingleVariate Struct Template Reference	1104
9.764	SITCurrency Class Reference	1105
9.765	SKKCurrency Class Reference	1106
9.766	Slovakia Class Reference	1107
9.767	SmileSection Class Reference	1109
9.768	SMMDriftCalculator Class Reference	1110
9.769	SobolBrownianGenerator Class Reference	1111
9.770	SobolRsg Class Reference	1112
9.771	SoftCallability Class Reference	1114
9.772	Solver1D Class Template Reference	1115
9.773	SouthAfrica Class Reference	1117
9.774	SouthKorea Class Reference	1118
9.775	SquareRootProcess Class Reference	1120
9.776	StatsHolder Class Reference	1121
9.777	SteepestDescent Class Reference	1122
9.778	step_iterator Class Template Reference	1123
9.779	StepCondition Class Template Reference	1124
9.780	StepConditionSet Class Template Reference	1125
9.781	StickyMaxPayoff Class Reference	1126
9.782	StickyMinPayoff Class Reference	1127
9.783	StickyPayoff Class Reference	1128
9.784	StochasticProcess Class Reference	1129
9.785	StochasticProcess1D Class Reference	1132
9.786	StochasticProcess1D::discretization Class Reference	1134
9.787	StochasticProcess::discretization Class Reference	1135
9.788	StochasticProcessArray Class Reference	1136
9.789	Stock Class Reference	1139
9.790	StrikedTypePayoff Class Reference	1140
9.791	StulzEngine Class Reference	1141
9.792	SuperFundPayoff Class Reference	1142
9.793	SuperSharePayoff Class Reference	1143
9.794	Surface Class Reference	1145

9.795	SVD Class Reference	1146
9.796	Swap Class Reference	1147
9.797	SwapIndex Class Reference	1149
9.798	SwapRateHelper Class Reference	1150
9.799	Swaption Class Reference	1152
9.800	Swaption::arguments Class Reference	1154
9.801	Swaption::engine Class Reference	1155
9.802	SwaptionConstantVolatility Class Reference	1156
9.803	SwaptionHelper Class Reference	1158
9.804	SwaptionVolatilityCube Class Reference	1159
9.805	SwaptionVolatilityMatrix Class Reference	1162
9.806	SwaptionVolatilityStructure Class Reference	1165
9.807	Sweden Class Reference	1168
9.808	Switzerland Class Reference	1169
9.809	SymmetricSchurDecomposition Class Reference	1170
9.810	TabulatedGaussLegendre Class Reference	1171
9.811	Taiwan Class Reference	1172
9.812	TARGET Class Reference	1174
9.813	TermStructure Class Reference	1175
9.814	TermStructureConsistentModel Class Reference	1177
9.815	TermStructureFittingParameter Class Reference	1178
9.816	THBCurrency Class Reference	1179
9.817	Thirty360 Class Reference	1180
9.818	Tian Class Reference	1181
9.819	Tibor Class Reference	1182
9.820	TimeBasket Class Reference	1183
9.821	TimeGrid Class Reference	1184
9.822	TimeSeries Class Template Reference	1186
9.823	TqrEigenDecomposition Class Reference	1188
9.824	TransformedGrid Class Reference	1189
9.825	TrapezoidIntegral Class Reference	1190
9.826	Tree Class Template Reference	1191
9.827	TreeCapFloorEngine Class Reference	1192
9.828	TreeLattice Class Template Reference	1193
9.829	TreeLattice1D Class Template Reference	1195
9.830	TreeLattice2D Class Template Reference	1196

9.831	TreeSwaptionEngine Class Reference	1197
9.832	TreeVanillaSwapEngine Class Reference	1198
9.833	TridiagonalOperator Class Reference	1199
9.834	TridiagonalOperator::TimeSetter Class Reference	1201
9.835	Trigeorgis Class Reference	1202
9.836	TrinomialTree Class Reference	1203
9.837	TRLCurrency Class Reference	1204
9.838	TRLibor Class Reference	1205
9.839	TRYCurrency Class Reference	1206
9.840	TsiveriotisFernandesLattice Class Template Reference	1207
9.841	TTDCurrency Class Reference	1209
9.842	Turkey Class Reference	1210
9.843	TWDCurrency Class Reference	1211
9.844	TwoFactorModel Class Reference	1212
9.845	TwoFactorModel::ShortRateDynamics Class Reference	1213
9.846	TwoFactorModel::ShortRateTree Class Reference	1214
9.847	TypePayoff Class Reference	1215
9.848	Ukraine Class Reference	1216
9.849	UnitedKingdom Class Reference	1218
9.850	UnitedStates Class Reference	1220
9.851	UpperBoundEngine Class Reference	1223
9.852	UpRounding Class Reference	1224
9.853	USDCurrency Class Reference	1225
9.854	USDLibor Class Reference	1226
9.855	VanillaOption Class Reference	1227
9.856	VanillaOption::engine Class Reference	1228
9.857	VanillaSwap::arguments Class Reference	1229
9.858	VanillaSwap::results Class Reference	1230
9.859	VarianceSwap Class Reference	1231
9.860	VarianceSwap::arguments Class Reference	1233
9.861	VarianceSwap::engine Class Reference	1234
9.862	VarianceSwap::results Class Reference	1235
9.863	Vasicek Class Reference	1236
9.864	Vasicek::Dynamics Class Reference	1238
9.865	VEBCurrency Class Reference	1239
9.866	Visitor Class Template Reference	1240

9.867	YieldTermStructure Class Reference	1241
9.868	ZARCurrency Class Reference	1245
9.869	ZeroCondition Class Template Reference	1246
9.870	ZeroCouponBond Class Reference	1247
9.871	ZeroSpreadedTermStructure Class Reference	1248
9.872	ZeroYield Struct Reference	1250
9.873	ZeroYieldStructure Class Reference	1251
9.874	Zibor Class Reference	1253
10	QuantLib File Documentation	1255
10.1	ql/capvolstructures.hpp File Reference	1255
10.2	ql/cashflow.hpp File Reference	1256
10.3	ql/cashflows/analysis.hpp File Reference	1257
10.4	ql/cashflows/capflooredcoupon.hpp File Reference	1258
10.5	ql/cashflows/cashflowvectors.hpp File Reference	1259
10.6	ql/cashflows/cmscoupon.hpp File Reference	1261
10.7	ql/cashflows/conundrumpricer.hpp File Reference	1262
10.8	ql/cashflows/coupon.hpp File Reference	1263
10.9	ql/cashflows/couponpricer.hpp File Reference	1264
10.10	ql/cashflows/dividend.hpp File Reference	1266
10.11	ql/cashflows/fixedratecoupon.hpp File Reference	1267
10.12	ql/cashflows/floatingratecoupon.hpp File Reference	1268
10.13	ql/cashflows/iborcoupon.hpp File Reference	1269
10.14	ql/cashflows/rangeaccrual.hpp File Reference	1270
10.15	ql/cashflows/simplecashflow.hpp File Reference	1271
10.16	ql/cashflows/timebasket.hpp File Reference	1272
10.17	ql/currencies/africa.hpp File Reference	1273
10.18	ql/currencies/america.hpp File Reference	1274
10.19	ql/currencies/asia.hpp File Reference	1276
10.20	ql/currencies/europe.hpp File Reference	1278
10.21	ql/currencies/exchangeratemanager.hpp File Reference	1281
10.22	ql/currencies/oceania.hpp File Reference	1282
10.23	ql/currency.hpp File Reference	1283
10.24	ql/daycounter.hpp File Reference	1284
10.25	ql/discretizedasset.hpp File Reference	1285
10.26	ql/errors.hpp File Reference	1286
10.27	ql/event.hpp File Reference	1289

10.28	ql/exchangerate.hpp File Reference	1290
10.29	ql/exercise.hpp File Reference	1291
10.30	ql/grid.hpp File Reference	1292
10.31	ql/handle.hpp File Reference	1293
10.32	ql/index.hpp File Reference	1294
10.33	ql/indexes/ibor/audlibor.hpp File Reference	1295
10.34	ql/indexes/ibor/cadlibor.hpp File Reference	1296
10.35	ql/indexes/ibor/cdor.hpp File Reference	1297
10.36	ql/indexes/ibor/chflibor.hpp File Reference	1298
10.37	ql/indexes/ibor/dkklbor.hpp File Reference	1299
10.38	ql/indexes/ibor/euribor.hpp File Reference	1300
10.39	ql/indexes/ibor/eurlibor.hpp File Reference	1303
10.40	ql/indexes/ibor/gbpibor.hpp File Reference	1305
10.41	ql/indexes/ibor/jibor.hpp File Reference	1306
10.42	ql/indexes/ibor/jpylibor.hpp File Reference	1307
10.43	ql/indexes/ibor/libor.hpp File Reference	1308
10.44	ql/indexes/ibor/nzdlibor.hpp File Reference	1309
10.45	ql/indexes/ibor/tibor.hpp File Reference	1310
10.46	ql/indexes/ibor/trlibor.hpp File Reference	1311
10.47	ql/indexes/ibor/usdlibor.hpp File Reference	1312
10.48	ql/indexes/ibor/zibor.hpp File Reference	1313
10.49	ql/indexes/iborindex.hpp File Reference	1314
10.50	ql/indexes/indexmanager.hpp File Reference	1315
10.51	ql/indexes/interestrategyindex.hpp File Reference	1316
10.52	ql/indexes/swap/euriborswapfixa.hpp File Reference	1317
10.53	ql/indexes/swap/euriborswapfixb.hpp File Reference	1319
10.54	ql/indexes/swap/euriborswapfixfr.hpp File Reference	1321
10.55	ql/indexes/swap/eurliborswapfixa.hpp File Reference	1323
10.56	ql/indexes/swap/eurliborswapfixb.hpp File Reference	1325
10.57	ql/indexes/swap/eurliborswapfixfr.hpp File Reference	1327
10.58	ql/indexes/swapindex.hpp File Reference	1329
10.59	ql/instrument.hpp File Reference	1330
10.60	ql/instruments/asianooption.hpp File Reference	1331
10.61	ql/instruments/assetswap.hpp File Reference	1332
10.62	ql/instruments/barrieroption.hpp File Reference	1333
10.63	ql/instruments/basketoption.hpp File Reference	1334

10.64	ql/instruments/bond.hpp File Reference	1335
10.65	ql/instruments/callabilityschedule.hpp File Reference	1337
10.66	ql/instruments/capfloor.hpp File Reference	1338
10.67	ql/instruments/cliquestoption.hpp File Reference	1340
10.68	ql/instruments/cmsratebond.hpp File Reference	1341
10.69	ql/instruments/compositeinstrument.hpp File Reference	1342
10.70	ql/instruments/convertiblebond.hpp File Reference	1343
10.71	ql/instruments/dividendschedule.hpp File Reference	1345
10.72	ql/instruments/dividendvanillaoption.hpp File Reference	1346
10.73	ql/instruments/europeanoption.hpp File Reference	1347
10.74	ql/instruments/fixedratebond.hpp File Reference	1348
10.75	ql/instruments/fixedratebondforward.hpp File Reference	1349
10.76	ql/instruments/floatingratebond.hpp File Reference	1350
10.77	ql/instruments/forward.hpp File Reference	1351
10.78	ql/instruments/forwardrateagreement.hpp File Reference	1352
10.79	ql/instruments/forwardvanillaoption.hpp File Reference	1353
10.80	ql/instruments/lookbackoption.hpp File Reference	1354
10.81	ql/instruments/makecapfloor.hpp File Reference	1355
10.82	ql/instruments/makecms.hpp File Reference	1356
10.83	ql/instruments/makevanillaswap.hpp File Reference	1357
10.84	ql/instruments/multiassetoption.hpp File Reference	1358
10.85	ql/instruments/oneassetoption.hpp File Reference	1359
10.86	ql/instruments/oneassetstrikedoption.hpp File Reference	1360
10.87	ql/instruments/payoffs.hpp File Reference	1361
10.88	ql/instruments/quantoforwardvanillaoption.hpp File Reference	1363
10.89	ql/instruments/quantovanillaoption.hpp File Reference	1364
10.90	ql/instruments/stickyratech.hpp File Reference	1365
10.91	ql/instruments/stock.hpp File Reference	1366
10.92	ql/instruments/swap.hpp File Reference	1367
10.93	ql/instruments/swaption.hpp File Reference	1368
10.94	ql/instruments/vanillaoption.hpp File Reference	1369
10.95	ql/instruments/vanillaswap.hpp File Reference	1370
10.96	ql/instruments/varianceswap.hpp File Reference	1371
10.97	ql/instruments/zerocouponbond.hpp File Reference	1372
10.98	ql/interestrates.hpp File Reference	1373
10.99	ql/legacy/libormarketmodels/lfmcovaproxy.hpp File Reference	1374

10.100	ql/legacy/libormarketmodels/liborforwardmodel.hpp File Reference	1375
10.101	ql/legacy/libormarketmodels/lmconstwrappercorrmodel.hpp File Reference . . .	1376
10.102	ql/legacy/libormarketmodels/lmconstwrappervolmodel.hpp File Reference . . .	1377
10.103	ql/legacy/libormarketmodels/lmcorrmodel.hpp File Reference	1378
10.104	ql/legacy/libormarketmodels/lmexpcorrmodel.hpp File Reference	1379
10.105	ql/legacy/libormarketmodels/lmextlinexpvolmodel.hpp File Reference	1380
10.106	ql/legacy/libormarketmodels/lmfixedvolmodel.hpp File Reference	1381
10.107	ql/legacy/libormarketmodels/lmlinexpcorrmodel.hpp File Reference	1382
10.108	ql/legacy/libormarketmodels/lmlinexpvolmodel.hpp File Reference	1383
10.109	ql/legacy/libormarketmodels/lmvolmodel.hpp File Reference	1384
10.110	ql/legacy/pricers/discretegeometricaso.hpp File Reference	1385
10.111	ql/legacy/pricers/mccliquetoption.hpp File Reference	1386
10.112	ql/legacy/pricers/mcdiscretearithmeticsaso.hpp File Reference	1387
10.113	ql/legacy/pricers/mceverest.hpp File Reference	1388
10.114	ql/legacy/pricers/mchimalaya.hpp File Reference	1389
10.115	ql/legacy/pricers/mcmaxbasket.hpp File Reference	1390
10.116	ql/legacy/pricers/mcpagoda.hpp File Reference	1391
10.117	ql/legacy/pricers/mcperformanceoption.hpp File Reference	1392
10.118	ql/legacy/pricers/mcpricer.hpp File Reference	1393
10.119	ql/legacy/pricers/singleassetoption.hpp File Reference	1394
10.120	ql/math/array.hpp File Reference	1395
10.121	ql/math/beta.hpp File Reference	1396
10.122	ql/math/comparison.hpp File Reference	1397
10.123	ql/math/curve.hpp File Reference	1398
10.124	ql/math/distributions/binomialdistribution.hpp File Reference	1399
10.125	ql/math/distributions/bivariatenormaldistribution.hpp File Reference	1400
10.126	ql/math/distributions/chisquaredistribution.hpp File Reference	1401
10.127	ql/math/distributions/gammadistribution.hpp File Reference	1402
10.128	ql/math/distributions/normaldistribution.hpp File Reference	1403
10.129	ql/math/distributions/poissondistribution.hpp File Reference	1404
10.130	ql/math/domain.hpp File Reference	1405
10.131	ql/math/errorfunction.hpp File Reference	1406
10.132	ql/math/factorial.hpp File Reference	1407
10.133	ql/math/functional.hpp File Reference	1408
10.134	ql/math/incompletegamma.hpp File Reference	1409
10.135	ql/math/integrals/gaussianorthogonalpolynomial.hpp File Reference	1410

10.136	ql/math/integrals/gaussianquadratures.hpp File Reference	1412
10.137	ql/math/integrals/integral.hpp File Reference	1414
10.138	ql/math/integrals/kronrodintegral.hpp File Reference	1415
10.139	ql/math/integrals/segmentintegral.hpp File Reference	1416
10.140	ql/math/integrals/simpsonintegral.hpp File Reference	1417
10.141	ql/math/integrals/trapezoidintegral.hpp File Reference	1418
10.142	ql/math/interpolation.hpp File Reference	1419
10.143	ql/math/interpolations/backwardflatinterpolation.hpp File Reference	1420
10.144	ql/math/interpolations/bicubicsplineinterpolation.hpp File Reference	1421
10.145	ql/math/interpolations/bilinearinterpolation.hpp File Reference	1422
10.146	ql/math/interpolations/cubicspline.hpp File Reference	1423
10.147	ql/math/interpolations/extrapolation.hpp File Reference	1424
10.148	ql/math/interpolations/flatextrapolation2d.hpp File Reference	1425
10.149	ql/math/interpolations/forwardflatinterpolation.hpp File Reference	1426
10.150	ql/math/interpolations/interpolation2d.hpp File Reference	1427
10.151	ql/math/interpolations/linearinterpolation.hpp File Reference	1428
10.152	ql/math/interpolations/loglinearinterpolation.hpp File Reference	1429
10.153	ql/math/interpolations/multicubicspline.hpp File Reference	1430
10.154	ql/math/interpolations/sabrinterpolation.hpp File Reference	1432
10.155	ql/math/lexicographicalview.hpp File Reference	1433
10.156	ql/math/linearleastquaresregression.hpp File Reference	1434
10.157	ql/math/matrix.hpp File Reference	1435
10.158	ql/math/matrixutilities/choleskydecomposition.hpp File Reference	1436
10.159	ql/math/matrixutilities/getcovariance.hpp File Reference	1437
10.160	ql/math/matrixutilities/pseudosqrt.hpp File Reference	1438
10.161	ql/math/matrixutilities/svd.hpp File Reference	1439
10.162	ql/math/matrixutilities/symmetricschurdecomposition.hpp File Reference	1440
10.163	ql/math/matrixutilities/tqreigendecomposition.hpp File Reference	1441
10.164	ql/math/optimization/armijo.hpp File Reference	1442
10.165	ql/math/optimization/conjugategradient.hpp File Reference	1443
10.166	ql/math/optimization/constraint.hpp File Reference	1444
10.167	ql/math/optimization/costfunction.hpp File Reference	1445
10.168	ql/math/optimization/endcriteria.hpp File Reference	1446
10.169	ql/math/optimization/leastsquare.hpp File Reference	1447
10.170	ql/math/optimization/levenbergmarquardt.hpp File Reference	1448
10.171	ql/math/optimization/linesearch.hpp File Reference	1449

10.172	ql/math/optimization/linebasedmethod.hpp File Reference	1450
10.173	ql/math/optimization/lmdif.hpp File Reference	1451
10.174	ql/math/optimization/method.hpp File Reference	1452
10.175	ql/math/optimization/problem.hpp File Reference	1453
10.176	ql/math/optimization/projectedcostfunction.hpp File Reference	1454
10.177	ql/math/optimization/simplex.hpp File Reference	1455
10.178	ql/math/optimization/steepestdescent.hpp File Reference	1456
10.179	ql/math/primenumbers.hpp File Reference	1457
10.180	ql/math/randomnumbers/boxmullergaussianrng.hpp File Reference	1458
10.181	ql/math/randomnumbers/centrallimitgaussianrng.hpp File Reference	1459
10.182	ql/math/randomnumbers/faurersg.hpp File Reference	1460
10.183	ql/math/randomnumbers/haltonrsg.hpp File Reference	1461
10.184	ql/math/randomnumbers/inversecumulativerng.hpp File Reference	1462
10.185	ql/math/randomnumbers/inversecumulativersg.hpp File Reference	1463
10.186	ql/math/randomnumbers/knuthuniformrng.hpp File Reference	1464
10.187	ql/math/randomnumbers/lecuyeruniformrng.hpp File Reference	1465
10.188	ql/math/randomnumbers/mt19937uniformrng.hpp File Reference	1466
10.189	ql/math/randomnumbers/randomizedlds.hpp File Reference	1467
10.190	ql/math/randomnumbers/randomsequencegenerator.hpp File Reference	1468
10.191	ql/math/randomnumbers/rngtraits.hpp File Reference	1469
10.192	ql/math/randomnumbers/seedgenerator.hpp File Reference	1470
10.193	ql/math/randomnumbers/sobolrsg.hpp File Reference	1471
10.194	ql/math/rounding.hpp File Reference	1472
10.195	ql/math/sampledcurve.hpp File Reference	1473
10.196	ql/math/solver1d.hpp File Reference	1474
10.197	ql/math/solvers1d/bisection.hpp File Reference	1475
10.198	ql/math/solvers1d/brent.hpp File Reference	1476
10.199	ql/math/solvers1d/falseposition.hpp File Reference	1477
10.200	ql/math/solvers1d/newton.hpp File Reference	1478
10.201	ql/math/solvers1d/newtonsafe.hpp File Reference	1479
10.202	ql/math/solvers1d/ridder.hpp File Reference	1480
10.203	ql/math/solvers1d/secant.hpp File Reference	1481
10.204	ql/math/statistics/convergencestatistics.hpp File Reference	1482
10.205	ql/math/statistics/discrepancystatistics.hpp File Reference	1483
10.206	ql/math/statistics/gaussianstatistics.hpp File Reference	1484
10.207	ql/math/statistics/generalstatistics.hpp File Reference	1485

10.208	ql/math/statistics/incrementalstatistics.hpp File Reference	1486
10.209	ql/math/statistics/riskstatistics.hpp File Reference	1487
10.210	ql/math/statistics/sequencestatistics.hpp File Reference	1488
10.211	ql/math/statistics/statistics.hpp File Reference	1490
10.212	ql/math/surface.hpp File Reference	1491
10.213	ql/math/transformedgrid.hpp File Reference	1492
10.214	ql/methods/finitedifferences/americancondition.hpp File Reference	1493
10.215	ql/methods/finitedifferences/boundarycondition.hpp File Reference	1494
10.216	ql/methods/finitedifferences/bsmoperator.hpp File Reference	1495
10.217	ql/methods/finitedifferences/bsmtermoperator.hpp File Reference	1496
10.218	ql/methods/finitedifferences/cranknicolson.hpp File Reference	1497
10.219	ql/methods/finitedifferences/dminus.hpp File Reference	1498
10.220	ql/methods/finitedifferences/dplus.hpp File Reference	1499
10.221	ql/methods/finitedifferences/dplusdminus.hpp File Reference	1500
10.222	ql/methods/finitedifferences/dzero.hpp File Reference	1501
10.223	ql/methods/finitedifferences/expliciteuler.hpp File Reference	1502
10.224	ql/methods/finitedifferences/fdtypedefs.hpp File Reference	1503
10.225	ql/methods/finitedifferences/finitedifferencemodel.hpp File Reference	1504
10.226	ql/methods/finitedifferences/impliciteuler.hpp File Reference	1505
10.227	ql/methods/finitedifferences/mixedscheme.hpp File Reference	1506
10.228	ql/methods/finitedifferences/onefactoroperator.hpp File Reference	1507
10.229	ql/methods/finitedifferences/operatorfactory.hpp File Reference	1508
10.230	ql/methods/finitedifferences/operatortraits.hpp File Reference	1509
10.231	ql/methods/finitedifferences/parallelevolver.hpp File Reference	1510
10.232	ql/methods/finitedifferences/pde.hpp File Reference	1511
10.233	ql/methods/finitedifferences/pdebsm.hpp File Reference	1512
10.234	ql/methods/finitedifferences/pdeshortrate.hpp File Reference	1513
10.235	ql/methods/finitedifferences/shoutcondition.hpp File Reference	1514
10.236	ql/methods/finitedifferences/stepcondition.hpp File Reference	1515
10.237	ql/methods/finitedifferences/tridiagonaloperator.hpp File Reference	1516
10.238	ql/methods/finitedifferences/zerocondition.hpp File Reference	1518
10.239	ql/methods/lattices/binomialtree.hpp File Reference	1519
10.240	ql/methods/lattices/bsmlattice.hpp File Reference	1521
10.241	ql/methods/lattices/lattice.hpp File Reference	1522
10.242	ql/methods/lattices/lattice1d.hpp File Reference	1523
10.243	ql/methods/lattices/lattice2d.hpp File Reference	1524

10.244	ql/methods/lattices/tflattice.hpp File Reference	1525
10.245	ql/methods/lattices/tree.hpp File Reference	1526
10.246	ql/methods/lattices/trinomialtree.hpp File Reference	1527
10.247	ql/methods/montecarlo/brownianbridge.hpp File Reference	1528
10.248	ql/methods/montecarlo/earlyexercisepathpricer.hpp File Reference	1529
10.249	ql/methods/montecarlo/longstaffschwartzpathpricer.hpp File Reference	1530
10.250	ql/methods/montecarlo/lsmbasissystem.hpp File Reference	1531
10.251	ql/methods/montecarlo/mctraits.hpp File Reference	1532
10.252	ql/methods/montecarlo/montecarlomodel.hpp File Reference	1533
10.253	ql/methods/montecarlo/multipath.hpp File Reference	1534
10.254	ql/methods/montecarlo/multipathgenerator.hpp File Reference	1535
10.255	ql/methods/montecarlo/path.hpp File Reference	1536
10.256	ql/methods/montecarlo/pathgenerator.hpp File Reference	1537
10.257	ql/methods/montecarlo/pathpricer.hpp File Reference	1538
10.258	ql/methods/montecarlo/sample.hpp File Reference	1539
10.259	ql/models/calibrationhelper.hpp File Reference	1540
10.260	ql/models/equity/batesmodel.hpp File Reference	1541
10.261	ql/models/equity/hestonmodel.hpp File Reference	1542
10.262	ql/models/equity/hestonmodelhelper.hpp File Reference	1543
10.263	ql/models/marketmodels/driftcomputation/cmsmmdriftcalculator.hpp File Reference	1544
10.264	ql/models/marketmodels/driftcomputation/lmmdriftcalculator.hpp File Reference	1545
10.265	ql/models/marketmodels/driftcomputation/lmmnormaldriftcalculator.hpp File Reference	1546
10.266	ql/models/marketmodels/driftcomputation/smmdriftcalculator.hpp File Reference	1547
10.267	ql/models/marketmodels/swapforwardmappings.hpp File Reference	1548
10.268	ql/models/model.hpp File Reference	1549
10.269	ql/models/parameter.hpp File Reference	1550
10.270	ql/models/shortrate/calibrationhelpers/caphelper.hpp File Reference	1551
10.271	ql/models/shortrate/calibrationhelpers/swaptionhelper.hpp File Reference	1552
10.272	ql/models/shortrate/onefactormodel.hpp File Reference	1553
10.273	ql/models/shortrate/onefactormodels/blackkarasinski.hpp File Reference	1554
10.274	ql/models/shortrate/onefactormodels/coxingersollross.hpp File Reference	1555
10.275	ql/models/shortrate/onefactormodels/extendedcoxingersollross.hpp File Reference	1556
10.276	ql/models/shortrate/onefactormodels/hullwhite.hpp File Reference	1557
10.277	ql/models/shortrate/onefactormodels/vasicek.hpp File Reference	1558

10.278	ql/models/shortrate/twofactormodel.hpp File Reference	1559
10.279	ql/models/shortrate/twofactormodels/g2.hpp File Reference	1560
10.280	ql/models/volatility/constantestimator.hpp File Reference	1561
10.281	ql/models/volatility/garch.hpp File Reference	1562
10.282	ql/models/volatility/garmanklass.hpp File Reference	1563
10.283	ql/models/volatility/simplelocalestimator.hpp File Reference	1564
10.284	ql/money.hpp File Reference	1565
10.285	ql/numericalmethod.hpp File Reference	1566
10.286	ql/option.hpp File Reference	1567
10.287	ql/patterns/composite.hpp File Reference	1568
10.288	ql/patterns/curiouslyrecurring.hpp File Reference	1569
10.289	ql/patterns/lazyobject.hpp File Reference	1570
10.290	ql/patterns/observable.hpp File Reference	1571
10.291	ql/patterns/singleton.hpp File Reference	1572
10.292	ql/patterns/visitor.hpp File Reference	1573
10.293	ql/payoff.hpp File Reference	1574
10.294	ql/position.hpp File Reference	1575
10.295	ql/prices.hpp File Reference	1576
10.296	ql/pricingengine.hpp File Reference	1577
10.297	ql/pricingengines/americanpayoffatexpiry.hpp File Reference	1578
10.298	ql/pricingengines/americanpayoffathit.hpp File Reference	1579
10.299	ql/pricingengines/asian/analytic_cont_geom_av_price.hpp File Reference	1580
10.300	ql/pricingengines/asian/analytic_discr_geom_av_price.hpp File Reference	1581
10.301	ql/pricingengines/asian/mc_discr_arith_av_price.hpp File Reference	1582
10.302	ql/pricingengines/asian/mc_discr_geom_av_price.hpp File Reference	1583
10.303	ql/pricingengines/asian/mcdiscreteasianengine.hpp File Reference	1584
10.304	ql/pricingengines/barrier/analyticbarrierengine.hpp File Reference	1585
10.305	ql/pricingengines/barrier/mcbarrierengine.hpp File Reference	1586
10.306	ql/pricingengines/basket/mcamericanbasketengine.hpp File Reference	1587
10.307	ql/pricingengines/basket/mcbasketengine.hpp File Reference	1588
10.308	ql/pricingengines/basket/stulzengine.hpp File Reference	1589
10.309	ql/pricingengines/blackcalculator.hpp File Reference	1590
10.310	ql/pricingengines/blackformula.hpp File Reference	1591
10.311	ql/pricingengines/blackscholescalculator.hpp File Reference	1593
10.312	ql/pricingengines/capfloor/analyticcapfloorengine.hpp File Reference	1594
10.313	ql/pricingengines/capfloor/blackcapfloorengine.hpp File Reference	1595

10.314	ql/pricingengines/capfloor/discretizedcapfloor.hpp File Reference	1596
10.315	ql/pricingengines/capfloor/marketmodelcapfloorengine.hpp File Reference . . .	1597
10.316	ql/pricingengines/capfloor/mchullwhiteengine.hpp File Reference	1598
10.317	ql/pricingengines/capfloor/treecapfloorengine.hpp File Reference	1599
10.318	ql/pricingengines/cliquet/analyticcliquetengine.hpp File Reference	1600
10.319	ql/pricingengines/cliquet/analyticperformanceengine.hpp File Reference	1601
10.320	ql/pricingengines/forward/forwardengine.hpp File Reference	1602
10.321	ql/pricingengines/forward/forwardperformanceengine.hpp File Reference	1603
10.322	ql/pricingengines/forward/mcvarianceswapengine.hpp File Reference	1604
10.323	ql/pricingengines/forward/replicatingvarianceswapengine.hpp File Reference .	1605
10.324	ql/pricingengines/genericmodelengine.hpp File Reference	1606
10.325	ql/pricingengines/greeks.hpp File Reference	1607
10.326	ql/pricingengines/hybrid/binomialconvertibleengine.hpp File Reference	1608
10.327	ql/pricingengines/hybrid/discretizedconvertible.hpp File Reference	1609
10.328	ql/pricingengines/latticeshortratemodelengine.hpp File Reference	1610
10.329	ql/pricingengines/lookback/analyticcontinuousfixedlookback.hpp File Reference	1611
10.330	ql/pricingengines/lookback/analyticcontinuousfloatinglookback.hpp File Refer- ence	1612
10.331	ql/pricingengines/mclongstaffschwartzengine.hpp File Reference	1613
10.332	ql/pricingengines/mcsimulation.hpp File Reference	1614
10.333	ql/pricingengines/quanto/quantoengine.hpp File Reference	1615
10.334	ql/pricingengines/swaption/blackswaptionengine.hpp File Reference	1616
10.335	ql/pricingengines/swaption/discretizedswaption.hpp File Reference	1617
10.336	ql/pricingengines/swaption/g2swaptionengine.hpp File Reference	1618
10.337	ql/pricingengines/swaption/jamshidianswaptionengine.hpp File Reference . . .	1619
10.338	ql/pricingengines/swaption/lfmswaptionengine.hpp File Reference	1620
10.339	ql/pricingengines/swaption/treeswaptionengine.hpp File Reference	1621
10.340	ql/pricingengines/vanilla/analyticdigitalamericanengine.hpp File Reference . . .	1622
10.341	ql/pricingengines/vanilla/analyticdividendeuropeanengine.hpp File Reference .	1623
10.342	ql/pricingengines/vanilla/analyticeuropeanengine.hpp File Reference	1624
10.343	ql/pricingengines/vanilla/analytichestonengine.hpp File Reference	1625
10.344	ql/pricingengines/vanilla/baroneadesiwhaleyengine.hpp File Reference	1626
10.345	ql/pricingengines/vanilla/batesengine.hpp File Reference	1627
10.346	ql/pricingengines/vanilla/binomialengine.hpp File Reference	1628
10.347	ql/pricingengines/vanilla/bjerkstundstenslandengine.hpp File Reference	1629
10.348	ql/pricingengines/vanilla/discretizedvanillaoption.hpp File Reference	1630
10.349	ql/pricingengines/vanilla/fdamericanengine.hpp File Reference	1631

10.350	ql/pricingengines/vanilla/fdbermudanengine.hpp File Reference	1632
10.351	ql/pricingengines/vanilla/fdconditions.hpp File Reference	1633
10.352	ql/pricingengines/vanilla/fddividendamericanengine.hpp File Reference	1634
10.353	ql/pricingengines/vanilla/fddividendengine.hpp File Reference	1635
10.354	ql/pricingengines/vanilla/fddividendeuropeanengine.hpp File Reference	1636
10.355	ql/pricingengines/vanilla/fddividendshoutengine.hpp File Reference	1637
10.356	ql/pricingengines/vanilla/fdeuropeanengine.hpp File Reference	1638
10.357	ql/pricingengines/vanilla/fdmultiperiodengine.hpp File Reference	1639
10.358	ql/pricingengines/vanilla/fdshoutengine.hpp File Reference	1640
10.359	ql/pricingengines/vanilla/fdstepconditionengine.hpp File Reference	1641
10.360	ql/pricingengines/vanilla/fdvanillaengine.hpp File Reference	1642
10.361	ql/pricingengines/vanilla/integralengine.hpp File Reference	1643
10.362	ql/pricingengines/vanilla/jumpdiffusionengine.hpp File Reference	1644
10.363	ql/pricingengines/vanilla/juquadraticengine.hpp File Reference	1645
10.364	ql/pricingengines/vanilla/mcamericanengine.hpp File Reference	1646
10.365	ql/pricingengines/vanilla/mcdigitalengine.hpp File Reference	1647
10.366	ql/pricingengines/vanilla/mceuropeanengine.hpp File Reference	1648
10.367	ql/pricingengines/vanilla/mceuropeanhestonengine.hpp File Reference	1649
10.368	ql/pricingengines/vanilla/mcvanillaengine.hpp File Reference	1650
10.369	ql/processes/blackscholesprocess.hpp File Reference	1651
10.370	ql/processes/eulerdiscretization.hpp File Reference	1652
10.371	ql/processes/forwardmeasureprocess.hpp File Reference	1653
10.372	ql/processes/g2process.hpp File Reference	1654
10.373	ql/processes/geometricbrownianprocess.hpp File Reference	1655
10.374	ql/processes/hestonprocess.hpp File Reference	1656
10.375	ql/processes/hullwhiteprocess.hpp File Reference	1657
10.376	ql/processes/lfmcovarParams.hpp File Reference	1658
10.377	ql/processes/lfmhullwhiteparam.hpp File Reference	1659
10.378	ql/processes/lfmpprocess.hpp File Reference	1660
10.379	ql/processes/merton76process.hpp File Reference	1661
10.380	ql/processes/ornsteinuhlenbeckprocess.hpp File Reference	1662
10.381	ql/processes/squarerootprocess.hpp File Reference	1663
10.382	ql/processes/stochasticprocessarray.hpp File Reference	1664
10.383	ql/qldefines.hpp File Reference	1665
10.384	ql/quote.hpp File Reference	1667
10.385	ql/quotes/compositequote.hpp File Reference	1668

10.386	ql/quotes/derivedquote.hpp File Reference	1669
10.387	ql/quotes/eurodollarfuturesquote.hpp File Reference	1670
10.388	ql/quotes/forwardvaluequote.hpp File Reference	1671
10.389	ql/quotes/futuresconadjustmentquote.hpp File Reference	1672
10.390	ql/quotes/impliestddevquote.hpp File Reference	1673
10.391	ql/quotes/simplequote.hpp File Reference	1674
10.392	ql/settings.hpp File Reference	1675
10.393	ql/stochasticprocess.hpp File Reference	1676
10.394	ql/swaptionvolstructure.hpp File Reference	1677
10.395	ql/termstructure.hpp File Reference	1678
10.396	ql/termstructures/volatilities/blackconstantvol.hpp File Reference	1679
10.397	ql/termstructures/volatilities/blackvariancecurve.hpp File Reference	1680
10.398	ql/termstructures/volatilities/blackvariancesurface.hpp File Reference	1681
10.399	ql/termstructures/volatilities/capflatvolvector.hpp File Reference	1682
10.400	ql/termstructures/volatilities/capletconstantvol.hpp File Reference	1683
10.401	ql/termstructures/volatilities/capletvariancecurve.hpp File Reference	1684
10.402	ql/termstructures/volatilities/capletvolatilitiesstructures.hpp File Reference . . .	1685
10.403	ql/termstructures/volatilities/capstripper.hpp File Reference	1686
10.404	ql/termstructures/volatilities/cmsmarket.hpp File Reference	1687
10.405	ql/termstructures/volatilities/IMPLIEDVOLTERMSTRUCTURE.hpp File Reference	1688
10.406	ql/termstructures/volatilities/interpolatedsmilesection.hpp File Reference	1689
10.407	ql/termstructures/volatilities/localconstantvol.hpp File Reference	1690
10.408	ql/termstructures/volatilities/localvolcurve.hpp File Reference	1691
10.409	ql/termstructures/volatilities/localvolsurface.hpp File Reference	1692
10.410	ql/termstructures/volatilities/sabr.hpp File Reference	1693
10.411	ql/termstructures/volatilities/sabrinterpolatedsmilesection.hpp File Reference . .	1694
10.412	ql/termstructures/volatilities/smilesection.hpp File Reference	1695
10.413	ql/termstructures/volatilities/swaptionconstantvol.hpp File Reference	1696
10.414	ql/termstructures/volatilities/swaptionvolcube.hpp File Reference	1697
10.415	ql/termstructures/volatilities/swaptionvolcube1.hpp File Reference	1698
10.416	ql/termstructures/volatilities/swaptionvolcube2.hpp File Reference	1699
10.417	ql/termstructures/volatilities/swaptionvoldiscrete.hpp File Reference	1700
10.418	ql/termstructures/volatilities/swaptionvolmatrix.hpp File Reference	1701
10.419	ql/termstructures/yieldcurves/bondhelpers.hpp File Reference	1702
10.420	ql/termstructures/yieldcurves/bootstraptraits.hpp File Reference	1703
10.421	ql/termstructures/yieldcurves/compoundforward.hpp File Reference	1704

10.422	ql/termstructures/yieldcurves/discountcurve.hpp File Reference	1705
10.423	ql/termstructures/yieldcurves/drifttermstructure.hpp File Reference	1706
10.424	ql/termstructures/yieldcurves/extendeddiscountcurve.hpp File Reference	1707
10.425	ql/termstructures/yieldcurves/flatforward.hpp File Reference	1708
10.426	ql/termstructures/yieldcurves/forwardcurve.hpp File Reference	1709
10.427	ql/termstructures/yieldcurves/forwardspreadedtermstructure.hpp File Reference	1710
10.428	ql/termstructures/yieldcurves/forwardstructure.hpp File Reference	1711
10.429	ql/termstructures/yieldcurves/IMPLIEDtermstructure.hpp File Reference	1712
10.430	ql/termstructures/yieldcurves/piecewiseyieldcurve.hpp File Reference	1713
10.431	ql/termstructures/yieldcurves/piecewisezerospreadedtermstructure.hpp File Reference	1714
10.432	ql/termstructures/yieldcurves/quantotermstructure.hpp File Reference	1715
10.433	ql/termstructures/yieldcurves/ratehelpers.hpp File Reference	1716
10.434	ql/termstructures/yieldcurves/zerocurve.hpp File Reference	1717
10.435	ql/termstructures/yieldcurves/zerospreadedtermstructure.hpp File Reference	1718
10.436	ql/termstructures/yieldcurves/zeroyieldstructure.hpp File Reference	1719
10.437	ql/time/businessdayconvention.hpp File Reference	1720
10.438	ql/time/calendar.hpp File Reference	1721
10.439	ql/time/calendars/argentina.hpp File Reference	1722
10.440	ql/time/calendars/australia.hpp File Reference	1723
10.441	ql/time/calendars/brazil.hpp File Reference	1724
10.442	ql/time/calendars/canada.hpp File Reference	1725
10.443	ql/time/calendars/china.hpp File Reference	1726
10.444	ql/time/calendars/czechrepublic.hpp File Reference	1727
10.445	ql/time/calendars/denmark.hpp File Reference	1728
10.446	ql/time/calendars/finland.hpp File Reference	1729
10.447	ql/time/calendars/germany.hpp File Reference	1730
10.448	ql/time/calendars/hongkong.hpp File Reference	1731
10.449	ql/time/calendars/hungary.hpp File Reference	1732
10.450	ql/time/calendars/iceland.hpp File Reference	1733
10.451	ql/time/calendars/india.hpp File Reference	1734
10.452	ql/time/calendars/indonesia.hpp File Reference	1735
10.453	ql/time/calendars/italy.hpp File Reference	1736
10.454	ql/time/calendars/japan.hpp File Reference	1737
10.455	ql/time/calendars/jointcalendar.hpp File Reference	1738
10.456	ql/time/calendars/mexico.hpp File Reference	1739
10.457	ql/time/calendars/newzealand.hpp File Reference	1740

10.458	ql/time/calendars/norway.hpp File Reference	1741
10.459	ql/time/calendars/nullcalendar.hpp File Reference	1742
10.460	ql/time/calendars/poland.hpp File Reference	1743
10.461	ql/time/calendars/saudi Arabia.hpp File Reference	1744
10.462	ql/time/calendars/singapore.hpp File Reference	1745
10.463	ql/time/calendars/slovakia.hpp File Reference	1746
10.464	ql/time/calendars/southafrica.hpp File Reference	1747
10.465	ql/time/calendars/southkorea.hpp File Reference	1748
10.466	ql/time/calendars/sweden.hpp File Reference	1749
10.467	ql/time/calendars/switzerland.hpp File Reference	1750
10.468	ql/time/calendars/taiwan.hpp File Reference	1751
10.469	ql/time/calendars/target.hpp File Reference	1752
10.470	ql/time/calendars/turkey.hpp File Reference	1753
10.471	ql/time/calendars/ukraine.hpp File Reference	1754
10.472	ql/time/calendars/unitedkingdom.hpp File Reference	1755
10.473	ql/time/calendars/unitedstates.hpp File Reference	1756
10.474	ql/time/date.hpp File Reference	1757
10.475	ql/time/daycounters/actual360.hpp File Reference	1759
10.476	ql/time/daycounters/actual365fixed.hpp File Reference	1760
10.477	ql/time/daycounters/actualactual.hpp File Reference	1761
10.478	ql/time/daycounters/business252.hpp File Reference	1762
10.479	ql/time/daycounters/one.hpp File Reference	1763
10.480	ql/time/daycounters/simplydaycounter.hpp File Reference	1764
10.481	ql/time/daycounters/thirty360.hpp File Reference	1765
10.482	ql/time/frequency.hpp File Reference	1766
10.483	ql/time/imm.hpp File Reference	1767
10.484	ql/time/period.hpp File Reference	1768
10.485	ql/time/schedule.hpp File Reference	1770
10.486	ql/time/timeunit.hpp File Reference	1771
10.487	ql/time/weekday.hpp File Reference	1772
10.488	ql/timegrid.hpp File Reference	1773
10.489	ql/timeseries.hpp File Reference	1774
10.490	ql/types.hpp File Reference	1775
10.491	ql/utilities/clone.hpp File Reference	1777
10.492	ql/utilities/dataformatters.hpp File Reference	1778
10.493	ql/utilities/dataparsers.hpp File Reference	1780

10.494	ql/utilities/disposable.hpp File Reference	1781
10.495	ql/utilities/null.hpp File Reference	1782
10.496	ql/utilities/observablevalue.hpp File Reference	1783
10.497	ql/utilities/steppingiterator.hpp File Reference	1784
10.498	ql/utilities/tracing.hpp File Reference	1785
10.499	ql/volatilitymodel.hpp File Reference	1787
10.500	ql/voltermstructure.hpp File Reference	1788
10.501	ql/yieldtermstructure.hpp File Reference	1789
11	QuantLib Example Documentation	1791
11.1	BermudanSwaption.cpp	1791
11.2	ConvertibleBonds.cpp	1797
11.3	DiscreteHedging.cpp	1802
11.4	EquityOption.cpp	1808
11.5	FRA.cpp	1814
11.6	Replication.cpp	1820
11.7	Repo.cpp	1826
11.8	swapvaluation.cpp	1830
11.9	tracing_example.cpp	1842
12	Caveats	1845
13	Test Suite	1853
14	Todo List	1865
15	Known Bugs	1869
16	Deprecated List	1871

Chapter 1

Getting started

1.1 Introduction

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

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

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

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

1.1.1 Disclaimer

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

1.2 Project overview

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

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

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

Date, calendars, and day count conventions

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

To do:

- Differentiate more calendars depending on exchange besides country.
- enable business day calculation in addition to calendar days calculation in DayCounter::daycount(). See DayTypeEnum in FpML.

Math

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

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

To do:

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

1-dimensional solvers

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

To do:

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

Optimization

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

Random-number generation

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

Patterns

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

Finite differences

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

To do:

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

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

Lattices

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

To do:

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

Monte Carlo

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

To do:

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

- Batch samples as $N=n_batches*batch_size$, and exploit it for randomized low discrepancy sequences (RQMC)

Pricing engines

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

To do:

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

- Define dividendRho for discrete dividends.

Pricers

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

Financial Instruments

- Instrument base class: npv(), isExpired(), etc.
- Interest-rate swap.
- Swaption.
- Cap/floor.
- Zero-coupon, fixed-rate coupon, and floating-rate coupon bond.
- Convertible 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.
- More...

Yield term structures

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

To do:

- Future convexity adjustment
- End of year effect

Volatility

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

To do:

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

Short rate models

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

Credit derivatives

To do:

- everything.

Test suite

Implemented by means of the Boost unit-test framework. More than 270 automated tests. A semi-automatically-generated list is available in chapter [13](#).

To do:

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

Miscellanea

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

To do:

- Implement currency as per OMG definition

Documentation

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

To do:

- Add a "Getting started" page to the site

1.3 Where to get QuantLib

1.3.1 QuantLib releases

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

1.3.2 Current CVS snapshot

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

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

1.4 Installation

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

1.4.1 Linux/Unix/Mac OS X/Cygwin

A tarball of the source distribution is available from

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

After uncompressing the sources:

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

1.4.2 Win32

An installer for the source distribution is available at

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

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

Projects files are supplied for building the library with Microsoft Visual C++ 7.1 and 8.0 and with Dev-C++.

1.5 User configuration

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

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

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

for a list of corresponding command-line options.

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

Such macros include:

```
#define QL_ERROR_FUNCTIONS
```

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

```
#define QL_ERROR_LINES
```

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

```
#define QL_ENABLE_TRACING
```

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

```
#define QL_NEGATIVE_RATES
```

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

```
#define QL_EXTRA_SAFETY_CHECKS
```

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

```
#define QL_TODAYS_PAYMENTS
```

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

```
#define QL_DISABLE_DEPRECATED
```

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

```
#define QL_USE_INDEXED_COUPON
```

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

```
#define QL_ENABLE_SESSIONS
```

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

1.6 Usage

To use QuantLib classes in your own code just add

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

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

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

1.6.1 Microsoft Visual C++

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

1. you won't have to explicitly link your application to the QuantLib library. This is done automatically by compiler directives embedded in the sources.
2. Your project must be compiled with the same options that were used in compiling the QuantLib library you're linking with, namely,
 - Property Pages -> C/C++ -> Code Generation -> Runtime Library: select the appropriate run-time library.
 - Property Pages -> C/C++ -> Code Generation -> Basic Runtime Checks: select "Both (/RTC1, equiv. to /RTCsu)".

1.7 Version history

Release 0.8.1 - June 2007

PORTABILITY

- Version 0.8.1 adds support for Boost 1.34 on Linux systems. If you are using version 0.8.0 on Windows systems, you do not need this upgrade.

Release 0.8.0 - May 30th, 2007

PORTABILITY

- Version 0.8.0 is the last QuantLib release to support the Metrowerks CodeWarrior compiler (which was discontinued by Metrowerks.) If you use such compiler and want support to continue, you can volunteer for maintaining the necessary patches: contact the QuantLib developers for information.

SOURCE TREE

- Files and folders in the source tree have been reorganized (hopefully for the better.) If you only included `<ql/quantlib.hpp>`, all changes were taken care of for you. If you included specific headers, you might want to check its current location; in particular, all folder names are now lowercase.

CALENDARS

- Added 2007 holidays for Indonesia, Saudi Arabia, and South Korea calendars.

CASH FLOWS

- Added floater range-accrual coupons.

INDEXES

- Added EuriborSwapFixB family.

INSTRUMENTS

- Added capped/floored floating-rate bond. It can also be used for reverse floaters.
- Added delta, gamma and theta to binomial option engines (thanks to Steve Cook.)
- Refactored basket engines to allow for more payoffs.

LIBOR MARKET MODEL

- This release includes an experimental implementation of a Libor market model developed with Mark Joshi. Improvements since release 0.4.0 include normal forward-rate market model, lognormal CMS market model, lognormal coterminal-swap market model, and calibration to caplets and coterminal swaptions. The interface of the model and its integration with the bulk of the library are still in development.

MATH

- Adaptive Gauss-Kronrod integration added.
- Added Higham's nearest correlation matrix method (thanks to Neil Firth)
- Refactored optimization framework.

PROCESSES

- Added new discretization schema to Heston process.

UTILITIES

- The Handle class was split into RelinkableHandle (behaving like the old Handle class) and Handle (which is notified when its copies are relinked, but cannot itself be relinked.) The former can safely be returned from inspectors.

Release 0.4.0 - February 20th, 2007

PORTABILITY

- Starting with release 0.4.0, the Borland free compiler 5.5 and Microsoft Visual C++ 6.0 are no longer supported. If you use one of these compilers and want support to continue, you can volunteer for maintaining the necessary patches: contact the QuantLib developers for information.

CALENDARS

- Added 2007 holidays for Hong Kong, India, Singapore, and Taiwan exchanges.

LIBOR MARKET MODEL

- This release includes an experimental implementation of a Libor market model developed with Mark Joshi. Improvements since release 0.3.14 include the use of quasi-random number generators and the calculation of Greeks and of upper bounds for instruments with early-exercise features. The interface of the model and its integration with the bulk of the library are still in development.

INSTRUMENTS

- Added helper classes to make it easier to instantiate swaps, caps/floors, and CMS instruments.

INTEREST RATES

- Added capped/floored floating-rate coupons (including convexity adjustment.)

MATH

- Curve, domain and surface interfaces added.

PROCESSES

- Added reversion level to Ornstein-Uhlenbeck process (thanks to Roland Lichters.)

VOLATILITY TERM STRUCTURES

- Added stripping of caplet-volatility term structure from cap quotes.
- Improved SABR interpolation and calibration.

Release 0.3.14 - November 6th, 2006

PORTABILITY

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

LIBOR MARKET MODEL

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

CURRENCIES

- Added Romanian new lev.

DATES, CALENDARS, AND DAY COUNTERS

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

INTEREST RATES

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

INSTRUMENTS

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

MATH

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

PRICING ENGINES

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

TERM STRUCTURE

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

Release 0.3.13 - July 31st, 2006

CALENDARS

- Added NERC calendar (thanks to Joe Byers.)

INSTRUMENTS AND PRICING ENGINES

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

INTEREST RATES

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

MODELS

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

PROCESSES

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

RANDOM NUMBERS

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

MATH

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

Release 0.3.12 - March 27th, 2006

CALENDARS

- Added Brazilian calendar (thanks to Piter Dias.)

- Added Argentinian, Icelandic, Indonesian, Mexican, and Ukrainian calendars.

INSTRUMENTS AND PRICING ENGINES

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

MODELS

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

OPTIMIZATION

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

EXAMPLES

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

Release 0.3.11 - October 20th, 2005

GLOBAL FEATURES

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

CALENDARS

- Added Bombay and Taipei calendars.

CURRENCIES

- Added new Turkish lira.

INDEXES

- More accurate LIBOR calendars (thanks to Daniele de Francesco.)
- Added DKKLibor, EURLibor, and NZDLibor indexes.
- Added TRLibor index (thanks to Sercan Atalik.)

PRICING ENGINES

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

TERM STRUCTURES

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

MATH

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

Release 0.3.10 - July 14th, 2005

GLOBAL FEATURES

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

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

CALENDARS

- Istanbul calendar added (thanks to Serkan Atalik.)

LATTICE FRAMEWORK

- Faster implementation of binomial and trinomial trees.

MONTE CARLO FRAMEWORK

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

INSTRUMENTS

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

MODELS

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

CASH FLOWS

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

MATH

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

Release 0.3.9 - May 2nd, 2005

GLOBAL FEATURES

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

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

INSTRUMENTS

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

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

CALENDARS

- Added Bratislava and Prague calendars.

INDICES

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

FINITE_DIFFERENCES FRAMEWORK

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

YIELD TERM STRUCTURES

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

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

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

MATH

- Added backward- and forward-flat interpolations.

Release 0.3.8 - December 22nd, 2004

REQUIRED PACKAGES

- Boost version 1.31.0 or later is now required.

DOCUMENTATION

- Documentation now includes a FAQ page.

GLOBAL FEATURES

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

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

INSTRUMENTS

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

DATE, CALENDARS, AND DAY COUNT CONVENTIONS

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

CURRENCIES AND FX RATES

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

MONTE CARLO FRAMEWORK

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

LATTICE FRAMEWORK

- Reworked the DiscretizedAsset interface.

PRICING ENGINES FRAMEWORK

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

FIXED INCOME

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

YIELD TERM STRUCTURE

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

PATTERNS

- Added Singleton pattern.

MATH

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

MISCELLANEA

- Renamed RelinkableHandle to Handle.

PORTABILITY

- Support for Dev-C++ IDE added.

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

Release 0.3.7 - July 23rd, 2004

IMPORTANT

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

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

DATE, CALENDARS, AND DAY COUNT CONVENTIONS

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

MATH

- Added rounding algorithms as per OMG enumeration/definition.

TEST SUITE

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

MISCELLANEA

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

PORTABILITY

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

Release 0.3.6 - April 15th, 2004

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

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

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

MONTE CARLO FRAMEWORK

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

PRICING ENGINES FRAMEWORK

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

SHORT RATE MODELS

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

VOLATILITY FRAMEWORK

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

OPTIMIZATION FRAMEWORK

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

PATTERNS

- Composite pattern

MATH

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

TEST SUITE

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

MISCELLANEA

- Inner namespaces have been deprecated.

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

Release 0.3.4 - November 21th, 2003

MONTE CARLO FRAMEWORK

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

PRICING ENGINES FRAMEWORK

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

FIXED INCOME

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

SHORT RATE MODELS

- An additional constraint can now be passed to the calibration

VOLATILITY FRAMEWORK

- Visitable volatility term structures

OPTIMIZATION FRAMEWORK

- Added composite constraint

PATTERNS

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

MATH

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

TEST SUITE

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

MISCELLANEA

- Conditionally allowed negative yields (disabled by default)
- Null calendar and simple day counter for reproducing theoretical calculations
- Fixed for VC++.Net compilation
- Added spec file for RPMs
- Added global flag for early/late payments
- Enabled test suite for Borland
- Removed OnTheEdge VC++ configurations
- Added VC++ configurations for static and dynamic Multithread libraries
- Upgraded to use Doxygen 1.3.4

Release 0.3.3 - September 3rd, 2003

MONTE CARLO FRAMEWORK

- Re-templated Monte Carlo model based on traits.
- New path generator based on DiffusionProcess, TimeGrid, and externally initialized random number generator.
- Added Halton low discrepancy sequence.
- Added sequence generators: random sequence generator creates a sequence generator out of a random number generator. InvCumGaussianRsg creates a gaussian sequence generator out of a uniform (random or low discrepancy) sequence generator.
- RNG as constructor input constructor(long seed) deprecated.

- Mersenne Twister random number generator added
- Old PathPricers, PathGenerators, etc are available with a trailing `_old`
- Added Jäckel's Brownian Bridge (not used yet.)
- Sobol Random Sequence Generator. Unit and Jäckel.
- Added randomized Halton sequences.

FINITE DIFFERENCE FRAMEWORK

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

LATTICE FRAMEWORK

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

PRICING ENGINES FRAMEWORK

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

YIELD TERM STRUCTURE

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

FIXED INCOME

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

OPTIMIZATION FRAMEWORK

- Solvers now take any function. ObjectiveFunction disappeared.

PATTERNS

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

DATE AND CALENDARS

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

MATH

- Added Gauss-Kronrod integration algorithm.
- Added primitive polynomial modulo 2 up to dimension 18 (available up to dimension 27.)
- Added BicubicSplineInterpolation.
- Numerical Recipes algorithm is back since there is a problem with Nicolas' code: it is unable to fit a straight line, it waves around the line.
- Prime number generation.
- Acklam's approximation for inverse cumulative normal distribution function (replaced Moro's algorithm as default.)
- Added error function.
- Improved Cumulative Normal Distribution function using the error function.
- Matrix pseudo square algorithm using salvaging algorithm(s).
- Added SequenceStatistics.
- Major Statistic reworking.
- Added DiscrepancyStatistic that inherits from SequenceStatistic and extends it with the calculation of L2-discrepancy.
- HStatistics.
- Added first and second derivative of cubic splines.

RISK MEASURES

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

MISCELLANEA

- QuEP 9 "generic disposable objects" implemented.

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

Release 0.3.1 - February 4th, 2003

FINITE DIFFERENCE FRAMEWORK

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

VOLATILITY FRAMEWORK

- added Black and local volatility interface

PRICING ENGINES FRAMEWORK

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

YIELD TERM STRUCTURE

- interface revisited
- added discrete time forward methods
- added `DiscountCurve` (loglinear interpolated) and `CompoundForward` term structures
- `ForwardSpreadedTermStructure` moved under `QuantLib::TermStructures` namespace

FIXED INCOME

- Modified coupons so that the payment date can be after the end of the accrual period

MISCELLANEA

- added/verified holidays of many calendars
- added new calendars

- added new currencies
- more date formatters
- added `Period(std::string&)`
- it is now possible to advance a calendar using a `Period`
- added `LogLinear Interpolation`
- the `allowExtrapolation` boolean in interpolation classes has been removed from constructors and added to the `operator()`
- Renamed `Solver1D::lowBound` and `hiBound`
- bug fixes

BUILD PROCESS

- More autoconfiscated time functions and types
- Migrated to latest autotools
- added patches for Darwin and Solaris

Release 0.3.0 - May 6th, 2002

MONTE CARLO FRAMEWORK

- `Path` and `MultiPath` are time-aware
- `McPricer`: extended interface, improved convergency algorithm

FINITE DIFFERENCE FRAMEWORK

- added mixed (implicit/explicit) scheme, from which `Crank-Nicolson`, `ImplicitEuler`, and `ExplicitEuler` are now derived
- Finite Difference exercise conditions are now in the `FiniteDifferences` folder/namespace
- Finite Difference pricers now start with 'Fd' letters
- `BSMNumericalOption` became `BsmFdOption`

LATTICE FRAMEWORK

- introduced first version of the framework
- CRR and JR binomial trees

VOLATILITY FRAMEWORK

- early works on reorganization of vol structures

YIELD TERM STRUCTURE

- new `TermStructure` class based on affine model

- yield curves can be spreaded in term of zeros (ZeroSpreadedTermStructure) and forwards (ForwardSpreadedTermStructure)
- Added dates() and times() to PiecewiseFlatForward
- discount factor accuracy in the yield curve bootstrapping is an input
- added single factor short-rate models (Hull-White, Black-Karasinski)
- added two factor short-rate models framework
- cap/floor and swaption calibration helpers
- added bermudan swaption pricing example (including BK and HW calibrations)

FIXED INCOME

- cap/floor and swaption tree pricer
- cap/floor analytical pricer
- vanilla swaption Jamshidian pricer
- Added accruedAmount() to coupons
- Made cash flow vector builders into functions

OPTIMIZATION FRAMEWORK

- added conjugate gradient, simplex

PATTERNS

- implemented QuEP 8 and 10

MISCELLANEA

- added allowExtrapolation parameter to interpolaton classes
- added 2D bilinear interpolation
- better spline interpolation algorithm
- Added non-central chi-square distribution function.
- Improved Inverse Cumulative Normal Distribution using Moro's algorithm
- Introduced class representing stochastic processes
- added isExpired() to Instrument interface
- added functions folder and namespace for QuantLibXL and any other function-like interface to QuantLib
- Handle is now castable to an Handle of a compatible type
- added downsideVariance to the Statistics class
- kurtosis() and skewness() now handles the case of stddev == 0 and/or variance == 0

- added Correlation Matrix to MultiVariateAccumulator
- enforced MS VC compilation settings
- added "-debug" to the QL_VERSION version string `ifdef QL_DEBUG`
- "make check" runs the example programs under Borland C++
- fixed compilation with "g++ -pedantic"
- Spread as market element
- new calendars introduced
- new Xibor Indexes introduced
- Added optional day count to libor indexes
- Shortened file names within 31 char limit to support HFS

Release 0.2.1 - December 3rd, 2001

MONTE CARLO FRAMEWORK

- Path and MultiPath are now classes on their own
- PathPricer now handles both Path and MultiPath
- MonteCarloModel now handles both single factor and multi factors simulations.
- McPricer now handles both single factor and multi factors pricing. New pricing interface
- antithetic variance-reduction technique made possible in Monte Carlo for both single factor and multi factors
- Control Variate specific class removed: control variation technique is now handled by the general MC model
- average price and average strike asian option refactored
- Sample as a (value,weight) struct
- random number generators moved under RandomNumbers folder and namespace

FINITE DIFFERENCE FRAMEWORK

- BackwardEuler and ForwardEuler renamed ImplicitEuler and ExplicitEuler, respectively
- refactoring of TridiagonalOperator and derived classes

YIELD TERM STRUCTURE AND FIXED INCOME

- Added some useful methods to term structure classes
- Allowed passing a quote to RateHelpers as double
- added FuturesRateHelpers (no convexity adjustment yet)
- PiecewiseFlatForward now observer of rates passed as MarketElements
- Unified Date and Time interface in TermStructure

- Added BPS to generic swap legs
- added term_structure+swap example
- Fixing days introduced for floating-coupon bond

PATTERNS

- Added factory pattern
- Calendar and DayCounter now use the Strategy pattern

VARIOUS

- used do-while-false idiom in QL_REQUIRE-like macros
- now using size_t where appropriate
- dividendYield is now a Spread instead of a Rate (that is: cost of carry is allowed)
- RelinkableHandle initialized with an optional Handle
- Worked around VC++ problems in History constructor
- added QL_VERSION and QL_HEX_VERSION
- generic bug fixes
- removed classes deprecated in 0.2.0

INSTALLATION FACILITIES

- improved and smoother Win32 binary installer

DOCUMENTATION

- general re-hauling
- improved and extended Monte Carlo documentation
- improved and extended examples
- Upgraded to Doxygen 1.2.11.1
- Added man pages for installed executables
- added docs in Windows Help format
- added info on "Win32 OnTheEdgeRelease" and "Win32 OnTheEdgeDebug" MS VC++ configurations
- additional information on how to create a MS VC++ project based on QuantLib

Release 0.2.0 - September 18th, 2001

- Library:
 - source code moved under ql, better GNU standards
 - gcc build dir can now be separated from source tree

- gcc 3.0.1 port
- clean compilation (no warnings)
- bootstrap script on cygwin
- Fixed automatic choice of seed for random number generators
- Actual/actual classes
- extended platform support (see table in documentation)
- antithetic variance-reduction technique made possible in Monte Carlo
- added dividend-Rho greek
- First implementation of segment integral (to be redesigned)
- Knuth random generator
- Cash flows, scheduler, and swap (both generic and simple) added
- added ICGaussian random generator
- generic bug fixes
- Installation facilities:
 - improved and smoother Win32 binary installer
 - better distribution
 - debian packages available
- Documentation:
 - general re-hauling
 - added examples of using QuantLib and of projects based on QL

Release 0.1.9 - May 31st, 2001

- Library:
 - Style guidelines introduced (see <http://quantlib.org/style.shtml>) and partially enforced
 - full support for Microsoft Visual Studio
 - full support for Linux/gcc
 - momentarily broken support for Metrowerks CodeWarrior
 - autoconfiscation (with specialized config*.hpp files for platforms without automake/autoconf support)
 - Include files moved under Include/ql folder and referenced as "ql/header.hpp"
 - Implemented expression templates techniques for array algebra optimization
 - Added custom iterators
 - Improved term structure
 - Added Asian, Bermudan, Shout, Cliquet, Himalaya, and Barrier options (all with greeks calculation, control variated where possible)
 - Added Helsinki and Wellington calendars
 - Improved Normal distribution related functions: cumulative, inverse cumulative, etc.
 - Added uniform and Gaussian random number generators
 - Added Statistics class (mean, variance, skewness, downside variance, etc.)

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

Release 0.1.1 - November 21st, 2000

Initial release.

1.8 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.

1.9 The QuantLib Group

1.9.1 Authors

The QuantLib Group members are:

- Ferdinando Ametrano, Banca Caboto SpA, administrator
- Luigi Ballabio, StatPro Italia srl, administrator
- Mario Aleppo, StatPro Italia srl
- Marco Bianchetti, Banca Caboto SpA
- Nicolas Di Césaré
- Cristina Duminuco, Banca Caboto SpA
- Dirk Eddelbuettel
- Giorgio Facchinetti, Banca Caboto SpA
- Neil Firth, Mathematical Institute, University of Oxford
- Chiara Fornarola, Banca Caboto SpA
- Nicola Jean, StatPro Italia srl
- Katuscia Manzoni, Banca Caboto SpA
- Marco Marchioro, StatPro Italia srl
- Mario Pucci, Banca Caboto SpA
- Sadruddin Rejeb
- Enrico Sirola, StatPro Italia srl
- Klaus Spanderen
- Niels Elken Sønderby
- François du Vignaud, Banca Caboto SpA
- Joseph Wang
- François du Vignaud, Banca Caboto SpA

1.9.2 Contributors

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

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

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

1.10 QuantLib License

QuantLib is

Copyright (C) 2000, 2001, 2002, 2003 RiskMap srl
Copyright (C) 2003, 2004, 2005, 2006, 2007 StatPro Italia srl
Copyright (C) 2002, 2003, 2004, 2005, 2006, 2007 Ferdinando Ametrano

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

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

Copyright (C) 2003, 2004, 2007 Neil Firth
Copyright (C) 2003, 2004 Roman Gitlin
Copyright (C) 2003 Niels Elken Søndersby
Copyright (C) 2003 Kawanishi Tomoya

Copyright (C) 2004 FIMAT Group
Copyright (C) 2004 M-Dimension Consulting Inc.
Copyright (C) 2004 Mike Parker
Copyright (C) 2004 Walter Penschke
Copyright (C) 2004 Gianni Piolanti
Copyright (C) 2004, 2005, 2006, 2007 Klaus Spanderen
Copyright (C) 2004 Jeff Yu

Copyright (C) 2005, 2006 Toyin Akin
Copyright (C) 2005 Sercan Atalik
Copyright (C) 2005, 2006 Theo Boafu
Copyright (C) 2005, 2006 Piter Dias
Copyright (C) 2005 Gary Kennedy
Copyright (C) 2005, 2006, 2007 Joseph Wang
Copyright (C) 2005 Charles Whitmore

Copyright (C) 2006, 2007 Banca Profilo S.p.A.
Copyright (C) 2006, 2007 Marco Bianchetti
Copyright (C) 2006 Yiping Chen
Copyright (C) 2006, 2007 Warren Chou
Copyright (C) 2006, 2007 Cristina Duminuco
Copyright (C) 2006, 2007 Giorgio Facchinetti
Copyright (C) 2006, 2007 Chiara Fornarola
Copyright (C) 2006 Silvia Frasson
Copyright (C) 2006 Richard Gould
Copyright (C) 2006, 2007 Mark Joshi
Copyright (C) 2006 Allen Kuo
Copyright (C) 2006 Roland Lichters
Copyright (C) 2006, 2007 Katiuscia Manzoni
Copyright (C) 2006, 2007 Mario Pucci
Copyright (C) 2006 François du Vignaud

Copyright (C) 2007 Affine Group Limited

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

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

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

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

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

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

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

1.10.1 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	101
Currencies and FX rates	104
Date and time calculations	108
Calendars	111
Day counters	114
Pricing engines	115
Asian option engines	116
Barrier option engines	117
Basket option engines	118
Cap/floor engines	119
Cliquet option engines	120
Forward option engines	121
Quanto option engines	122
Swaption engines	123
Vanilla option engines	124
Finite-differences framework	128
Short-rate modelling framework	130
Financial instruments	133
Lattice methods	137
Math tools	140
Monte Carlo framework	142
Design patterns	143
Stochastic processes	144
Term structures	146
Utilities	148
QuantLib macros	150
Generic macros	151
Numeric limits	152
Template capabilities	153
Iterator support	154
Debugging macros	157
Output manipulators	155

Chapter 3

QuantLib Namespace Index

3.1 QuantLib Namespace List

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

[std](#) (STL namespace) 161

Chapter 4

QuantLib Hierarchical Index

4.1 QuantLib Class Hierarchy

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

Abcd	163
AbcdFunction	166
AccountingEngine	169
AcyclicVisitor	173
AmericanCondition	176
AmericanPayoffAtExpiry	178
AmericanPayoffAtHit	179
AnalyticDigitalAmericanEngine	186
Array	195
AssetSwap::arguments	203
AssetSwap::results	204
Average	209
BackwardFlat	210
Barrier	213
BarrierOption::arguments	216
BasketOption::arguments	219
Bicubic	228
Bilinear	230
BinomialConvertibleEngine	232
BinomialDistribution	233
BinomialVanillaEngine	235
BivariateCumulativeNormalDistributionDr78	237
BivariateCumulativeNormalDistributionWe04DP	238
BlackCalculator	240
BlackScholesCalculator	250
BoundaryCondition	271
BoundaryCondition< TridiagonalOperator >	271
DirichletBC	415
NeumannBC	980
BoxMullerGaussianRng	274
BrownianBridge	279
Calendar	285
Argentina	192

Australia	208
Brazil	275
Canada	298
China	325
CzechRepublic	399
Denmark	411
Finland	642
Germany	746
HongKong	759
Hungary	770
Iceland	774
India	792
Indonesia	794
Italy	828
Japan	832
JointCalendar	835
Mexico	954
NewZealand	984
Norway	991
NullCalendar	994
Poland	1039
SaudiArabia	1080
Singapore	1098
Slovakia	1107
SouthAfrica	1117
SouthKorea	1118
Sweden	1168
Switzerland	1169
Taiwan	1172
TARGET	1174
Turkey	1210
Ukraine	1216
UnitedKingdom	1218
UnitedStates	1220
Calendar::Impl	289
Calendar::OrthodoxImpl	290
Calendar::WesternImpl	291
Callability::Price	297
CapFloor::arguments	302
CashFlows	317
CLGaussianRng	326
CliquetOption::arguments	329
Clone	331
CMSMMDriftCalculator	337
Composite	342
ConjugateGradient	349
ConstantEstimator	350
Constraint	353
BoundaryConstraint	273
CompositeConstraint	343
NoConstraint	986
PositiveConstraint	1040
Constraint::Impl	354

ContinuousAveragingAsianOption::arguments	357
ContinuousFixedLookbackOption::arguments	360
ContinuousFloatingLookbackOption::arguments	363
ConvergenceStatistics	369
CostFunction	376
LeastSquareFunction	848
ProjectedCostFunction	1045
CovarianceDecomposition	380
CoxIngersollRoss::Dynamics	383
Cubic	387
CumulativeBinomialDistribution	390
CumulativeNormalDistribution	391
CumulativePoissonDistribution	392
CuriouslyRecurringTemplate	393
Solver1D	1115
TreeLattice	1193
TreeLattice1D	1195
TreeLattice2D	1196
CuriouslyRecurringTemplate< AdditiveEQPBinomialTree >	393
Tree< AdditiveEQPBinomialTree >	1191
BinomialTree< AdditiveEQPBinomialTree >	234
EqualProbabilitiesBinomialTree< AdditiveEQPBinomialTree >	456
AdditiveEQPBinomialTree	174
CuriouslyRecurringTemplate< Bisection >	393
Solver1D< Bisection >	1115
Bisection	236
CuriouslyRecurringTemplate< BlackScholesLattice< T > >	393
TreeLattice< BlackScholesLattice< T > >	1193
TreeLattice1D< BlackScholesLattice< T > >	1195
BlackScholesLattice	252
TsiveriotisFernandesLattice	1207
CuriouslyRecurringTemplate< Brent >	393
Solver1D< Brent >	1115
Brent	277
CuriouslyRecurringTemplate< CoxRossRubinstein >	393
Tree< CoxRossRubinstein >	1191
BinomialTree< CoxRossRubinstein >	234
EqualJumpsBinomialTree< CoxRossRubinstein >	455
CoxRossRubinstein	384
CuriouslyRecurringTemplate< FalsePosition >	393
Solver1D< FalsePosition >	1115
FalsePosition	633
CuriouslyRecurringTemplate< JarrowRudd >	393
Tree< JarrowRudd >	1191
BinomialTree< JarrowRudd >	234
EqualProbabilitiesBinomialTree< JarrowRudd >	456
JarrowRudd	833
CuriouslyRecurringTemplate< Joshi4 >	393
Tree< Joshi4 >	1191
BinomialTree< Joshi4 >	234

CuriouslyRecurringTemplate< LeisenReimer >	393
Tree< LeisenReimer >	1191
BinomialTree< LeisenReimer >	234
LeisenReimer	851
CuriouslyRecurringTemplate< Newton >	393
Solver1D< Newton >	1115
Newton	982
CuriouslyRecurringTemplate< NewtonSafe >	393
Solver1D< NewtonSafe >	1115
NewtonSafe	983
CuriouslyRecurringTemplate< OneFactorModel::ShortRateTree >	393
TreeLattice< OneFactorModel::ShortRateTree >	1193
TreeLattice1D< OneFactorModel::ShortRateTree >	1195
OneFactorModel::ShortRateTree	1013
CuriouslyRecurringTemplate< Ridder >	393
Solver1D< Ridder >	1115
Ridder	1068
CuriouslyRecurringTemplate< Secant >	393
Solver1D< Secant >	1115
Secant	1082
CuriouslyRecurringTemplate< T >	393
Tree	1191
BinomialTree	234
EqualJumpsBinomialTree	455
EqualProbabilitiesBinomialTree	456
CuriouslyRecurringTemplate< Tian >	393
Tree< Tian >	1191
BinomialTree< Tian >	234
Tian	1181
CuriouslyRecurringTemplate< Trigeorgis >	393
Tree< Trigeorgis >	1191
BinomialTree< Trigeorgis >	234
EqualJumpsBinomialTree< Trigeorgis >	455
Trigeorgis	1202
CuriouslyRecurringTemplate< TrinomialTree >	393
Tree< TrinomialTree >	1191
TrinomialTree	1203
CuriouslyRecurringTemplate< TwoFactorModel::ShortRateTree >	393
TreeLattice< TwoFactorModel::ShortRateTree >	1193
TreeLattice2D< TwoFactorModel::ShortRateTree, TrinomialTree >	1196
TwoFactorModel::ShortRateTree	1214
Currency	394
ARSCurrency	199
ATSCurrency	205
AUDCurrency	206
BDTCurrency	224
BEFCurrency	225
BGLCurrency	227
BRLCurrency	278

BYRCurrency	282
CADCurrency	283
CHFCurrency	323
CLPCurrency	333
CNYCurrency	340
COPCurrency	375
CYPCurrency	398
CZKCurrency	401
DEMCurrency	410
DKKCurrency	437
EEKCurrency	452
ESPCurrency	459
EURCurrency	462
FIMCurrency	640
FRFCurrency	689
GBPCurrency	727
GRDCurrency	749
HKDCurrency	758
HUFCurrency	761
IEPCurrency	776
ILSCurrency	777
INRCurrency	796
IQDCurrency	825
IRRCurrency	826
ISKCurrency	827
ITLCurrency	830
JPYCurrency	836
KRWCurrency	841
KWDCurrency	842
LTLCurrency	895
LUFCurrency	896
LVLCurrency	897
MTLCurrency	965
MXNCurrency	977
NLGCurrency	985
NOKCurrency	987
NPRCurrency	992
NZDCurrency	997
PKRCurrency	1035
PLNCurrency	1037
PTECurrency	1046
ROLCurrency	1069
RONCurrency	1070
SARCurrency	1079
SEKCurrency	1085
SGDCurrency	1089
SITCurrency	1105
SKKCurrency	1106
THBCurrency	1179
TRLCurrency	1204
TRYCurrency	1206
TTDCurrency	1209
TWDCurrency	1211
USDCurrency	1225

VEBCurrency	1239
ZARCurrency	1245
Curve	396
CurveState	397
CMSwapCurveState	339
CoterminalSwapCurveState	377
LMMCurveState	875
Date	402
DayCounter	406
Actual360	170
Actual365Fixed	171
ActualActual	172
Business252	281
OneDayCounter	1009
SimpleDayCounter	1093
Thirty360	1180
DayCounter::Impl	408
DecInterpCapletVolStructure	409
Discount	417
DiscreteAveragingAsianOption::arguments	421
DiscretizedAsset	424
DiscretizedDiscountBond	427
DiscretizedOption	428
Disposable	430
DividendVanillaOption::arguments	435
Domain	440
Duration	448
EarlyExercisePathPricer	451
EarlyExercisePathPricer< MultiPath >	451
EarlyExercisePathPricer< Path >	451
EndCriteria	453
ErrorFunction	458
EvolutionDescription	617
exception	
Error	457
ExchangeRate	618
Exercise	622
EarlyExercise	450
AmericanExercise	177
BermudanExercise	226
EuropeanExercise	613
ExtendedCoxIngersollRoss::Dynamics	627
Extrapolator	631
Interpolation	812
BackwardFlatInterpolation	211
CubicSpline	388
MonotonicCubicSpline	960
NaturalCubicSpline	978
NaturalMonotonicCubicSpline	979
ForwardFlatInterpolation	668
LinearInterpolation	865
LogLinearInterpolation	887

SABRInterpolation	1074
Interpolation2D	814
BicubicSpline	229
BilinearInterpolation	231
TermStructure	1175
BlackVolTermStructure	264
BlackVarianceTermStructure	260
BlackVarianceCurve	256
BlackVarianceSurface	258
ImpliedVolTermStructure	784
BlackVolatilityTermStructure	262
BlackConstantVol	244
CapletVolatilityStructure	307
CapletConstantVolatility	305
CapVolatilityStructure	311
CapVolatilityVector	313
LocalVolTermStructure	884
LocalConstantVol	879
LocalVolCurve	880
LocalVolSurface	882
SwaptionVolatilityStructure	1165
SwaptionConstantVolatility	1156
YieldTermStructure	1241
FlatForward	653
ForwardRateStructure	677
CompoundForward	347
ForwardSpreadedTermStructure	679
InterpolatedForwardCurve	808
ImpliedTermStructure	782
InterpolatedDiscountCurve	806
InterpolatedDiscountCurve< LogLinear >	806
ExtendedDiscountCurve	629
ZeroYieldStructure	1251
DriftTermStructure	446
InterpolatedZeroCurve	810
PiecewiseZeroSpreadedTermStructure	1033
QuantoTermStructure	1052
ZeroSpreadedTermStructure	1248
Factorial	632
FaureRsg	634
FDDividendEngineMerton73	636
FDDividendEngineShiftScale	637
FDEuropeanEngine	638
FDStepConditionEngine	639
FiniteDifferenceModel	641
ForwardFlat	667
ForwardOptionArguments	671
ForwardRate	673
GammaFunction	701
Garch11	704
GarmanKlassAbstract	705
GaussianOrthogonalPolynomial	717

GaussHermitePolynomial	714
GaussHyperbolicPolynomial	716
GaussJacobiPolynomial	720
GaussChebyshev2thPolynomial	708
GaussChebyshevPolynomial	710
GaussGegenbauerPolynomial	712
GaussLegendrePolynomial	726
GaussLaguerrePolynomial	724
GaussianQuadrature	718
GaussChebyshev2thIntegration	707
GaussChebyshevIntegration	709
GaussGegenbauerIntegration	711
GaussHermiteIntegration	713
GaussHyperbolicIntegration	715
GaussJacobiIntegration	719
GaussLaguerreIntegration	723
GaussLegendreIntegration	725
GaussKronrodAdaptive	721
GaussKronrodNonAdaptive	722
GeneralStatistics	731
GenericGaussianStatistics	736
GenericRiskStatistics	740
GenericSequenceStatistics	743
DiscrepancyStatistics	418
Greeks	750
MultiAssetOption::results	969
OneAssetOption::results	1006
HaltonRsg	751
Handle	752
RelinkableHandle	1066
HullWhite::Dynamics	764
IMM	778
IncrementalStatistics	786
IntegralEngine	800
InterestRate	801
Interpolation2D::Impl	816
Interpolation2D::templateImpl	817
Interpolation::Impl	818
Interpolation::templateImpl	819
IntervalPrice	820
InverseCumulativeNormal	821
InverseCumulativePoisson	822
InverseCumulativeRng	823
InverseCumulativeRsg	824
KnuthUniformRng	840
Lattice	843
TreeLattice	1193
TreeLattice< BlackScholesLattice< T > >	1193
TreeLattice< OneFactorModel::ShortRateTree >	1193
TreeLattice< TwoFactorModel::ShortRateTree >	1193
LeastSquareProblem	849

LecuyerUniformRng	850
LexicographicalView	853
LfmCovarianceParameterization	855
LfmCovarianceProxy	856
LfmHullWhiteParameterization	857
Linear	864
LinearLeastSquaresRegression	866
LineSearch	867
ArmijoLineSearch	194
LmCorrelationModel	870
LmExponentialCorrelationModel	871
LmLinearExponentialCorrelationModel	873
LMMDriftCalculator	876
LMMNormalDriftCalculator	877
LmVolatilityModel	878
LmConstWrapperVolatilityModel	869
LmLinearExponentialVolatilityModel	874
LmExtLinearExponentialVolModel	872
LogLinear	886
MakeCapFloor	898
MakeCms	899
MakeMCAmericanEngine	900
MakeMCDigitalEngine	901
MakeMCEuropeanEngine	902
MakeMCEuropeanHestonEngine	903
MakeMCHullWhiteCapFloorEngine	904
MakeMCVarianceSwapEngine	905
MakeSchedule	906
MakeVanillaSwap	907
map< Date, Real >	
TimeBasket	1183
MarketModel	908
AbcdVol	168
MarketModelEvolver	912
ConstrainedEvolver	352
LogNormalFwdRateEulerConstrained	891
LogNormalCmSwapRatePc	888
LogNormalCotSwapRatePc	889
LogNormalFwdRateEuler	890
LogNormalFwdRateIpc	892
LogNormalFwdRatePc	893
NormalFwdRatePc	990
MarketModelMultiProduct	914
MarketModelComposite	910
MultiProductComposite	973
SingleProductComposite	1102
MultiProductMultiStep	974
MultiProductOneStep	975
Matrix	915
McPricer	945
McPricer< MultiVariate, PseudoRandom >	945

McEverest	936
McHimalaya	937
McMaxBasket	942
McPagoda	943
McPricer< SingleVariate, PseudoRandom >	945
McCliquetOption	926
McDiscreteArithmeticASO	930
McPerformanceOption	944
McSimulation	946
MCLongstaffSchwartzEngine	940
MCLongstaffSchwartzEngine< BasketOption::engine, MultiVariate, RNG >	940
MCAmericanBasketEngine	920
MCLongstaffSchwartzEngine< VanillaOption::engine, SingleVariate, RNG, S >	940
MCAmericanEngine	921
MCVanillaEngine	948
MCVanillaEngine< MultiVariate, RNG, S >	948
MCEuropeanHestonEngine	935
MCVanillaEngine< SingleVariate, RNG, S >	948
MCDigitalEngine	927
MCEuropeanEngine	934
McSimulation< MultiVariate, RNG, S >	946
MCBasketEngine	924
McSimulation< MultiVariate, RNG, Statistics >	946
McSimulation< SingleVariate, RNG, S >	946
MCBarrierEngine	922
MCDiscreteAveragingAsianEngine	931
MCDiscreteArithmeticAPEngine	928
MCDiscreteGeometricAPEngine	933
MCHullWhiteCapFloorEngine	938
MCVarianceSwapEngine	949
MersenneTwisterUniformRng	951
MixedScheme	955
CrankNicolson	385
ExplicitEuler	623
ImplicitEuler	780
Money	957
MonteCarloModel	961
MoreGreeks	962
OneAssetOption::results	1006
MoroInverseCumulativeNormal	963
MTBrownianGenerator	964
MultiAssetOption::arguments	968
MultiCubicSpline	970
MultiPath	971
MultiPathGenerator	972
MultiVariate	976
NonLinearLeastSquare	988
NormalDistribution	989
Null	993
Observable	999
AffineModel	175

G2	693
LiborForwardModel	860
OneFactorAffineModel	1010
CoxIngersollRoss	381
ExtendedCoxIngersollRoss	625
Vasicek	1236
HullWhite	762
CalibratedModel	292
HestonModel	754
BatesModel	223
LiborForwardModel	860
ShortRateModel	1090
OneFactorModel	1011
BlackKarasinski	247
OneFactorAffineModel	1010
TwoFactorModel	1212
G2	693
CalibrationHelper	294
CapHelper	304
HestonModelHelper	755
SwaptionHelper	1158
Event	615
Callability	296
SoftCallability	1114
CashFlow	315
Coupon	378
FixedRateCoupon	651
FloatingRateCoupon	656
CappedFlooredCoupon	309
CmsCoupon	334
IborCoupon	771
Dividend	431
FixedDividend	645
FractionalDividend	685
SimpleCashFlow	1092
FloatingRateCouponPricer	659
CmsCouponPricer	335
ConundrumPricer	365
ConundrumPricerByBlack	367
ConundrumPricerByNumericalIntegration	368
IborCouponPricer	772
BlackIborCouponPricer	246
Index	789
InterestRateIndex	804
IborIndex	773
Cdor	321
Euribor	463
Euribor10M	464
Euribor11M	465
Euribor1M	466
Euribor1Y	467

Euribor2M	468
Euribor2W	469
Euribor3M	486
Euribor3W	487
Euribor4M	488
Euribor5M	489
Euribor6M	490
Euribor7M	491
Euribor8M	492
Euribor9M	493
EuriborSW	494
Euribor365	470
Euribor365_10M	471
Euribor365_11M	472
Euribor365_1M	473
Euribor365_1Y	474
Euribor365_2M	475
Euribor365_2W	476
Euribor365_3M	477
Euribor365_3W	478
Euribor365_4M	479
Euribor365_5M	480
Euribor365_6M	481
Euribor365_7M	482
Euribor365_8M	483
Euribor365_9M	484
Euribor365_SW	485
EURLibor	546
EURLibor10M	547
EURLibor11M	548
EURLibor1M	549
EURLibor1Y	550
EURLibor2M	551
EURLibor2W	552
EURLibor3M	553
EURLibor4M	554
EURLibor5M	555
EURLibor6M	556
EURLibor7M	557
EURLibor8M	558
EURLibor9M	559
EURLiborSW	560
Jibar	834
Libor	859
AUDLibor	207
CADLibor	284
CHFLibor	324
DKKLibor	438
GBPLibor	728
JPYLibor	837
NZDLibor	998
USDLibor	1226
Tibor	1182

TRLibor	1205
Zibor	1253
SwapIndex	1149
EuriborSwapFixAvs3M	510
EuriborSwapFixA1Y	498
EuriborSwapFixAvs6M	511
EuriborSwapFixA10Y	495
EuriborSwapFixA12Y	496
EuriborSwapFixA15Y	497
EuriborSwapFixA20Y	499
EuriborSwapFixA25Y	500
EuriborSwapFixA2Y	501
EuriborSwapFixA30Y	502
EuriborSwapFixA3Y	503
EuriborSwapFixA4Y	504
EuriborSwapFixA5Y	505
EuriborSwapFixA6Y	506
EuriborSwapFixA7Y	507
EuriborSwapFixA8Y	508
EuriborSwapFixA9Y	509
EuriborSwapFixBvs3M	527
EuriborSwapFixB1Y	515
EuriborSwapFixBvs6M	528
EuriborSwapFixB10Y	512
EuriborSwapFixB12Y	513
EuriborSwapFixB15Y	514
EuriborSwapFixB20Y	516
EuriborSwapFixB25Y	517
EuriborSwapFixB2Y	518
EuriborSwapFixB30Y	519
EuriborSwapFixB3Y	520
EuriborSwapFixB4Y	521
EuriborSwapFixB5Y	522
EuriborSwapFixB6Y	523
EuriborSwapFixB7Y	524
EuriborSwapFixB8Y	525
EuriborSwapFixB9Y	526
EuriborSwapFixIFRvs3M	544
EuriborSwapFixIFR1Y	532
EuriborSwapFixIFRvs6M	545
EuriborSwapFixIFR10Y	529
EuriborSwapFixIFR12Y	530
EuriborSwapFixIFR15Y	531
EuriborSwapFixIFR20Y	533
EuriborSwapFixIFR25Y	534
EuriborSwapFixIFR2Y	535
EuriborSwapFixIFR30Y	536
EuriborSwapFixIFR3Y	537
EuriborSwapFixIFR4Y	538
EuriborSwapFixIFR5Y	539
EuriborSwapFixIFR6Y	540
EuriborSwapFixIFR7Y	541
EuriborSwapFixIFR8Y	542

EuriborSwapFixIFR9Y	543
EurliborSwapFixAvs3M	576
EurliborSwapFixA1Y	564
EurliborSwapFixAvs6M	577
EurliborSwapFixA10Y	561
EurliborSwapFixA12Y	562
EurliborSwapFixA15Y	563
EurliborSwapFixA20Y	565
EurliborSwapFixA25Y	566
EurliborSwapFixA2Y	567
EurliborSwapFixA30Y	568
EurliborSwapFixA3Y	569
EurliborSwapFixA4Y	570
EurliborSwapFixA5Y	571
EurliborSwapFixA6Y	572
EurliborSwapFixA7Y	573
EurliborSwapFixA8Y	574
EurliborSwapFixA9Y	575
EurliborSwapFixBvs3M	593
EurliborSwapFixB1Y	581
EurliborSwapFixBvs6M	594
EurliborSwapFixB10Y	578
EurliborSwapFixB12Y	579
EurliborSwapFixB15Y	580
EurliborSwapFixB20Y	582
EurliborSwapFixB25Y	583
EurliborSwapFixB2Y	584
EurliborSwapFixB30Y	585
EurliborSwapFixB3Y	586
EurliborSwapFixB4Y	587
EurliborSwapFixB5Y	588
EurliborSwapFixB6Y	589
EurliborSwapFixB7Y	590
EurliborSwapFixB8Y	591
EurliborSwapFixB9Y	592
EurliborSwapFixIFRvs3M	610
EurliborSwapFixIFR1Y	598
EurliborSwapFixIFRvs6M	611
EurliborSwapFixIFR10Y	595
EurliborSwapFixIFR12Y	596
EurliborSwapFixIFR15Y	597
EurliborSwapFixIFR20Y	599
EurliborSwapFixIFR25Y	600
EurliborSwapFixIFR2Y	601
EurliborSwapFixIFR30Y	602
EurliborSwapFixIFR3Y	603
EurliborSwapFixIFR4Y	604
EurliborSwapFixIFR5Y	605
EurliborSwapFixIFR6Y	606
EurliborSwapFixIFR7Y	607
EurliborSwapFixIFR8Y	608
EurliborSwapFixIFR9Y	609
LazyObject	846

CmsMarket	336
EurodollarFuturesImpliedStdDevQuote	612
ImpliedStdDevQuote	781
Instrument	797
Bond	267
CmsRateBond	338
ConvertibleBond	370
ConvertibleFixedCouponBond	372
ConvertibleFloatingRateBond	373
ConvertibleZeroCouponBond	374
FixedRateBond	647
FloatingRateBond	655
ZeroCouponBond	1247
CapFloor	300
Cap	299
Collar	341
Floor	661
CompositeInstrument	344
Forward	663
FixedRateBondForward	648
ForwardRateAgreement	674
Option	1016
MultiAssetOption	966
BasketOption	218
OneAssetOption	1002
ContinuousFloatingLookbackOption	362
OneAssetStrikedOption	1007
BarrierOption	214
CliquetOption	327
ContinuousAveragingAsianOption	355
ContinuousFixedLookbackOption	359
DiscreteAveragingAsianOption	419
VanillaOption	1227
DividendVanillaOption	433
EuropeanOption	614
ForwardVanillaOption	683
QuantoVanillaOption	1054
QuantoForwardVanillaOption	1049
Swaption	1152
Stock	1139
Swap	1147
AssetSwap	201
VarianceSwap	1231
PiecewiseYieldCurve	1031
SwaptionVolatilityCube	1159
SwaptionVolatilityMatrix	1162
MarketModelFactory	913
PricingEngine	1041
GenericEngine	735
GenericModelEngine	739
MCLongstaffSchwartzEngine	940
GenericEngine< Arguments, Results >	735

GenericModelEngine< ShortRateModel, Arguments, Results >	739
LatticeShortRateModelEngine	845
GenericEngine< BarrierOption::arguments, BarrierOption::results >	735
BarrierOption::engine	217
AnalyticBarrierEngine	180
MCBarrierEngine	922
GenericEngine< BasketOption::arguments, BasketOption::results >	735
BasketOption::engine	220
MCBasketEngine	924
MCLongstaffSchwartzEngine< BasketOption::engine, MultiVariate, RNG >	940
StulzEngine	1141
GenericEngine< Bond::arguments, Bond::results >	735
GenericEngine< CapFloor::arguments, CapFloor::results >	735
CapFloor::engine	303
BlackCapFloorEngine	243
MarketModelCapFloorEngine	909
MCHullWhiteCapFloorEngine	938
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >	739
AnalyticCapFloorEngine	181
GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results >	739
LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >	845
TreeCapFloorEngine	1192
GenericEngine< CliquetOption::arguments, CliquetOption::results >	735
CliquetOption::engine	330
AnalyticCliquetEngine	182
AnalyticPerformanceEngine	191
GenericEngine< ContinuousAveragingAsianOption::arguments, ContinuousAveragingAsianOption::results >	735
ContinuousAveragingAsianOption::engine	358
AnalyticContinuousGeometricAveragePriceAsianEngine	185
GenericEngine< ContinuousFixedLookbackOption::arguments, ContinuousFixedLookbackOption::results >	735
ContinuousFixedLookbackOption::engine	361
AnalyticContinuousFixedLookbackEngine	183
GenericEngine< ContinuousFloatingLookbackOption::arguments, ContinuousFloatingLookbackOption::results >	735
ContinuousFloatingLookbackOption::engine	364
AnalyticContinuousFloatingLookbackEngine	184
GenericEngine< ConvertibleBond::option::arguments, ConvertibleBond::option::results >	735
GenericEngine< DiscreteAveragingAsianOption::arguments, DiscreteAveragingAsianOption::results >	735
DiscreteAveragingAsianOption::engine	422
AnalyticDiscreteGeometricAveragePriceAsianEngine	187
MCDiscreteAveragingAsianEngine	931
GenericEngine< DividendVanillaOption::arguments, DividendVanillaOption::results >	735
DividendVanillaOption::engine	436
AnalyticDividendEuropeanEngine	188
GenericEngine< ForwardOptionArguments< ArgumentsType >, ResultsType >	735

ForwardEngine	666
ForwardPerformanceEngine	672
GenericEngine< OneAssetOption::arguments, OneAssetOption::results >	735
GenericEngine< QuantoOptionArguments< ArgumentsType >, QuantoOptionResults< ResultsType > >	735
QuantoEngine	1047
GenericEngine< Swaption::arguments, Swaption::results >	735
GenericModelEngine< G2, Swaption::arguments, Swaption::results >	739
G2SwaptionEngine	700
GenericModelEngine< LiborForwardModel, Swaption::arguments, Swaption::results >	739
LfmSwaptionEngine	858
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swaption::results >	739
JamshidianSwaptionEngine	831
GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results >	739
LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >	845
TreeSwaptionEngine	1197
Swaption::engine	1155
BlackSwaptionEngine	255
GenericEngine< VanillaOption::arguments, VanillaOption::results >	735
GenericModelEngine< HestonModel, VanillaOption::arguments, VanillaOption::results >	739
AnalyticHestonEngine	190
BatesEngine	221
GenericEngine< VanillaSwap::arguments, VanillaSwap::results >	735
GenericModelEngine< ShortRateModel, VanillaSwap::arguments, VanillaSwap::results >	739
LatticeShortRateModelEngine< VanillaSwap::arguments, VanillaSwap::results >	845
TreeVanillaSwapEngine	1198
GenericEngine< VarianceSwap::arguments, VarianceSwap::results >	735
VarianceSwap::engine	1234
MCVarianceSwapEngine	949
ReplicatingVarianceSwapEngine	1067
Quote	1056
CompositeQuote	346
DerivedQuote	414
EurodollarFuturesImpliedStdDevQuote	612
ForwardValueQuote	682
FuturesConvAdjustmentQuote	690
ImpliedStdDevQuote	781
SimpleQuote	1095
RateHelper	1063
FixedCouponBondHelper	643
FuturesRateHelper	692
RelativeDateRateHelper	1065
DepositRateHelper	412
FraRateHelper	687
SwapRateHelper	1150
SmileSection	1109

StochasticProcess	1129
ForwardMeasureProcess	669
G2ForwardProcess	696
G2Process	698
HestonProcess	756
LiborForwardModelProcess	862
StochasticProcess1D	1132
ForwardMeasureProcess1D	670
HullWhiteForwardProcess	766
GeneralizedBlackScholesProcess	729
BlackProcess	249
BlackScholesMertonProcess	253
BlackScholesProcess	254
GarmanKohlagenProcess	706
GeometricBrownianMotionProcess	745
HullWhiteProcess	768
Merton76Process	952
OrnsteinUhlenbeckProcess	1019
SquareRootProcess	1120
StochasticProcessArray	1136
TermStructure	1175
TermStructureConsistentModel	1177
BlackKarasinski	247
ExtendedCoxIngersollRoss	625
G2	693
HullWhite	762
ObservableValue	1000
ObservableValue< Date >	1000
Observer	1001
BlackCapFloorEngine	243
BlackSwaptionEngine	255
CalibratedModel	292
CalibrationHelper	294
CompositeQuote	346
DerivedQuote	414
FloatingRateCoupon	656
FloatingRateCouponPricer	659
ForwardValueQuote	682
FuturesConvAdjustmentQuote	690
GenericModelEngine	739
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >	739
GenericModelEngine< G2, Swaption::arguments, Swaption::results >	739
GenericModelEngine< HestonModel, VanillaOption::arguments, VanillaOption::results >	739
GenericModelEngine< LiborForwardModel, Swaption::arguments, Swaption::results >	739
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swaption::results >	739
GenericModelEngine< ShortRateModel, Arguments, Results >	739
GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results >	739
GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results >	739
GenericModelEngine< ShortRateModel, VanillaSwap::arguments, VanillaSwap::results >	739

InterestRateIndex	804
LazyObject	846
MarketModelCapFloorEngine	909
MCHullWhiteCapFloorEngine	938
RateHelper	1063
StochasticProcess	1129
TermStructure	1175
OneAssetOption::arguments	1005
OneFactorModel::ShortRateDynamics	1012
BlackKarasinski::Dynamics	248
OperatorFactory	1014
OptimizationMethod	1015
LevenbergMarquardt	852
Simplex	1096
Option::arguments	1018
Parameter	1021
ConstantParameter	351
NullParameter	996
PiecewiseConstantParameter	1030
TermStructureFittingParameter	1178
ExtendedCoxIngersollRoss::FittingParameter	628
G2::FittingParameter	695
HullWhite::FittingParameter	765
Parameter::Impl	1022
Path	1023
PathGenerator	1024
PathPricer	1025
PathPricer< MultiPath >	1025
PathPricer< Path >	1025
PathPricer< PathType >	1025
LongstaffSchwartzPathPricer	894
Payoff	1026
DoubleStickyRatchetPayoff	441
RatchetMaxPayoff	1060
RatchetMinPayoff	1061
RatchetPayoff	1062
StickyMaxPayoff	1126
StickyMinPayoff	1127
StickyPayoff	1128
ForwardTypePayoff	681
TypePayoff	1215
FloatingTypePayoff	660
StrikedTypePayoff	1140
AssetOrNothingPayoff	200
CashOrNothingPayoff	320
GapPayoff	702
PercentageStrikePayoff	1027
PlainVanillaPayoff	1036
SuperFundPayoff	1142
SuperSharePayoff	1143
Period	1028
PoissonDistribution	1038

PrimeNumbers	1042
Problem	1043
QuantoOptionArguments	1050
QuantoOptionResults	1051
RandomizedLDS	1057
RandomSequenceGenerator	1059
Rounding	1071
CeilingTruncation	322
ClosestRounding	332
DownRounding	443
FloorTruncation	662
UpRounding	1224
SABR	1073
SalvagingAlgorithm	1075
Sample	1076
SampledCurve	1077
Schedule	1081
SegmentIntegral	1084
Settlement	1088
ShoutCondition	1091
SimpleLocalEstimator	1094
SingleAssetOption	1100
DiscreteGeometricASO	423
Singleton	1103
Singleton< ExchangeRateManager >	1103
ExchangeRateManager	620
Singleton< IndexManager >	1103
IndexManager	791
Singleton< ProblemData >	1103
Singleton< SeedGenerator >	1103
SeedGenerator	1083
Singleton< Settings >	1103
Settings	1086
Singleton< Tracing >	1103
SingleVariate	1104
SMMDriftCalculator	1110
SobolBrownianGenerator	1111
SobolRsg	1112
StatsHolder	1121
SteepestDescent	1122
step_iterator	1123
StepCondition	1124
NullCondition	995
ZeroCondition	1246
StepConditionSet	1125
StochasticProcess1D::discretization	1134
EulerDiscretization	460
StochasticProcess::discretization	1135
EulerDiscretization	460
Surface	1145
SVD	1146

Swaption::arguments	1154
SymmetricSchurDecomposition	1170
TabulatedGaussLegendre	1171
TimeGrid	1184
TimeSeries	1186
TqrEigenDecomposition	1188
TransformedGrid	1189
TrapezoidIntegral	1190
SimpsonIntegral	1097
TridiagonalOperator	1199
BSMOperator	280
DMinus	439
DPlus	444
DPlusDMinus	445
DZero	449
TridiagonalOperator::TimeSetter	1201
TwoFactorModel::ShortRateDynamics	1213
UpperBoundEngine	1223
VanillaOption::engine	1228
AnalyticEuropeanEngine	189
BaroneAdesiWhaleyApproximationEngine	212
Bjerk SundStenslandApproximationEngine	239
FDBermudanEngine	635
JumpDiffusionEngine	838
JuQuadraticApproximationEngine	839
MCLongstaffSchwartzEngine< VanillaOption::engine, SingleVariate, RNG, S >	940
MCVanillaEngine< MultiVariate, RNG, S >	948
MCVanillaEngine< SingleVariate, RNG, S >	948
VanillaSwap::arguments	1229
VanillaSwap::results	1230
VarianceSwap::arguments	1233
VarianceSwap::results	1235
Vasicek::Dynamics	1238
Visitor	1240
ZeroYield	1250

Chapter 5

QuantLib Class Index

5.1 QuantLib Class List

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

Abcd (Abcd functional form for instantaneous volatility)	163
AbcdFunction (Abcd functional form for instantaneous volatility)	166
AbcdVol (Abcd-interpolated volatility structure)	168
AccountingEngine (Engine collecting cash flows along a market-model simulation) . .	169
Actual360 (Actual/360 day count convention)	170
Actual365Fixed (Actual/365 (Fixed) day count convention)	171
ActualActual (Actual/Actual day count)	172
AcyclicVisitor (Degenerate base class for the Acyclic Visitor pattern)	173
AdditiveEQPBinomialTree (Additive equal probabilities binomial tree)	174
AffineModel (Affine model class)	175
AmericanCondition (American exercise condition)	176
AmericanExercise (American exercise)	177
AmericanPayoffAtExpiry (Analytic formula for American exercise payoff at-expiry op- tions)	178
AmericanPayoffAtHit (Analytic formula for American exercise payoff at-hit options) .	179
AnalyticBarrierEngine (Pricing engine for barrier options using analytical formulae) .	180
AnalyticCapFloorEngine (Analytic engine for cap/floor)	181
AnalyticCliquetEngine (Pricing engine for Cliquet options using analytical formulae) .	182
AnalyticContinuousFixedLookbackEngine (Pricing engine for European continuous fixed-strike lookback)	183
AnalyticContinuousFloatingLookbackEngine (Pricing engine for European continuous floating-strike lookback)	184
AnalyticContinuousGeometricAveragePriceAsianEngine (Pricing engine for European continuous geometric average price Asian)	185
AnalyticDigitalAmericanEngine (Analytic pricing engine for American vanilla options with digital payoff)	186
AnalyticDiscreteGeometricAveragePriceAsianEngine (Pricing engine for European dis- crete geometric average price Asian)	187
AnalyticDividendEuropeanEngine (Analytic pricing engine for European options with discrete dividends)	188
AnalyticEuropeanEngine (Pricing engine for European vanilla options using analytical formulae)	189
AnalyticHestonEngine (Analytic Heston-model engine based on Fourier transform) . .	190

AnalyticPerformanceEngine (Pricing engine for performance options using analytical formulae)	191
Argentina (Argentinian calendars)	192
ArmijoLineSearch (Armijo line search)	194
Array (1-D array used in linear algebra)	195
ARSCurrency (Argentinian peso)	199
AssetOrNothingPayoff (Binary asset-or-nothing payoff)	200
AssetSwap (Bullet bond vs Libor swap)	201
AssetSwap::arguments (Arguments for asset swap calculation)	203
AssetSwap::results (Results from simple swap calculation)	204
ATSCurrency (Austrian shilling)	205
AUDCurrency (Australian dollar)	206
AUDLibor (AUD LIBOR rate)	207
Australia (Australian calendar)	208
Average (Placeholder for enumerated averaging types)	209
BackwardFlat (Backward-flat interpolation factory and traits)	210
BackwardFlatInterpolation (Backward-flat interpolation between discrete points)	211
BaroneAdesiWhaleyApproximationEngine (Barone-Adesi and Whaley pricing engine for American options (1987))	212
Barrier (Placeholder for enumerated barrier types)	213
BarrierOption (Barrier option on a single asset)	214
BarrierOption::arguments (Arguments for barrier option calculation)	216
BarrierOption::engine (Barrier-option engine base class)	217
BasketOption (Basket option on a number of assets)	218
BasketOption::arguments (Arguments for basket option calculation)	219
BasketOption::engine (Basket-option engine base class)	220
BatesEngine (Bates model engines based on Fourier transform)	221
BatesModel (Bates stochastic-volatility model)	223
BDTCurrency (Bangladesh taka)	224
BEFCurrency (Belgian franc)	225
BermudanExercise (Bermudan exercise)	226
BGLCurrency (Bulgarian lev)	227
Bicubic (Bicubic-spline-interpolation factory)	228
BicubicSpline (Bicubic-spline interpolation between discrete points)	229
Bilinear (Bilinear-interpolation factory)	230
BilinearInterpolation (bilinear interpolation between discrete points)	231
BinomialConvertibleEngine (Binomial Tsiveriotis-Fernandes engine for convertible bonds)	232
BinomialDistribution (Binomial probability distribution function)	233
BinomialTree (Binomial tree base class)	234
BinomialVanillaEngine (Pricing engine for vanilla options using binomial trees)	235
Bisection (Bisection 1-D solver)	236
BivariateCumulativeNormalDistributionDr78 (Cumulative bivariate normal distribution function)	237
BivariateCumulativeNormalDistributionWe04DP (Cumulative bivariate normal distribution function (West 2004))	238
Bjerk SundStenslandApproximationEngine (Bjerk Sund and Stensland pricing engine for American options (1993))	239
BlackCalculator (Black 1976 calculator class)	240
BlackCapFloorEngine (Black-formula cap/floor engine)	243
BlackConstantVol (Constant Black volatility, no time-strike dependence)	244
BlackIborCouponPricer (Black-formula pricer for capped/floored Ibor coupons)	246
BlackKarasinski (Standard Black-Karasinski model class)	247
BlackKarasinski::Dynamics (Short-rate dynamics in the Black-Karasinski model)	248

BlackProcess (Black (1976) stochastic process)	249
BlackScholesCalculator (Black-Scholes 1973 calculator class)	250
BlackScholesLattice (Simple binomial lattice approximating the Black-Scholes model)	252
BlackScholesMertonProcess (Merton (1973) extension to the Black-Scholes stochastic process)	253
BlackScholesProcess (Black-Scholes (1973) stochastic process)	254
BlackSwaptionEngine (Black-formula swaption engine)	255
BlackVarianceCurve (Black volatility curve modelled as variance curve)	256
BlackVarianceSurface (Black volatility surface modelled as variance surface)	258
BlackVarianceTermStructure (Black variance term structure)	260
BlackVolatilityTermStructure (Black-volatility term structure)	262
BlackVolTermStructure (Black-volatility term structure)	264
Bond (Base bond class)	267
BoundaryCondition (Abstract boundary condition class for finite difference problems)	271
BoundaryConstraint (Constraint imposing all arguments to be in [low,high])	273
BoxMullerGaussianRng (Gaussian random number generator)	274
Brazil (Brazilian calendar)	275
Brent (Brent 1-D solver)	277
BRLCurrency (Brazilian real)	278
BrownianBridge (Builds Wiener process paths using Gaussian variates)	279
BSMOperator (Black-Scholes-Merton differential operator)	280
Business252 (Business/252 day count convention)	281
BYRCurrency (Belarussian ruble)	282
CADCurrency (Canadian dollar)	283
CADLibor (CAD LIBOR rate)	284
Calendar (calendar class)	285
Calendar::Impl (Abstract base class for calendar implementations)	289
Calendar::OrthodoxImpl (Partial calendar implementation)	290
Calendar::WesternImpl (Partial calendar implementation)	291
CalibratedModel (Calibrated model class)	292
CalibrationHelper (Liquid market instrument used during calibration)	294
Callability (instrument callability)	296
Callability::Price (Amount to be paid upon callability)	297
Canada (Canadian calendar)	298
Cap (Concrete cap class)	299
CapFloor (Base class for cap-like instruments)	300
CapFloor::arguments (Arguments for cap/floor calculation)	302
CapFloor::engine (Base class for cap/floor engines)	303
CapHelper (Calibration helper for ATM cap)	304
CapletConstantVolatility (Constant caplet volatility, no time-strike dependence)	305
CapletVolatilityStructure (Caplet/floorlet forward-volatility structure)	307
CappedFlooredCoupon (Capped and/or floored floating-rate coupon)	309
CapVolatilityStructure (Cap/floor term-volatility structure)	311
CapVolatilityVector (Cap/floor at-the-money term-volatility vector)	313
CashFlow (Base class for cash flows)	315
CashFlows (cashflow-analysis functions)	317
CashOrNothingPayoff (Binary cash-or-nothing payoff)	320
Cdor (CDOR rate)	321
CeilingTruncation (Ceiling truncation)	322
CHFCurrency (Swiss franc)	323
CHFLibor (CHF LIBOR rate)	324
China (Chinese calendar)	325
CLGaussianRng (Gaussian random number generator)	326
CliquetOption (Cliquet (Ratchet) option)	327

CliquetOption::arguments (Arguments for cliquet option calculation)	329
CliquetOption::engine (Cliquet engine base class)	330
Clone (Cloning proxy to an underlying object)	331
ClosestRounding (Closest rounding)	332
CLPCurrency (Chilean peso)	333
CmsCoupon (CMS coupon class)	334
CmsCouponPricer (Base pricer for vanilla CMS coupons)	335
CmsMarket (Set of CMS quotes)	336
CMSMMDriftCalculator (Drift computation for CMS market models)	337
CmsRateBond (CMS-rate bond)	338
CMSwapCurveState (Curve state for constant-maturity-swap market models)	339
CNYCurrency (Chinese yuan)	340
Collar (Concrete collar class)	341
Composite (Composite pattern)	342
CompositeConstraint (Constraint enforcing both given sub-constraints)	343
CompositeInstrument (Composite instrument)	344
CompositeQuote (Market element whose value depends on two other market element)	346
CompoundForward (Compound-forward structure)	347
ConjugateGradient (Multi-dimensional Conjugate Gradient class)	349
ConstantEstimator (Constant-estimator volatility model)	350
ConstantParameter (Standard constant parameter $a(t) = a$)	351
ConstrainedEvolver (Constrained market-model evolver)	352
Constraint (Base constraint class)	353
Constraint::Impl (Base class for constraint implementations)	354
ContinuousAveragingAsianOption (Continuous-averaging Asian option)	355
ContinuousAveragingAsianOption::arguments (Extra arguments for single-asset continuous-average Asian option)	357
ContinuousAveragingAsianOption::engine (Continuous-averaging Asian engine base class)	358
ContinuousFixedLookbackOption (Continuous-fixed lookback option)	359
ContinuousFixedLookbackOption::arguments (Arguments for continuous fixed lookback option calculation)	360
ContinuousFixedLookbackOption::engine (Continuous fixed lookback engine base class)	361
ContinuousFloatingLookbackOption (Continuous-floating lookback option)	362
ContinuousFloatingLookbackOption::arguments (Arguments for continuous floating lookback option calculation)	363
ContinuousFloatingLookbackOption::engine (Continuous floating lookback engine base class)	364
ConundrumPricer (CMS-coupon pricer)	365
ConundrumPricerByBlack (CMS-coupon pricer)	367
ConundrumPricerByNumericalIntegration (CMS-coupon pricer)	368
ConvergenceStatistics (Statistics class with convergence table)	369
ConvertibleBond (Base class for convertible bonds)	370
ConvertibleFixedCouponBond (Convertible fixed-coupon bond)	372
ConvertibleFloatingRateBond (Convertible floating-rate bond)	373
ConvertibleZeroCouponBond (Convertible zero-coupon bond)	374
COPCurrency (Colombian peso)	375
CostFunction (Cost function abstract class for optimization problem)	376
CoterminalSwapCurveState (Curve state for coterminal-swap market models)	377
Coupon (coupon accruing over a fixed period)	378
CovarianceDecomposition (Covariance decomposition into correlation and variances)	380
CoxIngersollRoss (Cox-Ingersoll-Ross model class)	381
CoxIngersollRoss::Dynamics (Dynamics of the short-rate under the Cox-Ingersoll-Ross model)	383

CoxRossRubinstein (Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree) .	384
CrankNicolson (Crank-Nicolson scheme for finite difference methods)	385
Cubic (cubic-spline interpolation factory and traits)	387
CubicSpline (Cubic spline interpolation between discrete points)	388
CumulativeBinomialDistribution (Cumulative binomial distribution function)	390
CumulativeNormalDistribution (Cumulative normal distribution function)	391
CumulativePoissonDistribution (Cumulative Poisson distribution function)	392
CuriouslyRecurringTemplate (Support for the curiously recurring template pattern) . .	393
Currency (Currency specification)	394
Curve (Abstract curve class)	396
CurveState (Curve state for market-model simulations)	397
CYPCurrency (Cyprus pound)	398
CzechRepublic (Czech calendars)	399
CZKCurrency (Czech koruna)	401
Date (Concrete date class)	402
DayCounter (Day counter class)	406
DayCounter::Impl (Abstract base class for day counter implementations)	408
DecInterpCapletVolStructure (This class is interpolating caplets volatilities linealy in two steps (instead of)	409
DEMCurrency (Deutsche mark)	410
Denmark (Danish calendar)	411
DepositRateHelper (Rate helper for bootstrapping over deposit rates)	412
DerivedQuote (Market quote whose value depends on another quote)	414
DirichletBC (Neumann boundary condition (i.e., constant value))	415
Discount (Discount-curve traits)	417
DiscrepancyStatistics (Statistic tool for sequences with discrepancy calculation)	418
DiscreteAveragingAsianOption (Discrete-averaging Asian option)	419
DiscreteAveragingAsianOption::arguments (Extra arguments for single-asset discrete-average Asian option)	421
DiscreteAveragingAsianOption::engine (Discrete-averaging Asian engine base class) . .	422
DiscreteGeometricASO (Discrete geometric average-strike Asian option (European style))	423
DiscretizedAsset (Discretized asset class used by numerical methods)	424
DiscretizedDiscountBond (Useful discretized discount bond asset)	427
DiscretizedOption (Discretized option on a given asset)	428
Disposable (Generic disposable object with move semantics)	430
Dividend (Predetermined cash flow)	431
DividendVanillaOption (Single-asset vanilla option (no barriers) with discrete dividends)	433
DividendVanillaOption::arguments (Arguments for dividend vanilla option calculation)	435
DividendVanillaOption::engine (Dividend-vanilla-option engine base class)	436
DKKCurrency (Danish krone)	437
DKKLibor (DKK LIBOR rate)	438
DMinus (D_- matricial representation)	439
Domain (domain abstract lclass)	440
DoubleStickyRatchetPayoff (Intermediate class for single/double sticky/ratchet payoffs)	441
DownRounding (Down-rounding)	443
DPlus (D_+ matricial representation)	444
DPlusDMinus (D_+D_- matricial representation)	445
DriftTermStructure (Drift term structure)	446
Duration (duration type)	448
DZero (D_0 matricial representation)	449
EarlyExercise (Early-exercise base class)	450
EarlyExercisePathPricer (Base class for early exercise path pricers)	451
EEKCurrency (Estonian kroon)	452
EndCriteria (Criteria to end optimization process:)	453

EqualJumpsBinomialTree (Base class for equal jumps binomial tree)	455
EqualProbabilitiesBinomialTree (Base class for equal probabilities binomial tree)	456
Error (Base error class)	457
ErrorFunction (Error function)	458
ESPCurrency (Spanish peseta)	459
EulerDiscretization (Euler discretization for stochastic processes)	460
EURCurrency (European Euro)	462
Euribor (Euribor index)	463
Euribor10M (10-months Euribor index)	464
Euribor11M (11-months Euribor index)	465
Euribor1M (1-month Euribor index)	466
Euribor1Y (1-year Euribor index)	467
Euribor2M (2-months Euribor index)	468
Euribor2W (2-weeks Euribor index)	469
Euribor365 (Actual/365 Euribor index)	470
Euribor365_10M (10-months Euribor365 index)	471
Euribor365_11M (11-months Euribor365 index)	472
Euribor365_1M (1-month Euribor365 index)	473
Euribor365_1Y (1-year Euribor365 index)	474
Euribor365_2M (2-months Euribor365 index)	475
Euribor365_2W (2-weeks Euribor365 index)	476
Euribor365_3M (3-months Euribor365 index)	477
Euribor365_3W (3-weeks Euribor365 index)	478
Euribor365_4M (4-months Euribor365 index)	479
Euribor365_5M (5-months Euribor365 index)	480
Euribor365_6M (6-months Euribor365 index)	481
Euribor365_7M (7-months Euribor365 index)	482
Euribor365_8M (8-months Euribor365 index)	483
Euribor365_9M (9-months Euribor365 index)	484
Euribor365_SW (1-week Euribor365 index)	485
Euribor3M (3-months Euribor index)	486
Euribor3W (3-weeks Euribor index)	487
Euribor4M (4-months Euribor index)	488
Euribor5M (5-months Euribor index)	489
Euribor6M (6-months Euribor index)	490
Euribor7M (7-months Euribor index)	491
Euribor8M (8-months Euribor index)	492
Euribor9M (9-months Euribor index)	493
EuriborSW (1-week Euribor index)	494
EuriborSwapFixA10Y (10-year EuriborSwapFixAvs6M index)	495
EuriborSwapFixA12Y (12-year EuriborSwapFixAvs6M index)	496
EuriborSwapFixA15Y (15-year EuriborSwapFixAvs6M index)	497
EuriborSwapFixA1Y (1-year EuriborSwapFixAvs3M index)	498
EuriborSwapFixA20Y (20-year EuriborSwapFixAvs6M index)	499
EuriborSwapFixA25Y (25-year EuriborSwapFixAvs6M index)	500
EuriborSwapFixA2Y (2-year EuriborSwapFixAvs6M index)	501
EuriborSwapFixA30Y (30-year EuriborSwapFixAvs6M index)	502
EuriborSwapFixA3Y (3-year EuriborSwapFixAvs6M index)	503
EuriborSwapFixA4Y (4-year EuriborSwapFixAvs6M index)	504
EuriborSwapFixA5Y (5-year EuriborSwapFixAvs6M index)	505
EuriborSwapFixA6Y (6-year EuriborSwapFixAvs6M index)	506
EuriborSwapFixA7Y (7-year EuriborSwapFixAvs6M index)	507
EuriborSwapFixA8Y (8-year EuriborSwapFixAvs6M index)	508
EuriborSwapFixA9Y (9-year EuriborSwapFixAvs6M index)	509

EuriborSwapFixAvs3M (EuriborSwapFixA vs 3M index base class)	510
EuriborSwapFixAvs6M (EuriborSwapFixA vs 6M index base class)	511
EuriborSwapFixB10Y (10-year EuriborSwapFixBvs6M index)	512
EuriborSwapFixB12Y (12-year EuriborSwapFixBvs6M index)	513
EuriborSwapFixB15Y (15-year EuriborSwapFixBvs6M index)	514
EuriborSwapFixB1Y (1-year EuriborSwapFixBvs3M index)	515
EuriborSwapFixB20Y (20-year EuriborSwapFixBvs6M index)	516
EuriborSwapFixB25Y (25-year EuriborSwapFixBvs6M index)	517
EuriborSwapFixB2Y (2-year EuriborSwapFixBvs6M index)	518
EuriborSwapFixB30Y (30-year EuriborSwapFixBvs6M index)	519
EuriborSwapFixB3Y (3-year EuriborSwapFixBvs6M index)	520
EuriborSwapFixB4Y (4-year EuriborSwapFixBvs6M index)	521
EuriborSwapFixB5Y (5-year EuriborSwapFixBvs6M index)	522
EuriborSwapFixB6Y (6-year EuriborSwapFixBvs6M index)	523
EuriborSwapFixB7Y (7-year EuriborSwapFixBvs6M index)	524
EuriborSwapFixB8Y (8-year EuriborSwapFixBvs6M index)	525
EuriborSwapFixB9Y (9-year EuriborSwapFixBvs6M index)	526
EuriborSwapFixBvs3M (EuriborSwapFixB vs 3M index base class)	527
EuriborSwapFixBvs6M (EuriborSwapFixB vs 6M index base class)	528
EuriborSwapFixIFR10Y (10-year EuriborSwapFixIFRvs6M index)	529
EuriborSwapFixIFR12Y (12-year EuriborSwapFixIFRvs6M index)	530
EuriborSwapFixIFR15Y (15-year EuriborSwapFixIFRvs6M index)	531
EuriborSwapFixIFR1Y (1-year EuriborSwapFixIFR3M index)	532
EuriborSwapFixIFR20Y (20-year EuriborSwapFixIFRvs6M index)	533
EuriborSwapFixIFR25Y (25-year EuriborSwapFixIFRvs6M index)	534
EuriborSwapFixIFR2Y (2-year EuriborSwapFixIFRvs6M index)	535
EuriborSwapFixIFR30Y (30-year EuriborSwapFixIFRvs6M index)	536
EuriborSwapFixIFR3Y (3-year EuriborSwapFixIFRvs6M index)	537
EuriborSwapFixIFR4Y (4-year EuriborSwapFixIFRvs6M index)	538
EuriborSwapFixIFR5Y (5-year EuriborSwapFixIFRvs6M index)	539
EuriborSwapFixIFR6Y (6-year EuriborSwapFixIFRvs6M index)	540
EuriborSwapFixIFR7Y (7-year EuriborSwapFixIFRvs6M index)	541
EuriborSwapFixIFR8Y (8-year EuriborSwapFixIFRvs6M index)	542
EuriborSwapFixIFR9Y (9-year EuriborSwapFixIFRvs6M index)	543
EuriborSwapFixIFRvs3M (EuriborSwapFixIFR vs 3M index base class)	544
EuriborSwapFixIFRvs6M (EuriborSwapFixIFR vs 6M index base class)	545
EURLibor (EUR LIBOR rate)	546
EURLibor10M (10-months EURLibor index)	547
EURLibor11M (11-months EURLibor index)	548
EURLibor1M (1-month EURLibor index)	549
EURLibor1Y (1-year EURLibor index)	550
EURLibor2M (2-months EURLibor index)	551
EURLibor2W (2-weeks Euribor index)	552
EURLibor3M (3-months EURLibor index)	553
EURLibor4M (4-months EURLibor index)	554
EURLibor5M (5-months EURLibor index)	555
EURLibor6M (6-months EURLibor index)	556
EURLibor7M (7-months EURLibor index)	557
EURLibor8M (8-months EURLibor index)	558
EURLibor9M (9-months EURLibor index)	559
EURLiborSW (1-week EURLibor index)	560
EurliborSwapFixA10Y (10-year EurliborSwapFixAvs6M index)	561
EurliborSwapFixA12Y (12-year EurliborSwapFixAvs6M index)	562
EurliborSwapFixA15Y (15-year EurliborSwapFixAvs6M index)	563

EurliborSwapFixA1Y (1-year EurliborSwapFixAvs3M index)	564
EurliborSwapFixA20Y (20-year EurliborSwapFixAvs6M index)	565
EurliborSwapFixA25Y (25-year EurliborSwapFixAvs6M index)	566
EurliborSwapFixA2Y (2-year EurliborSwapFixAvs6M index)	567
EurliborSwapFixA30Y (30-year EurliborSwapFixAvs6M index)	568
EurliborSwapFixA3Y (3-year EurliborSwapFixAvs6M index)	569
EurliborSwapFixA4Y (4-year EurliborSwapFixAvs6M index)	570
EurliborSwapFixA5Y (5-year EurliborSwapFixAvs6M index)	571
EurliborSwapFixA6Y (6-year EurliborSwapFixAvs6M index)	572
EurliborSwapFixA7Y (7-year EurliborSwapFixAvs6M index)	573
EurliborSwapFixA8Y (8-year EurliborSwapFixAvs6M index)	574
EurliborSwapFixA9Y (9-year EurliborSwapFixAvs6M index)	575
EurliborSwapFixAvs3M (EurliborSwapFixA vs 3M index base class)	576
EurliborSwapFixAvs6M (EurliborSwapFixA vs 6M index base class)	577
EurliborSwapFixB10Y (10-year EurliborSwapFixBvs6M index)	578
EurliborSwapFixB12Y (12-year EurliborSwapFixBvs6M index)	579
EurliborSwapFixB15Y (15-year EurliborSwapFixBvs6M index)	580
EurliborSwapFixB1Y (1-year EurliborSwapFixBvs3M index)	581
EurliborSwapFixB20Y (20-year EurliborSwapFixBvs6M index)	582
EurliborSwapFixB25Y (25-year EurliborSwapFixBvs6M index)	583
EurliborSwapFixB2Y (2-year EurliborSwapFixBvs6M index)	584
EurliborSwapFixB30Y (30-year EurliborSwapFixBvs6M index)	585
EurliborSwapFixB3Y (3-year EurliborSwapFixBvs6M index)	586
EurliborSwapFixB4Y (4-year EurliborSwapFixBvs6M index)	587
EurliborSwapFixB5Y (5-year EurliborSwapFixBvs6M index)	588
EurliborSwapFixB6Y (6-year EurliborSwapFixBvs6M index)	589
EurliborSwapFixB7Y (7-year EurliborSwapFixBvs6M index)	590
EurliborSwapFixB8Y (8-year EurliborSwapFixBvs6M index)	591
EurliborSwapFixB9Y (9-year EurliborSwapFixBvs6M index)	592
EurliborSwapFixBvs3M (EurliborSwapFixB vs 3M index base class)	593
EurliborSwapFixBvs6M (EurliborSwapFixB vs 6M index base class)	594
EurliborSwapFixIFR10Y (10-year EurliborSwapFixIFRvs6M index)	595
EurliborSwapFixIFR12Y (12-year EurliborSwapFixIFRvs6M index)	596
EurliborSwapFixIFR15Y (15-year EurliborSwapFixIFRvs6M index)	597
EurliborSwapFixIFR1Y (1-year EurliborSwapFixIFRvs3M index)	598
EurliborSwapFixIFR20Y (20-year EurliborSwapFixIFRvs6M index)	599
EurliborSwapFixIFR25Y (25-year EurliborSwapFixIFRvs6M index)	600
EurliborSwapFixIFR2Y (2-year EurliborSwapFixIFRvs6M index)	601
EurliborSwapFixIFR30Y (30-year EurliborSwapFixIFRvs6M index)	602
EurliborSwapFixIFR3Y (3-year EurliborSwapFixIFRvs6M index)	603
EurliborSwapFixIFR4Y (4-year EurliborSwapFixIFRvs6M index)	604
EurliborSwapFixIFR5Y (5-year EurliborSwapFixIFRvs6M index)	605
EurliborSwapFixIFR6Y (6-year EurliborSwapFixIFRvs6M index)	606
EurliborSwapFixIFR7Y (7-year EurliborSwapFixIFRvs6M index)	607
EurliborSwapFixIFR8Y (8-year EurliborSwapFixIFRvs6M index)	608
EurliborSwapFixIFR9Y (9-year EurliborSwapFixIFRvs6M index)	609
EurliborSwapFixIFRvs3M (EurliborSwapFixIFR vs 3M index base class)	610
EurliborSwapFixIFRvs6M (EurliborSwapFixIFR vs 6M index base class)	611
EurodollarFuturesImpliedStdDevQuote (quote for the Eurodollar-future implied standard deviation)	612
EuropeanExercise (European exercise)	613
EuropeanOption (European option on a single asset)	614
Event (Base class for event)	615
EvolutionDescription (Market-model evolution description)	617

ExchangeRate (Exchange rate between two currencies)	618
ExchangeRateManager (Exchange-rate repository)	620
Exercise (Base exercise class)	622
ExplicitEuler (Forward Euler scheme for finite difference methods)	623
ExtendedCoxIngersollRoss (Extended Cox-Ingersoll-Ross model class)	625
ExtendedCoxIngersollRoss::Dynamics (Short-rate dynamics in the extended Cox-Ingersoll-Ross model)	627
ExtendedCoxIngersollRoss::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	628
ExtendedDiscountCurve (Term structure based on loglinear interpolation of discount factors)	629
Extrapolator (Base class for classes possibly allowing extrapolation)	631
Factorial (Factorial numbers calculator)	632
FalsePosition (False position 1-D solver)	633
FaureRsg (Faure low-discrepancy sequence generator)	634
FDBermudanEngine (Finite-differences Bermudan engine)	635
FDDividendEngineMerton73 (Finite-differences pricing engine for dividend options using)	636
FDDividendEngineShiftScale (Finite-differences pricing engine for dividend options using)	637
FDEuropeanEngine (Pricing engine for European options using finite-differences)	638
FDStepConditionEngine (Finite-differences pricing engine for American-style vanilla options)	639
FIMCurrency (Finnish markka)	640
FiniteDifferenceModel (Generic finite difference model)	641
Finland (Finnish calendar)	642
FixedCouponBondHelper (Fixed-coupon bond helper)	643
FixedDividend (Predetermined cash flow)	645
FixedRateBond (Fixed-rate bond)	647
FixedRateBondForward (Forward contract on a fixed-rate bond)	648
FixedRateCoupon (Coupon paying a fixed interest rate)	651
FlatForward (Flat interest-rate curve)	653
FloatingRateBond (Floating-rate bond (possibly capped and/or floored))	655
FloatingRateCoupon (Base floating-rate coupon class)	656
FloatingRateCouponPricer (Generic pricer for floating-rate coupons)	659
FloatingTypePayoff (Payoff based on a floating strike)	660
Floor (Concrete floor class)	661
FloorTruncation (Floor truncation)	662
Forward (Abstract base forward class)	663
ForwardEngine (Forward-engine base class)	666
ForwardFlat (Forward-flat interpolation factory and traits)	667
ForwardFlatInterpolation (Forward-flat interpolation between discrete points)	668
ForwardMeasureProcess (Forward-measure stochastic process)	669
ForwardMeasureProcess1D (Forward-measure 1-D stochastic process)	670
ForwardOptionArguments (Arguments for forward (strike-resetting) option calculation)	671
ForwardPerformanceEngine (Forward performance engine)	672
ForwardRate (Forward-curve traits)	673
ForwardRateAgreement (Forward rate agreement (FRA) class)	674
ForwardRateStructure (Forward-rate term structure)	677
ForwardSpreadedTermStructure (Term structure with added spread on the instantaneous forward rate)	679
ForwardTypePayoff (Class for forward type payoffs)	681
ForwardValueQuote (quote for the forward value of an index)	682
ForwardVanillaOption (Forward version of a vanilla option)	683

FractionalDividend (Predetermined cash flow)	685
FraRateHelper (Rate helper for bootstrapping over FRA rates)	687
FRFCurrency (French franc)	689
FuturesConvAdjustmentQuote (quote for the futures-convexity adjustment of an index)	690
FuturesRateHelper (Rate helper for bootstrapping over interest-rate futures prices)	692
G2 (Two-additive-factor gaussian model class)	693
G2::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	695
G2ForwardProcess (Forward G2 stochastic process)	696
G2Process (G2 stochastic process)	698
G2SwaptionEngine (Swaption priced by means of the Black formula)	700
GammaFunction (Gamma function class)	701
GapPayoff (Binary gap payoff)	702
Garch11 (GARCH volatility model)	704
GarmanKlassAbstract (Garman-Klass volatility model)	705
GarmanKohlhagenProcess (Garman-Kohlhagen (1983) stochastic process)	706
GaussChebyshev2thIntegration (Gauss-Chebyshev integration (second kind))	707
GaussChebyshev2thPolynomial (Gauss-Chebyshev polynomial (second kind))	708
GaussChebyshevIntegration (Gauss-Chebyshev integration)	709
GaussChebyshevPolynomial (Gauss-Chebyshev polynomial)	710
GaussGegenbauerIntegration (Gauss-Gegenbauer integration)	711
GaussGegenbauerPolynomial (Gauss-Gegenbauer polynomial)	712
GaussHermiteIntegration (Generalized Gauss-Hermite integration)	713
GaussHermitePolynomial (Gauss-Hermite polynomial)	714
GaussHyperbolicIntegration (Gauss-Hyperbolic integration)	715
GaussHyperbolicPolynomial (Gauss hyperbolic polynomial)	716
GaussianOrthogonalPolynomial (Orthogonal polynomial for Gaussian quadratures)	717
GaussianQuadrature (Integral of a 1-dimensional function using the Gauss quadratures method)	718
GaussJacobiIntegration (Gauss-Jacobi integration)	719
GaussJacobiPolynomial (Gauss-Jacobi polynomial)	720
GaussKronrodAdaptive (Integral of a 1-dimensional function using the Gauss-Kronrod methods)	721
GaussKronrodNonAdaptive (Integral of a 1-dimensional function using the Gauss-Kronrod methods)	722
GaussLaguerreIntegration (Generalized Gauss-Laguerre integration)	723
GaussLaguerrePolynomial (Gauss-Laguerre polynomial)	724
GaussLegendreIntegration (Gauss-Legendre integration)	725
GaussLegendrePolynomial (Gauss-Legendre polynomial)	726
GBPCurrency (British pound sterling)	727
GBPLibor (GBP LIBOR rate)	728
GeneralizedBlackScholesProcess (Generalized Black-Scholes stochastic process)	729
GeneralStatistics (Statistics tool)	731
GenericEngine (Template base class for option pricing engines)	735
GenericGaussianStatistics (Statistics tool for gaussian-assumption risk measures)	736
GenericModelEngine (Base class for some pricing engine on a particular model)	739
GenericRiskStatistics (Empirical-distribution risk measures)	740
GenericSequenceStatistics (Statistics analysis of N-dimensional (sequence) data)	743
GeometricBrownianMotionProcess (Geometric brownian-motion process)	745
Germany (German calendars)	746
GRDCurrency (Greek drachma)	749
Greeks (Additional option results)	750
HaltonRsg (Halton low-discrepancy sequence generator)	751
Handle (Shared handle to an observable)	752
HestonModel (Heston model for the stochastic volatility of an asset)	754

HestonModelHelper (Calibration helper for Heston model)	755
HestonProcess (Square-root stochastic-volatility Heston process)	756
HKDCurrency (Honk Kong dollar)	758
HongKong (Hong Kong calendars)	759
HUFCurrency (Hungarian forint)	761
HullWhite (Single-factor Hull-White (extended Vasicek) model class)	762
HullWhite::Dynamics (Short-rate dynamics in the Hull-White model)	764
HullWhite::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	765
HullWhiteForwardProcess (Forward Hull-White stochastic process)	766
HullWhiteProcess (Hull-White stochastic process)	768
Hungary (Hungarian calendar)	770
IborCoupon (Coupon paying a Libor-type index)	771
IborCouponPricer (Base pricer for capped/floored Ibor coupons)	772
IborIndex (Base class for Inter-Bank-Offered-Rate indexes (e.g. Libor, etc.))	773
Iceland (Icelandic calendars)	774
IEPCurrency (Irish punt)	776
ILSCurrency (Israeli shekel)	777
IMM (Main cycle of the International Money Market (a.k.a. IMM) months)	778
ImplicitEuler (Backward Euler scheme for finite difference methods)	780
ImpliedStdDevQuote (quote for the implied standard deviation of an underlying)	781
ImpliedTermStructure (Implied term structure at a given date in the future)	782
ImpliedVolTermStructure (Implied vol term structure at a given date in the future)	784
IncrementalStatistics (Statistics tool based on incremental accumulation)	786
Index (Purely virtual base class for indexes)	789
IndexManager (Global repository for past index fixings)	791
India (Indian calendars)	792
Indonesia (Indonesian calendars)	794
INRCurrency (Indian rupee)	796
Instrument (Abstract instrument class)	797
IntegralEngine (Pricing engine for European vanilla options using integral approach)	800
InterestRate (Concrete interest rate class)	801
InterestRateIndex (Base class for interest rate indexes)	804
InterpolatedDiscountCurve (Term structure based on interpolation of discount factors)	806
InterpolatedForwardCurve (Term structure based on interpolation of forward rates)	808
InterpolatedZeroCurve (Term structure based on interpolation of zero yields)	810
Interpolation (Base class for 1-D interpolations)	812
Interpolation2D (Base class for 2-D interpolations)	814
Interpolation2D::Impl (Abstract base class for 2-D interpolation implementations)	816
Interpolation2D::templateImpl (Basic template implementation)	817
Interpolation::Impl (Abstract base class for interpolation implementations)	818
Interpolation::templateImpl (Basic template implementation)	819
IntervalPrice (Interval price)	820
InverseCumulativeNormal (Inverse cumulative normal distribution function)	821
InverseCumulativePoisson (Inverse cumulative Poisson distribution function)	822
InverseCumulativeRng (Inverse cumulative random number generator)	823
InverseCumulativeRsg (Inverse cumulative random sequence generator)	824
IQDCurrency (Iraqi dinar)	825
IRRCurrency (Iranian rial)	826
ISKCurrency (Icelandic krona)	827
Italy (Italian calendars)	828
ITLCurrency (Italian lira)	830
JamshidianSwaptionEngine (Jamshidian swaption engine)	831
Japan (Japanese calendar)	832
JarrowRudd (Jarrow-Rudd (multiplicative) equal probabilities binomial tree)	833

Jibar (JIBAR rate)	834
JointCalendar (Joint calendar)	835
JPYCurrency (Japanese yen)	836
JPYLibor (JPY LIBOR rate)	837
JumpDiffusionEngine (Jump-diffusion engine for vanilla options)	838
JuQuadraticApproximationEngine (Pricing engine for American options with Ju quadratic approximation)	839
KnuthUniformRng (Uniform random number generator)	840
KRWCurrency (South-Korean won)	841
KWDCurrency (Kuwaiti dinar)	842
Lattice (Lattice (tree, finite-differences) base class)	843
LatticeShortRateModelEngine (Engine for a short-rate model specialized on a lattice)	845
LazyObject (Framework for calculation on demand and result caching)	846
LeastSquareFunction (Cost function for least-square problems)	848
LeastSquareProblem (Base class for least square problem)	849
LecuyerUniformRng (Uniform random number generator)	850
LeisenReimer (Leisen & Reimer tree: multiplicative approach)	851
LevenbergMarquardt (Levenberg-Marquardt optimization method)	852
LexicographicalView (Lexicographical 2-D view of a contiguous set of data)	853
LfmCovarianceParameterization (Libor market model parameterization)	855
LfmCovarianceProxy (Proxy for a libor forward model covariance parameterization)	856
LfmHullWhiteParameterization (Libor market model parameterization based on Hull White paper)	857
LfmSwaptionEngine (Libor forward model swaption engine based on Black formula)	858
Libor (Base class for all BBA LIBOR indexes but the EUR ones)	859
LiborForwardModel (Libor forward model)	860
LiborForwardModelProcess (Libor-forward-model process)	862
Linear (Linear-interpolation factory and traits)	864
LinearInterpolation (Linear interpolation between discrete points)	865
LinearLeastSquaresRegression (General linear least squares regression)	866
LineSearch (Base class for line search)	867
LmConstWrapperVolatilityModel (Caplet const volatility model)	869
LmCorrelationModel (libor forward correlation model)	870
LmExponentialCorrelationModel (Exponential correlation model)	871
LmExtLinearExponentialVolModel (Extended linear exponential volatility model)	872
LmLinearExponentialCorrelationModel (linear exponential correlation model)	873
LmLinearExponentialVolatilityModel (linear exponential volatility model)	874
LMMCurveState (Curve state for Libor market models)	875
LMMDriftCalculator (Drift computation for log-normal Libor market models)	876
LMMNormalDriftCalculator (Drift computation for normal Libor market models)	877
LmVolatilityModel (Caplet volatility model)	878
LocalConstantVol (Constant local volatility, no time-strike dependence)	879
LocalVolCurve (Local volatility curve derived from a Black curve)	880
LocalVolSurface (Local volatility surface derived from a Black vol surface)	882
LocalVolTermStructure (Local-volatility term structure)	884
LogLinear (Log-linear interpolation factory and traits)	886
LogLinearInterpolation (log-linear interpolation between discrete points)	887
LogNormalCmSwapRatePc (Predictor-Corrector)	888
LogNormalCotSwapRatePc (Predictor-Corrector)	889
LogNormalFwdRateEuler (Euler)	890
LogNormalFwdRateEulerConstrained (Euler stepping)	891
LogNormalFwdRateIpc (Iterative Predictor-Corrector)	892
LogNormalFwdRatePc (Predictor-Corrector)	893
LongstaffSchwartzPathPricer (Longstaff-Schwarz path pricer for early exercise options)	894

LTLCurrency (Lithuanian litas)	895
LUFCurrency (Luxembourg franc)	896
LVLCurrency (Latvian lat)	897
MakeCapFloor (Helper class)	898
MakeCms (Helper class)	899
MakeMCAmericanEngine (Monte Carlo American engine factory)	900
MakeMCDigitalEngine (Monte Carlo digital engine factory)	901
MakeMCEuropeanEngine (Monte Carlo European engine factory)	902
MakeMCEuropeanHestonEngine (Monte Carlo Heston European engine factory)	903
MakeMCHullWhiteCapFloorEngine (Monte Carlo Hull-White cap-floor engine factory)	904
MakeMCVarianceSwapEngine (Monte Carlo variance-swap engine factory)	905
MakeSchedule (Helper class)	906
MakeVanillaSwap (Helper class)	907
MarketModel (Base class for market models)	908
MarketModelCapFloorEngine (Market-model cap/floor engine)	909
MarketModelComposite (Composition of two or more market-model products)	910
MarketModelEvolver (Market-model evolver)	912
MarketModelFactory (Base class for market-model factories)	913
MarketModelMultiProduct (Market-model product)	914
Matrix (Matrix used in linear algebra)	915
MCAmericanBasketEngine (Least-square Monte Carlo engine)	920
MCAmericanEngine (American Monte Carlo engine)	921
MCBarrierEngine (Pricing engine for barrier options using Monte Carlo simulation)	922
MCBasketEngine (Pricing engine for basket options using Monte Carlo simulation)	924
McCliquetOption (Simple example of Monte Carlo pricer)	926
MCDigitalEngine (Pricing engine for digital options using Monte Carlo simulation)	927
MCDiscreteArithmeticAPEngine (Monte Carlo pricing engine for discrete arithmetic average price Asian)	928
McDiscreteArithmeticASO (Discrete arithmetic average-strike Asian option)	930
MCDiscreteAveragingAsianEngine (Pricing engine for discrete average Asians using Monte Carlo simulation)	931
MCDiscreteGeometricAPEngine (Monte Carlo pricing engine for discrete geometric average price Asian)	933
MCEuropeanEngine (European option pricing engine using Monte Carlo simulation)	934
MCEuropeanHestonEngine (Monte Carlo Heston-model engine for European options)	935
McEverest (Everest-type option pricer)	936
McHimalaya (Himalayan-type option pricer)	937
MCHullWhiteCapFloorEngine (Monte Carlo Hull-White engine for cap/floors)	938
MCLongstaffSchwartzEngine (Longstaff-Schwarz Monte Carlo engine for early exercise options)	940
McMaxBasket (Max-basket Monte Carlo pricer)	942
McPagoda (Roofed Asian option)	943
McPerformanceOption (Performance option computed using Monte Carlo simulation)	944
McPricer (Base class for Monte Carlo pricers)	945
McSimulation (Base class for Monte Carlo engines)	946
MCVanillaEngine (Pricing engine for vanilla options using Monte Carlo simulation)	948
MCVarianceSwapEngine (Variance-swap pricing engine using Monte Carlo simulation,)	949
MersenneTwisterUniformRng (Uniform random number generator)	951
Merton76Process (Merton-76 jump-diffusion process)	952
Mexico (Mexican calendars)	954
MixedScheme (Mixed (explicit/implicit) scheme for finite difference methods)	955
Money (Amount of cash)	957
MonotonicCubicSpline (Cubic spline with monotonicity constraint)	960
MonteCarloModel (General-purpose Monte Carlo model for path samples)	961

MoreGreeks (More additional option results)	962
MoroInverseCumulativeNormal (Moro Inverse cumulative normal distribution class) .	963
MTBrownianGenerator (Mersenne-twister Brownian generator for market-model simulations)	964
MtlCurrency (Maltese lira)	965
MultiAssetOption (Base class for options on multiple assets)	966
MultiAssetOption::arguments (Arguments for multi-asset option calculation)	968
MultiAssetOption::results (Results from multi-asset option calculation)	969
MultiCubicSpline (N-dimensional cubic spline interpolation between discrete points) .	970
MultiPath (Correlated multiple asset paths)	971
MultiPathGenerator (Generates a multipath from a random number generator)	972
MultiProductComposite (Composition of one or more market-model products)	973
MultiProductMultiStep (Multiple-step market-model product)	974
MultiProductOneStep (Single-step market-model product)	975
MultiVariate (Default Monte Carlo traits for multi-variate models)	976
MXNCurrency (Mexican peso)	977
NaturalCubicSpline (Cubic spline with null second derivative at end points)	978
NaturalMonotonicCubicSpline (Natural cubic spline with monotonicity constraint) . .	979
NeumannBC (Neumann boundary condition (i.e., constant derivative))	980
Newton (Newton 1-D solver)	982
NewtonSafe (Safe Newton 1-D solver)	983
NewZealand (New Zealand calendar)	984
NLGCurrency (Dutch guilder)	985
NoConstraint (No constraint)	986
NOKCurrency (Norwegian krone)	987
NonLinearLeastSquare (Non-linear least-square method)	988
NormalDistribution (Normal distribution function)	989
NormalFwdRatePc (Predictor-Corrector)	990
Norway (Norwegian calendar)	991
NPRCurrency (Nepal rupee)	992
Null (Template class providing a null value for a given type)	993
NullCalendar (Calendar for reproducing theoretical calculations)	994
NullCondition (null step condition)	995
NullParameter (Parameter which is always zero $a(t) = 0$)	996
NZDCurrency (New Zealand dollar)	997
NZDLibor (NZD LIBOR rate)	998
Observable (Object that notifies its changes to a set of observables)	999
ObservableValue (observable and assignable proxy to concrete value)	1000
Observer (Object that gets notified when a given observable changes)	1001
OneAssetOption (Base class for options on a single asset)	1002
OneAssetOption::arguments (Arguments for single-asset option calculation)	1005
OneAssetOption::results (Results from single-asset option calculation)	1006
OneAssetStrikedOption (Base class for options on a single asset with striked payoff) .	1007
OneDayCounter (1/1 day count convention)	1009
OneFactorAffineModel (Single-factor affine base class)	1010
OneFactorModel (Single-factor short-rate model abstract class)	1011
OneFactorModel::ShortRateDynamics (Base class describing the short-rate dynamics) .	1012
OneFactorModel::ShortRateTree (Recombining trinomial tree discretizing the state variable)	1013
OperatorFactory (Black-Scholes-Merton differential operator)	1014
OptimizationMethod (Abstract class for constrained optimization method)	1015
Option (Base option class)	1016
Option::arguments (Basic option arguments)	1018
OrnsteinUhlenbeckProcess (Ornstein-Uhlenbeck process class)	1019

Parameter (Base class for model arguments)	1021
Parameter::Impl (Base class for model parameter implementation)	1022
Path (Single-factor random walk)	1023
PathGenerator (Generates random paths using a sequence generator)	1024
PathPricer (Base class for path pricers)	1025
Payoff (Abstract base class for option payoffs)	1026
PercentageStrikePayoff (Payoff with strike expressed as percentage)	1027
Period (Time period described by a number of a given time unit)	1028
PiecewiseConstantParameter (Piecewise-constant parameter)	1030
PiecewiseYieldCurve (Piecewise yield term structure)	1031
PiecewiseZeroSpreadedTermStructure (Term structure with an added vector of spreads on the zero-yield rate)	1033
PKRCurrency (Pakistani rupee)	1035
PlainVanillaPayoff (Plain-vanilla payoff)	1036
PLNCurrency (Polish zloty)	1037
PoissonDistribution (Normal distribution function)	1038
Poland (Polish calendar)	1039
PositiveConstraint (Constraint imposing positivity to all arguments)	1040
PricingEngine (Interface for pricing engines)	1041
PrimeNumbers (Prime numbers calculator)	1042
Problem (Constrained optimization problem)	1043
ProjectedCostFunction (Parameterized cost function)	1045
PTECurrency (Portuguese escudo)	1046
QuantoEngine (Quanto engine base class)	1047
QuantoForwardVanillaOption (Quanto version of a forward vanilla option)	1049
QuantoOptionArguments (Arguments for quanto option calculation)	1050
QuantoOptionResults (Results from quanto option calculation)	1051
QuantoTermStructure (Quanto term structure)	1052
QuantoVanillaOption (Quanto version of a vanilla option)	1054
Quote (Purely virtual base class for market observables)	1056
RandomizedLDS (Randomized (random shift) low-discrepancy sequence)	1057
RandomSequenceGenerator (Random sequence generator based on a pseudo-random number generator)	1059
RatchetMaxPayoff (RatchetMax payoff (double option))	1060
RatchetMinPayoff (RatchetMin payoff (double option))	1061
RatchetPayoff (Ratchet payoff (single option))	1062
RateHelper (Base helper class for yield-curve bootstrapping)	1063
RelativeDateRateHelper (Rate helper with date schedule relative to the global evaluation date)	1065
RelinkableHandle (Relinkable handle to an observable)	1066
ReplicatingVarianceSwapEngine (Variance-swap pricing engine using replicating cost,)	1067
Ridder (Ridder 1-D solver)	1068
ROLCurrency (Romanian leu)	1069
RONCurrency (Romanian new leu)	1070
Rounding (Basic rounding class)	1071
SABR (SABR interpolation factory)	1073
SABRInterpolation (SABR smile interpolation between discrete volatility points)	1074
SalvagingAlgorithm (Algorithm used for matricial pseudo square root)	1075
Sample (Weighted sample)	1076
SampledCurve (This class contains a sampled curve)	1077
SARCurrency (Saudi riyal)	1079
SaudiArabia (Saudi Arabian calendar)	1080
Schedule (Payment schedule)	1081
Secant (Secant 1-D solver)	1082

SeedGenerator (Random seed generator)	1083
SegmentIntegral (Integral of a one-dimensional function)	1084
SEKCurrency (Swedish krona)	1085
Settings (Global repository for run-time library settings)	1086
Settlement (settlement information)	1088
SGDCurrency (Singapore dollar)	1089
ShortRateModel (Abstract short-rate model class)	1090
ShoutCondition (Shout option condition)	1091
SimpleCashFlow (Predetermined cash flow)	1092
SimpleDayCounter (Simple day counter for reproducing theoretical calculations)	1093
SimpleLocalEstimator (Local-estimator volatility model)	1094
SimpleQuote (Market element returning a stored value)	1095
Simplex (Multi-dimensional simplex class)	1096
SimpsonIntegral (Integral of a one-dimensional function)	1097
Singapore (Singapore calendars)	1098
SingleAssetOption (Black-Scholes-Merton option)	1100
SingleProductComposite (Composition of one or more market-model products)	1102
Singleton (Basic support for the singleton pattern)	1103
SingleVariate (Default Monte Carlo traits for single-variate models)	1104
SITCurrency (Slovenian tolar)	1105
SKKCurrency (Slovak koruna)	1106
Slovakia (Slovak calendars)	1107
SmileSection (Interest rate volatility smile section)	1109
SMMDriftCalculator (Drift computation for coterminial swap market models)	1110
SobolBrownianGenerator (Sobol Brownian generator for market-model simulations)	1111
SobolRsg (Sobol low-discrepancy sequence generator)	1112
SoftCallability (callability leaving to the holder the possibility to convert)	1114
Solver1D (Base class for 1-D solvers)	1115
SouthAfrica (South-African calendar)	1117
SouthKorea (South Korean calendars)	1118
SquareRootProcess (Square-root process class)	1120
StatsHolder (Helper class for precomputed distributions)	1121
SteepestDescent (Multi-dimensional steepest-descent class)	1122
step_iterator (Iterator advancing in constant steps)	1123
StepCondition (Condition to be applied at every time step)	1124
StepConditionSet (Parallel evolver for multiple arrays)	1125
StickyMaxPayoff (StickyMax payoff (double option))	1126
StickyMinPayoff (StickyMin payoff (double option))	1127
StickyPayoff (Sticky payoff (single option))	1128
StochasticProcess (Multi-dimensional stochastic process class)	1129
StochasticProcess1D (1-dimensional stochastic process)	1132
StochasticProcess1D::discretization (Discretization of a 1-D stochastic process)	1134
StochasticProcess::discretization (Discretization of a stochastic process over a given time interval)	1135
StochasticProcessArray (Array of correlated 1-D stochastic processes)	1136
Stock (Simple stock class)	1139
StrikedTypePayoff (Intermediate class for payoffs based on a fixed strike)	1140
StulzEngine (Pricing engine for 2D European Baskets)	1141
SuperFundPayoff (Binary superfund payoff)	1142
SuperSharePayoff (Binary supershare payoff)	1143
Surface (Surface abstract class)	1145
SVD (Singular value decomposition)	1146
Swap (Interest rate swap)	1147
SwapIndex (Base class for swap-rate indexes)	1149

SwapRateHelper (Rate helper for bootstrapping over swap rates)	1150
Swaption (Swaption class)	1152
Swaption::arguments (Arguments for swaption calculation)	1154
Swaption::engine (Base class for swaption engines)	1155
SwaptionConstantVolatility (Constant swaption volatility, no time-strike dependence)	1156
SwaptionHelper (Calibration helper for ATM swaption)	1158
SwaptionVolatilityCube (Swaption-volatility cube)	1159
SwaptionVolatilityMatrix (At-the-money swaption-volatility matrix)	1162
SwaptionVolatilityStructure (Swaption-volatility structure)	1165
Sweden (Swedish calendar)	1168
Switzerland (Swiss calendar)	1169
SymmetricSchurDecomposition (Symmetric threshold Jacobi algorithm)	1170
TabulatedGaussLegendre (Tabulated Gauss-Legendre quadratures)	1171
Taiwan (Taiwanese calendars)	1172
TARGET (TARGET calendar)	1174
TermStructure (Basic term-structure functionality)	1175
TermStructureConsistentModel (Term-structure consistent model class)	1177
TermStructureFittingParameter (Deterministic time-dependent parameter used for yield-curve fitting)	1178
THBCurrency (Thai baht)	1179
Thirty360 (30/360 day count convention)	1180
Tian (Tian tree: third moment matching, multiplicative approach)	1181
Tibor (JPY TIBOR index)	1182
TimeBasket (Distribution over a number of dates)	1183
TimeGrid (Time grid class)	1184
TimeSeries (Container for historical data)	1186
TqrEigenDecomposition (Tridiag. QR eigen decomposition with explicite shift aka Wilkinson)	1188
TransformedGrid (Transformed grid)	1189
TrapezoidIntegral (Integral of a one-dimensional function)	1190
Tree (Tree approximating a single-factor diffusion)	1191
TreeCapFloorEngine (Numerical lattice engine for cap/floors)	1192
TreeLattice (Tree-based lattice-method base class)	1193
TreeLattice1D (One-dimensional tree-based lattice)	1195
TreeLattice2D (Two-dimensional tree-based lattice)	1196
TreeSwaptionEngine (Numerical lattice engine for swaptions)	1197
TreeVanillaSwapEngine (Numerical lattice engine for simple swaps)	1198
TridiagonalOperator (Base implementation for tridiagonal operator)	1199
TridiagonalOperator::TimeSetter (Encapsulation of time-setting logic)	1201
Trigeorgis (Trigeorgis (additive equal jumps) binomial tree)	1202
TrinomialTree (Recombining trinomial tree class)	1203
TRLCurrency (Turkish lira)	1204
TRLibor (TRY LIBOR rate)	1205
TRYCurrency (New Turkish lira)	1206
TsiveriotisFernandesLattice (Binomial lattice approximating the Tsiveriotis-Fernandes model)	1207
TTDCurrency (Trinidad & Tobago dollar)	1209
Turkey (Turkish calendar)	1210
TWDCurrency (Taiwan dollar)	1211
TwoFactorModel (Abstract base-class for two-factor models)	1212
TwoFactorModel::ShortRateDynamics (Class describing the dynamics of the two state variables)	1213
TwoFactorModel::ShortRateTree (Recombining two-dimensional tree discretizing the state variable)	1214

TypePayoff (Intermediate class for put/call payoffs)	1215
Ukraine (Ukrainian calendars)	1216
UnitedKingdom (United Kingdom calendars)	1218
UnitedStates (United States calendars)	1220
UpperBoundEngine (Market-model engine for upper-bound estimation)	1223
UpRounding (Up-rounding)	1224
USDCurrency (U.S. dollar)	1225
USDLibor (USD LIBOR rate)	1226
VanillaOption (Vanilla option (no discrete dividends, no barriers) on a single asset)	1227
VanillaOption::engine (Vanilla option engine base class)	1228
VanillaSwap::arguments (Arguments for simple swap calculation)	1229
VanillaSwap::results (Results from simple swap calculation)	1230
VarianceSwap (Variance swap)	1231
VarianceSwap::arguments (Arguments for forward fair-variance calculation)	1233
VarianceSwap::engine (Base class for variance-swap engines)	1234
VarianceSwap::results (Results from variance-swap calculation)	1235
Vasicek (Vasicek model class)	1236
Vasicek::Dynamics (Short-rate dynamics in the Vasicek model)	1238
VEBCurrency (Venezuelan bolivar)	1239
Visitor (Visitor for a specific class)	1240
YieldTermStructure (Interest-rate term structure)	1241
ZARCurrency (South-African rand)	1245
ZeroCondition (Zero exercise condition)	1246
ZeroCouponBond (Zero-coupon bond)	1247
ZeroSpreadedTermStructure (Term structure with an added spread on the zero yield rate)	1248
ZeroYield (Zero-curve traits)	1250
ZeroYieldStructure (Zero-yield term structure)	1251
Zibor (CHF ZIBOR rate)	1253

Chapter 6

QuantLib File Index

6.1 QuantLib File List

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

ql/capvolstructures.hpp (Cap/floor volatility structures)	1255
ql/cashflow.hpp (Base class for cash flows)	1256
ql/currency.hpp (Currency specification)	1283
ql/daycounter.hpp (Day counter class)	1284
ql/discretizedasset.hpp (Discretized asset classes)	1285
ql/errors.hpp (Classes and functions for error handling)	1286
ql/event.hpp (Base class for events associated with a given date)	1289
ql/exchangerate.hpp (Exchange rate between two currencies)	1290
ql/exercise.hpp (Option exercise classes and payoff function)	1291
ql/grid.hpp (Grid constructors)	1292
ql/handle.hpp (Globally accessible relinkable pointer)	1293
ql/index.hpp (Purely virtual base class for indexes)	1294
ql/instrument.hpp (Abstract instrument class)	1330
ql/interestrates.hpp (Instrument rate class)	1373
ql/money.hpp (Cash amount in a given currency)	1565
ql/numericalmethod.hpp (Numerical method class)	1566
ql/option.hpp (Base option class)	1567
ql/payoff.hpp (Option payoff classes)	1574
ql/position.hpp (Short or long position)	1575
ql/prices.hpp (Price classes)	1576
ql/pricingengine.hpp (Base class for pricing engines)	1577
ql/qldefines.hpp (Global definitions and compiler switches)	1665
ql/quote.hpp (Purely virtual base class for market observables)	1667
ql/settings.hpp (Global repository for run-time library settings)	1675
ql/stochasticprocess.hpp (Stochastic processes)	1676
ql/swaptionvolstructure.hpp (Swaption volatility structure)	1677
ql/termstructure.hpp (Base class for term structures)	1678
ql/timegrid.hpp (Discrete time grid)	1773
ql/timeseries.hpp (Container for historical data)	1774
ql/types.hpp (Custom types)	1775
ql/volatilitymodel.hpp (Volatility term structures)	1787
ql/voltermstructure.hpp (Volatility term structures)	1788
ql/yieldtermstructure.hpp (Interest-rate term structure)	1789

ql/cashflows/ analysis.hpp (Cash-flow analysis functions)	1257
ql/cashflows/ capflooredcoupon.hpp (Floating rate coupon with additional cap/floor) . .	1258
ql/cashflows/ cashflowvectors.hpp (Cash flow vector builders)	1259
ql/cashflows/ cmscoupon.hpp (CMS coupon)	1261
ql/cashflows/ conundrumpricer.hpp (CMS-coupon pricer)	1262
ql/cashflows/ coupon.hpp (Coupon accruing over a fixed period)	1263
ql/cashflows/ couponpricer.hpp (Coupon pricers)	1264
ql/cashflows/ dividend.hpp (A stock dividend)	1266
ql/cashflows/ fixedratecoupon.hpp (Coupon paying a fixed annual rate)	1267
ql/cashflows/ floatingratecoupon.hpp (Coupon paying a variable index-based rate) . . .	1268
ql/cashflows/ iborcoupon.hpp (Coupon paying a Libor-type index)	1269
ql/cashflows/ rangeaccrual.hpp (Range-accrual coupon)	1270
ql/cashflows/ simplecashflow.hpp (Predetermined cash flow)	1271
ql/cashflows/ timebasket.hpp (Distribution over a number of date ranges)	1272
ql/currencies/ africa.hpp (African currencies)	1273
ql/currencies/ america.hpp (American currencies)	1274
ql/currencies/ asia.hpp (Asian currencies)	1276
ql/currencies/ europe.hpp (European currencies)	1278
ql/currencies/ exchangeratemanager.hpp (Exchange-rate repository)	1281
ql/currencies/ oceania.hpp (Oceania currencies)	1282
ql/indexes/ iborindex.hpp (Base class for Inter-Bank-Offered-Rate indexes)	1314
ql/indexes/ indexmanager.hpp (Global repository for past index fixings)	1315
ql/indexes/ interestrateindex.hpp (Base class for interest rate indexes)	1316
ql/indexes/ swapindex.hpp (Swap-rate indexes)	1329
ql/indexes/ibor/ audlibor.hpp (AUD LIBOR rate)	1295
ql/indexes/ibor/ cadlibor.hpp (CAD LIBOR rate)	1296
ql/indexes/ibor/ cdor.hpp (CDOR rate)	1297
ql/indexes/ibor/ chflibor.hpp (CHF LIBOR rate)	1298
ql/indexes/ibor/ dkklibor.hpp (DKK LIBOR rate)	1299
ql/indexes/ibor/ euribor.hpp (Euribor index)	1300
ql/indexes/ibor/ eurlibor.hpp (EUR LIBOR rate)	1303
ql/indexes/ibor/ gbplibor.hpp (GBP LIBOR rate)	1305
ql/indexes/ibor/ jibar.hpp (JIBAR rate)	1306
ql/indexes/ibor/ jpylibor.hpp (JPY LIBOR rate)	1307
ql/indexes/ibor/ libor.hpp (Base class for BBA LIBOR indexes)	1308
ql/indexes/ibor/ nzdlibor.hpp (NZD LIBOR rate)	1309
ql/indexes/ibor/ tibor.hpp (JPY TIBOR rate)	1310
ql/indexes/ibor/ trlibor.hpp (TRY LIBOR rate)	1311
ql/indexes/ibor/ usdlibor.hpp (USD LIBOR rate)	1312
ql/indexes/ibor/ zibor.hpp (CHF ZIBOR rate)	1313
ql/indexes/swap/ euriborswapfixa.hpp (EuriborSwapFixA indexes)	1317
ql/indexes/swap/ euriborswapfixb.hpp (EuriborSwapFixB indexes)	1319
ql/indexes/swap/ euriborswapfixifr.hpp (EuriborSwapFixIFR indexes)	1321
ql/indexes/swap/ eurliborswapfixa.hpp (EurliborSwapFixA indexes)	1323
ql/indexes/swap/ eurliborswapfixb.hpp (EurliborSwapFixB indexes)	1325
ql/indexes/swap/ eurliborswapfixifr.hpp (EurliborSwapFixIFR indexes)	1327
ql/instruments/ asianoption.hpp (Asian option on a single asset)	1331
ql/instruments/ assetswap.hpp (Bullet bond vs Libor swap)	1332
ql/instruments/ barrieroption.hpp (Barrier option on a single asset)	1333
ql/instruments/ basketoption.hpp (Basket option on a number of assets)	1334
ql/instruments/ bond.hpp (Concrete bond class)	1335
ql/instruments/ callabilityschedule.hpp (Schedule of put/call dates)	1337
ql/instruments/ capfloor.hpp (Cap and floor class)	1338
ql/instruments/ cliquetoption.hpp (Cliquet option)	1340

ql/instruments/ cmsratebond.hpp (CMS-rate bond)	1341
ql/instruments/ compositeinstrument.hpp (Composite instrument class)	1342
ql/instruments/ convertiblebond.hpp (Convertible bond class)	1343
ql/instruments/ dividendschedule.hpp (Schedule of dividend dates)	1345
ql/instruments/ dividendvanillaoption.hpp (Vanilla option on a single asset with discrete dividends)	1346
ql/instruments/ europeanoption.hpp (European option on a single asset)	1347
ql/instruments/ fixedratebond.hpp (Fixed-rate bond)	1348
ql/instruments/ fixedratebondforward.hpp (Forward contract on a fixed-rate bond) . . .	1349
ql/instruments/ floatingratebond.hpp (Floating-rate bond)	1350
ql/instruments/ forward.hpp (Base forward class)	1351
ql/instruments/ forwardrateagreement.hpp (Forward rate agreement)	1352
ql/instruments/ forwardvanillaoption.hpp (Forward version of a vanilla option)	1353
ql/instruments/ lookbackoption.hpp (Lookback option on a single asset)	1354
ql/instruments/ makecapfloor.hpp (Helper class to instantiate standard market cap/floor)	1355
ql/instruments/ makecms.hpp (Helper class to instantiate standard market CMS)	1356
ql/instruments/ makevanillaswap.hpp (Helper class to instantiate standard market swaps)	1357
ql/instruments/ multiassetoption.hpp (Option on multiple assets)	1358
ql/instruments/ oneassetoption.hpp (Option on a single asset)	1359
ql/instruments/ oneassetstrikedoption.hpp (Option on a single asset with striked payoff)	1360
ql/instruments/ payoffs.hpp (Payoffs for various options)	1361
ql/instruments/ quantoforwardvanillaoption.hpp (Quanto version of a forward vanilla option)	1363
ql/instruments/ quantovanillaoption.hpp (Quanto version of a vanilla option)	1364
ql/instruments/ stickyratchet.hpp (Payoffs for double nested options of sticky or ratchet type)	1365
ql/instruments/ stock.hpp (Concrete stock class)	1366
ql/instruments/ swap.hpp (Interest rate swap)	1367
ql/instruments/ swaption.hpp (Swaption class)	1368
ql/instruments/ vanillaoption.hpp (Vanilla option on a single asset)	1369
ql/instruments/ vanillaswap.hpp (Simple fixed-rate vs Libor swap)	1370
ql/instruments/ varianceswap.hpp (Variance swap)	1371
ql/instruments/ zerocouponbond.hpp (Zero-coupon bond)	1372
ql/legacy/libormarketmodels/ lfmcovaproxy.hpp (Proxy for libor forward covariance parameterization)	1374
ql/legacy/libormarketmodels/ liborforwardmodel.hpp (Libor forward model incl. exact cap pricing Rebonato formula to approximate swaption prices)	1375
ql/legacy/libormarketmodels/ lmconstwrappercorrmodel.hpp (Const wrapper for correlation model for libor market models)	1376
ql/legacy/libormarketmodels/ lmconstwrappervolmodel.hpp (Const wrapper for a volatility model for libor market models)	1377
ql/legacy/libormarketmodels/ lmcrrmodel.hpp (Correlation model for libor market models)	1378
ql/legacy/libormarketmodels/ lmexpcorrmodel.hpp (Exponential correlation model for libor market models)	1379
ql/legacy/libormarketmodels/ lmextlinexpvolmodel.hpp (Volatility model for libor market models)	1380
ql/legacy/libormarketmodels/ lmfixedvolmodel.hpp (Model of constant volatilities for libor market models)	1381
ql/legacy/libormarketmodels/ lmliexpcorrmodel.hpp (Exponential correlation model for libor market models)	1382
ql/legacy/libormarketmodels/ lmliexpvolmodel.hpp (Volatility model for libor market models)	1383
ql/legacy/libormarketmodels/ lmvolmodel.hpp (Volatility model for libor market models)	1384

ql/legacy/pricers/ discretegeometricasos.hpp (Discrete geometric average-strike Asian option)	1385
ql/legacy/pricers/ mccliqetoption.hpp (Cliquet option priced with Monte Carlo simulation)	1386
ql/legacy/pricers/ mcdiscretearithmeticasos.hpp (Discrete arithmetic average-strike Asian option)	1387
ql/legacy/pricers/ mceverest.hpp (Everest-type option pricer)	1388
ql/legacy/pricers/ mchimalaya.hpp (Himalayan-type option pricer)	1389
ql/legacy/pricers/ mcmaxbasket.hpp (Max-basket Monte Carlo pricer)	1390
ql/legacy/pricers/ mcpagoda.hpp (Roofed multi asset Asian option)	1391
ql/legacy/pricers/ mcperformanceoption.hpp (Performance option priced with Monte Carlo simulation)	1392
ql/legacy/pricers/ mcpricer.hpp (Base class for Monte Carlo pricers)	1393
ql/legacy/pricers/ singleassetoption.hpp (Common code for option evaluation)	1394
ql/math/array.hpp (1-D array used in linear algebra)	1395
ql/math/beta.hpp (Beta and beta incomplete functions)	1396
ql/math/comparison.hpp (Floating-point comparisons)	1397
ql/math/curve.hpp (Curve)	1398
ql/math/domain.hpp (Domain)	1405
ql/math/errorfunction.hpp (Error function)	1406
ql/math/factorial.hpp (Factorial numbers calculator)	1407
ql/math/functional.hpp (Functionals and combinators not included in the STL)	1408
ql/math/incompletemgamma.hpp (Incomplete Gamma function)	1409
ql/math/interpolation.hpp (Base class for 1-D interpolations)	1419
ql/math/lexicographicalview.hpp (Lexicographical 2-D view of a contiguous set of data)	1433
ql/math/linearleastquaresregression.hpp (General linear least square regression)	1434
ql/math/matrix.hpp (Matrix used in linear algebra)	1435
ql/math/primenumbers.hpp (Prime numbers calculator)	1457
ql/math/rounding.hpp (Rounding implementation)	1472
ql/math/sampledcurve.hpp (Class that contains a sampled curve)	1473
ql/math/solver1d.hpp (Abstract 1-D solver class)	1474
ql/math/surface.hpp (Surface)	1491
ql/math/transformedgrid.hpp (Encapsulates a grid)	1492
ql/math/distributions/binomialdistribution.hpp (Binomial distribution)	1399
ql/math/distributions/bivariatenormaldistribution.hpp (Bivariate cumulative normal distribution)	1400
ql/math/distributions/chisquaredistribution.hpp (Chi-square (central and non-central) distributions)	1401
ql/math/distributions/gammadistribution.hpp (Gamma distribution)	1402
ql/math/distributions/normaldistribution.hpp (Normal, cumulative and inverse cumulative distributions)	1403
ql/math/distributions/poissondistribution.hpp (Poisson distribution)	1404
ql/math/integrals/gaussianorthogonalpolynomial.hpp (Orthogonal polynomials for gaussian quadratures)	1410
ql/math/integrals/gaussianquadratures.hpp (Integral of a 1-dimensional function using the Gauss quadratures)	1412
ql/math/integrals/integral.hpp (Integrators base class definition)	1414
ql/math/integrals/kronrodintegral.hpp (Integral of a 1-dimensional function using the Gauss-Kronrod method)	1415
ql/math/integrals/segmentintegral.hpp (Integral of a one-dimensional function using segment algorithm)	1416
ql/math/integrals/simpsonintegral.hpp (Integral of a one-dimensional function using Simpson formula)	1417

ql/math/integrals/ trapezoidintegral.hpp (Integral of a one-dimensional function using the trapezoid formula)	1418
ql/math/interpolations/ backwardflatinterpolation.hpp (Backward-flat interpolation between discrete points)	1420
ql/math/interpolations/ bicubicsplineinterpolation.hpp (Bicubic spline interpolation between discrete points)	1421
ql/math/interpolations/ bilinearinterpolation.hpp (Bilinear interpolation between discrete points)	1422
ql/math/interpolations/ cubicspline.hpp (Cubic spline interpolation between discrete points)	1423
ql/math/interpolations/ extrapolation.hpp (Class-wide extrapolation settings)	1424
ql/math/interpolations/ flatextrapolation2d.hpp (Abstract base classes for 2-D flat extrapolations)	1425
ql/math/interpolations/ forwardflatinterpolation.hpp (Forward-flat interpolation between discrete points)	1426
ql/math/interpolations/ interpolation2d.hpp (Abstract base classes for 2-D interpolations)	1427
ql/math/interpolations/ linearinterpolation.hpp (Linear interpolation between discrete points)	1428
ql/math/interpolations/ loglinearinterpolation.hpp (Log-linear interpolation between discrete points)	1429
ql/math/interpolations/ multicubicspline.hpp (N-dimensional cubic spline interpolation between discrete points)	1430
ql/math/interpolations/ sabrinterpolation.hpp (SABR interpolation interpolation between discrete points)	1432
ql/math/matrixutilities/ choleskydecomposition.hpp (Cholesky decomposition)	1436
ql/math/matrixutilities/ getcovariance.hpp (Covariance matrix calculation)	1437
ql/math/matrixutilities/ pseudosqrt.hpp (Pseudo square root of a real symmetric matrix)	1438
ql/math/matrixutilities/ svd.hpp (Singular value decomposition)	1439
ql/math/matrixutilities/ symmetricschurdecomposition.hpp (Eigenvalues/eigenvectors of a real symmetric matrix)	1440
ql/math/matrixutilities/ tqreigendecomposition.hpp (Tridiag. QR eigen decomposition with explicite shift aka Wilkinson)	1441
ql/math/optimization/ armijo.hpp (Armijo line-search class)	1442
ql/math/optimization/ conjugategradient.hpp (Conjugate gradient optimization method)	1443
ql/math/optimization/ constraint.hpp (Abstract constraint class)	1444
ql/math/optimization/ costfunction.hpp (Optimization cost function class)	1445
ql/math/optimization/ endcriteria.hpp (Optimization criteria class)	1446
ql/math/optimization/ leastsquare.hpp (Least square cost function)	1447
ql/math/optimization/ levenbergmarquardt.hpp (Levenberg-Marquardt optimization method)	1448
ql/math/optimization/ linesearch.hpp (Line search abstract class)	1449
ql/math/optimization/ linesearchbasedmethod.hpp (Abstract optimization method class)	1450
ql/math/optimization/ lmdif.hpp (Wrapper for MINPACK minimization routine)	1451
ql/math/optimization/ method.hpp (Abstract optimization method class)	1452
ql/math/optimization/ problem.hpp (Abstract optimization problem class)	1453
ql/math/optimization/ projectedcostfunction.hpp (Cost function utility)	1454
ql/math/optimization/ simplex.hpp (Simplex optimization method)	1455
ql/math/optimization/ steepestdescent.hpp (Steepest descent optimization method)	1456
ql/math/randomnumbers/ boxmullergaussianrng.hpp (Box-Muller Gaussian random-number generator)	1458
ql/math/randomnumbers/ centrallimitgaussianrng.hpp (Central limit Gaussian random-number generator)	1459
ql/math/randomnumbers/ faurersg.hpp (Faure low-discrepancy sequence generator)	1460
ql/math/randomnumbers/ haltonrsg.hpp (Halton low-discrepancy sequence generator)	1461

ql/math/randomnumbers/inversecumulativerng.hpp (Inverse cumulative Gaussian random-number generator)	1462
ql/math/randomnumbers/inversecumulativersg.hpp (Inverse cumulative random sequence generator)	1463
ql/math/randomnumbers/knuthuniformrng.hpp (Knuth uniform random number generator)	1464
ql/math/randomnumbers/lecuyeruniformrng.hpp (L'Ecuyer uniform random number generator)	1465
ql/math/randomnumbers/mt19937uniformrng.hpp (Mersenne Twister uniform random number generator)	1466
ql/math/randomnumbers/randomizedlds.hpp (Randomized low-discrepancy sequence)	1467
ql/math/randomnumbers/randomsequencegenerator.hpp (Random sequence generator based on a pseudo-random number generator)	1468
ql/math/randomnumbers/rngtraits.hpp (Random-number generation policies)	1469
ql/math/randomnumbers/seedgenerator.hpp (Random seed generator)	1470
ql/math/randomnumbers/sobolrngs.hpp (Sobol low-discrepancy sequence generator)	1471
ql/math/solvers1d/bisection.hpp (Bisection 1-D solver)	1475
ql/math/solvers1d/brent.hpp (Brent 1-D solver)	1476
ql/math/solvers1d/falseposition.hpp (False-position 1-D solver)	1477
ql/math/solvers1d/newton.hpp (Newton 1-D solver)	1478
ql/math/solvers1d/newtonsafe.hpp (Safe (bracketed) Newton 1-D solver)	1479
ql/math/solvers1d/ridder.hpp (Ridder 1-D solver)	1480
ql/math/solvers1d/secant.hpp (Secant 1-D solver)	1481
ql/math/statistics/convergencestatistics.hpp (Statistics tool with risk measures)	1482
ql/math/statistics/discrepancystatistics.hpp (Statistic tool for sequences with discrepancy calculation)	1483
ql/math/statistics/gaussianstatistics.hpp (Statistics tool for gaussian-assumption risk measures)	1484
ql/math/statistics/generalstatistics.hpp (Statistics tool)	1485
ql/math/statistics/incrementalstatistics.hpp (Statistics tool based on incremental accumulation)	1486
ql/math/statistics/riskstatistics.hpp (Empirical-distribution risk measures)	1487
ql/math/statistics/sequencestatistics.hpp (Statistics tools for sequence (vector, list, array) samples)	1488
ql/math/statistics/statistics.hpp (Statistics tool with risk measures)	1490
ql/methods/finitedifferences/americancondition.hpp (American option exercise condition)	1493
ql/methods/finitedifferences/boundarycondition.hpp (Boundary conditions for differential operators)	1494
ql/methods/finitedifferences/bsmoperator.hpp (Differential operator for Black-Scholes-Merton equation)	1495
ql/methods/finitedifferences/bsmtermoperator.hpp (Differential operator for Black-Scholes-Merton equation)	1496
ql/methods/finitedifferences/cranknicolson.hpp (Crank-Nicolson scheme for finite difference methods)	1497
ql/methods/finitedifferences/dminus.hpp (D_- matricial representation)	1498
ql/methods/finitedifferences/dplus.hpp (D_+ matricial representation)	1499
ql/methods/finitedifferences/dplusdminus.hpp ($D_+ D_-$ matricial representation)	1500
ql/methods/finitedifferences/dzero.hpp (D_0 matricial representation)	1501
ql/methods/finitedifferences/expliciteuler.hpp (Explicit Euler scheme for finite difference methods)	1502
ql/methods/finitedifferences/fdtypedefs.hpp (Default choices for template instantiations)	1503
ql/methods/finitedifferences/finitedifferencemodel.hpp (Generic finite difference model)	1504

ql/methods/finitedifferences/ impliciteuler.hpp (Implicit Euler scheme for finite difference methods)	1505
ql/methods/finitedifferences/ mixedscheme.hpp (Mixed (explicit/implicit) scheme for finite difference methods)	1506
ql/methods/finitedifferences/ onefactoroperator.hpp (General differential operator for one-factor interest rate models)	1507
ql/methods/finitedifferences/ operatorfactory.hpp (Factory for finite difference operators)	1508
ql/methods/finitedifferences/ operatortraits.hpp (Differential operator traits)	1509
ql/methods/finitedifferences/ parallelevolver.hpp (Parallel evolver for multiple arrays)	1510
ql/methods/finitedifferences/ pde.hpp (General class for one dimensional PDE's)	1511
ql/methods/finitedifferences/ pdebsm.hpp (Black-Scholes-Merton PDE)	1512
ql/methods/finitedifferences/ pdeshortrate.hpp (Adapter to short rate)	1513
ql/methods/finitedifferences/ shoutcondition.hpp (Shout option exercise condition)	1514
ql/methods/finitedifferences/ stepcondition.hpp (Conditions to be applied at every time step)	1515
ql/methods/finitedifferences/ tridiagonaloperator.hpp (Tridiagonal operator)	1516
ql/methods/finitedifferences/ zerocondition.hpp (Zero option exercise condition)	1518
ql/methods/lattices/ binomialtree.hpp (Binomial tree class)	1519
ql/methods/lattices/ bsmlattice.hpp (Binomial trees under the BSM model)	1521
ql/methods/lattices/ lattice.hpp (Tree-based lattice-method class)	1522
ql/methods/lattices/ lattice1d.hpp (One-dimensional lattice class)	1523
ql/methods/lattices/ lattice2d.hpp (Two-dimensional lattice class)	1524
ql/methods/lattices/ tlattice.hpp (Binomial Tsiveriotis-Fernandes tree model)	1525
ql/methods/lattices/ tree.hpp (Tree class)	1526
ql/methods/lattices/ trinomialtree.hpp (Trinomial tree class)	1527
ql/methods/montecarlo/ brownianbridge.hpp (Brownian bridge)	1528
ql/methods/montecarlo/ earlyexercisepathpricer.hpp (Base class for early exercise single-path pricers)	1529
ql/methods/montecarlo/ longstaffschwartzpathpricer.hpp (Longstaff-Schwarz path pricer for early exercise options)	1530
ql/methods/montecarlo/ lsmbasisystem.hpp (Utility classes for Longstaff-Schwarz early-exercise Monte Carlo)	1531
ql/methods/montecarlo/ mctrails.hpp (Monte Carlo policies)	1532
ql/methods/montecarlo/ montecarlomodel.hpp (General-purpose Monte Carlo model)	1533
ql/methods/montecarlo/ multipath.hpp (Correlated multiple asset paths)	1534
ql/methods/montecarlo/ multipathgenerator.hpp (Generates a multi path from a random-array generator)	1535
ql/methods/montecarlo/ path.hpp (Single factor random walk)	1536
ql/methods/montecarlo/ pathgenerator.hpp (Generates random paths using a sequence generator)	1537
ql/methods/montecarlo/ pathpricer.hpp (Base class for single-path pricers)	1538
ql/methods/montecarlo/ sample.hpp (Weighted sample)	1539
ql/models/ calibrationhelper.hpp (Calibration helper class)	1540
ql/models/ model.hpp (Abstract interest rate model class)	1549
ql/models/ parameter.hpp (Model parameter classes)	1550
ql/models/equity/ batesmodel.hpp (Extended versions of the Heston model)	1541
ql/models/equity/ hestonmodel.hpp (Heston model for the stochastic volatility of an asset)	1542
ql/models/equity/ hestonmodelhelper.hpp (Heston-model calibration helper)	1543
ql/models/marketmodels/ swapforwardmappings.hpp (Utility functions for mapping between swap rate and forward rate)	1548
ql/models/marketmodels/driftcomputation/ cmsmmdriftcalculator.hpp (Drift computation for CMS market model)	1544
ql/models/marketmodels/driftcomputation/ lmmmdriftcalculator.hpp (Drift computation for Libor market model)	1545

ql/models/marketmodels/driftcomputation/ lmmnormaldriftcalculator.hpp (Drift computation for normal Libor market model)	1546
ql/models/marketmodels/driftcomputation/ smmdriftcalculator.hpp (Drift computation for coterminial-swap market model)	1547
ql/models/shortrate/ onefactormodel.hpp (Abstract one-factor interest rate model class)	1553
ql/models/shortrate/ twofactormodel.hpp (Abstract two-factor interest rate model class)	1559
ql/models/shortrate/calibrationhelpers/ caphelper.hpp (CapHelper calibration helper) .	1551
ql/models/shortrate/calibrationhelpers/ swaptionhelper.hpp (Swaption calibration helper)	1552
ql/models/shortrate/onefactormodels/ blackkarasinski.hpp (Black-Karasinski model) . .	1554
ql/models/shortrate/onefactormodels/ coxingersollross.hpp (Cox-Ingersoll-Ross model)	1555
ql/models/shortrate/onefactormodels/ extendedcoxingersollross.hpp (Extended Cox-Ingersoll-Ross model)	1556
ql/models/shortrate/onefactormodels/ hullwhite.hpp (Hull & White (HW) model)	1557
ql/models/shortrate/onefactormodels/ vasicek.hpp (Vasicek model class)	1558
ql/models/shortrate/twofactormodels/ g2.hpp (Two-factor additive Gaussian Model G2++)	1560
ql/models/volatility/ constantestimator.hpp (Constant volatility estimator)	1561
ql/models/volatility/ garch.hpp (GARCH volatility model)	1562
ql/models/volatility/ garmanklass.hpp (Volatility estimators using high low data)	1563
ql/models/volatility/ simplelocalestimator.hpp (Constant volatility estimator)	1564
ql/patterns/ composite.hpp (Composite pattern)	1568
ql/patterns/ curiouslyrecurring.hpp (Curiously recurring template pattern)	1569
ql/patterns/ lazyobject.hpp (Framework for calculation on demand and result caching) .	1570
ql/patterns/ observable.hpp (Observer/observable pattern)	1571
ql/patterns/ singleton.hpp (Basic support for the singleton pattern)	1572
ql/patterns/ visitor.hpp (Degenerate base class for the Acyclic Visitor pattern)	1573
ql/pricingengines/ americanpayoffatexpiry.hpp (Analytical formulae for american exercise with payoff at expiry)	1578
ql/pricingengines/ americanpayoffathit.hpp (Analytical formulae for american exercise with payoff at hit)	1579
ql/pricingengines/ blackcalculator.hpp (Black-formula calculator class)	1590
ql/pricingengines/ blackformula.hpp (Black formula)	1591
ql/pricingengines/ blackscholescalculator.hpp (Black-Scholes formula calculator class) .	1593
ql/pricingengines/ genericmodelengine.hpp (Generic option engine based on a model) .	1606
ql/pricingengines/ greeks.hpp (Default greek calculations)	1607
ql/pricingengines/ latticeshortratemodelengine.hpp (Engine for a short-rate model specialized on a lattice)	1610
ql/pricingengines/ mclongstaffschwartzengine.hpp (Longstaff Schwartz Monte Carlo engine for early exercise options)	1613
ql/pricingengines/ mcsimulation.hpp (Framework for Monte Carlo engines)	1614
ql/pricingengines/asian/ analytic_cont_geom_av_price.hpp (Analytic engine for continuous geometric average price Asian)	1580
ql/pricingengines/asian/ analytic_discr_geom_av_price.hpp (Analytic engine for discrete geometric average price Asian)	1581
ql/pricingengines/asian/ mc_discr_arith_av_price.hpp (Monte Carlo engine for discrete arithmetic average price Asian)	1582
ql/pricingengines/asian/ mc_discr_geom_av_price.hpp (Monte Carlo engine for discrete geometric average price Asian)	1583
ql/pricingengines/asian/ mcdiscreteasianengine.hpp (Monte Carlo pricing engine for discrete average Asians)	1584
ql/pricingengines/barrier/ analyticbarrierengine.hpp (Analytic barrier option engines) .	1585
ql/pricingengines/barrier/ mcbarrierengine.hpp (Monte Carlo barrier option engines) . .	1586

ql/pricingengines/basket/ mcamericanbasketengine.hpp (Least-square Monte Carlo engines)	1587
ql/pricingengines/basket/ mcbasketengine.hpp (European basket MC Engine)	1588
ql/pricingengines/basket/ stulzengine.hpp (2D European Basket formulae, due to Stulz (1982))	1589
ql/pricingengines/capfloor/ analyticcapfloorengine.hpp (Analytic engine for caps/floors)	1594
ql/pricingengines/capfloor/ blackcapfloorengine.hpp (Black-formula cap/floor engine) .	1595
ql/pricingengines/capfloor/ discretizedcapfloor.hpp (Discretized cap/floor)	1596
ql/pricingengines/capfloor/ marketmodelcapfloorengine.hpp (Market-model cap/floor engine)	1597
ql/pricingengines/capfloor/ mchullwhiteengine.hpp (Monte Carlo Hull-White engine for cap/floors)	1598
ql/pricingengines/capfloor/ treecapfloorengine.hpp (Numerical lattice engine for cap/floors)	1599
ql/pricingengines/cliquet/ analyticcliquetengine.hpp (Analytic Cliquet engine)	1600
ql/pricingengines/cliquet/ analyticperformanceengine.hpp (Analytic performance engine)	1601
ql/pricingengines/forward/ forwardengine.hpp (Forward (strike-resetting) option engine)	1602
ql/pricingengines/forward/ forwardperformanceengine.hpp (Forward (strike-resetting) performance option engines)	1603
ql/pricingengines/forward/ mcvarianceswapengine.hpp (Monte Carlo variance-swap engine)	1604
ql/pricingengines/forward/ replicatingvarianceswapengine.hpp (Replicating engine for variance swaps)	1605
ql/pricingengines/hybrid/ binomialconvertibleengine.hpp (Binomial engine for convertible bonds)	1608
ql/pricingengines/hybrid/ discretizedconvertible.hpp (Discretized convertible)	1609
ql/pricingengines/lookback/ analyticcontinuousfixedlookback.hpp (Analytic engine for continuous fixed-strike lookback)	1611
ql/pricingengines/lookback/ analyticcontinuousfloatinglookback.hpp (Analytic engine for continuous floating-strike lookback)	1612
ql/pricingengines/quanto/ quantoengine.hpp (Quanto option engine)	1615
ql/pricingengines/swaption/ blackswaptionengine.hpp (Black-formula swaption engine)	1616
ql/pricingengines/swaption/ discretizedswaption.hpp (Discretized swaption class) . . .	1617
ql/pricingengines/swaption/ g2swaptionengine.hpp (Swaption pricing engine for two-factor additive Gaussian Model G2++)	1618
ql/pricingengines/swaption/ jamshidianswaptionengine.hpp (Swaption engine using Jamshidian's decomposition)	1619
ql/pricingengines/swaption/ lfmswaptionengine.hpp (Libor forward model swaption engine based on black formula)	1620
ql/pricingengines/swaption/ treeswaptionengine.hpp (Numerical lattice engines for swaps and swaptions)	1621
ql/pricingengines/vanilla/ analyticdigitalamericanengine.hpp (Analytic digital American option engine)	1622
ql/pricingengines/vanilla/ analyticdividendeuropeanengine.hpp (Analytic discrete-dividend European engine)	1623
ql/pricingengines/vanilla/ analyticeuropeanengine.hpp (Analytic European engine) . . .	1624
ql/pricingengines/vanilla/ analytichestonengine.hpp (Analytic Heston-model engine) . .	1625
ql/pricingengines/vanilla/ baroneadesiwhaleyengine.hpp (Barone-Adesi and Whaley approximation engine)	1626
ql/pricingengines/vanilla/ batesengine.hpp (Analytic Bates model engine)	1627
ql/pricingengines/vanilla/ binomialengine.hpp (Binomial option engine)	1628
ql/pricingengines/vanilla/ bjerksundstenslandengine.hpp (Bjerksund and Stensland approximation engine)	1629

ql/pricingengines/vanilla/discretizedvanillaoption.hpp (Discretized vanilla option) . . .	1630
ql/pricingengines/vanilla/fdamericanengine.hpp (Finite-differences American option engine)	1631
ql/pricingengines/vanilla/fdbermudanengine.hpp (Finite-difference Bermudan engine)	1632
ql/pricingengines/vanilla/fdconditions.hpp (Finite-difference templates to generate engines)	1633
ql/pricingengines/vanilla/fddividendamericaneengine.hpp (American engine with discrete deterministic dividends)	1634
ql/pricingengines/vanilla/fddividendengine.hpp (Base engine for option with dividends)	1635
ql/pricingengines/vanilla/fddividendeuropeanengine.hpp (Finite-differences engine for European option with dividends)	1636
ql/pricingengines/vanilla/fddividendshoutengine.hpp (Base class for shout engine with dividends)	1637
ql/pricingengines/vanilla/fdeuropeanengine.hpp (Finite-difference European engine) .	1638
ql/pricingengines/vanilla/fdmultiengine.hpp (Base engine for options with events happening at specific times)	1639
ql/pricingengines/vanilla/fdshoutengine.hpp (Finite-differences shout engine)	1640
ql/pricingengines/vanilla/fdstepconditionengine.hpp (Finite-differences step-condition engine)	1641
ql/pricingengines/vanilla/fdvanillaengine.hpp (Finite-differences vanilla-option engine)	1642
ql/pricingengines/vanilla/integralengine.hpp (Integral option engine)	1643
ql/pricingengines/vanilla/jumpdiffusionengine.hpp (Jump diffusion (Merton 1976) engine)	1644
ql/pricingengines/vanilla/juquadraticengine.hpp (Ju quadratic (1999) approximation engine)	1645
ql/pricingengines/vanilla/mcamericanengine.hpp (American Monte Carlo engine) . . .	1646
ql/pricingengines/vanilla/mcdigitalengine.hpp (Digital option Monte Carlo engine) . .	1647
ql/pricingengines/vanilla/mceuropeanengine.hpp (Monte Carlo European option engine)	1648
ql/pricingengines/vanilla/mceuropeanhestonengine.hpp (Monte Carlo Heston-model engine for European options)	1649
ql/pricingengines/vanilla/mcvanillaengine.hpp (Monte Carlo vanilla option engine) . .	1650
ql/processes/blackscholesprocess.hpp (Black-Scholes processes)	1651
ql/processes/eulerdiscretization.hpp (Euler discretization for stochastic processes) . . .	1652
ql/processes/forwardmeasureprocess.hpp (Forward-measure stochastic processes) . . .	1653
ql/processes/g2process.hpp (G2 stochastic processes)	1654
ql/processes/geometricbrownianprocess.hpp (Geometric Brownian-motion process) . .	1655
ql/processes/hestonprocess.hpp (Heston stochastic process)	1656
ql/processes/hullwhiteprocess.hpp (Hull-White stochastic processes)	1657
ql/processes/lfmcovarParams.hpp (Volatility & correlation function for libor forward model process)	1658
ql/processes/lfmhullwhiteParams.hpp (Libor market model parameterization based on Hull White)	1659
ql/processes/lfmprocess.hpp (Stochastic process of a libor forward model)	1660
ql/processes/merton76process.hpp (Merton-76 process)	1661
ql/processes/ornsteinuhlenbeckprocess.hpp (Ornstein-Uhlenbeck process)	1662
ql/processes/squarerootprocess.hpp (Square-root process)	1663
ql/processes/stochasticprocessarray.hpp (Array of correlated 1-D stochastic processes) .	1664
ql/quotes/compositequote.hpp (Purely virtual base class for market observables)	1668
ql/quotes/derivedquote.hpp (Market quote whose value depends on another quote) . .	1669
ql/quotes/eurodollarfuturesquote.hpp (Quote for the Eurodollar-future implied standard deviation)	1670
ql/quotes/forwardvaluequote.hpp (Quote for the forward value of an index)	1671
ql/quotes/futuresconvadjustmentquote.hpp (Quote for the futures-convexity adjustment of an index)	1672

ql/quotes/ impliedstddevquote.hpp (Quote for the implied standard deviation of an underlying)	1673
ql/quotes/ simplequote.hpp (Simple quote class)	1674
ql/termstructures/volatilities/ blackconstantvol.hpp (Black constant volatility, no time dependence, no strike dependence)	1679
ql/termstructures/volatilities/ blackvariancecurve.hpp (Black volatility curve modelled as variance curve)	1680
ql/termstructures/volatilities/ blackvariancesurface.hpp (Black volatility surface modelled as variance surface)	1681
ql/termstructures/volatilities/ capflatvolvector.hpp (Cap/floor at-the-money flat volatility vector)	1682
ql/termstructures/volatilities/ capletconstantvol.hpp (Constant caplet volatility)	1683
ql/termstructures/volatilities/ capletvariancecurve.hpp (Caplet variance curve)	1684
ql/termstructures/volatilities/ capletvolatilitiesstructures.hpp (Caplet Volatilities Structures used during bootstrapping procedure)	1685
ql/termstructures/volatilities/ capstripper.hpp (Caplet volatility stripper)	1686
ql/termstructures/volatilities/ cmsmarket.hpp (Set of CMS quotes)	1687
ql/termstructures/volatilities/ impliedvoltermstructure.hpp (Implied Black Vol Term Structure)	1688
ql/termstructures/volatilities/ interpolatedsmilesection.hpp (Interpolated smile section class)	1689
ql/termstructures/volatilities/ localconstantvol.hpp (Local constant volatility, no time dependence, no asset dependence)	1690
ql/termstructures/volatilities/ localvolcurve.hpp (Local volatility curve derived from a Black curve)	1691
ql/termstructures/volatilities/ localvolsurface.hpp (Local volatility surface derived from a Black vol surface)	1692
ql/termstructures/volatilities/ sabr.hpp (SABR functions)	1693
ql/termstructures/volatilities/ sabrinterpolatedsmilesection.hpp (Interpolated smile section class)	1694
ql/termstructures/volatilities/ smilesection.hpp (Swaption volatility structure)	1695
ql/termstructures/volatilities/ swaptionconstantvol.hpp (Constant swaption volatility)	1696
ql/termstructures/volatilities/ swaptionvolcube.hpp (Swaption volatility cube)	1697
ql/termstructures/volatilities/ swaptionvolcube1.hpp (Swaption volatility cube, fit-early-interpolate-later approach)	1698
ql/termstructures/volatilities/ swaptionvolcube2.hpp (Swaption volatility cube, fit-later-interpolate-early approach)	1699
ql/termstructures/volatilities/ swaptionvoldiscrete.hpp (Discretized swaption volatility)	1700
ql/termstructures/volatilities/ swaptionvolmatrix.hpp (Swaption at-the-money volatility matrix)	1701
ql/termstructures/yieldcurves/ bondhelpers.hpp (Bond rate helpers)	1702
ql/termstructures/yieldcurves/ bootstraptraits.hpp (Bootstrap traits)	1703
ql/termstructures/yieldcurves/ compoundforward.hpp (Compounded forward term structure)	1704
ql/termstructures/yieldcurves/ discountcurve.hpp (Interpolated discount factor structure)	1705
ql/termstructures/yieldcurves/ drifttermstructure.hpp (Drift term structure)	1706
ql/termstructures/yieldcurves/ extendeddiscountcurve.hpp (Discount factor structure with detailed compound-forward calculation)	1707
ql/termstructures/yieldcurves/ flatforward.hpp (Flat forward rate term structure)	1708
ql/termstructures/yieldcurves/ forwardcurve.hpp (Interpolated forward-rate structure)	1709
ql/termstructures/yieldcurves/ forwardspreadedtermstructure.hpp (Forward-spreaded term structure)	1710
ql/termstructures/yieldcurves/ forwardstructure.hpp (Forward-based yield term structure)	1711

ql/termstructures/yieldcurves/ impliedtermstructure.hpp (Implied term structure) . . .	1712
ql/termstructures/yieldcurves/ piecisewiseyieldcurve.hpp (Piecewise-interpolated term structure)	1713
ql/termstructures/yieldcurves/ piecisewisezerospreadedtermstructure.hpp (Piecewise-zero-spreaded term structure)	1714
ql/termstructures/yieldcurves/ quantotermstructure.hpp (Quanto term structure)	1715
ql/termstructures/yieldcurves/ ratehelpers.hpp (Deposit, FRA, futures, and swap rate helpers)	1716
ql/termstructures/yieldcurves/ zerocurve.hpp (Interpolated zero-rates structure)	1717
ql/termstructures/yieldcurves/ zerospreadedtermstructure.hpp (Zero spreaded term structure)	1718
ql/termstructures/yieldcurves/ zeroyieldstructure.hpp (Zero-yield based term structure) .	1719
ql/time/ businessdayconvention.hpp (BusinessDayConvention enumeration)	1720
ql/time/ calendar.hpp (calendar class)	1721
ql/time/ date.hpp (Date- and time-related classes, typedefs and enumerations)	1757
ql/time/ frequency.hpp (Frequency enumeration)	1766
ql/time/ imm.hpp (IMM-related date functions)	1767
ql/time/ period.hpp (Period- and frequency-related classes and enumerations)	1768
ql/time/ schedule.hpp (Date schedule)	1770
ql/time/ timeunit.hpp (TimeUnit enumeration)	1771
ql/time/ weekday.hpp (Weekday enumeration)	1772
ql/time/calendars/ argentina.hpp (Argentinian calendars)	1722
ql/time/calendars/ australia.hpp (Australian calendar)	1723
ql/time/calendars/ brazil.hpp (Brazilian calendar)	1724
ql/time/calendars/ canada.hpp (Canadian calendar)	1725
ql/time/calendars/ china.hpp (Chinese calendar)	1726
ql/time/calendars/ czechrepublic.hpp (Czech calendars)	1727
ql/time/calendars/ denmark.hpp (Danish calendar)	1728
ql/time/calendars/ finland.hpp (Finnish calendar)	1729
ql/time/calendars/ germany.hpp (German calendars)	1730
ql/time/calendars/ hongkong.hpp (Hong Kong calendars)	1731
ql/time/calendars/ hungary.hpp (Hungarian calendar)	1732
ql/time/calendars/ iceland.hpp (Icelandic calendars)	1733
ql/time/calendars/ india.hpp (Indian calendars)	1734
ql/time/calendars/ indonesia.hpp (Indonesian calendars)	1735
ql/time/calendars/ italy.hpp (Italian calendars)	1736
ql/time/calendars/ japan.hpp (Japanese calendar)	1737
ql/time/calendars/ jointcalendar.hpp (Joint calendar)	1738
ql/time/calendars/ mexico.hpp (Mexican calendars)	1739
ql/time/calendars/ newzealand.hpp (New Zealand calendar)	1740
ql/time/calendars/ norway.hpp (Norwegian calendar)	1741
ql/time/calendars/ nullcalendar.hpp (Calendar for reproducing theoretical calculations) .	1742
ql/time/calendars/ poland.hpp (Polish calendar)	1743
ql/time/calendars/ saudiarabia.hpp (Saudi Arabian calendar)	1744
ql/time/calendars/ singapore.hpp (Singapore calendars)	1745
ql/time/calendars/ slovakia.hpp (Slovak calendars)	1746
ql/time/calendars/ southafrica.hpp (South-African calendar)	1747
ql/time/calendars/ southkorea.hpp (South Korean calendars)	1748
ql/time/calendars/ sweden.hpp (Swedish calendar)	1749
ql/time/calendars/ switzerland.hpp (Swiss calendar)	1750
ql/time/calendars/ taiwan.hpp (Taiwanese calendars)	1751
ql/time/calendars/ target.hpp (TARGET calendar)	1752
ql/time/calendars/ turkey.hpp (Turkish calendar)	1753
ql/time/calendars/ ukraine.hpp (Ukrainian calendars)	1754

ql/time/calendars/ unitedkingdom.hpp (UK calendars)	1755
ql/time/calendars/ unitedstates.hpp (US calendars)	1756
ql/time/daycounters/ actual360.hpp (Act/360 day counter)	1759
ql/time/daycounters/ actual365fixed.hpp (Actual/365 (Fixed) day counter)	1760
ql/time/daycounters/ actualactual.hpp (Act/act day counters)	1761
ql/time/daycounters/ business252.hpp (Business/252 day counter)	1762
ql/time/daycounters/ one.hpp (1/1 day counter)	1763
ql/time/daycounters/ simplifiedaycounter.hpp (Simple day counter for reproducing theo- retical calculations)	1764
ql/time/daycounters/ thirty360.hpp (30/360 day counters)	1765
ql/utilities/ clone.hpp (Cloning proxy to an underlying object)	1777
ql/utilities/ dataformatters.hpp (Output manipulators)	1778
ql/utilities/ dataparsers.hpp (Classes used to parse data for input)	1780
ql/utilities/ disposable.hpp (Generic disposable object with move semantics)	1781
ql/utilities/ null.hpp (Null values)	1782
ql/utilities/ observablevalue.hpp (Observable and assignable proxy to concrete value)	1783
ql/utilities/ steppingiterator.hpp (Iterator advancing in constant steps)	1784
ql/utilities/ tracing.hpp (Tracing facilities)	1785

Chapter 7

QuantLib Module Documentation

7.1 Numeric types

7.1.1 Detailed Description

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

Typedefs

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

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

7.1.2 Typedef Documentation

7.1.2.1 typedef QL_INTEGER Integer

integer number

Examples:

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

7.1.2.2 typedef QL_BIG_INTEGER BigInteger

large integer number

7.1.2.3 typedef unsigned QL_INTEGER Natural

positive integer

7.1.2.4 typedef QL_REAL Real

real number

Examples:

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

7.1.2.5 typedef Real Decimal

decimal number

7.1.2.6 typedef std::size_t Size

size of a container

Examples:

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

7.1.2.7 `typedef Real Time`

continuous quantity with 1-year units

Examples:

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

7.1.2.8 `typedef Real DiscountFactor`

discount factor between dates

Examples:

[DiscreteHedging.cpp](#).

7.1.2.9 `typedef Real Rate`

interest rates

Examples:

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

7.1.2.10 `typedef Real Spread`

spreads on interest rates

Examples:

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

7.1.2.11 `typedef Real Volatility`

volatility

Examples:

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

7.2 Currencies and FX rates

Classes

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

Iraqi dinar.

- class [IRRCurrency](#)

Iranian rial.

- class [JPYCurrency](#)

Japanese yen.

- class [KRWCurrency](#)

South-Korean won.

- class [KWDCurrency](#)

Kuwaiti dinar.

- class [NPRCurrency](#)

Nepal rupee.

- class [PKRCurrency](#)

Pakistani rupee.

- class [SARCurrency](#)

Saudi riyal.

- class [SGDCurrency](#)

Singapore dollar

- class [THBCurrency](#)

Thai baht.

- class [TWDCurrency](#)

Taiwan dollar

- class [BGLCurrency](#)

Bulgarian lev.

- class [BYRCurrency](#)

Belarussian ruble.

- class [CHFCurrency](#)

Swiss franc.

- class [CYPCurrency](#)

Cyprus pound.

- class [CZKCurrency](#)

Czech koruna.

- class [DKKCurrency](#)

Danish krone.

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

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

7.3 Date and time calculations

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

Typedefs

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

Enumerations

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

7.3.2 Typedef Documentation

7.3.2.1 typedef Integer Day

Day number.

7.3.2.2 typedef Integer Year

Year number.

7.3.3 Enumeration Type Documentation

7.3.3.1 enum BusinessDayConvention

Business Day conventions.

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

Enumerator:

Following Choose the first business day after the given holiday.

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

Preceding Choose the first business day before the given holiday.

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

Unadjusted Do not adjust.

7.3.3.2 enum Month

Month names.

7.3.3.3 enum Frequency

Frequency of events.

Enumerator:

NoFrequency null frequency

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

Annual once a year

Semiannual twice a year

EveryFourthMonth every fourth month

Quarterly every third month

Bimonthly every second month

Monthly once a month

Biweekly every second week

Weekly once a week

Daily once a day

7.3.3.4 enum TimeUnit

Units used to describe time periods.

7.3.3.5 enum Weekday

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

7.4 Calendars

7.4.1 Detailed Description

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

Classes

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

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

Taiwanese calendars.

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

7.5 Day counters

7.5.1 Detailed Description

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

Classes

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

7.6 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](#)

7.7 Asian option engines

7.7.1 Detailed Description

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.

7.8 Barrier option engines

7.8.1 Detailed Description

Classes

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

7.9 Basket option engines

7.9.1 Detailed Description

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.

7.10 Cap/floor engines

7.10.1 Detailed Description

Classes

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

7.11 Cliquet option engines

7.11.1 Detailed Description

Classes

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

7.12 Forward option engines

7.12.1 Detailed Description

Classes

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

7.13 Quanto option engines

7.13.1 Detailed Description

Classes

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

7.14 Swaption engines

7.14.1 Detailed Description

Classes

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

7.15 Vanilla option engines

7.15.1 Detailed Description

Classes

- class [AnalyticDigitalAmericanEngine](#)
Analytic pricing engine for American vanilla options with digital payoff.
- class [AnalyticDividendEuropeanEngine](#)
Analytic pricing engine for European options with discrete dividends.
- class [AnalyticEuropeanEngine](#)
Pricing engine for European vanilla options using analytical formulae.
- class [AnalyticHestonEngine](#)
analytic Heston-model engine based on Fourier transform
- class [BaroneAdesiWhaleyApproximationEngine](#)
Barone-Adesi and Whaley pricing engine for American options (1987).
- class [BatesEngine](#)
Bates model engines based on Fourier transform.
- class [BinomialVanillaEngine](#)
Pricing engine for vanilla options using binomial trees.
- class [Bjerk SundStenslandApproximationEngine](#)
Bjerk Sund and Stensland pricing engine for American options (1993).
- class [FDBermudanEngine](#)
Finite-differences Bermudan engine.
- class [FDDividendEngineMerton73](#)
Finite-differences pricing engine for dividend options using.
- class [FDDividendEngineShiftScale](#)
Finite-differences pricing engine for dividend options using.
- class [FDEuropeanEngine](#)
Pricing engine for European options using finite-differences.
- class [FDStepConditionEngine](#)
Finite-differences pricing engine for American-style vanilla options.
- class [IntegralEngine](#)
Pricing engine for European vanilla options using integral approach.
- class [JumpDiffusionEngine](#)

Jump-diffusion engine for vanilla options.

- class [JuQuadraticApproximationEngine](#)
Pricing engine for American options with Ju quadratic approximation.
- class [MCAmericanEngine](#)
American Monte Carlo engine.
- class [MCDigitalEngine](#)
Pricing engine for digital options using Monte Carlo simulation.
- class [MCEuropeanEngine](#)
European option pricing engine using Monte Carlo simulation.
- class [MCEuropeanHestonEngine](#)
Monte Carlo Heston-model engine for European options.
- class [MCVanillaEngine](#)
Pricing engine for vanilla options using Monte Carlo simulation.

Typedefs

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

7.15.2 Typedef Documentation

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

Finite-differences pricing engine for American one asset options.

Tests

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

Examples:

[EquityOption.cpp](#).

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

Finite-differences pricing engine for dividend American options.

Tests

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

Bug

results are not overly reliable.

Bug

method `impliedVolatility()` utterly fails

7.15.2.3 `typedef FDEngineAdapter<FDDividendEngine, DividendVanillaOption::engine> FDDividendEuropeanEngine`

Finite-differences pricing engine for dividend European options.

Tests

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

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

Finite-differences shout engine with dividends.

Bug

results are not overly reliable.

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

Finite-differences pricing engine for shout vanilla options.

Tests

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

7.16 Finite-differences framework

7.16.1 Detailed Description

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

Classes

- class [BoundaryCondition](#)
Abstract boundary condition class for finite difference problems.
- class [NeumannBC](#)
Neumann boundary condition (i.e., constant derivative).
- class [DirichletBC](#)
Neumann boundary condition (i.e., constant value).
- class [BSMOperator](#)
Black-Scholes-Merton differential operator.
- class [CrankNicolson](#)
Crank-Nicolson scheme for finite difference methods.
- class [DMinus](#)
 D_- matricial representation
- class [DPlus](#)
 D_+ matricial representation
- class [DPlusDMinus](#)
 D_+D_- matricial representation
- class [DZero](#)
 D_0 matricial representation
- class [ExplicitEuler](#)
Forward Euler scheme for finite difference methods
- class [FiniteDifferenceModel](#)
Generic finite difference model.
- class [ImplicitEuler](#)
Backward Euler scheme for finite difference methods.
- class [MixedScheme](#)
Mixed (explicit/implicit) scheme for finite difference methods.

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

Typedefs

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

7.16.2 Typedef Documentation

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

Black-Scholes-Merton differential operator.

Tests

coefficients are tested against constant BSM operator

7.16.2.2 typedef `PdeOperator<PdeShortRate>` [OneFactorOperator](#)

Interest-rate single factor model differential operator.

7.17 Short-rate modelling framework

7.17.1 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).

7.17.2 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).

7.17.3 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.

7.17.4 Two-factor models

7.17.5 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

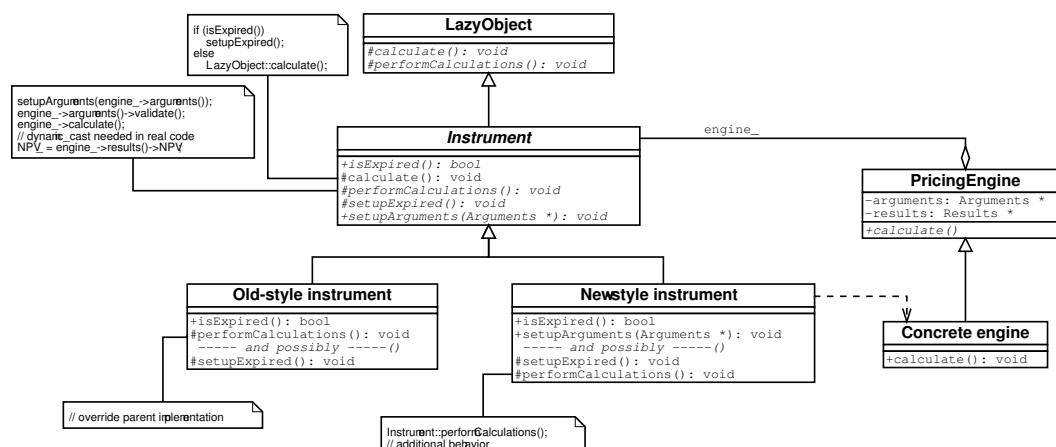
- class [AffineModel](#)
Affine model class.
- class [TermStructureConsistentModel](#)
Term-structure consistent model class.
- class [ShortRateModel](#)
Abstract short-rate model class.

- 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.

7.18 Financial instruments

7.18.1 Detailed Description

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



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

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

```

class FooArguments : public Arguments { ... };

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

class FooOptionArguments : public FooArguments { ... };

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

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

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

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

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

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

Classes

- class [ContinuousAveragingAsianOption](#)

Continuous-averaging Asian option.

- class [DiscreteAveragingAsianOption](#)

Discrete-averaging Asian option.

- class [AssetSwap](#)

Bullet bond vs Libor swap.

- class [BarrierOption](#)

Barrier option on a single asset.

- class [BasketOption](#)

Basket option on a number of assets.

- class [Bond](#)

Base bond class.

- class [CapFloor](#)

Base class for cap-like instruments.

- class [Cap](#)

Concrete cap class.

- class [Floor](#)

Concrete floor class.

- class [Collar](#)

Concrete collar class.

- class [CliquetOption](#)

cliquet (Ratchet) option

- class [CmsRateBond](#)

CMS-rate bond.

- class [CompositeInstrument](#)
Composite instrument
- class [DividendVanillaOption](#)
Single-asset vanilla option (no barriers) with discrete dividends.
- class [EuropeanOption](#)
European option on a single asset.
- class [FixedRateBond](#)
fixed-rate bond
- class [FixedRateBondForward](#)
Forward contract on a fixed-rate bond
- class [FloatingRateBond](#)
floating-rate bond (possibly capped and/or floored)
- class [Forward](#)
Abstract base forward class.
- class [ForwardRateAgreement](#)
Forward rate agreement (FRA) class
- class [ForwardVanillaOption](#)
Forward version of a vanilla option
- class [ContinuousFloatingLookbackOption](#)
Continuous-floating lookback option.
- class [ContinuousFixedLookbackOption](#)
Continuous-fixed lookback option.
- class [QuantoForwardVanillaOption](#)
Quanto version of a forward vanilla option.
- class [QuantoVanillaOption](#)
quanto version of a vanilla option
- class [Stock](#)
Simple stock class.
- class [Swap](#)
Interest rate swap.
- class [Swaption](#)
Swaption class
- class [VanillaOption](#)
Vanilla option (no discrete dividends, no barriers) on a single asset.

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

7.19 Lattice methods

7.19.1 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.

7.19.2 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 `bsmlattice.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.

7.19.3 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::TrinomialTree` 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

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

7.19.4 Bidimensional lattices

To come...

7.19.5 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 [TreeLattice](#)
Tree-based lattice-method base class.
- class [TreeLattice1D](#)

One-dimensional tree-based lattice.

- class [TreeLattice2D](#)

Two-dimensional tree-based lattice.

- class [TsiveriotisFernandesLattice](#)

Binomial lattice approximating the Tsiveriotis-Fernandes model.

- class [Tree](#)

Tree approximating a single-factor diffusion

- class [TrinomialTree](#)

Recombining trinomial tree class.

7.20 Math tools

Math facilities of the library include:

7.20.1 Pseudo-random number and low-discrepancy sequence generators

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

7.20.2 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.

7.20.3 Optimizers

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

The simplex method

This method, implemented in [QuantLib::Simplex](#), is rather raw and requires quite a lot of computing resources, but it has the advantage that it does not need any evaluation of the cost function's gradient, and that it is quite easily implemented. First, we must choose $N+1$ starting points, given here by a starting point P_0 and N points such that

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

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

The conjugate gradient method

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

by

$$\mathbf{d}_i = -\nabla f(\mathbf{x}_i) + \frac{\|\nabla f(\mathbf{x}_i)\|^2}{\|\nabla f(\mathbf{x}_{i-1})\|^2} \mathbf{d}_{i-1},$$
$$\mathbf{d}_0 = -\nabla f(\mathbf{x}_0).$$

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

7.21 Monte Carlo framework

7.21.1 Detailed Description

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

Classes

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

7.22 Design patterns

Classes

- 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

7.23 Stochastic processes

7.23.1 Detailed Description

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

Classes

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

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

7.24 Term structures

7.24.1 Detailed Description

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

Classes

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

Typedefs

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

7.24.2 Typedef Documentation

7.24.2.1 `typedef InterpolatedDiscountCurve<LogLinear> DiscountCurve`

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

Log-linear interpolation guarantees piecewise-constant forward rates.

7.24.2.2 `typedef InterpolatedForwardCurve<BackwardFlat> ForwardCurve`

Term structure based on flat interpolation of forward rates.

7.24.2.3 `typedef InterpolatedZeroCurve<Linear> ZeroCurve`

Term structure based on linear interpolation of zero yields.

7.25 Utilities

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

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

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

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

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

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

`QuantLib::coupling_iterator` allows us to do the same thing without allocating the extra vector: what we do is simply:

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

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

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

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

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

7.26 QuantLib macros

7.26.1 Detailed Description

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

Modules

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

Defines

- `#define QL_VERSION "0.8.1"`
version string
- `#define QL_HEX_VERSION 0x000801f0`
version hexadecimal number
- `#define QL_LIB_VERSION "0_8_1"`
version string for output lib name

7.27 Generic macros

7.27.1 Detailed Description

Miscellaneous macros for compiler idiosyncrasies not fitting other categories.

Defines

- `#define QL_DUMMY_RETURN(x)`
Is a dummy return statement required?

7.27.2 Define Documentation

7.27.2.1 `#define QL_DUMMY_RETURN(x)`

Is a dummy return statement required?

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

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

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

7.28 Numeric limits

7.28.1 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)())`

7.28.2 Define Documentation

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

Defines the value of the largest representable negative integer value

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

Defines the value of the largest representable integer value

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

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

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

Defines the value of the smallest representable positive double value

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

Defines the value of the largest representable floating-point value

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

Defines the machine precision for operations over doubles

7.29 Template capabilities

7.29.1 Detailed Description

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

Defines

- `#define QL_TYPENAME typename`

7.29.2 Define Documentation

7.29.2.1 `#define QL_TYPENAME typename`

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

7.30 Iterator support

7.30.1 Detailed Description

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

Defines

- `#define QL_FULL_ITERATOR_SUPPORT`

7.30.2 Define Documentation

7.30.2.1 `#define QL_FULL_ITERATOR_SUPPORT`

Some compilers (most notably, Visual C++ 6) still do not fully support iterators in their STL implementation. This macro can be used to select between alternate implementations of blocks of code, namely, one that takes advantage of full iterator support and a less efficient one which is compatible with all compilers.

7.31 Output manipulators

7.31.1 Detailed Description

Helper functions for creating formatted output.

Functions

- detail::short_date_holder [short_date](#) (const Date &)
output dates in short format (mm/dd/yyyy)
- detail::long_date_holder [long_date](#) (const Date &)
output dates in long format (Month ddth, yyyy)
- detail::iso_date_holder [iso_date](#) (const Date &)
output dates in ISO format (yyyy-mm-dd)
- detail::long_period_holder [long_period](#) (const Period &)
output periods in long format (e.g. "2 weeks")
- detail::short_period_holder [short_period](#) (const Period &)
output periods in short format (e.g. "2w")
- detail::long_weekday_holder [long_weekday](#) (Weekday)
output weekdays in long format
- detail::short_weekday_holder [short_weekday](#) (Weekday)
output weekdays in short format (three letters)
- detail::shortest_weekday_holder [shortest_weekday](#) (Weekday)
output weekdays in shortest format (two letters)
- template<typename T>
detail::null_checker< T > [checknull](#) (T)
check for nulls before output
- detail::ordinal_holder [ordinal](#) (Size)
outputs naturals as 1st, 2nd, 3rd...
- template<typename T>
detail::power_of_two_holder< T > [power_of_two](#) (T)
output integers as powers of two
- detail::percent_holder [percent](#) (Real)
output reals as percentages
- detail::percent_holder [rate](#) (Rate)
output rates and spreads as percentages

- `detail::percent_holder` [volatility](#) (Volatility)
output volatilities as percentages

7.31.2 Function Documentation

7.31.2.1 `detail::short_date_holder` `QuantLib::io::short_date` (const Date &)

output dates in short format (mm/dd/yyyy)

7.31.2.2 `detail::long_date_holder` `QuantLib::io::long_date` (const Date &)

output dates in long format (Month ddth, yyyy)

7.31.2.3 `detail::iso_date_holder` `QuantLib::io::iso_date` (const Date &)

output dates in ISO format (yyyy-mm-dd)

7.31.2.4 `detail::long_period_holder` `QuantLib::io::long_period` (const Period &)

output periods in long format (e.g. "2 weeks")

7.31.2.5 `detail::short_period_holder` `QuantLib::io::short_period` (const Period &)

output periods in short format (e.g. "2w")

7.31.2.6 `detail::long_weekday_holder` `QuantLib::io::long_weekday` (Weekday)

output weekdays in long format

7.31.2.7 `detail::short_weekday_holder` `QuantLib::io::short_weekday` (Weekday)

output weekdays in short format (three letters)

7.31.2.8 `detail::shortest_weekday_holder` `QuantLib::io::shortest_weekday` (Weekday)

output weekdays in shortest format (two letters)

7.32 Debugging macros

7.32.1 Detailed Description

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

Defines

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

7.32.2 Define Documentation

7.32.2.1 `#define QL_TRACE_ENABLE`

enable tracing

The statement

```
QL_TRACE_ENABLE;
```

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

Examples:

[tracing_example.cpp](#).

7.32.2.2 **#define QL_TRACE_DISABLE**

disable tracing

The statement

```
QL_TRACE_DISABLE;
```

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

7.32.2.3 **#define QL_TRACE_ON(out)**

set tracing stream

The statement

```
QL_TRACE_ON(stream);
```

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

7.32.2.4 **#define QL_TRACE(message)**

output tracing information

The statement

```
QL_TRACE(message);
```

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

Examples:

[tracing_example.cpp](#).

7.32.2.5 **#define QL_TRACE_ENTER_FUNCTION**

output tracing information

The statement

```
QL_TRACE_ENTER_FUNCTION;
```

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

Examples:

[tracing_example.cpp](#).

7.32.2.6 **#define QL_TRACE_EXIT_FUNCTION**

output tracing information

The statement

```
QL_TRACE_EXIT_FUNCTION;
```

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

Examples:

[tracing_example.cpp](#).

7.32.2.7 **#define QL_TRACE_LOCATION**

output tracing information

The statement

```
QL_TRACE_LOCATION;
```

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

Examples:

[tracing_example.cpp](#).

7.32.2.8 **#define QL_TRACE_VARIABLE(variable)**

output tracing information

The statement

```
QL_TRACE_VARIABLE(variable);
```

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

Examples:

[tracing_example.cpp](#).

Chapter 8

QuantLib Namespace Documentation

8.1 `std` Namespace Reference

8.1.1 Detailed Description

STL namespace.

Chapter 9

QuantLib Class Documentation

9.1 Abcd Class Reference

```
#include <ql/termstructures/volatilities/abcd.hpp>
```

9.1.1 Detailed Description

Abcd functional form for instantaneous volatility

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

following Rebonato notation.

Public Member Functions

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

instantaneous volatility at time to maturity u:

$$f(u)$$

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

instantaneous volatility when time to maturity = 0.0

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

Friends

- class [AbcdCostFunction](#)

9.1.2 Member Function Documentation

9.1.2.1 Real instantaneousVolatility (Time t , Time T) const

instantaneous volatility at time t of the T-fixing rate:

$$f(T - t)$$

9.1.2.2 Real instantaneousVariance (Time t , Time T) const

instantaneous variance at time t of T-fixing rate:

$$f(T - t)f(T - t)$$

9.1.2.3 Real instantaneousCovariance (Time u , Time T , Time S) const

instantaneous covariance at time t between T and S fixing rates:

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

9.1.2.4 Real volatility (Time $tMin$, Time $tMax$, Time T) const

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

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

9.1.2.5 Real variance (Time $tMin$, Time $tMax$, Time T) const

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

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

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

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

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

9.2 AbcdFunction Struct Reference

```
#include <ql/termstructures/volatilities/abcd.hpp>
```

9.2.1 Detailed Description

Abcd functional form for instantaneous volatility

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

following Rebonato's notation.

Public Member Functions

- **AbcdFunction** (Real a=-0.06, Real b=0.17, Real c=0.54, Real d=0.17)

- Real **operator()** (Time u) const

volatility function value at time u:

$$f(u)$$

- Real **maximumLocation** () const

time at which the volatility function reaches maximum (if any)

- Real **maximumValue** () const

maximum value of the volatility function

- Real **shortTermValue** () const

volatility function value at time 0:

$$f(0)$$

- Real **longTermValue** () const

volatility function value at time +inf:

$$f(\text{inf})$$

- Real **covariance** (Time t, Time T, Time S) const
- Real **covariance** (Time t1, Time t2, Time T, Time S) const
- Real **primitive** (Time t, Time T, Time S) const
- Real **volatility** (Time T, Time tMax, Time tMin) const
- Real **variance** (Time T, Time tMax, Time tMin) const

Public Attributes

- Real **a_**
- Real **b_**
- Real **c_**
- Real **d_**

9.2.2 Member Function Documentation

9.2.2.1 Real covariance (Time t , Time T , Time S) const

instantaneous covariance function at time t between T-fixing and S-fixing rates

$$f(T - t)f(S - t)$$

9.2.2.2 Real covariance (Time $t1$, Time $t2$, Time T , Time S) const

integral of the instantaneous covariance function between time $t1$ and $t2$ for T-fixing and S-fixing rates

$$\int_{t1}^{t2} f(T - t)f(S - t)dt$$

9.2.2.3 Real primitive (Time t , Time T , Time S) const

indefinite integral of the instantaneous covariance function at time t between T-fixing and S-fixing rates

$$\int f(T - t)f(S - t)dt$$

9.2.2.4 Real volatility (Time T , Time $tMax$, Time $tMin$) const

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

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

9.2.2.5 Real variance (Time T , Time $tMax$, Time $tMin$) const

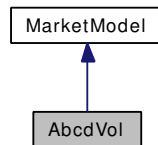
variance in $[tMin, tMax]$ of T-fixing rate:

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

9.3 AbcdVol Class Reference

```
#include <ql/models/marketmodels/models/abcdvol.hpp>
```

Inheritance diagram for AbcdVol:



9.3.1 Detailed Description

Abcd-interpolated volatility structure

Public Member Functions

- **AbcdVol** (Real a, Real b, Real c, Real d, const std::vector< Real > &ks, const boost::shared_ptr< PiecewiseConstantCorrelation > &corr, const [EvolutionDescription](#) &evolution, const Size numberOfFactors, const std::vector< Rate > &initialRates, const std::vector< Spread > &displacements)

MarketModel interface

- const std::vector< Rate > & **initialRates** () const
- const std::vector< Spread > & **displacements** () const
- const [EvolutionDescription](#) & **evolution** () const
- Size **numberOfRates** () const
- Size **numberOfFactors** () const
- Size **numberOfSteps** () const
- const [Matrix](#) & **pseudoRoot** (Size i) const

9.4 AccountingEngine Class Reference

```
#include <ql/models/marketmodels/accountingengine.hpp>
```

9.4.1 Detailed Description

Engine collecting cash flows along a market-model simulation.

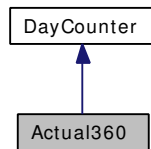
Public Member Functions

- **AccountingEngine** (const boost::shared_ptr< [MarketModelEvolver](#) > &evolver, const [Clone](#)< [MarketModelMultiProduct](#) > &product, Real initialNumeraireValue)
- void **multiplePathValues** ([SequenceStatistics](#) &stats, Size numberOfPaths)

9.5 Actual360 Class Reference

```
#include <ql/time/daycounters/actual360.hpp>
```

Inheritance diagram for Actual360:



9.5.1 Detailed Description

Actual/360 day count convention.

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

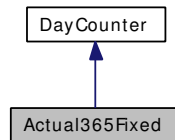
Examples:

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

9.6 Actual365Fixed Class Reference

```
#include <ql/time/daycounters/actual365fixed.hpp>
```

Inheritance diagram for Actual365Fixed:



9.6.1 Detailed Description

Actual/365 (Fixed) day count convention.

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

Warning

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

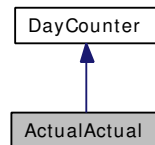
Examples:

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

9.7 ActualActual Class Reference

```
#include <ql/time/daycounters/actualactual.hpp>
```

Inheritance diagram for ActualActual:



9.7.1 Detailed Description

Actual/Actual day count.

The day count can be calculated according to:

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

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

Tests

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

Examples:

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

Public Types

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

Public Member Functions

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

9.8 AcyclicVisitor Class Reference

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

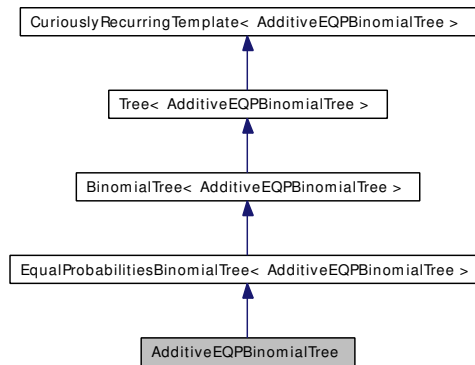
9.8.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

9.9 AdditiveEQPBinoomialTree Class Reference

```
#include <ql/methods/lattices/binomialtree.hpp>
```

Inheritance diagram for AdditiveEQPBinoomialTree:



9.9.1 Detailed Description

Additive equal probabilities binomial tree.

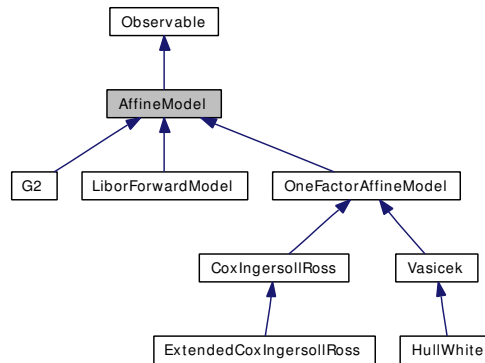
Public Member Functions

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

9.10 AffineModel Class Reference

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

Inheritance diagram for AffineModel:



9.10.1 Detailed Description

Affine model class.

Base class for analytically tractable models.

Public Member Functions

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

9.11 AmericanCondition Class Reference

```
#include <ql/methods/finitedifferences/americancondition.hpp>
```

9.11.1 Detailed Description

American exercise condition.

Todo

unify the intrinsicValues/Payoff thing

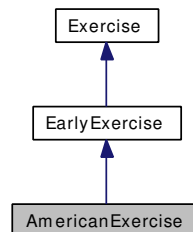
Public Member Functions

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

9.12 AmericanExercise Class Reference

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

Inheritance diagram for AmericanExercise:



9.12.1 Detailed Description

American exercise.

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

Todo

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

Examples:

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

Public Member Functions

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

9.13 AmericanPayoffAtExpiry Class Reference

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

9.13.1 Detailed Description

Analytic formula for American exercise payoff at-expiry options.

Todo

calculate greeks

Public Member Functions

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

9.14 AmericanPayoffAtHit Class Reference

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

9.14.1 Detailed Description

Analytic formula for American exercise payoff at-hit options.

Todo

calculate greeks

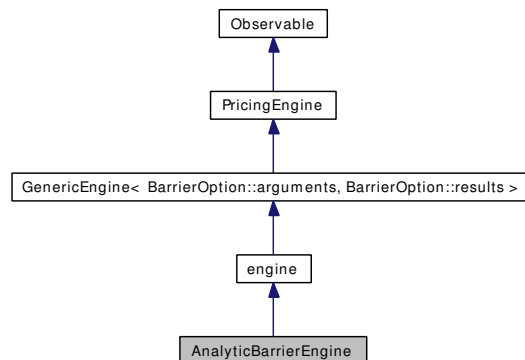
Public Member Functions

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

9.15 AnalyticBarrierEngine Class Reference

```
#include <ql/pricingengines/barrier/analyticbarrierengine.hpp>
```

Inheritance diagram for AnalyticBarrierEngine:



9.15.1 Detailed Description

Pricing engine for barrier options using analytical formulae.

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

Tests

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

Todo

rework to avoid repeated casts inside utility methods

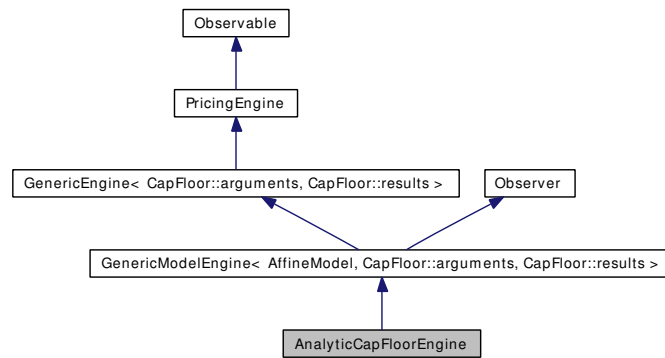
Public Member Functions

- void **calculate** () const

9.16 AnalyticCapFloorEngine Class Reference

```
#include <ql/pricingengines/capfloor/analyticcapfloorengine.hpp>
```

Inheritance diagram for AnalyticCapFloorEngine:



9.16.1 Detailed Description

Analytic engine for cap/floor.

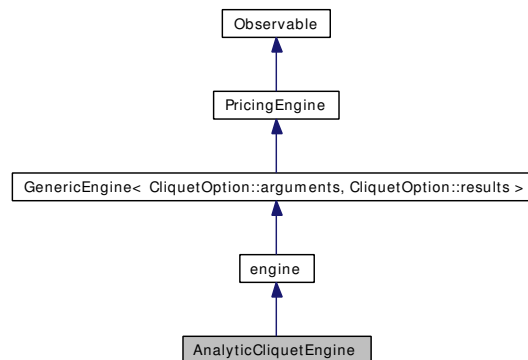
Public Member Functions

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

9.17 AnalyticCliquetEngine Class Reference

```
#include <ql/pricingengines/cliquet/analyticcliquetengine.hpp>
```

Inheritance diagram for AnalyticCliquetEngine:



9.17.1 Detailed Description

Pricing engine for Cliquet options using analytical formulae.

Tests

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

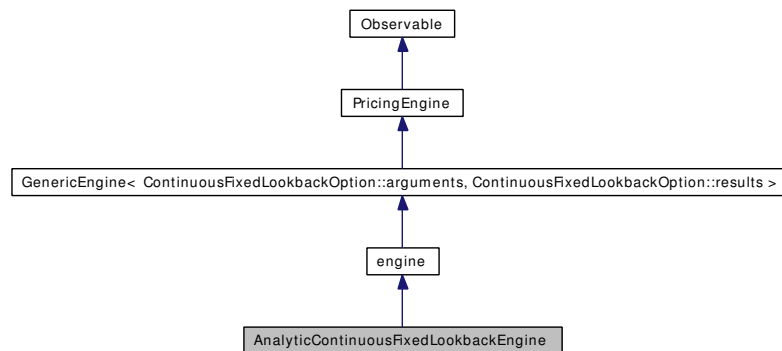
Public Member Functions

- void **calculate** () const

9.18 AnalyticContinuousFixedLookbackEngine Class Reference

```
#include <ql/pricingengines/lookback/analyticcontinuousfixedlookback.hpp>
```

Inheritance diagram for AnalyticContinuousFixedLookbackEngine:



9.18.1 Detailed Description

Pricing engine for European continuous fixed-strike lookback.

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

Tests

returned values are verified against results from literature

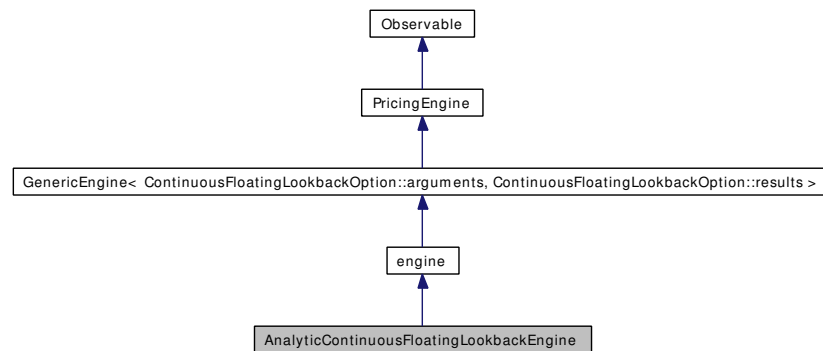
Public Member Functions

- `void calculate () const`

9.19 AnalyticContinuousFloatingLookbackEngine Class Reference

```
#include <ql/pricingengines/lookback/analyticcontinuousfloatinglookback.hpp>
```

Inheritance diagram for AnalyticContinuousFloatingLookbackEngine:



9.19.1 Detailed Description

Pricing engine for European continuous floating-strike lookback.

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

Tests

returned values verified against results from literature

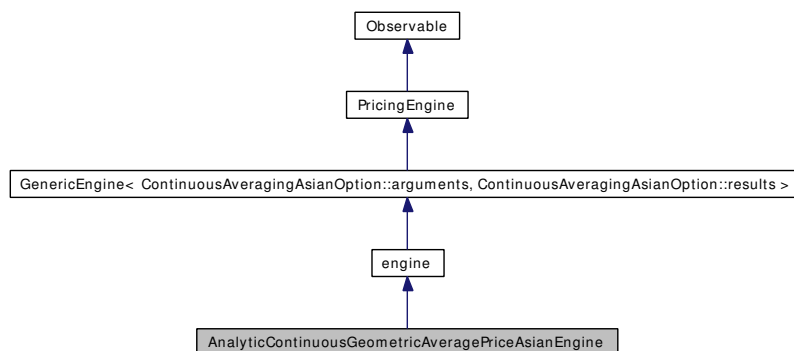
Public Member Functions

- `void calculate () const`

9.20 AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference

```
#include <ql/pricingengines/asian/analytic_cont_geom_av_price.hpp>
```

Inheritance diagram for AnalyticContinuousGeometricAveragePriceAsianEngine:



9.20.1 Detailed Description

Pricing engine for European continuous geometric average price Asian.

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

Tests

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

Todo

handle seasoned options

Public Member Functions

- void **calculate** () const

9.21 AnalyticDigitalAmericanEngine Class Reference

```
#include <ql/pricingengines/vanilla/analyticdigitalamericanengine.hpp>
```

9.21.1 Detailed Description

Analytic pricing engine for American vanilla options with digital payoff.

Todo

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

Tests

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

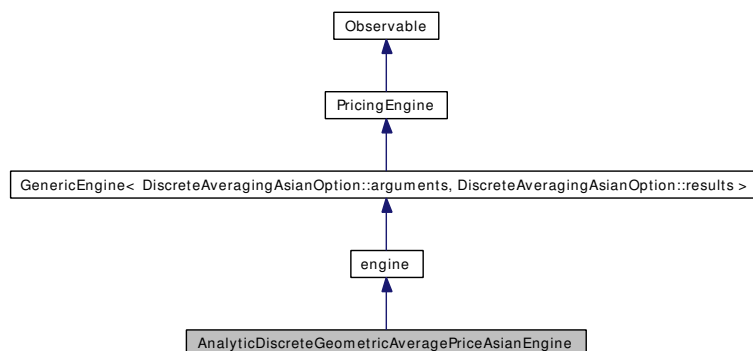
Public Member Functions

- void **calculate** () const

9.22 AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference

```
#include <ql/pricingengines/asian/analytic_discr_geom_av_price.hpp>
```

Inheritance diagram for AnalyticDiscreteGeometricAveragePriceAsianEngine:



9.22.1 Detailed Description

Pricing engine for European discrete geometric average price Asian.

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

Todo

implement correct theta, rho, and dividend-rho calculation

Tests

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

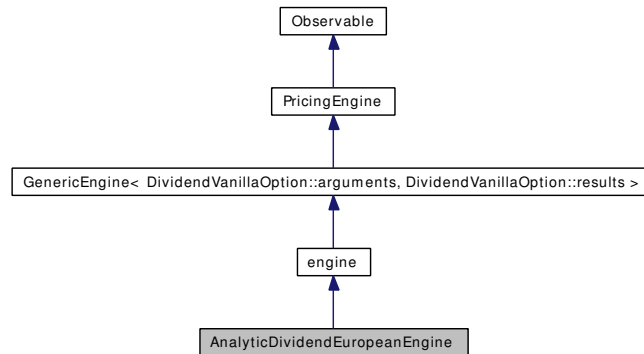
Public Member Functions

- `void calculate () const`

9.23 AnalyticDividendEuropeanEngine Class Reference

```
#include <ql/pricingengines/vanilla/analyticdividend europeanengine.hpp>
```

Inheritance diagram for AnalyticDividendEuropeanEngine:



9.23.1 Detailed Description

Analytic pricing engine for European options with discrete dividends.

Tests

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

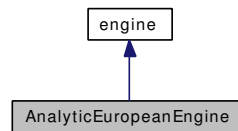
Public Member Functions

- void **calculate** () const

9.24 AnalyticEuropeanEngine Class Reference

```
#include <ql/pricingengines/vanilla/analyticeuropeanengine.hpp>
```

Inheritance diagram for AnalyticEuropeanEngine:



9.24.1 Detailed Description

Pricing engine for European vanilla options using analytical formulae.

Tests

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

Examples:

[EquityOption.cpp](#).

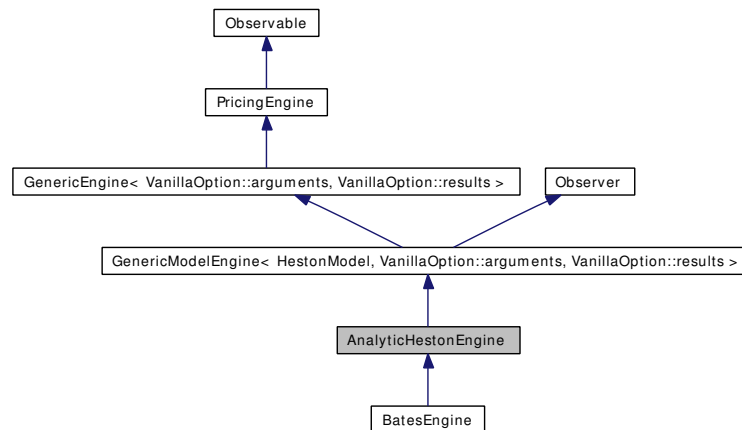
Public Member Functions

- void **calculate** () const

9.25 AnalyticHestonEngine Class Reference

```
#include <ql/pricingengines/vanilla/analytichestonengine.hpp>
```

Inheritance diagram for AnalyticHestonEngine:



9.25.1 Detailed Description

analytic Heston-model engine based on Fourier transform

References:

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

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

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

Tests

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

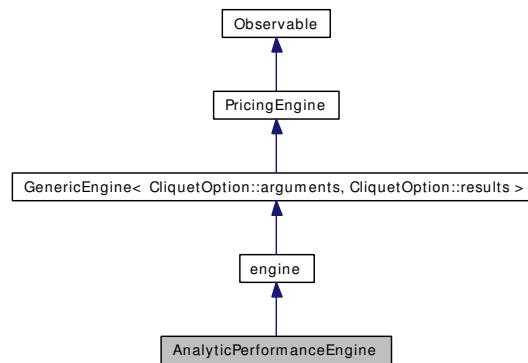
Public Member Functions

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

9.26 AnalyticPerformanceEngine Class Reference

```
#include <ql/pricingengines/cliquet/analyticperformanceengine.hpp>
```

Inheritance diagram for AnalyticPerformanceEngine:



9.26.1 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

9.27 Argentina Class Reference

```
#include <ql/time/calendars/argentina.hpp>
```

Inheritance diagram for Argentina:



9.27.1 Detailed Description

Argentinian calendars.

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

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

Public Types

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

Public Member Functions

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

9.27.2 Member Enumeration Documentation

9.27.2.1 enum Market

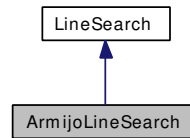
Enumerator:

Merval Buenos Aires stock exchange calendar.

9.28 ArmijoLineSearch Class Reference

```
#include <ql/math/optimization/armijo.hpp>
```

Inheritance diagram for ArmijoLineSearch:



9.28.1 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, NY, 1997)

Public Member Functions

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

9.29 Array Class Reference

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

9.29.1 Detailed Description

1-D array used in linear algebra.

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

Tests

construction of arrays is checked in a number of cases

Public Types

- typedef Real **value_type**
- 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](#) (const std::vector< Real > &)
creates the array as a copy of a given stl vector
- [Array](#) & **operator=** (const [Array](#) &)
- [Array](#) & **operator=** (const [Disposable](#)< [Array](#) > &)
- bool **operator==** (const [Array](#) &) const
- bool **operator!=** (const [Array](#) &) const

Vector algebra

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

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

Precondition:

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

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

Element access

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

Inspectors

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

Iterator access

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

Utilities

- void **swap** ([Array](#) &)

Related Functions

(Note that these are not member functions.)

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

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

9.29.2 Friends And Related Function Documentation

- 9.29.2.1 `Real DotProduct (const Array &, const Array &)` [related]
- 9.29.2.2 `const Disposable< Array > operator+ (const Array & v)` [related]
- 9.29.2.3 `const Disposable< Array > operator- (const Array & v)` [related]
- 9.29.2.4 `const Disposable< Array > operator+ (const Array &, const Array &)` [related]
- 9.29.2.5 `const Disposable< Array > operator+ (const Array &, Real)` [related]
- 9.29.2.6 `const Disposable< Array > operator+ (Real, const Array &)` [related]
- 9.29.2.7 `const Disposable< Array > operator- (const Array &, const Array &)` [related]
- 9.29.2.8 `const Disposable< Array > operator- (const Array &, Real)` [related]
- 9.29.2.9 `const Disposable< Array > operator- (Real, const Array &)` [related]
- 9.29.2.10 `const Disposable< Array > operator * (const Array &, const Array &)` [related]
- 9.29.2.11 `const Disposable< Array > operator * (const Array &, Real)` [related]
- 9.29.2.12 `const Disposable< Array > operator * (Real, const Array &)` [related]
- 9.29.2.13 `const Disposable< Array > operator/ (const Array &, const Array &)` [related]
- 9.29.2.14 `const Disposable< Array > operator/ (const Array &, Real)` [related]
- 9.29.2.15 `const Disposable< Array > operator/ (Real, const Array &)` [related]
- 9.29.2.16 `const Disposable< Array > Abs (const Array &)` [related]
- 9.29.2.17 `const Disposable< Array > Sqrt (const Array &)` [related]
- 9.29.2.18 `const Disposable< Array > Log (const Array &)` [related]
- 9.29.2.19 `const Disposable< Array > Exp (const Array &)` [related]
- 9.29.2.20 `void swap (Array &, Array &)` [related]

Examples:

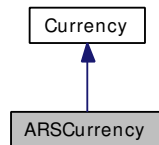
[BermudanSwaption.cpp](#).

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

9.30 ARSCurrency Class Reference

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

Inheritance diagram for ARSCurrency:



9.30.1 Detailed Description

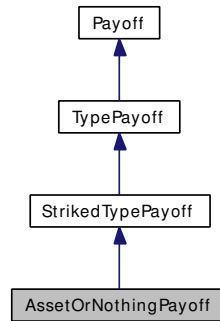
Argentinian peso.

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

9.31 AssetOrNothingPayoff Class Reference

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

Inheritance diagram for AssetOrNothingPayoff:



9.31.1 Detailed Description

Binary asset-or-nothing payoff.

Definitions of Binary path-independent payoffs used below, can be found in M. Rubinstein, E. Reiner: "Unscrambling The Binary Code", Risk, Vol.4 no.9,1991. (see: <http://www.in-the-money.com/artandpap/Binary%20Options.doc>)

Public Member Functions

- **AssetOrNothingPayoff** (Option::Type type, Real strike)

Payoff interface

- std::string **name** () const
- Real **operator()** (Real price) const
- virtual void **accept** (AcyclicVisitor &)

9.31.2 Member Function Documentation

9.31.2.1 std::string name () const [virtual]

Warning

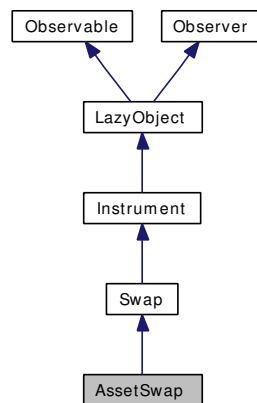
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implements [Payoff](#).

9.32 AssetSwap Class Reference

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

Inheritance diagram for AssetSwap:



9.32.1 Detailed Description

Bullet bond vs Libor swap.

for mechanics of par asset swap and market asset swap, refer to "Introduction to Asset Swap", Lehman Brothers European Fixed Income Research - January 2000, D. O’Kane

Bug

fair prices are not calculated correctly when using indexed coupons.

Public Member Functions

- **AssetSwap** (bool payFixedRate, const boost::shared_ptr< [Bond](#) > &bond, Real bondCleanPrice, const boost::shared_ptr< [IborIndex](#) > &index, Spread spread, const [Handle](#)< [YieldTermStructure](#) > &discountCurve, const [Schedule](#) &floatSchedule=[Schedule](#)(), const [DayCounter](#) &floatingDayCount=[DayCounter](#)(), bool parAssetSwap=true)
- Spread **fairSpread** () const
- Real **floatingLegBPS** () const
- Real **fairPrice** () const
- Spread **spread** () const
- Real **nominal** () const
- bool **payFixedRate** () const
- const Leg & **bondLeg** () const
- const Leg & **floatingLeg** () const
- void **setupArguments** ([PricingEngine::arguments](#) *args) const
- void **fetchResults** (const [PricingEngine::results](#) *) const

Classes

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

9.32.2 Member Function Documentation

9.32.2.1 `void fetchResults (const PricingEngine::results *) const` [virtual]

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

Reimplemented from [Instrument](#).

9.33 AssetSwap::arguments Class Reference

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

9.33.1 Detailed Description

Arguments for asset swap calculation

Public Member Functions

- void **validate** () const

Public Attributes

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

9.34 AssetSwap::results Class Reference

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

9.34.1 Detailed Description

Results from simple swap calculation

Public Member Functions

- void `reset()`

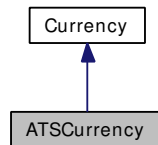
Public Attributes

- Real `floatingLegBPS`
- Spread `fairSpread`
- Real `fairPrice`

9.35 ATSCurrency Class Reference

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

Inheritance diagram for ATSCurrency:



9.35.1 Detailed Description

Austrian shilling.

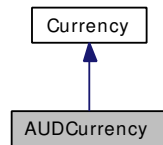
The ISO three-letter code was ATS; the numeric code was 40. It was divided in 100 groschen.

Obsoleted by the Euro since 1999.

9.36 AUDCurrency Class Reference

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

Inheritance diagram for AUDCurrency:



9.36.1 Detailed Description

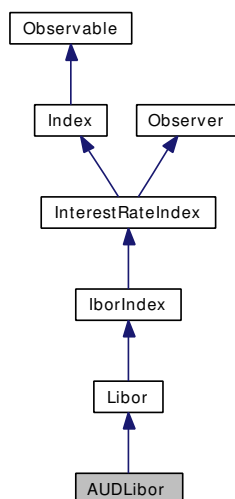
Australian dollar.

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

9.37 AUDLibor Class Reference

```
#include <ql/indexes/ibor/audlibor.hpp>
```

Inheritance diagram for AUDLibor:



9.37.1 Detailed Description

AUD LIBOR rate

Australian Dollar LIBOR fixed by BBA.

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

Public Member Functions

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

9.38 Australia Class Reference

```
#include <ql/time/calendars/australia.hpp>
```

Inheritance diagram for Australia:



9.38.1 Detailed Description

Australian calendar.

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- [Australia](#) Day, January 26th (possibly moved to Monday)
- Good Friday
- Easter Monday
- ANZAC Day. April 25th (possibly moved to Monday)
- Queen's Birthday, second Monday in June
- Bank Holiday, first Monday in August
- Labour Day, first Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

9.39 Average Struct Reference

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

9.39.1 Detailed Description

placeholder for enumerated averaging types

Public Types

- enum Type { Arithmetic, Geometric }

9.40 BackwardFlat Class Reference

```
#include <ql/math/interpolations/backwardflatinterpolation.hpp>
```

9.40.1 Detailed Description

Backward-flat interpolation factory and traits.

Public Types

- enum { **global** = 0 }

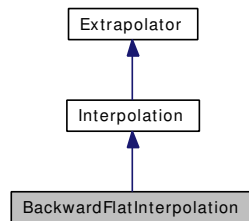
Public Member Functions

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

9.41 BackwardFlatInterpolation Class Reference

```
#include <ql/math/interpolations/backwardflatinterpolation.hpp>
```

Inheritance diagram for BackwardFlatInterpolation:



9.41.1 Detailed Description

Backward-flat interpolation between discrete points.

Public Member Functions

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

9.41.2 Constructor & Destructor Documentation

9.41.2.1 BackwardFlatInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

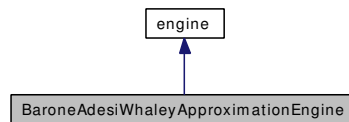
Precondition:

the *x* values must be sorted.

9.42 BaroneAdesiWhaleyApproximationEngine Class Reference

```
#include <ql/pricingengines/vanilla/baroneadesiwhaleyengine.hpp>
```

Inheritance diagram for BaroneAdesiWhaleyApproximationEngine:



9.42.1 Detailed Description

Barone-Adesi and Whaley pricing engine for American options (1987).

Tests

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

Examples:

[EquityOption.cpp](#).

Public Member Functions

- void **calculate** () const

Static Public Member Functions

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

9.43 Barrier Struct Reference

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

9.43.1 Detailed Description

Placeholder for enumerated barrier types.

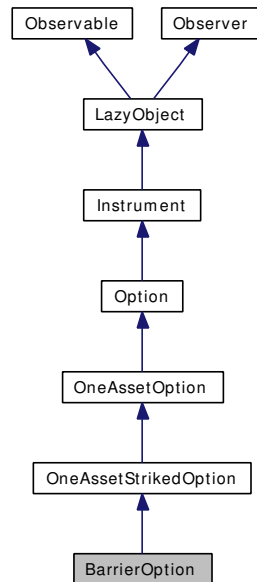
Public Types

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

9.44 BarrierOption Class Reference

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

Inheritance diagram for BarrierOption:



9.44.1 Detailed Description

Barrier option on a single asset.

The analytic pricing [engine](#) will be used if none is passed.

Examples:

[Replication.cpp](#).

Public Member Functions

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

Protected Attributes

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

Classes

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

9.45 BarrierOption::arguments Class Reference

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

9.45.1 Detailed Description

Arguments for barrier option calculation

Public Member Functions

- void **validate** () const

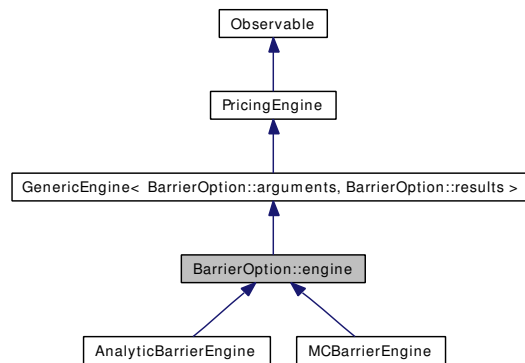
Public Attributes

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

9.46 BarrierOption::engine Class Reference

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

Inheritance diagram for BarrierOption::engine:



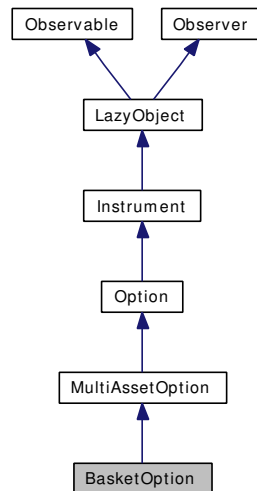
9.46.1 Detailed Description

Barrier-option engine base class

9.47 BasketOption Class Reference

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

Inheritance diagram for BasketOption:



9.47.1 Detailed Description

Basket option on a number of assets.

Todo

Replace with STL algorithms

Public Member Functions

- **BasketOption** (const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [BasketPayoff](#) > &, const boost::shared_ptr< [Exercise](#) > &, const boost::shared_ptr< [PricingEngine](#) > & **engine**=boost::shared_ptr< [PricingEngine](#) >())
- void **setupArguments** ([PricingEngine::arguments](#) *) const

Classes

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

9.48 BasketOption::arguments Class Reference

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

9.48.1 Detailed Description

Arguments for basket option calculation

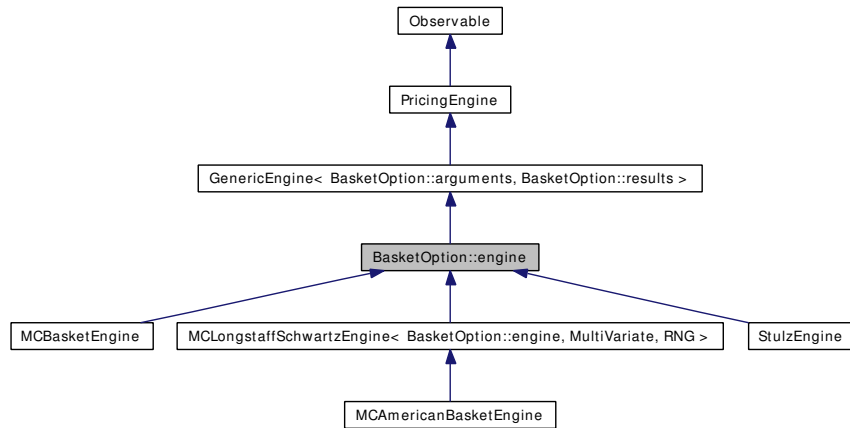
Public Member Functions

- void **validate** () const

9.49 BasketOption::engine Class Reference

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

Inheritance diagram for BasketOption::engine:



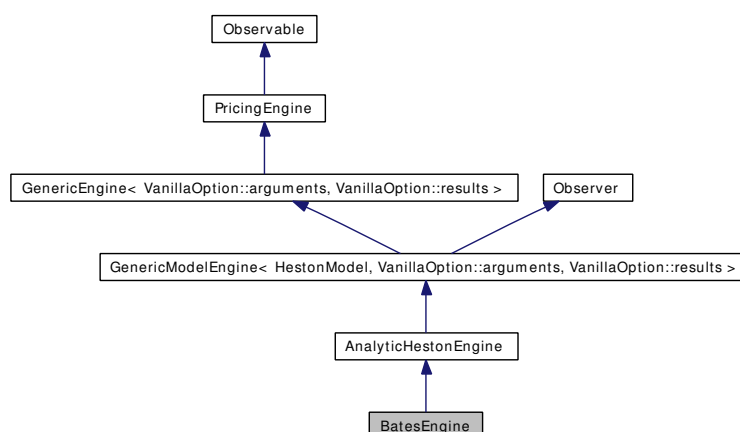
9.49.1 Detailed Description

Basket-option engine base class

9.50 BatesEngine Class Reference

```
#include <ql/pricingengines/vanilla/batesengine.hpp>
```

Inheritance diagram for BatesEngine:



9.50.1 Detailed Description

Bates model engines based on Fourier transform.

this classes price european options under the following processes

1. Jump-Diffusion with Stochastic Volatility

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

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

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

Logarithm of the jump size J is normally distributed

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

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

The jump size has an asymmetric double exponential distribution

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

2. Stochastic Volatility with Jump Diffusion and Deterministic Jump Intensity

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

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

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

References:

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

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

Tests

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

Public Member Functions

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

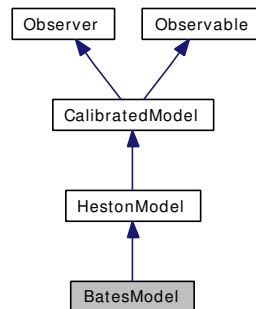
Protected Member Functions

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

9.51 BatesModel Class Reference

```
#include <ql/models/equity/batesmodel.hpp>
```

Inheritance diagram for BatesModel:



9.51.1 Detailed Description

Bates stochastic-volatility model.

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

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

Tests

calibration is tested against known values.

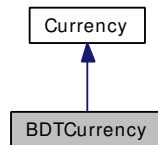
Public Member Functions

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

9.52 BDTCurrency Class Reference

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

Inheritance diagram for BDTCurrency:



9.52.1 Detailed Description

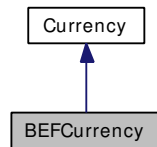
Bangladesh taka.

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

9.53 BEFCurrency Class Reference

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

Inheritance diagram for BEFCurrency:



9.53.1 Detailed Description

Belgian franc.

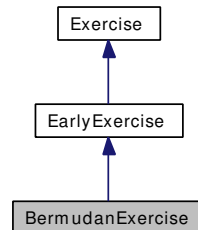
The ISO three-letter code was BEF; the numeric code was 56. It had no subdivisions.

Obsoleted by the Euro since 1999.

9.54 BermudanExercise Class Reference

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

Inheritance diagram for BermudanExercise:



9.54.1 Detailed Description

Bermudan exercise.

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

Todo

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

Examples:

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

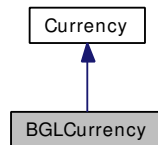
Public Member Functions

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

9.55 BGLCurrency Class Reference

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

Inheritance diagram for BGLCurrency:



9.55.1 Detailed Description

Bulgarian lev.

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

9.56 Bicubic Class Reference

```
#include <ql/math/interpolations/bicubicsplineinterpolation.hpp>
```

9.56.1 Detailed Description

bicubic-spline-interpolation factory

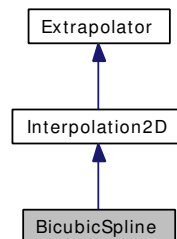
Public Member Functions

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

9.57 BicubicSpline Class Reference

```
#include <ql/math/interpolations/bicubicsplineinterpolation.hpp>
```

Inheritance diagram for BicubicSpline:



9.57.1 Detailed Description

bicubic-spline interpolation between discrete points

Todo

revise end conditions

Public Member Functions

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

9.57.2 Constructor & Destructor Documentation

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

Precondition:

the x and y values must be sorted.

9.58 Bilinear Class Reference

```
#include <ql/math/interpolations/bilinearinterpolation.hpp>
```

9.58.1 Detailed Description

bilinear-interpolation factory

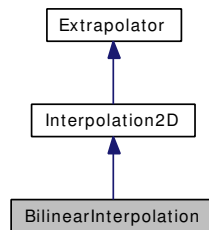
Public Member Functions

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

9.59 BilinearInterpolation Class Reference

```
#include <ql/math/interpolations/bilinearinterpolation.hpp>
```

Inheritance diagram for BilinearInterpolation:



9.59.1 Detailed Description

bilinear interpolation between discrete points

Public Member Functions

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

9.59.2 Constructor & Destructor Documentation

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

Precondition:

the x and y values must be sorted.

9.60 BinomialConvertibleEngine Class Template Reference

```
#include <ql/pricingengines/hybrid/binomialconvertibleengine.hpp>
```

9.60.1 Detailed Description

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

Binomial Tsiveriotis-Fernandes engine for convertible bonds.

Examples:

[ConvertibleBonds.cpp](#).

Public Member Functions

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

9.61 BinomialDistribution Class Reference

```
#include <ql/math/distributions/binomialdistribution.hpp>
```

9.61.1 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.

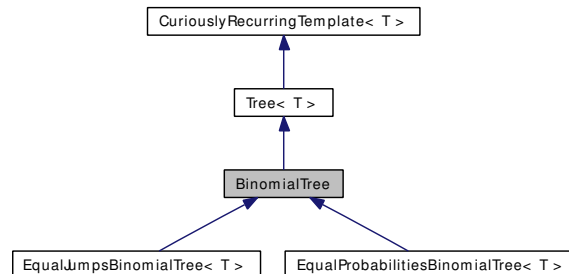
Public Member Functions

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

9.62 BinomialTree Class Template Reference

```
#include <ql/methods/lattices/binomialtree.hpp>
```

Inheritance diagram for BinomialTree:



9.62.1 Detailed Description

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

Binomial tree base class.

Public Types

- enum **Branches** { **branches** = 2 }

Public Member Functions

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

Protected Attributes

- Real **x0_**
- Real **driftPerStep_**
- Time **dt_**

9.63 BinomialVanillaEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/binomialengine.hpp>
```

9.63.1 Detailed Description

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

Pricing engine for vanilla options using binomial trees.

Tests

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

Todo

[Greeks](#) are not overly accurate. They could be improved by building a tree so that it has three points at the current time. The value would be fetched from the middle one, while the two side points would be used for estimating partial derivatives.

Examples:

[EquityOption.cpp](#).

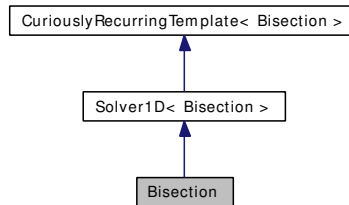
Public Member Functions

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

9.64 Bisection Class Reference

```
#include <ql/math/solvers1d/bisection.hpp>
```

Inheritance diagram for Bisection:



9.64.1 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`

9.65 BivariateCumulativeNormalDistributionDr78 Class Reference

```
#include <ql/math/distributions/bivariatenormaldistribution.hpp>
```

9.65.1 Detailed Description

Cumulative bivariate normal distribution function.

Drezner (1978) algorithm, six decimal places accuracy.

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

Todo

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

Tests

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

Public Member Functions

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

9.66 BivariateCumulativeNormalDistributionWe04DP Class Reference

```
#include <ql/math/distributions/bivariatenormaldistribution.hpp>
```

9.66.1 Detailed Description

Cumulative bivariate normal distribution function (West 2004).

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

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

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

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

Tests

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

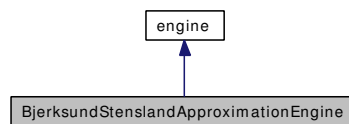
Public Member Functions

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

9.67 BjerksundStenslandApproximationEngine Class Reference

```
#include <ql/pricingengines/vanilla/bjerksundstenslandengine.hpp>
```

Inheritance diagram for BjerksundStenslandApproximationEngine:



9.67.1 Detailed Description

Bjerksund and Stensland pricing engine for American options (1993).

Tests

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

Examples:

[EquityOption.cpp](#).

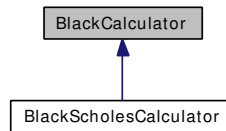
Public Member Functions

- void `calculate ()` const

9.68 BlackCalculator Class Reference

```
#include <ql/pricingengines/blackcalculator.hpp>
```

Inheritance diagram for BlackCalculator:



9.68.1 Detailed Description

Black 1976 calculator class.

Bug

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

Examples:

[DiscreteHedging.cpp](#).

Public Member Functions

- **BlackCalculator**(const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, Real forward, Real stdDev, Real discount=1.0)
- Real **value** () const
- Real [deltaForward](#) () const
- virtual Real [delta](#) (Real spot) const
- Real [elasticityForward](#) () const
- virtual Real [elasticity](#) (Real spot) const
- Real [gammaForward](#) () const
- virtual Real [gamma](#) (Real spot) const
- virtual Real [theta](#) (Real spot, Time maturity) const
- virtual 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

Protected Attributes

- Real **strike_**
- Real **forward_**
- Real **stdDev_**
- Real **discount_**
- Real **variance_**
- Real **D1_**
- Real **D2_**
- Real **alpha_**
- Real **beta_**
- Real **DalphaDd1_**
- Real **DbetaDd2_**
- Real **n_d1_**
- Real **cum_d1_**
- Real **n_d2_**
- Real **cum_d2_**
- Real **X_**
- Real **DXDs_**
- Real **DXDstrike_**

Friends

- class **Calculator**

9.68.2 Member Function Documentation

9.68.2.1 Real deltaForward () const

Sensitivity to change in the underlying forward price.

9.68.2.2 virtual Real delta (Real *spot*) const [virtual]

Sensitivity to change in the underlying spot price.

9.68.2.3 Real elasticityForward () const

Sensitivity in percent to a percent change in the underlying forward price.

9.68.2.4 virtual Real elasticity (Real *spot*) const [virtual]

Sensitivity in percent to a percent change in the underlying spot price.

9.68.2.5 Real gammaForward () const

Second order derivative with respect to change in the underlying forward price.

9.68.2.6 virtual Real gamma (Real *spot*) const [virtual]

Second order derivative with respect to change in the underlying spot price.

9.68.2.7 virtual Real theta (Real *spot*, Time *maturity*) const [virtual]

Sensitivity to time to maturity.

9.68.2.8 virtual Real thetaPerDay (Real *spot*, Time *maturity*) const [virtual]

Sensitivity to time to maturity per day, assuming 365 day per year.

9.68.2.9 Real vega (Time *maturity*) const

Sensitivity to volatility.

9.68.2.10 Real rho (Time *maturity*) const

Sensitivity to discounting rate.

9.68.2.11 Real dividendRho (Time *maturity*) const

Sensitivity to dividend/growth rate.

9.68.2.12 Real itmCashProbability () const

Probability of being in the money in the bond martingale measure, i.e. $N(d_2)$. It is a risk-neutral probability, not the real world one.

9.68.2.13 Real itmAssetProbability () const

Probability of being in the money in the asset martingale measure, i.e. $N(d_1)$. It is a risk-neutral probability, not the real world one.

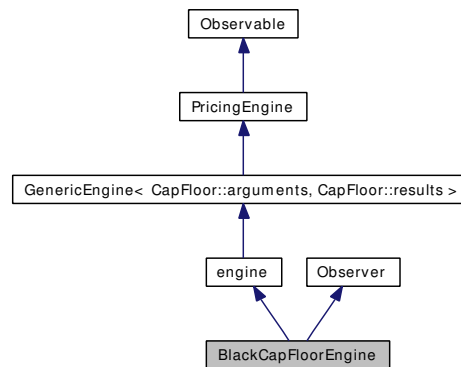
9.68.2.14 Real strikeSensitivity () const

Sensitivity to strike.

9.69 BlackCapFloorEngine Class Reference

```
#include <ql/pricingengines/capfloor/blackcapfloorengine.hpp>
```

Inheritance diagram for BlackCapFloorEngine:



9.69.1 Detailed Description

Black-formula cap/floor engine.

Public Member Functions

- **BlackCapFloorEngine** (const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dc=[Actual365Fixed](#)())
- **BlackCapFloorEngine** (const [Handle](#)< [CapletVolatilityStructure](#) > &v)
- void **calculate** () const
- void **update** ()

9.69.2 Member Function Documentation

9.69.2.1 void update () [virtual]

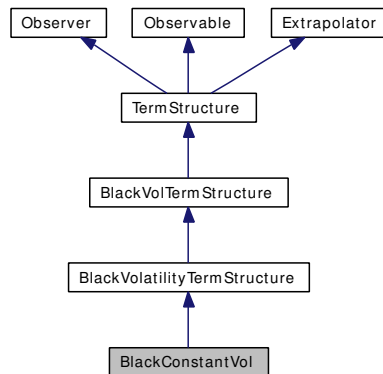
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.70 BlackConstantVol Class Reference

```
#include <ql/termstructures/volatilities/blackconstantvol.hpp>
```

Inheritance diagram for BlackConstantVol:



9.70.1 Detailed Description

Constant Black volatility, no time-strike dependence.

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

Examples:

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

Public Member Functions

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

BlackVolTermStructure interface

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

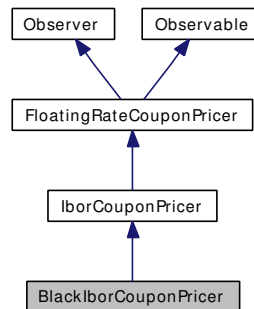
Protected Member Functions

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

9.71 BlackIborCouponPricer Class Reference

```
#include <ql/cashflows/couponpricer.hpp>
```

Inheritance diagram for BlackIborCouponPricer:



9.71.1 Detailed Description

Black-formula pricer for capped/floored Ibor coupons.

Public Member Functions

- **BlackIborCouponPricer** (const [Handle](#)< [CapletVolatilityStructure](#) > &capletVol=[Handle](#)< [CapletVolatilityStructure](#) >())
- void **initialize** (const [FloatingRateCoupon](#) &coupon)
- Real **swapletPrice** () const
- Rate **swapletRate** () const
- Real **capletPrice** (Rate effectiveCap) const
- Rate **capletRate** (Rate effectiveCap) const
- Real **floorletPrice** (Rate effectiveFloor) const
- Rate **floorletRate** (Rate effectiveFloor) const

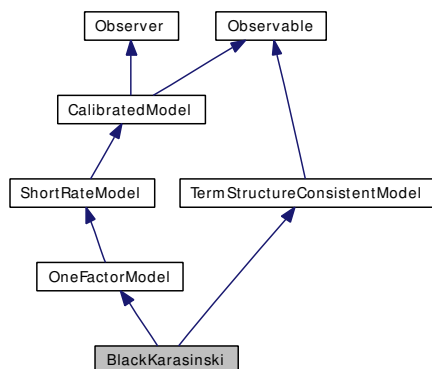
Protected Member Functions

- Real **optionletPrice** (Option::Type optionType, Real effStrike) const

9.72 BlackKarasinski Class Reference

```
#include <ql/models/shortrate/onefactormodels/blackkarasinski.hpp>
```

Inheritance diagram for BlackKarasinski:



9.72.1 Detailed Description

Standard Black-Karasinski model class.

This class implements the standard Black-Karasinski model defined by

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

where *alpha* and *sigma* are constants.

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

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

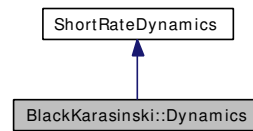
Classes

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

9.73 BlackKarasinski::Dynamics Class Reference

```
#include <ql/models/shortrate/onefactormodels/blackkarasinski.hpp>
```

Inheritance diagram for BlackKarasinski::Dynamics:



9.73.1 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.

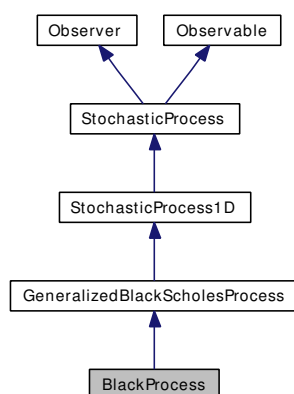
Public Member Functions

- **Dynamics** (const [Parameter](#) &fitting, Real alpha, Real sigma)
- Real [variable](#) (Time t, Rate r) const
Compute state variable from short rate.
- Real [shortRate](#) (Time t, Real x) const
Compute short rate from state variable.

9.74 BlackProcess Class Reference

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

Inheritance diagram for BlackProcess:



9.74.1 Detailed Description

Black (1976) stochastic process.

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

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

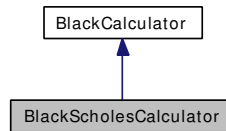
Public Member Functions

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

9.75 BlackScholesCalculator Class Reference

```
#include <ql/pricingengines/blackscholescalculator.hpp>
```

Inheritance diagram for BlackScholesCalculator:



9.75.1 Detailed Description

Black-Scholes 1973 calculator class.

Public Member Functions

- **BlackScholesCalculator** (const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, Real spot, DiscountFactor growth, Real stdDev, DiscountFactor discount)
- Real [delta](#) () const
- Real [elasticity](#) () const
- Real [gamma](#) () const
- Real [theta](#) (Time maturity) const
- Real [thetaPerDay](#) (Time maturity) const

Protected Attributes

- Real **spot_**
- DiscountFactor **growth_**

9.75.2 Member Function Documentation

9.75.2.1 Real delta () const

Sensitivity to change in the underlying spot price.

9.75.2.2 Real elasticity () const

Sensitivity in percent to a percent change in the underlying spot price.

9.75.2.3 Real gamma () const

Second order derivative with respect to change in the underlying spot price.

9.75.2.4 Real theta (Time *maturity*) const

Sensitivity to time to maturity.

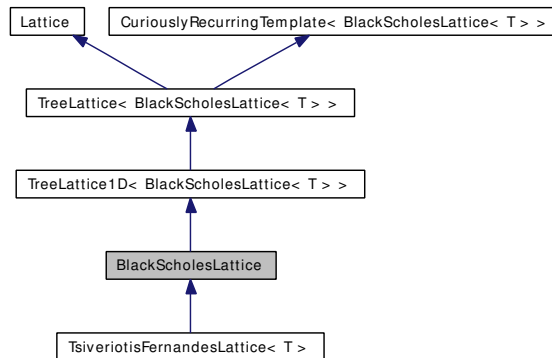
9.75.2.5 Real thetaPerDay (Time *maturity*) const

Sensitivity to time to maturity per day (assuming 365 day in a year).

9.76 BlackScholesLattice Class Template Reference

```
#include <ql/methods/lattices/bsmlattice.hpp>
```

Inheritance diagram for BlackScholesLattice:



9.76.1 Detailed Description

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

Simple binomial lattice approximating the Black-Scholes model.

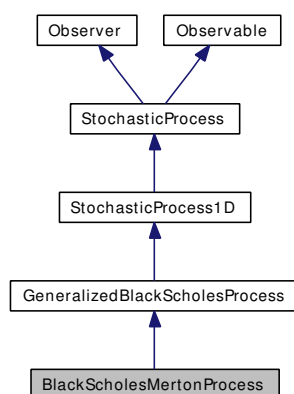
Public Member Functions

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

9.77 BlackScholesMertonProcess Class Reference

#include <ql/processes/blackscholesprocess.hpp>

Inheritance diagram for BlackScholesMertonProcess:



9.77.1 Detailed Description

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

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

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

Examples:

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

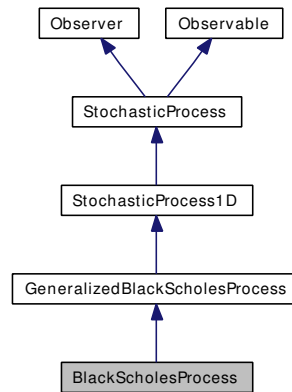
Public Member Functions

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

9.78 BlackScholesProcess Class Reference

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

Inheritance diagram for BlackScholesProcess:



9.78.1 Detailed Description

Black-Scholes (1973) stochastic process.

This class describes the stochastic process for a stock given by

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

Examples:

[Replication.cpp](#).

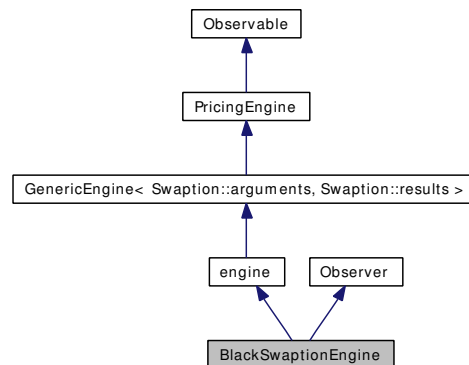
Public Member Functions

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

9.79 BlackSwaptionEngine Class Reference

```
#include <ql/pricingengines/swaption/blackswaptionengine.hpp>
```

Inheritance diagram for BlackSwaptionEngine:



9.79.1 Detailed Description

Black-formula swaption engine.

Warning

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

Public Member Functions

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

9.79.2 Member Function Documentation

9.79.2.1 void update () [virtual]

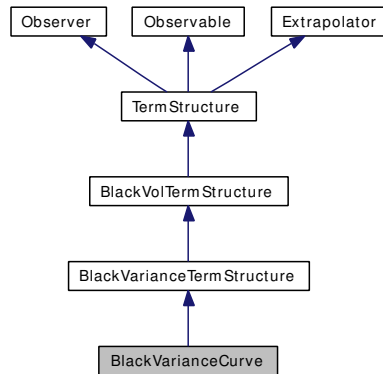
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.80 BlackVarianceCurve Class Reference

```
#include <ql/termstructures/volatilities/blackvariancecurve.hpp>
```

Inheritance diagram for BlackVarianceCurve:



9.80.1 Detailed Description

Black volatility curve modelled as variance curve.

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

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

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

Todo

check time extrapolation

Public Member Functions

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

BlackVolTermStructure interface

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

the maximum strike for which the term structure can return vols

Modifiers

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

Visitability

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

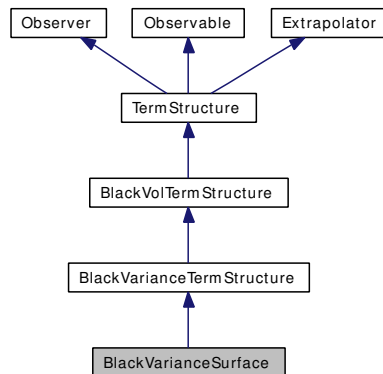
Protected Member Functions

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

9.81 BlackVarianceSurface Class Reference

```
#include <ql/termstructures/volatilities/blackvariancesurface.hpp>
```

Inheritance diagram for BlackVarianceSurface:



9.81.1 Detailed Description

Black volatility surface modelled as variance surface.

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

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

Todo

check time extrapolation

Public Types

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

Public Member Functions

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

BlackVolTermStructure interface

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

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

Modifiers

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

Visitability

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

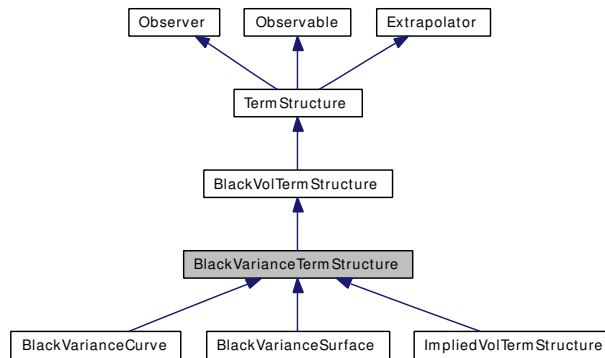
Protected Member Functions

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

9.82 BlackVarianceTermStructure Class Reference

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

Inheritance diagram for BlackVarianceTermStructure:



9.82.1 Detailed Description

Black variance term structure.

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

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

See the TermStructure documentation for issues regarding constructors.

- **BlackVarianceTermStructure** (const DayCounter &dc=Actual365Fixed())
default constructor
- **BlackVarianceTermStructure** (const Date &referenceDate, const Calendar &cal=Calendar(), const DayCounter &dc=Actual365Fixed())
initialize with a fixed reference date
- **BlackVarianceTermStructure** (Natural settlementDays, const Calendar &, const DayCounter &dc=Actual365Fixed())
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

9.82.2 Constructor & Destructor Documentation

9.82.2.1 BlackVarianceTermStructure (const DayCounter & *dc* = Actual365Fixed())

default constructor

Warning

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

9.82.3 Member Function Documentation

9.82.3.1 Volatility blackVolImpl (Time *maturity*, Real *strike*) const [protected, virtual]

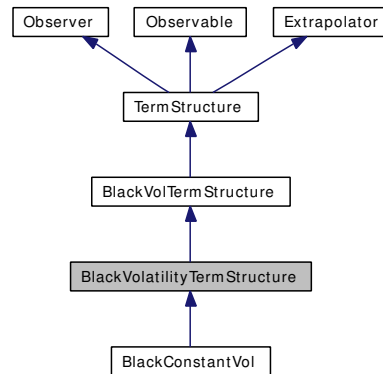
Returns the volatility for the given strike and date calculating it from the variance.

Implements [BlackVolTermStructure](#).

9.83 BlackVolatilityTermStructure Class Reference

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

Inheritance diagram for BlackVolatilityTermStructure:



9.83.1 Detailed Description

Black-volatility term structure.

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

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

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

- [BlackVolatilityTermStructure](#) (const [DayCounter](#) &dc=[Actual365Fixed](#)())
default constructor
- [BlackVolatilityTermStructure](#) (const [Date](#) &referenceDate, const [Calendar](#) &cal=[Calendar](#)(), const [DayCounter](#) &dc=[Actual365Fixed](#)())
initialize with a fixed reference date
- [BlackVolatilityTermStructure](#) (Natural settlementDays, const [Calendar](#) &, const [DayCounter](#) &dc=[Actual365Fixed](#)())
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

9.83.2 Constructor & Destructor Documentation

9.83.2.1 BlackVolatilityTermStructure (const DayCounter & *dc* = Actual365Fixed())

default constructor

Warning

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

9.83.3 Member Function Documentation

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

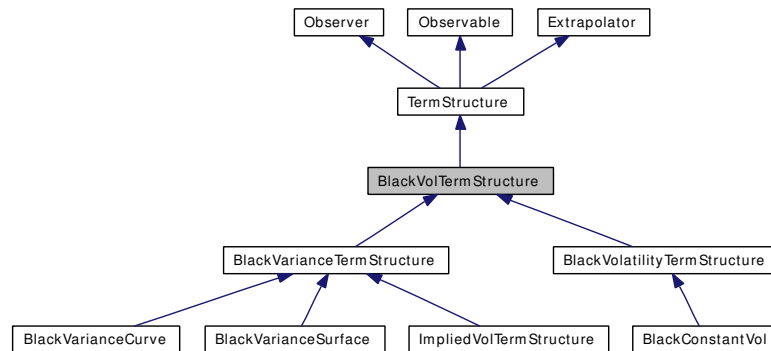
Returns the variance for the given strike and date calculating it from the volatility.

Implements [BlackVolTermStructure](#).

9.84 BlackVolTermStructure Class Reference

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

Inheritance diagram for BlackVolTermStructure:



9.84.1 Detailed Description

Black-volatility term structure.

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

Volatilities are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

See the *TermStructure* documentation for issues regarding constructors.

- **BlackVolTermStructure** (const [DayCounter](#) &dc=[Actual365Fixed](#)())
default constructor
- **BlackVolTermStructure** (const [Date](#) &referenceDate, const [Calendar](#) &cal=[Calendar](#)()),
const [DayCounter](#) &dc=[Actual365Fixed](#)())
initialize with a fixed reference date
- **BlackVolTermStructure** (Natural settlementDays, const [Calendar](#) &, const [DayCounter](#) &dc=[Actual365Fixed](#)())
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 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.

9.84.2 Constructor & Destructor Documentation

9.84.2.1 BlackVolTermStructure (const DayCounter & *dc* = Actual365Fixed())

default constructor

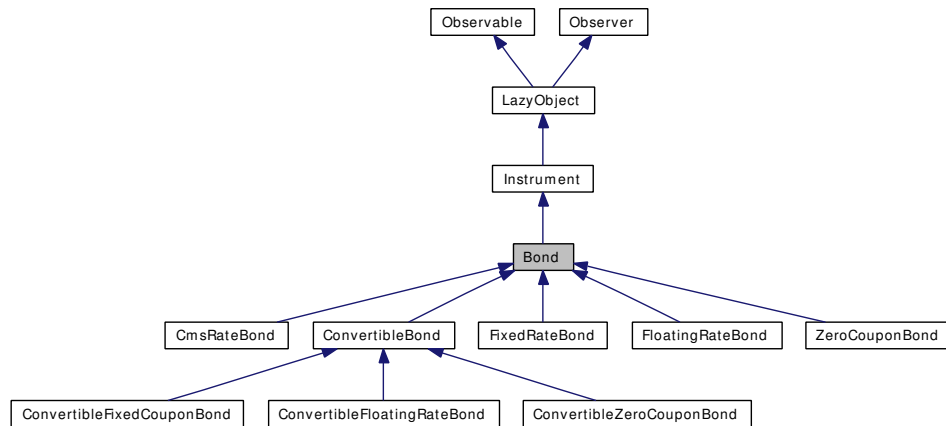
Warning

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

9.85 Bond Class Reference

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

Inheritance diagram for Bond:



9.85.1 Detailed Description

Base bond class.

Derived classes must fill the uninitialized data members.

Warning

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

Tests

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

Public Member Functions

Inspectors

- **Date** **settlementDate** (const **Date** &d=**Date**()) const
- **Date** **issueDate** () const
- **Date** **maturityDate** () const
- **Date** **interestAccrualDate** () const
- const Leg & **cashflows** () const
- const boost::shared_ptr< **CashFlow** > & **redemption** () const
- const **Calendar** & **calendar** () const
- **BusinessDayConvention** **paymentConvention** () const
- Real **faceAmount** () const
- const **DayCounter** & **dayCounter** () const
- **Frequency** **frequency** () const
- **Handle**< **YieldTermStructure** > **discountCurve** () const

Calculations

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

Protected Member Functions

- **Bond** (Natural settlementDays, Real faceAmount, const [Calendar](#) &calendar, const [DayCounter](#) &paymentDayCounter, [BusinessDayConvention](#) paymentConvention, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())
- void [performCalculations](#) () const
- void [setupArguments](#) (PricingEngine::arguments *) const

Protected Attributes

- Natural [settlementDays_](#)
- Real [faceAmount_](#)
- [Calendar](#) [calendar_](#)
- [DayCounter](#) [paymentDayCounter_](#)
- [BusinessDayConvention](#) [paymentConvention_](#)
- [Handle](#)< [YieldTermStructure](#) > [discountCurve_](#)
- [Date](#) [issueDate_](#)
- [Date](#) [datedDate_](#)
- [Date](#) [maturityDate_](#)
- [Frequency](#) [frequency_](#)
- Leg [cashflows_](#)

9.85.2 Member Function Documentation

9.85.2.1 `const Leg & cashflows () const`

Warning

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

9.85.2.2 `Real cleanPrice () const`

theoretical clean price

The default bond settlement is used for calculation.

Warning

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

9.85.2.3 `Real dirtyPrice () const`

theoretical dirty price

The default bond settlement is used for calculation.

Warning

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

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

theoretical bond yield

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

9.85.2.5 `Real cleanPrice (Rate yield, Compounding compounding, Date settlementDate = Date()) const`

clean price given a yield and settlement date

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

9.85.2.6 `Real dirtyPrice (Rate yield, Compounding compounding, Date settlementDate = Date()) const`

dirty price given a yield and settlement date

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

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

yield given a (clean) price and settlement date

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

9.85.2.8 virtual Real accruedAmount (Date *d* = Date()) const [virtual]

accrued amount at a given date

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

9.85.2.9 void performCalculations () const [protected, virtual]

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

Reimplemented from [Instrument](#).

Reimplemented in [ConvertibleBond](#).

9.86 BoundaryCondition Class Template Reference

```
#include <ql/methods/finitedifferences/boundarycondition.hpp>
```

9.86.1 Detailed Description

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

Abstract boundary condition class for finite difference problems.

Public Types

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

Public Member Functions

- virtual void [applyBeforeApplying](#) (operator_type &) const=0
- virtual void [applyAfterApplying](#) (array_type &) const =0
- virtual void [applyBeforeSolving](#) (operator_type &, array_type &rhs) const=0
- virtual void [applyAfterSolving](#) (array_type &) const =0
- virtual void [setTime](#) (Time t)=0

9.86.2 Member Enumeration Documentation

9.86.2.1 enum Side

[Todo](#)

Generalize for n-dimensional conditions

9.86.3 Member Function Documentation

9.86.3.1 virtual void [applyBeforeApplying](#) (operator_type &) const [pure virtual]

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

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

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

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

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

9.86.3.3 virtual void applyBeforeSolving (operator_type &, array_type & rhs) const [pure virtual]

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

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

9.86.3.4 virtual void applyAfterSolving (array_type &) const [pure virtual]

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

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

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

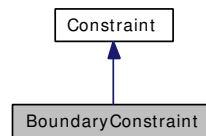
This method sets the current time for time-dependent boundary conditions.

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

9.87 BoundaryConstraint Class Reference

```
#include <ql/math/optimization/constraint.hpp>
```

Inheritance diagram for BoundaryConstraint:



9.87.1 Detailed Description

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

Public Member Functions

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

9.88 BoxMullerGaussianRng Class Template Reference

```
#include <ql/math/randomnumbers/boxmullergaussianrng.hpp>
```

9.88.1 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

9.89 Brazil Class Reference

```
#include <ql/time/calendars/brazil.hpp>
```

Inheritance diagram for Brazil:



9.89.1 Detailed Description

Brazilian calendar.

Banking holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Tiradentes's Day, April 21th
- Labour Day, May 1st
- Independence Day, September 21th
- Nossa Sra. Aparecida Day, October 12th
- Dead Day, October 2nd
- Republic Day, November 15th
- Christmas, December 25th
- Passion of Christ
- Carnival
- Corpus Christi

Tests

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

Public Types

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

Brazilian calendars.

Public Member Functions

- **Brazil** ([Market](#) market=[Settlement](#))

9.89.2 Member Enumeration Documentation

9.89.2.1 enum Market

Brazilian calendars.

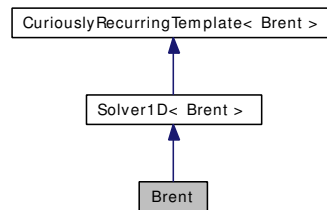
Enumerator:

Settlement generic settlement calendar

9.90 Brent Class Reference

```
#include <ql/math/solvers1d/brent.hpp>
```

Inheritance diagram for Brent:



9.90.1 Detailed Description

Brent 1-D solver

Tests

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

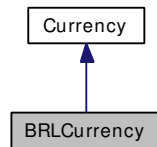
Public Member Functions

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

9.91 BRLCurrency Class Reference

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

Inheritance diagram for BRLCurrency:



9.91.1 Detailed Description

Brazilian real.

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

9.92 BrownianBridge Class Reference

```
#include <ql/methods/montecarlo/brownianbridge.hpp>
```

9.92.1 Detailed Description

Builds Wiener process paths using Gaussian variates.

This class generates normalized (i.e., unit-variance) paths as sequences of variations. In order to obtain the actual path of the underlying, the returned variations must be multiplied by the integrated variance (including time) over the corresponding time step.

Brownian-bridge constructor

- `template<class RandomAccessIterator1, class RandomAccessIterator2>`
`void transform (RandomAccessIterator1 begin, RandomAccessIterator1 end, RandomAccessIterator2 output) const`

Public Member Functions

- [BrownianBridge](#) (Size steps)
unit-time path
- [BrownianBridge](#) (const std::vector< Time > ×)
generic times
- [BrownianBridge](#) (const [TimeGrid](#) &timeGrid)
generic times

inspectors

- Size `size ()` const
- const std::vector< Time > & `times ()` const

9.92.2 Constructor & Destructor Documentation

9.92.2.1 BrownianBridge (const std::vector< Time > & times)

generic times

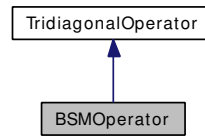
Note:

the starting time of the path is assumed to be 0 and must not be included

9.93 BSMOperator Class Reference

```
#include <ql/methods/finitedifferences/bsmoperator.hpp>
```

Inheritance diagram for BSMOperator:



9.93.1 Detailed Description

Black-Scholes-Merton differential operator.

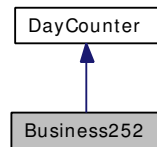
Public Member Functions

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

9.94 Business252 Class Reference

```
#include <ql/time/daycounters/business252.hpp>
```

Inheritance diagram for Business252:



9.94.1 Detailed Description

Business/252 day count convention.

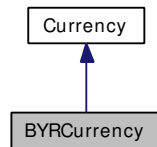
Public Member Functions

- **Business252** ([Calendar](#) c)

9.95 BYRCurrency Class Reference

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

Inheritance diagram for BYRCurrency:



9.95.1 Detailed Description

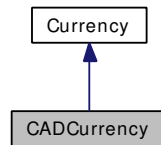
Belarussian ruble.

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

9.96 CADCurrency Class Reference

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

Inheritance diagram for CADCurrency:



9.96.1 Detailed Description

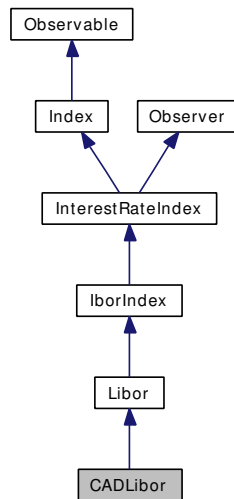
Canadian dollar.

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

9.97 CADLibor Class Reference

```
#include <ql/indexes/ibor/cadlibor.hpp>
```

Inheritance diagram for CADLibor:



9.97.1 Detailed Description

CAD LIBOR rate

Canadian Dollar LIBOR fixed by BBA.

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

Warning

This is the rate fixed in London by BBA. Use CDOR if you're interested in the Canadian fixing by IDA.

Public Member Functions

- **CADLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), Natural settlementDays=2)

9.98 Calendar Class Reference

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

Inheritance diagram for Calendar:



9.98.1 Detailed Description

calendar class

This class provides methods for determining whether a date is a business day or a holiday for a given market, and for incrementing/decrementing a date of a given number of business days.

The Bridge pattern is used to provide the base behavior of the calendar, namely, to determine whether a date is a business day.

A calendar should be defined for specific exchange holiday schedule or for general country holiday schedule. Legacy city holiday schedule calendars will be moved to the exchange/country convention.

Tests

the methods for adding and removing holidays are tested by inspecting the calendar before and after their invocation.

Examples:

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

Calendar interface

- bool [empty](#) () const
Returns whether or not the calendar is initialized.
- std::string [name](#) () const
Returns the name of the calendar.
- bool [isBusinessDay](#) (const [Date](#) &d) const
- bool [isHoliday](#) (const [Date](#) &d) const
- bool [isWeekend](#) ([Weekday](#) w) const
- bool [isEndOfMonth](#) (const [Date](#) &d) const
- [Date](#) [endOfMonth](#) (const [Date](#) &d) const
last business day of the month to which the given date belongs
- void [addHoliday](#) (const [Date](#) &)
- void [removeHoliday](#) (const [Date](#) &)
- [Date](#) [adjust](#) (const [Date](#) &, [BusinessDayConvention](#) convention=Following) const
- [Date](#) [advance](#) (const [Date](#) &, Integer n, [TimeUnit](#) unit, [BusinessDayConvention](#) convention=Following, bool endOfMonth=false) const

- [Date advance](#) (const [Date](#) &date, const [Period](#) &period, [BusinessDayConvention](#) convention=Following, bool endOfMonth=false) const
- BigInteger [businessDaysBetween](#) (const [Date](#) &from, const [Date](#) &to, bool includeFirst=true, bool includeLast=false) const
- static std::vector< [Date](#) > [holidayList](#) (const [Calendar](#) &calendar, const [Date](#) &from, const [Date](#) &to, bool includeWeekEnds=false)

Returns the holidays between two dates.

Public Member Functions

- [Calendar](#) ()

Protected Attributes

- boost::shared_ptr< [Impl](#) > [impl_](#)

Related Functions

(Note that these are not member functions.)

- bool [operator==](#) (const [Calendar](#) &, const [Calendar](#) &)
- bool [operator!=](#) (const [Calendar](#) &, const [Calendar](#) &)
- std::ostream & [operator<<](#) (std::ostream &, const [Calendar](#) &)

Classes

- class [Impl](#)
abstract base class for calendar implementations
- class [OrthodoxImpl](#)
partial calendar implementation
- class [WesternImpl](#)
partial calendar implementation

9.98.2 Constructor & Destructor Documentation

9.98.2.1 [Calendar](#) ()

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

9.98.3 Member Function Documentation

9.98.3.1 `std::string name () const`

Returns the name of the calendar.

Warning

This method is used for output and comparison between calendars. It is **not** meant to be used for writing switch-on-type code.

9.98.3.2 `bool isBusinessDay (const Date & d) const`

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

9.98.3.3 `bool isHoliday (const Date & d) const`

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

9.98.3.4 `bool isWeekend (Weekday w) const`

Returns true iff the weekday is part of the weekend for the given market.

9.98.3.5 `bool isEndOfMonth (const Date & d) const`

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

9.98.3.6 `void addHoliday (const Date &)`

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

9.98.3.7 `void removeHoliday (const Date &)`

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

9.98.3.8 `Date adjust (const Date &, BusinessDayConvention convention = Following) const`

Adjusts a non-business day to the appropriate near business day with respect to the given convention.

Examples:

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

9.98.3.9 Date advance (const Date &, Integer *n*, TimeUnit *unit*, BusinessDayConvention *convention* = Following, bool *endOfMonth* = false) const

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

Note:

The input date is not modified.

Examples:

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

9.98.3.10 Date advance (const Date & *date*, const Period & *period*, BusinessDayConvention *convention* = Following, bool *endOfMonth* = false) const

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

Note:

The input date is not modified.

9.98.3.11 BigInteger businessDaysBetween (const Date & *from*, const Date & *to*, bool *includeFirst* = true, bool *includeLast* = false) const

Calculates the number of business days between two given dates and returns the result.

9.98.4 Friends And Related Function Documentation

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

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

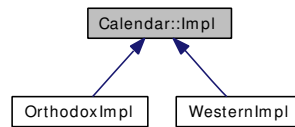
9.98.4.2 bool operator!= (const Calendar &, const Calendar &) [related]

9.98.4.3 std::ostream & operator<< (std::ostream &, const Calendar &) [related]

9.99 Calendar::Impl Class Reference

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

Inheritance diagram for Calendar::Impl:



9.99.1 Detailed Description

abstract base class for calendar implementations

Public Member Functions

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

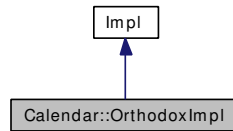
Public Attributes

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

9.100 Calendar::OrthodoxImpl Class Reference

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

Inheritance diagram for Calendar::OrthodoxImpl:



9.100.1 Detailed Description

partial calendar implementation

This class provides the means of determining the Orthodox Easter Monday for a given year, as well as specifying Saturdays and Sundays as weekend days.

Protected Member Functions

- `bool isWeekend` ([Weekday](#)) `const`

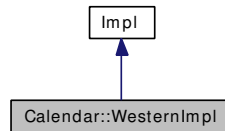
Static Protected Member Functions

- `static Day easterMonday` (Year)
expressed relative to first day of year

9.101 Calendar::WesternImpl Class Reference

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

Inheritance diagram for Calendar::WesternImpl:



9.101.1 Detailed Description

partial calendar implementation

This class provides the means of determining the Easter Monday for a given year, as well as specifying Saturdays and Sundays as weekend days.

Protected Member Functions

- `bool isWeekend (Weekday) const`

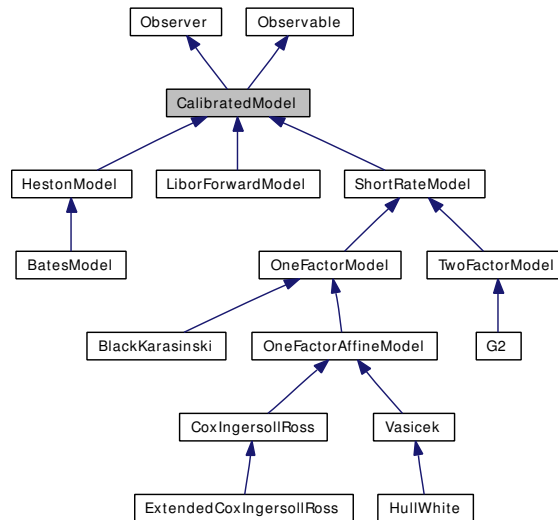
Static Protected Member Functions

- `static Day easterMonday (Year)`
expressed relative to first day of year

9.102 CalibratedModel Class Reference

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

Inheritance diagram for CalibratedModel:



9.102.1 Detailed Description

Calibrated model class.

Public Member Functions

- **CalibratedModel** (Size nArguments)
- void **update** ()
- void **calibrate** (const std::vector< boost::shared_ptr< CalibrationHelper > > &, OptimizationMethod &method, const EndCriteria &endCriteria, const Constraint &constraint=Constraint(), const std::vector< Real > &weights=std::vector< Real >())

Calibrate to a set of market instruments (caps/swaptions).

- const boost::shared_ptr< Constraint > & **constraint** () const
- Disposable< Array > **params** () const

Returns array of arguments on which calibration is done.

- virtual void **setParams** (const Array ¶ms)

Protected Member Functions

- virtual void **generateArguments** ()

Protected Attributes

- `std::vector< Parameter > arguments_`
- `boost::shared_ptr< Constraint > constraint_`

Friends

- class `CalibrationFunction`

9.102.2 Member Function Documentation

9.102.2.1 `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](#).

9.102.2.2 `void calibrate(const std::vector< boost::shared_ptr< CalibrationHelper > > &, OptimizationMethod & method, const EndCriteria & endCriteria, const Constraint & constraint = Constraint(), const std::vector< Real > & weights = std::vector< Real >())`

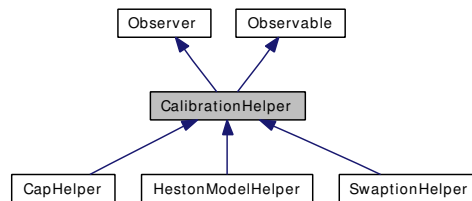
Calibrate to a set of market instruments (caps/swaptions).

An additional constraint can be passed which must be satisfied in addition to the constraints of the model.

9.103 CalibrationHelper Class Reference

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

Inheritance diagram for CalibrationHelper:



9.103.1 Detailed Description

liquid market instrument used during calibration

Public Member Functions

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

Protected Attributes

- Real [marketValue_](#)
- [Handle](#)< [Quote](#) > [volatility_](#)
- [Handle](#)< [YieldTermStructure](#) > [termStructure_](#)
- boost::shared_ptr< [PricingEngine](#) > [engine_](#)

9.103.2 Member Function Documentation

9.103.2.1 void update () [virtual]

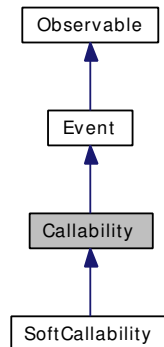
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.104 Callability Class Reference

```
#include <ql/instruments/callabilityschedule.hpp>
```

Inheritance diagram for Callability:



9.104.1 Detailed Description

instrument callability

Examples:

[ConvertibleBonds.cpp](#).

Public Types

- enum `Type` { `Call`, `Put` }
type of the callability

Public Member Functions

- `Callability` (const `Price` &price, `Type` type, const `Date` &date)
- const `Price` & `price` () const
- `Type` `type` () const
- `Date` `date` () const
returns the date at which the event occurs

Classes

- class `Price`
amount to be paid upon callability

9.105 Callability::Price Class Reference

```
#include <ql/instruments/callabilityschedule.hpp>
```

9.105.1 Detailed Description

amount to be paid upon callability

Examples:

[ConvertibleBonds.cpp](#).

Public Types

- enum **Type** { **Dirty**, **Clean** }

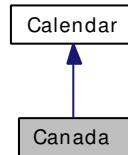
Public Member Functions

- **Price** (Real amount, Type type)
- Real **amount** () const
- Type **type** () const

9.106 Canada Class Reference

```
#include <ql/time/calendars/canada.hpp>
```

Inheritance diagram for Canada:



9.106.1 Detailed Description

Canadian calendar.

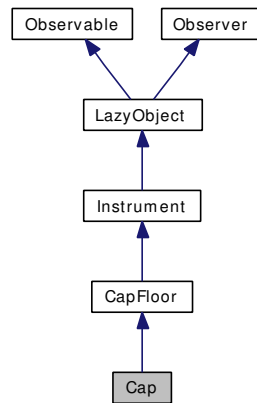
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Victoria Day, The Monday on or preceding 24 May
- [Canada](#) Day, July 1st (possibly moved to Monday)
- Provincial Holiday, first Monday of August
- Labour Day, first Monday of September
- Thanksgiving Day, second Monday of October
- Remembrance Day, November 11th
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

9.107 Cap Class Reference

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

Inheritance diagram for Cap:



9.107.1 Detailed Description

Concrete cap class.

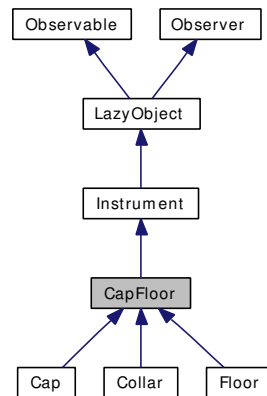
Public Member Functions

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

9.108 CapFloor Class Reference

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

Inheritance diagram for CapFloor:



9.108.1 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 Leg &floatingLeg, const std::vector< Rate > &capRates, const std::vector< Rate > &floorRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared_ptr< [PricingEngine](#) > &engine)
- **CapFloor** (Type type, const Leg &floatingLeg, const std::vector< Rate > &strikes, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared_ptr< [PricingEngine](#) > &engine)
- Volatility [impliedVolatility](#) (Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility minVol=1.0e-7, Volatility maxVol=4.0) const
implied term volatility

Instrument interface

- bool `isExpired ()` const
returns whether the instrument is still tradable.
- void `setupArguments (PricingEngine::arguments *)` const

Inspectors

- Type `type ()` const
- const Leg & `leg ()` const
- const std::vector< Rate > & `capRates ()` const
- const std::vector< Rate > & `floorRates ()` const
- const Leg & `floatingLeg ()` const
- Rate `atmRate ()` const
- Date `startDate ()` const
- Date `maturityDate ()` const
- Date `lastFixingDate ()` const

Classes

- class `arguments`
Arguments for cap/floor calculation
- class `engine`
base class for cap/floor engines

9.109 CapFloor::arguments Class Reference

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

9.109.1 Detailed Description

Arguments for cap/floor calculation

Public Member Functions

- void **validate** () const

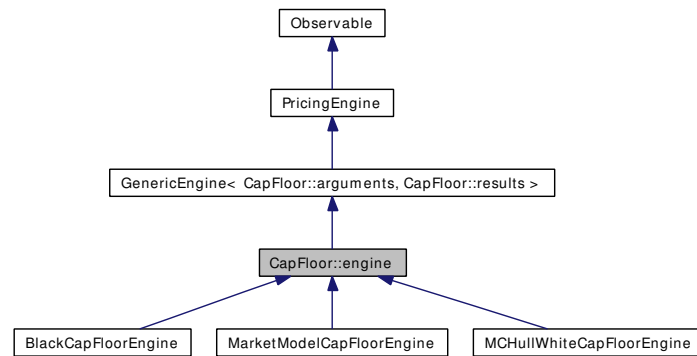
Public Attributes

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

9.110 CapFloor::engine Class Reference

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

Inheritance diagram for CapFloor::engine:



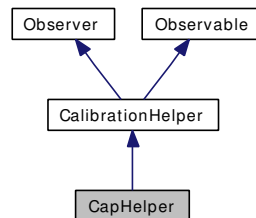
9.110.1 Detailed Description

base class for cap/floor engines

9.111 CapHelper Class Reference

```
#include <ql/models/shortrate/calibrationhelpers/caphelper.hpp>
```

Inheritance diagram for CapHelper:



9.111.1 Detailed Description

calibration helper for ATM cap

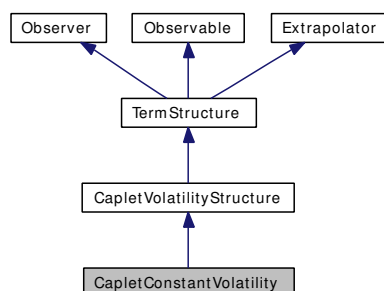
Public Member Functions

- **CapHelper** (const [Period](#) &length, const [Handle](#)< [Quote](#) > &volatility, const boost::shared_ptr< [IborIndex](#) > &index, [Frequency](#) fixedLegFrequency, const [DayCounter](#) &fixedLegDayCounter, bool includeFirstSwaplet, const [Handle](#)< [YieldTermStructure](#) > &termStructure, bool calibrateVolatility=false)
- virtual void **addTimesTo** (std::list< Time > ×) const
- virtual Real [modelValue](#) () const
returns the price of the instrument according to the model
- virtual Real [blackPrice](#) (Volatility volatility) const
Black price given a volatility.

9.112 CapletConstantVolatility Class Reference

#include <ql/termstructures/volatilities/capletconstantvol.hpp>

Inheritance diagram for CapletConstantVolatility:



9.112.1 Detailed Description

Constant caplet volatility, no time-strike dependence.

Public Member Functions

- **CapletConstantVolatility** (const [Date](#) &referenceDate, Volatility volatility, const [DayCounter](#) &dayCounter)
- **CapletConstantVolatility** (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)
- **CapletConstantVolatility** (Volatility volatility, const [DayCounter](#) &dayCounter)
- **CapletConstantVolatility** (const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)
- **Date** [maxDate](#) () const
the latest date for which the curve can return values
- **Real** [minStrike](#) () const
the minimum strike for which the term structure can return vols
- **Real** [maxStrike](#) () const
the maximum strike for which the term structure can return vols

TermStructure interface

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

Protected Member Functions

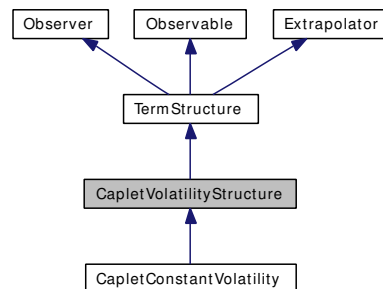
CapletVolatilityStructure interface

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

9.113 CapletVolatilityStructure Class Reference

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

Inheritance diagram for CapletVolatilityStructure:



9.113.1 Detailed Description

Caplet/floorlet forward-volatility structure.

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

Public Member Functions

Constructors

See the *TermStructure* documentation for issues regarding constructors.

- [CapletVolatilityStructure](#) (const [DayCounter](#) &dc=[Actual365Fixed](#)())
default constructor
- [CapletVolatilityStructure](#) (const [Date](#) &referenceDate, const [Calendar](#) &cal=[Calendar](#)()),
const [DayCounter](#) &dc=[Actual365Fixed](#)())
initialize with a fixed reference date
- [CapletVolatilityStructure](#) (Natural settlementDays, const [Calendar](#) &, const [DayCounter](#) &dc=[Actual365Fixed](#)())
calculate the reference date based on the global evaluation date

Volatility and Variance

- Volatility [volatility](#) (const [Date](#) &exerciseDate, Rate strike, bool extrapolate=false) const
returns the volatility for a given exercise date and strike rate
- Volatility [volatility](#) (Time t, Rate strike, bool extrapolate=false) const
returns the volatility for a given exercise time and strike rate
- Volatility [volatility](#) (const [Period](#) &optionTenor, Rate strike, bool extrapolate=false) const
returns the volatility for a given option tenor and strike rate

- Real [blackVariance](#) (const [Date](#) &exerciseDate, Rate strike, bool extrapolate=false) const
returns the Black variance for a given exercise date and strike rate
- Real [blackVariance](#) (Time t, Rate strike, bool extrapolate=false) const
returns the Black variance for a given start time and strike rate
- Volatility [blackVariance](#) (const [Period](#) &optionTenor, Rate strike, bool extrapolate=false) const
returns the Black variance for a given option tenor and strike rate

Limits

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

Protected Member Functions

- virtual Volatility [volatilityImpl](#) (Time length, Rate strike) const=0
implements the actual volatility calculation in derived classes

9.113.2 Constructor & Destructor Documentation

9.113.2.1 CapletVolatilityStructure (const DayCounter & dc = Actual365Fixed())

default constructor

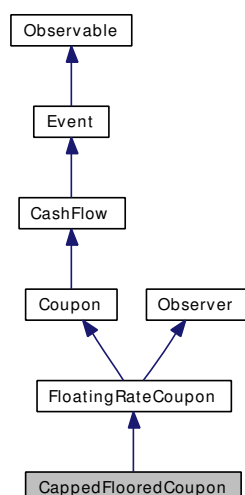
Warning

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

9.114 CappedFlooredCoupon Class Reference

```
#include <ql/cashflows/capflooredcoupon.hpp>
```

Inheritance diagram for CappedFlooredCoupon:



9.114.1 Detailed Description

Capped and/or floored floating-rate coupon.

The payoff P of a capped floating-rate coupon is:

$$P = N \times T \times \min(aL + b, C).$$

The payoff of a floored floating-rate coupon is:

$$P = N \times T \times \max(aL + b, F).$$

The payoff of a collared floating-rate coupon is:

$$P = N \times T \times \min(\max(aL + b, F), C).$$

where N is the notional, T is the accrual time, L is the floating rate, a is its gearing, b is the spread, and C and F the strikes.

They can be decomposed in the following manner. Decomposition of a capped floating rate coupon:

$$R = \min(aL + b, C) = (aL + b) + \min(C - b - \xi|a|L, 0)$$

where $\xi = \text{sgn}(a)$. Then:

$$R = (aL + b) + |a| \min\left(\frac{C - b}{|a|} - \xi L, 0\right)$$

Visitability

- `boost::shared_ptr< FloatingRateCoupon > underlying_`
- `bool isCapped_`

- bool **isFloored**_
- Rate **cap**_
- Rate **floor**_
- virtual void **accept** ([AcyclicVisitor](#) &)
- bool **isCapped** () const
- bool **isFloored** () const
- void **setPricer** (const boost::shared_ptr< [FloatingRateCouponPricer](#) > &pricer)

Public Member Functions

- **CappedFlooredCoupon** (const boost::shared_ptr< [FloatingRateCoupon](#) > &underlying, Rate cap=[Null](#)< Rate >(), Rate floor=[Null](#)< Rate >())
- Rate [cap](#) () const
cap
- Rate [floor](#) () const
floor
- Rate [effectiveCap](#) () const
effective cap of fixing
- Rate [effectiveFloor](#) () const
effective floor of fixing

Coupon interface

- Rate [rate](#) () const
accrued rate
- Rate [convexityAdjustment](#) () const
convexity adjustment

Observer interface

- void [update](#) ()

9.114.2 Member Function Documentation

9.114.2.1 void update () [virtual]

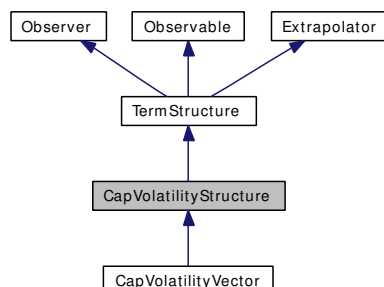
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [FloatingRateCoupon](#).

9.115 CapVolatilityStructure Class Reference

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

Inheritance diagram for CapVolatilityStructure:



9.115.1 Detailed Description

Cap/floor term-volatility structure.

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

Public Member Functions

Constructors

See the *TermStructure* documentation for issues regarding constructors.

- **CapVolatilityStructure** (const [DayCounter](#) &dc=[Actual365Fixed](#)())
default constructor
- **CapVolatilityStructure** (const [Date](#) &referenceDate, const [Calendar](#) &cal=[Calendar](#)()),
const [DayCounter](#) &dc=[Actual365Fixed](#)())
initialize with a fixed reference date
- **CapVolatilityStructure** (Natural settlementDays, const [Calendar](#) &, const [DayCounter](#) &dc=[Actual365Fixed](#)())
calculate the reference date based on the global evaluation date

Volatility

- Volatility **volatility** (const [Date](#) &end, Rate strike, bool extrapolate=false) const
- Volatility **volatility** (const [Period](#) &length, Rate strike, bool extrapolate=false) const
returns the volatility for a given cap/floor length and strike rate
- Volatility **volatility** (Time t, Rate strike, bool extrapolate=false) const
returns the volatility for a given end time and strike rate

Limits

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

Protected Member Functions

- virtual Volatility [volatilityImpl](#) (Time length, Rate strike) const=0
implements the actual volatility calculation in derived classes

9.115.2 Constructor & Destructor Documentation

9.115.2.1 CapVolatilityStructure (const DayCounter & dc = Actual365Fixed())

default constructor

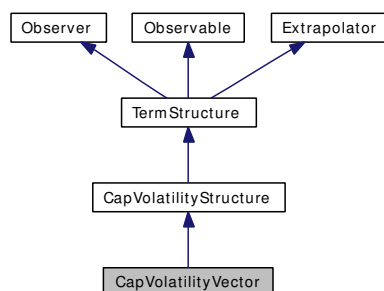
Warning

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

9.116 CapVolatilityVector Class Reference

```
#include <ql/termstructures/volatilities/capflatvolvector.hpp>
```

Inheritance diagram for CapVolatilityVector:



9.116.1 Detailed Description

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

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

Todo

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

Public Member Functions

- **CapVolatilityVector** (const [Date](#) &settlementDate, const std::vector< [Period](#) > &lengths, const std::vector< Volatility > &volatilities, const [DayCounter](#) &dayCounter)
- **CapVolatilityVector** (Natural settlementDays, const [Calendar](#) &calendar, const std::vector< [Period](#) > &lengths, const std::vector< Volatility > &volatilities, const [DayCounter](#) &dayCounter)
- [DayCounter](#) dayCounter () const
the day counter used for date/time conversion
- [Date](#) maxDate () const
the latest date for which the curve can return values
- Real minStrike () const
the minimum strike for which the term structure can return vols
- Real maxStrike () const
the maximum strike for which the term structure can return vols
- void update ()

9.116.2 Member Function Documentation

9.116.2.1 `void update ()` [virtual]

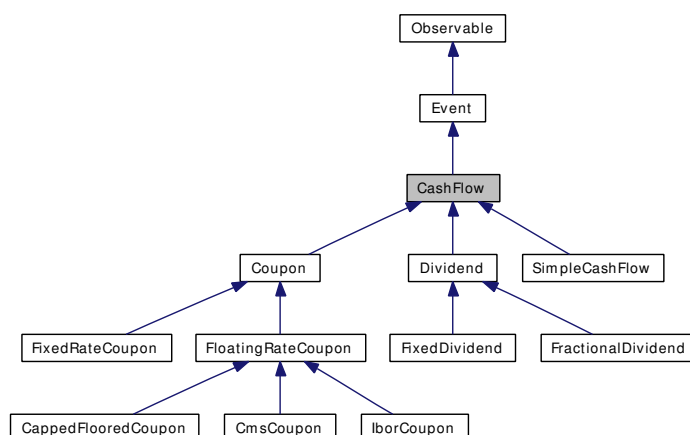
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

9.117 CashFlow Class Reference

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

Inheritance diagram for CashFlow:



9.117.1 Detailed Description

Base class for cash flows.

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

Public Member Functions

CashFlow interface

- virtual Real [amount](#) () const=0
returns the amount of the cash flow
- virtual [Date date](#) () const=0
Note:
This is inherited from the event class

Visitability

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

9.117.2 Member Function Documentation

9.117.2.1 virtual Real amount () const [pure virtual]

returns the amount of the cash flow

Note:

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

Implemented in [Dividend](#), [FixedDividend](#), [FractionalDividend](#), [FixedRateCoupon](#), [FloatingRateCoupon](#), and [SimpleCashFlow](#).

9.118 CashFlows Class Reference

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

9.118.1 Detailed Description

cashflow-analysis functions

Todo

add tests

Static Public Member Functions

- static [Date](#) **startDate** (const Leg &leg)
- static [Date](#) **maturityDate** (const Leg &leg)
- static Real **npv** (const Leg &leg, const [Handle](#)< [YieldTermStructure](#) > &discountCurve, const [Date](#) &settlementDate=[Date](#)(), const [Date](#) &npvDate=[Date](#)(), Integer exDividendDays=0)
NPV of the cash flows.
- static Real **npv** (const Leg &leg, const [InterestRate](#) &, [Date](#) settlementDate=[Date](#)())
NPV of the cash flows.
- static Real **bps** (const Leg &leg, const [Handle](#)< [YieldTermStructure](#) > &discountCurve, const [Date](#) &settlementDate=[Date](#)(), const [Date](#) &npvDate=[Date](#)(), Integer exDividendDays=0)
Basis-point sensitivity of the cash flows.
- static Real **bps** (const Leg &leg, const [InterestRate](#) &, [Date](#) settlementDate=[Date](#)())
Basis-point sensitivity of the cash flows.
- static Rate **atmRate** (const Leg &leg, const [Handle](#)< [YieldTermStructure](#) > &discountCurve, const [Date](#) &settlementDate=[Date](#)(), const [Date](#) &npvDate=[Date](#)(), Integer exDividendDays=0, Real npv=[Null](#)< Real >())
At-the-money rate of the cash flows.
- static Rate **irr** (const Leg &leg, Real marketPrice, const [DayCounter](#) &dayCounter, Compounding compounding, [Frequency](#) frequency=NoFrequency, [Date](#) settlementDate=[Date](#)(), Real tolerance=1.0e-10, Size maxIterations=10000, Rate guess=0.05)
Internal rate of return.
- static Time **duration** (const Leg &leg, const [InterestRate](#) &y, Duration::Type type=Duration::Modified, [Date](#) settlementDate=[Date](#)())
Cash-flow duration.
- static Real **convexity** (const Leg &leg, const [InterestRate](#) &y, [Date](#) settlementDate=[Date](#)())
Cash-flow convexity.

9.118.2 Member Function Documentation

9.118.2.1 `static Real npv (const Leg & leg, const Handle< YieldTermStructure > & discountCurve, const Date & settlementDate = Date(), const Date & npvDate = Date(), Integer exDividendDays = 0) [static]`

NPV of the cash flows.

The NPV is the sum of the cash flows, each discounted according to the given term structure.

9.118.2.2 `static Real npv (const Leg & leg, const InterestRate &, Date settlementDate = Date()) [static]`

NPV of the cash flows.

The NPV is the sum of the cash flows, each discounted according to the given constant interest rate. The result is affected by the choice of the interest-rate compounding and the relative frequency and day counter.

9.118.2.3 `static Real bps (const Leg & leg, const Handle< YieldTermStructure > & discountCurve, const Date & settlementDate = Date(), const Date & npvDate = Date(), Integer exDividendDays = 0) [static]`

Basis-point sensitivity of the cash flows.

The result is the change in NPV due to a uniform 1-basis-point change in the rate paid by the cash flows. The change for each coupon is discounted according to the given term structure.

9.118.2.4 `static Real bps (const Leg & leg, const InterestRate &, Date settlementDate = Date()) [static]`

Basis-point sensitivity of the cash flows.

The result is the change in NPV due to a uniform 1-basis-point change in the rate paid by the cash flows. The change for each coupon is discounted according to the given constant interest rate. The result is affected by the choice of the interest-rate compounding and the relative frequency and day counter.

9.118.2.5 `static Rate atmRate (const Leg & leg, const Handle< YieldTermStructure > & discountCurve, const Date & settlementDate = Date(), const Date & npvDate = Date(), Integer exDividendDays = 0, Real npv = Null< Real >()) [static]`

At-the-money rate of the cash flows.

The result is the fixed rate for which a fixed rate cash flow vector, equivalent to the input vector, has the required NPV according to the given term structure. If the required NPV is not given, the input cash flow vector's NPV is used instead.

9.118.2.6 static Rate irr (const Leg & *leg*, Real *marketPrice*, const DayCounter & *dayCounter*, Compounding *compounding*, Frequency *frequency* = NoFrequency, Date *settlementDate* = Date(), Real *tolerance* = 1.0e-10, Size *maxIterations* = 10000, Rate *guess* = 0.05) [static]

Internal rate of return.

The IRR is the interest rate at which the NPV of the cash flows equals the given market price. The function verifies the theoretical existence of an IRR and numerically establishes the IRR to the desired precision.

9.118.2.7 static Time duration (const Leg & *leg*, const InterestRate & *y*, Duration::Type *type* = Duration::Modified, Date *settlementDate* = Date()) [static]

Cash-flow duration.

The simple duration of a string of cash flows is defined as

$$D_{\text{simple}} = \frac{\sum t_i c_i B(t_i)}{\sum c_i B(t_i)}$$

where c_i is the amount of the i -th cash flow, t_i is its payment time, and $B(t_i)$ is the corresponding discount according to the passed yield.

The modified duration is defined as

$$D_{\text{modified}} = -\frac{1}{P} \frac{\partial P}{\partial y}$$

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

The Macaulay duration is defined for a compounded IRR as

$$D_{\text{Macaulay}} = \left(1 + \frac{y}{N}\right) D_{\text{modified}}$$

where y is the IRR and N is the number of cash flows per year.

9.118.2.8 static Real convexity (const Leg & *leg*, const InterestRate & *y*, Date *settlementDate* = Date()) [static]

Cash-flow convexity.

The convexity of a string of cash flows is defined as

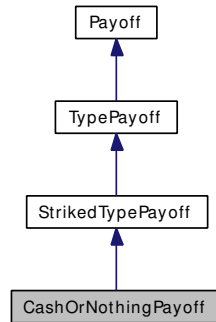
$$C = \frac{1}{P} \frac{\partial^2 P}{\partial y^2}$$

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

9.119 CashOrNothingPayoff Class Reference

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

Inheritance diagram for CashOrNothingPayoff:



9.119.1 Detailed Description

Binary cash-or-nothing payoff.

Examples:

[Replication.cpp](#).

Public Member Functions

- **CashOrNothingPayoff** (Option::Type type, Real strike, Real cashPayoff)
- Real **cashPayoff** () const

Payoff interface

- std::string **name** () const
- std::string **description** () const
- Real **operator()** (Real price) const
- virtual void **accept** ([AcyclicVisitor](#) &)

Protected Attributes

- Real **cashPayoff_**

9.119.2 Member Function Documentation

9.119.2.1 std::string name () const [virtual]

Warning

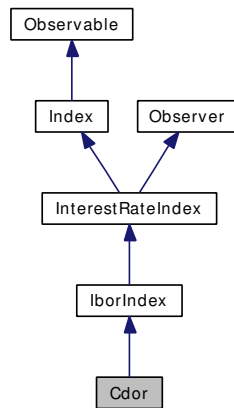
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implements [Payoff](#).

9.120 Cdor Class Reference

```
#include <ql/indexes/ibor/cdor.hpp>
```

Inheritance diagram for Cdor:



9.120.1 Detailed Description

CDOR rate

Canadian Dollar Offered Rate fixed by IDA.

Warning

This is the rate fixed in [Canada](#) by IDA. Use [CADLibor](#) if you're interested in the London fixing by BBA.

Todo

check settlement days, end-of-month adjustment, and day-count convention.

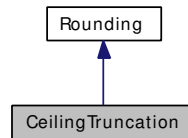
Public Member Functions

- **Cdor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.121 CeilingTruncation Class Reference

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

Inheritance diagram for CeilingTruncation:



9.121.1 Detailed Description

Ceiling truncation.

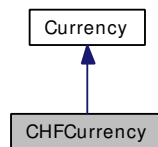
Public Member Functions

- **CeilingTruncation** (Integer precision, Integer digit=5)

9.122 CHFCurrency Class Reference

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

Inheritance diagram for CHFCurrency:



9.122.1 Detailed Description

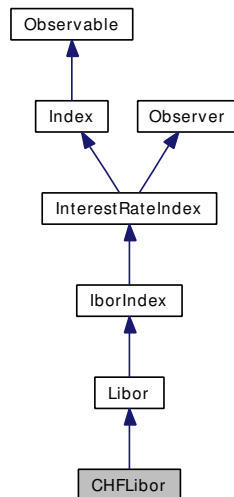
Swiss franc.

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

9.123 CHFLibor Class Reference

```
#include <ql/indexes/ibor/chflibor.hpp>
```

Inheritance diagram for CHFLibor:



9.123.1 Detailed Description

CHF LIBOR rate

Swiss Franc LIBOR fixed by BBA.

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

Warning

This is the rate fixed in London by BBA. Use ZIBOR if you're interested in the Zurich fixing.

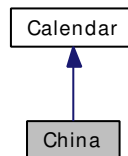
Public Member Functions

- **CHFLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), Natural settlementDays=2)

9.124 China Class Reference

```
#include <ql/time/calendars/china.hpp>
```

Inheritance diagram for China:



9.124.1 Detailed Description

Chinese calendar.

Holidays:

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

Other holidays for which no rule is given:

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

9.125 CLGaussianRng Class Template Reference

```
#include <ql/math/randomnumbers/centrallimitgaussianrng.hpp>
```

9.125.1 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**

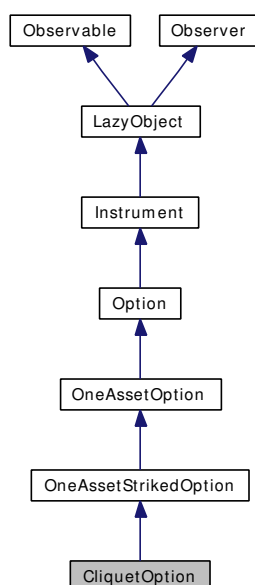
Public Member Functions

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

9.126 CliquetOption Class Reference

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

Inheritance diagram for CliquetOption:



9.126.1 Detailed Description

cliquet (Ratchet) option

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

Todo

- add local/global caps/floors
- add accrued coupon and last fixing

Public Member Functions

- **CliquetOption** (const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [PercentageStrikePayoff](#) > &, const boost::shared_ptr< [EuropeanExercise](#) > & maturity, const std::vector< [Date](#) > &resetDates, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void **setupArguments** ([PricingEngine::arguments](#) *) const

Classes

- class [arguments](#)
Arguments for cliquet option calculation

- class [engine](#)

Cliquet engine base class.

9.127 CliquetOption::arguments Class Reference

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

9.127.1 Detailed Description

Arguments for cliquet option calculation

Public Member Functions

- void **validate** () const

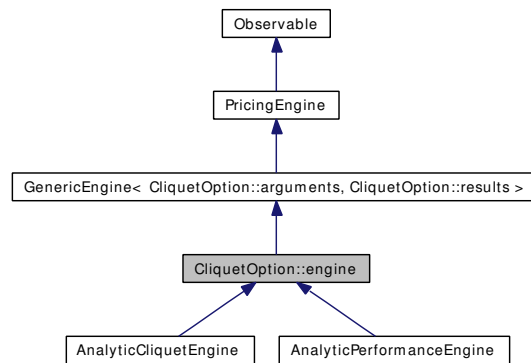
Public Attributes

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

9.128 CliquetOption::engine Class Reference

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

Inheritance diagram for CliquetOption::engine:



9.128.1 Detailed Description

Cliquet engine base class.

9.129 Clone Class Template Reference

```
#include <ql/utilities/clone.hpp>
```

9.129.1 Detailed Description

template<class T> class QuantLib::Clone< T >

cloning proxy to an underlying object

When copied, this class will make a clone of its underlying object (which must provide a `clone()` method returning a `std::auto_ptr` to a newly-allocated instance.)

Public Member Functions

- **Clone** (`std::auto_ptr< T >`)
- **Clone** (`const T &`)
- **Clone** (`const Clone< T > &`)
- **Clone**< T > & **operator=** (`const T &`)
- **Clone**< T > & **operator=** (`const Clone< T > &`)
- `T & operator *` (`() const`)
- `T * operator →` (`() const`)
- `bool empty` (`() const`)
- `void swap` (`Clone< T > &t`)

Related Functions

(Note that these are not member functions.)

- `void swap` (`Clone< T > &`, `Clone< T > &`)

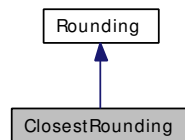
9.129.2 Friends And Related Function Documentation

9.129.2.1 `void swap` (`Clone< T > &`, `Clone< T > &`) [related]

9.130 ClosestRounding Class Reference

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

Inheritance diagram for ClosestRounding:



9.130.1 Detailed Description

Closest rounding.

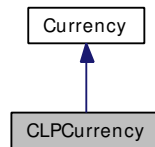
Public Member Functions

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

9.131 CLPCurrency Class Reference

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

Inheritance diagram for CLPCurrency:



9.131.1 Detailed Description

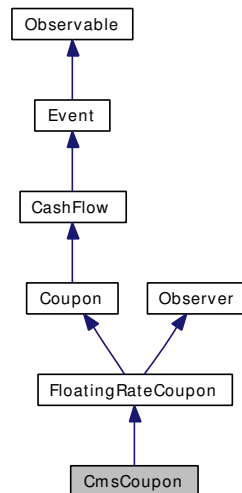
Chilean peso.

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

9.132 CmsCoupon Class Reference

```
#include <ql/cashflows/cmscoupon.hpp>
```

Inheritance diagram for CmsCoupon:



9.132.1 Detailed Description

CMS coupon class.

Warning

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **CmsCoupon** (const [Date](#) &paymentDate, const Real nominal, const [Date](#) &startDate, const [Date](#) &endDate, const Natural fixingDays, const boost::shared_ptr< [SwapIndex](#) > &index, const Real gearing=1.0, const Spread spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)(), bool isInArrears=false)

Inspectors

- const boost::shared_ptr< [SwapIndex](#) > & **swapIndex** () const

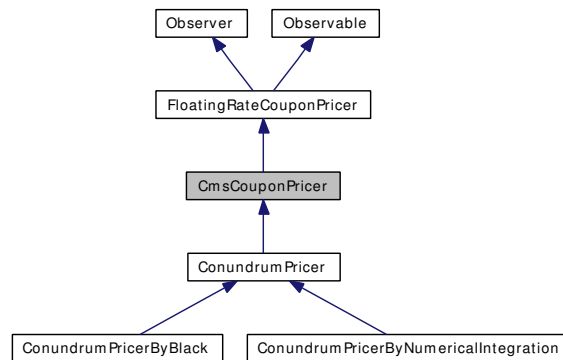
Visitability

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

9.133 CmsCouponPricer Class Reference

```
#include <ql/cashflows/couponpricer.hpp>
```

Inheritance diagram for CmsCouponPricer:



9.133.1 Detailed Description

base pricer for vanilla CMS coupons

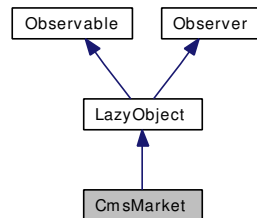
Public Member Functions

- **CmsCouponPricer** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &swaptionVol)
- [Handle](#)< [SwaptionVolatilityStructure](#) > **swaptionVolatility** () const
- void **setSwaptionVolatility** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &swaptionVol)

9.134 CmsMarket Class Reference

```
#include <ql/termstructures/volatilities/cmsmarket.hpp>
```

Inheritance diagram for CmsMarket:



9.134.1 Detailed Description

set of CMS quotes

Public Member Functions

- **CmsMarket** (const std::vector< [Period](#) > &expiries, const std::vector< boost::shared_ptr< [SwapIndex](#) > > &swapIndices, const std::vector< std::vector< [Handle](#)< [Quote](#) > > > &bidAskSpreads, const std::vector< boost::shared_ptr< [CmsCouponPricer](#) > > &pricers, const [Handle](#)< [YieldTermStructure](#) > &yieldTermStructure)
- void **reprice** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &volStructure, Real meanReversion)
- const std::vector< [Period](#) > & **swapTenors** () const
- [Matrix](#) **meanReversions** ()
- [Matrix](#) **impliedCmsSpreads** ()
- [Matrix](#) **spreadErrors** ()
- [Matrix](#) **browse** () const
- Real **weightedError** (const [Matrix](#) &weights)
- Real **weightedPriceError** (const [Matrix](#) &weights)
- Real **weightedForwardPriceError** (const [Matrix](#) &weights)
- [Disposable](#)< [Array](#) > **weightedErrors** (const [Matrix](#) &weights)
- [Disposable](#)< [Array](#) > **weightedPriceErrors** (const [Matrix](#) &weights)
- [Disposable](#)< [Array](#) > **weightedForwardPriceErrors** (const [Matrix](#) &weights)

LazyObject interface

- void [update](#) ()

9.134.2 Member Function Documentation

9.134.2.1 void update () [virtual]

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

Reimplemented from [LazyObject](#).

9.135 CMSMMDriftCalculator Class Reference

```
#include <ql/models/marketmodels/driftcomputation/cmsmmdriftcalculator.hpp>
```

9.135.1 Detailed Description

Drift computation for CMS market models.

Returns the drift $\mu\Delta t$. See Mark Joshi, *Rapid Computation of Drifts in a Reduced Factor [Libor](#) Market Model*, Wilmott Magazine, May 2003.

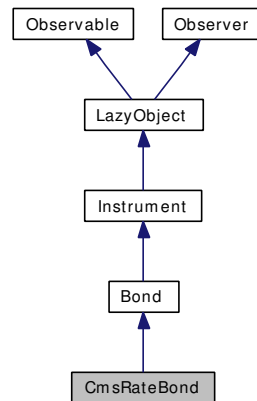
Public Member Functions

- **CMSMMDriftCalculator** (const [Matrix](#) &pseudo, const std::vector< Spread > &displacements, const std::vector< Time > &taus, Size numeraire, Size alive, Size spanningFwds)
- void **compute** (const [CMSwapCurveState](#) &cs, std::vector< Real > &drifts) const
Computes the drifts.

9.136 CmsRateBond Class Reference

```
#include <ql/instruments/cmsratebond.hpp>
```

Inheritance diagram for CmsRateBond:



9.136.1 Detailed Description

CMS-rate bond.

Tests

calculations are tested by checking results against cached values.

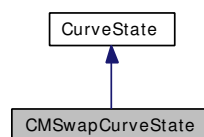
Public Member Functions

- **CmsRateBond** (Natural settlementDays, Real faceAmount, const [Schedule](#) &schedule, const boost::shared_ptr< [SwapIndex](#) > &index, const [DayCounter](#) &paymentDayCounter, [BusinessDayConvention](#) paymentConvention=Following, Natural fixingDays=[Null](#)< Natural >(), const std::vector< Real > &gearings=std::vector< Real >(1, 1.0), const std::vector< Spread > &spreads=std::vector< Spread >(1, 0.0), const std::vector< Rate > &caps=std::vector< Rate >(), const std::vector< Rate > &floors=std::vector< Rate >(), bool inArrears=false, Real redemption=100.0, const [Date](#) &issueDate=[Date](#)(), const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())

9.137 CMSwapCurveState Class Reference

```
#include <ql/models/marketmodels/curvestates/cmswapcurvestate.hpp>
```

Inheritance diagram for CMSwapCurveState:



9.137.1 Detailed Description

Curve state for constant-maturity-swap market models

Public Member Functions

- **CMSwapCurveState** (const std::vector< Time > &rateTimes, Size spanningForwards)

Modifiers

- void **setOnCMSwapRates** (const std::vector< Rate > &cmSwapRates, Size firstValidIndex=0)

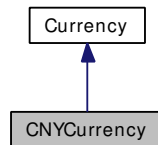
Inspectors

- Real **discountRatio** (Size i, Size j) const
- Rate **forwardRate** (Size i) const
- Rate **coterminalSwapRate** (Size i) const
- Rate **coterminalSwapAnnuity** (Size numeraire, Size i) const
- Rate **cmSwapRate** (Size i, Size spanningForwards) const
- Rate **cmSwapAnnuity** (Size numeraire, Size i, Size spanningForwards) const
- const std::vector< Rate > & **forwardRates** () const
- const std::vector< Rate > & **coterminalSwapRates** () const
- const std::vector< Rate > & **cmSwapRates** (Size spanningForwards) const
- std::auto_ptr< [CurveState](#) > **clone** () const

9.138 CNYCurrency Class Reference

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

Inheritance diagram for CNYCurrency:



9.138.1 Detailed Description

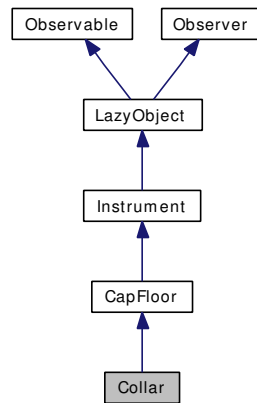
Chinese yuan.

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

9.139 Collar Class Reference

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

Inheritance diagram for Collar:



9.139.1 Detailed Description

Concrete collar class.

Public Member Functions

- **Collar** (const Leg &floatingLeg, const std::vector< Rate > &capRates, const std::vector< Rate > &floorRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared_ptr< [PricingEngine](#) > &engine)

9.140 Composite Class Template Reference

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

9.140.1 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)

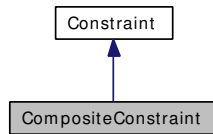
Protected Attributes

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

9.141 CompositeConstraint Class Reference

```
#include <ql/math/optimization/constraint.hpp>
```

Inheritance diagram for CompositeConstraint:



9.141.1 Detailed Description

Constraint enforcing both given sub-constraints

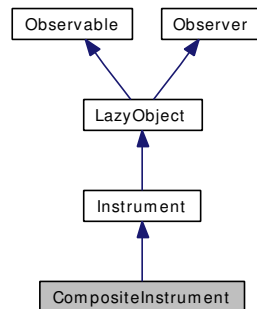
Public Member Functions

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

9.142 CompositeInstrument Class Reference

```
#include <ql/instruments/compositeinstrument.hpp>
```

Inheritance diagram for CompositeInstrument:



9.142.1 Detailed Description

Composite instrument

This instrument is an aggregate of other instruments. Its NPV is the sum of the NPVs of its components, each possibly multiplied by a given factor.

Example: [static replication of a down-and-out barrier option](#)

Warning

Methods that drive the calculation directly (such as [recalculate\(\)](#), [freeze\(\)](#) and others) might not work correctly.

Examples:

[Replication.cpp](#).

Instrument interface

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

Public Member Functions

- void [add](#) (const boost::shared_ptr< [Instrument](#) > &instrument, Real multiplier=1.0)
adds an instrument to the composite
- void [subtract](#) (const boost::shared_ptr< [Instrument](#) > &instrument, Real multiplier=1.0)
shorts an instrument from the composite

9.142.2 Member Function Documentation

9.142.2.1 void performCalculations () const [protected, virtual]

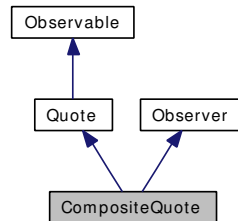
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

9.143 CompositeQuote Class Template Reference

```
#include <ql/quotes/compositequote.hpp>
```

Inheritance diagram for CompositeQuote:



9.143.1 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](#) ()

9.143.2 Member Function Documentation

9.143.2.1 void update () [virtual]

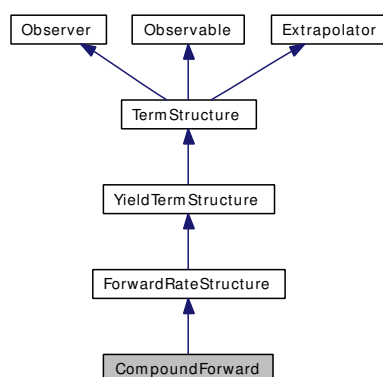
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.144 CompoundForward Class Reference

```
#include <ql/termstructures/yieldcurves/compoundforward.hpp>
```

Inheritance diagram for CompoundForward:



9.144.1 Detailed Description

compound-forward structure

Tests

- the correctness of the curve is tested by reproducing the input data.
- the correctness of the curve is tested by checking the consistency between returned rates and swaps priced on the curve.

Bug

swap rates are not reproduced exactly when using indexed coupons. Apparently, some assumption about the swap fixings is hard-coded into the bootstrapping algorithm.

Public Member Functions

- **CompoundForward** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &dates, const std::vector< [Rate](#) > &forwards, const [Calendar](#) &calendar, const [BusinessDayConvention](#) conv, const Integer compounding, const [DayCounter](#) &dayCounter)
- [BusinessDayConvention](#) **businessDayConvention** () const
- Integer **compounding** () const
- [Date](#) **maxDate** () const
the latest date for which the curve can return values
- const std::vector< [Time](#) > & **times** () const
- const std::vector< [Date](#) > & **dates** () const
- const std::vector< [Rate](#) > & **forwards** () const
- boost::shared_ptr< [ExtendedDiscountCurve](#) > **discountCurve** () const
- [Rate](#) **compoundForward** (const [Date](#) &d1, Integer f, bool extrapolate=false) const
- [Rate](#) **compoundForward** ([Time](#) t1, Integer f, bool extrapolate=false) const

Protected Member Functions

- void **calibrateNodes** () const
- boost::shared_ptr< [YieldTermStructure](#) > **bootstrap** () const
- Rate [zeroYieldImpl](#) (Time) const
- DiscountFactor [discountImpl](#) (Time) const
- Size **referenceNode** (Time) const
- Rate [forwardImpl](#) (Time) const
instantaneous forward-rate calculation
- Rate **compoundForwardImpl** (Time, Integer) const

9.144.2 Member Function Documentation

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

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

[Warning](#)

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

Reimplemented from [ForwardRateStructure](#).

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

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

Reimplemented from [ForwardRateStructure](#).

9.145 ConjugateGradient Class Reference

```
#include <ql/math/optimization/conjugategradient.hpp>
```

9.145.1 Detailed Description

Multi-dimensional Conjugate Gradient class.

User has to provide line-search method and optimization end criteria. Search direction $d_i = -f'(x_i) + c_i * d_{i-1}$ where $c_i = \|f'(x_i)\|^2 / \|f'(x_{i-1})\|^2$ and $d_1 = -f'(x_1)$

Public Member Functions

- **ConjugateGradient** (const boost::shared_ptr< [LineSearch](#) > &lineSearch=boost::shared_ptr< [LineSearch](#) >())
- virtual EndCriteria::Type [minimize](#) ([Problem](#) &P, const [EndCriteria](#) &endCriteria)
solve the optimization problem P

9.146 ConstantEstimator Class Reference

```
#include <ql/models/volatility/constantestimator.hpp>
```

9.146.1 Detailed Description

Constant-estimator volatility model.

Volatilities are assumed to be expressed on an annual basis.

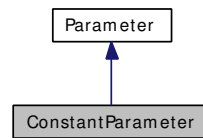
Public Member Functions

- **ConstantEstimator** (Size size)
- [TimeSeries](#)< Volatility > **calculate** (const [TimeSeries](#)< Volatility > &)
- void **calibrate** (const [TimeSeries](#)< Volatility > &)

9.147 ConstantParameter Class Reference

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

Inheritance diagram for ConstantParameter:



9.147.1 Detailed Description

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

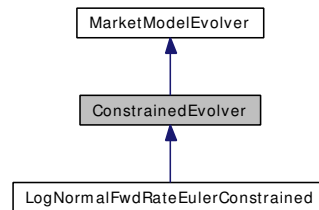
Public Member Functions

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

9.148 ConstrainedEvolver Class Reference

```
#include <ql/models/marketmodels/constrainedevolver.hpp>
```

Inheritance diagram for ConstrainedEvolver:



9.148.1 Detailed Description

Constrained market-model evolver.

Abstract base class. Requires extra methods above that of marketmodelevolver to let you fix rates via importance sampling.

The evolver does the actual gritty work of evolving the forward rates from one time to the next.

This is intended to be used for the Fries-Joshi proxy simulation approach to [Greeks](#)

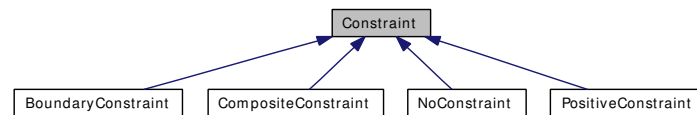
Public Member Functions

- virtual void [setConstraintType](#) (const std::vector< Size > &startIndexOfSwapRate, const std::vector< Size > &EndIndexOfSwapRate)=0
call once
- virtual void [setThisConstraint](#) (const std::vector< Rate > &rateConstraints, const std::vector< bool > &isConstraintActive)=0
call before each path

9.149 Constraint Class Reference

```
#include <ql/math/optimization/constraint.hpp>
```

Inheritance diagram for Constraint:



9.149.1 Detailed Description

Base constraint class.

Public Member Functions

- `bool empty () const`
- `bool test (const Array &p) const`
- `Real update (Array &p, const Array &direction, Real beta)`
- `Constraint (const boost::shared_ptr< Impl > &impl=boost::shared_ptr< Impl >())`

Protected Attributes

- `boost::shared_ptr< Impl > impl_`

Classes

- class [Impl](#)
Base class for constraint implementations.

9.150 Constraint::Impl Class Reference

```
#include <ql/math/optimization/constraint.hpp>
```

9.150.1 Detailed Description

Base class for constraint implementations.

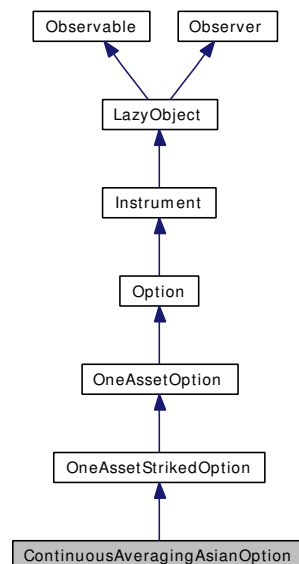
Public Member Functions

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

9.151 ContinuousAveragingAsianOption Class Reference

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

Inheritance diagram for ContinuousAveragingAsianOption:



9.151.1 Detailed Description

Continuous-averaging Asian option.

Todo

add running average

Public Member Functions

- **ContinuousAveragingAsianOption** (Average::Type averageType, const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void **setupArguments** ([PricingEngine::arguments](#) *) const

Protected Attributes

- Average::Type **averageType_**

Classes

- class [arguments](#)

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

- class [engine](#)

Continuous-averaging Asian engine base class.

9.152 ContinuousAveragingAsianOption::arguments Class Reference

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

9.152.1 Detailed Description

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

Public Member Functions

- void **validate** () const

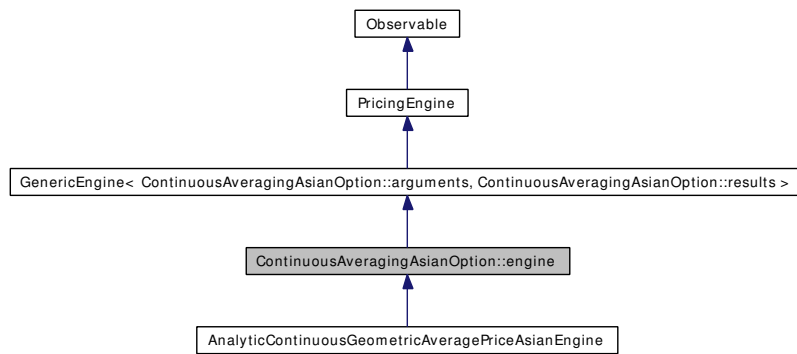
Public Attributes

- Average::Type **averageType**

9.153 ContinuousAveragingAsianOption::engine Class Reference

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

Inheritance diagram for ContinuousAveragingAsianOption::engine:



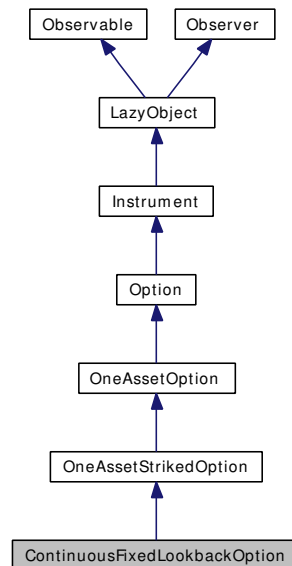
9.153.1 Detailed Description

Continuous-averaging Asian engine base class.

9.154 ContinuousFixedLookbackOption Class Reference

```
#include <ql/instruments/lookbackoption.hpp>
```

Inheritance diagram for ContinuousFixedLookbackOption:



9.154.1 Detailed Description

Continuous-fixed lookback option.

Public Member Functions

- **ContinuousFixedLookbackOption** (Real currentMinmax, const boost::shared_ptr< [StochasticProcess](#) > &process, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void **setupArguments** ([PricingEngine::arguments](#) *) const

Protected Attributes

- Real minmax_

Classes

- class [arguments](#)
Arguments for continuous fixed lookback option calculation
- class [engine](#)
Continuous fixed lookback engine base class

9.155 ContinuousFixedLookbackOption::arguments Class Reference

```
#include <ql/instruments/lookbackoption.hpp>
```

9.155.1 Detailed Description

Arguments for continuous fixed lookback option calculation

Public Member Functions

- void **validate** () const

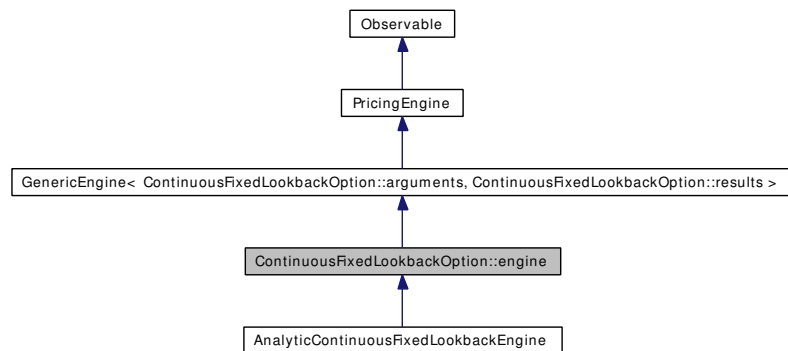
Public Attributes

- Real **minmax**

9.156 ContinuousFixedLookbackOption::engine Class Reference

```
#include <ql/instruments/lookbackoption.hpp>
```

Inheritance diagram for ContinuousFixedLookbackOption::engine:



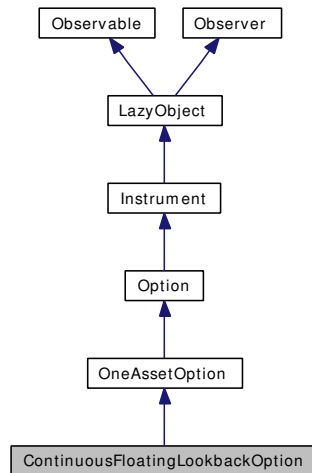
9.156.1 Detailed Description

Continuous fixed lookback engine base class

9.157 ContinuousFloatingLookbackOption Class Reference

```
#include <ql/instruments/lookbackoption.hpp>
```

Inheritance diagram for ContinuousFloatingLookbackOption:



9.157.1 Detailed Description

Continuous-floating lookback option.

Public Member Functions

- **ContinuousFloatingLookbackOption** (Real currentMinmax, const boost::shared_ptr< [StochasticProcess](#) > &process, const boost::shared_ptr< [TypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void **setupArguments** ([PricingEngine::arguments](#) *) const

Protected Attributes

- Real **minmax_**

Classes

- class [arguments](#)
Arguments for continuous floating lookback option calculation
- class [engine](#)
Continuous floating lookback engine base class

9.158 ContinuousFloatingLookbackOption::arguments Class Reference

```
#include <ql/instruments/lookbackoption.hpp>
```

9.158.1 Detailed Description

Arguments for continuous floating lookback option calculation

Public Member Functions

- void **validate** () const

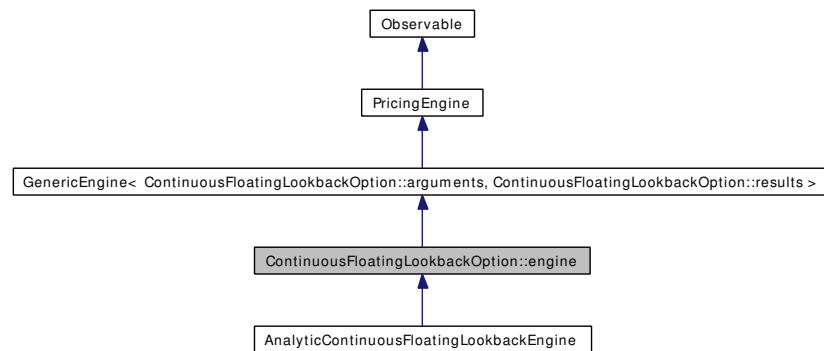
Public Attributes

- Real **minmax**

9.159 ContinuousFloatingLookbackOption::engine Class Reference

```
#include <ql/instruments/lookbackoption.hpp>
```

Inheritance diagram for ContinuousFloatingLookbackOption::engine:



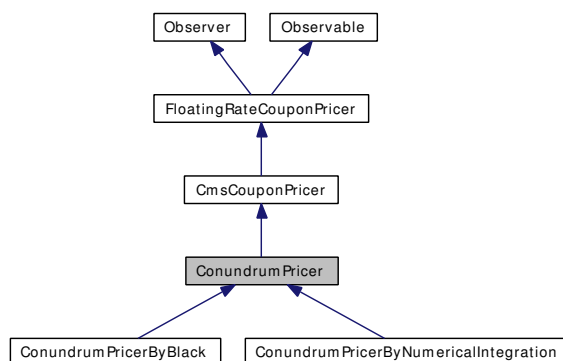
9.159.1 Detailed Description

Continuous floating lookback engine base class

9.160 ConundrumPricer Class Reference

```
#include <ql/cashflows/conundrumpricer.hpp>
```

Inheritance diagram for ConundrumPricer:



9.160.1 Detailed Description

CMS-coupon pricer.

Base class for the pricing of a CMS coupon via static replication as in Hagan's "Conundrums..." article

Public Member Functions

- virtual Real **swapletPrice** () const=0
- virtual Rate **swapletRate** () const
- virtual Real **capletPrice** (Rate effectiveCap) const
- virtual Rate **capletRate** (Rate effectiveCap) const
- virtual Real **floorletPrice** (Rate effectiveFloor) const
- virtual Rate **floorletRate** (Rate effectiveFloor) const
- Real **meanReversion** () const
- void **setMeanReversion** (const [Handle](#)< [Quote](#) > &meanReversion)

Protected Member Functions

- **ConundrumPricer** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &swaptionVol, GFunctionFactory::ModelOfYieldCurve modelOfYieldCurve, const [Handle](#)< [Quote](#) > &meanReversion)
- void **initialize** (const [FloatingRateCoupon](#) &coupon)
- virtual Real **optionletPrice** (Option::Type optionType, Real strike) const=0

Protected Attributes

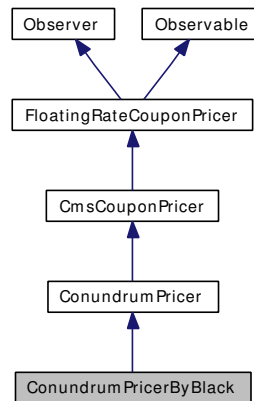
- boost::shared_ptr< [YieldTermStructure](#) > **rateCurve_**
- GFunctionFactory::ModelOfYieldCurve **modelOfYieldCurve_**
- boost::shared_ptr< GFunction > **gFunction_**

- const [CmsCoupon](#) * coupon_
- [Date](#) paymentDate_
- [Date](#) fixingDate_
- Rate swapRateValue_
- DiscountFactor discount_
- Real annuity_
- Real gearing_
- Spread spread_
- Real spreadLegValue_
- Rate cutoffForCaplet_
- Rate cutoffForFloorlet_
- [Handle](#)< [Quote](#) > meanReversion_
- [Period](#) swapTenor_
- boost::shared_ptr< VanillaOptionPricer > vanillaOptionPricer_

9.161 ConundrumPricerByBlack Class Reference

```
#include <ql/cashflows/conundrumpricer.hpp>
```

Inheritance diagram for ConundrumPricerByBlack:



9.161.1 Detailed Description

CMS-coupon pricer.

Public Member Functions

- **ConundrumPricerByBlack** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &swaptionVol, [GFunctionFactory::ModelOfYieldCurve](#) modelOfYieldCurve, const [Handle](#)< [Quote](#) > &meanReversion)

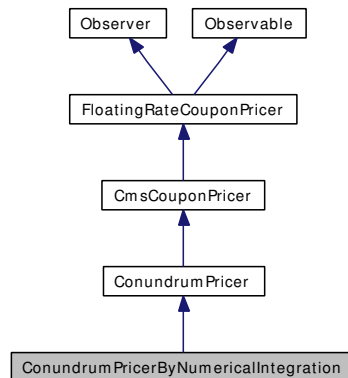
Protected Member Functions

- Real **optionletPrice** (Option::Type optionType, Real strike) const
- Real **swapletPrice** () const

9.162 ConundrumPricerByNumericalIntegration Class Reference

```
#include <ql/cashflows/conundrumpricer.hpp>
```

Inheritance diagram for ConundrumPricerByNumericalIntegration:



9.162.1 Detailed Description

CMS-coupon pricer.

Prices a cms coupon via static replication as in Hagan's "Conundrums..." article via numerical integration based on prices of vanilla swaptions

Public Member Functions

- **ConundrumPricerByNumericalIntegration** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &swaptionVol, [GFunctionFactory::ModelOfYieldCurve](#) modelOfYieldCurve, const [Handle](#)< [Quote](#) > &meanReversion, Rate lowerLimit=0.0, Rate upperLimit=1.0, Real precision=1.0e-6)
- Real **upperLimit** ()
- Real **stdDeviations** ()
- Real **integrate** (Real a, Real b, const [ConundrumIntegrand](#) &Integrand) const
- virtual Real **optionletPrice** ([Option::Type](#) optionType, Rate strike) const
- virtual Real **swapletPrice** () const
- Real **resetUpperLimit** (Real stdDeviationsForUpperLimit) const
- Real **refineIntegration** (Real integralValue, const [ConundrumIntegrand](#) &integrand) const

Public Attributes

- Real **upperLimit_**
- Real **stdDeviationsForUpperLimit_**
- const Real **lowerLimit_**
- const Real **requiredStdDeviations_**
- const Real **precision_**
- const Real **refiningIntegrationTolerance_**

9.163 ConvergenceStatistics Class Template Reference

```
#include <ql/math/statistics/convergencestatistics.hpp>
```

9.163.1 Detailed Description

```
template<class T, class U = DoublingConvergenceSteps> class
QuantLib::ConvergenceStatistics< T, U >
```

statistics class with convergence table

This class decorates another statistics class adding a convergence table calculation. The table tracks the convergence of the mean.

It is possible to specify the number of samples at which the mean should be stored by mean of the second template parameter; the default is to store 2^{n-1} samples at the n -th step. Any passed class must implement the following interface:

```
Size initialSamples() const;
Size nextSamples(Size currentSamples) const;
```

as well as a copy constructor.

Tests

results are tested against known good values.

Public Types

- typedef T::value_type **value_type**
- typedef std::vector< std::pair< Size, value_type > > **table_type**

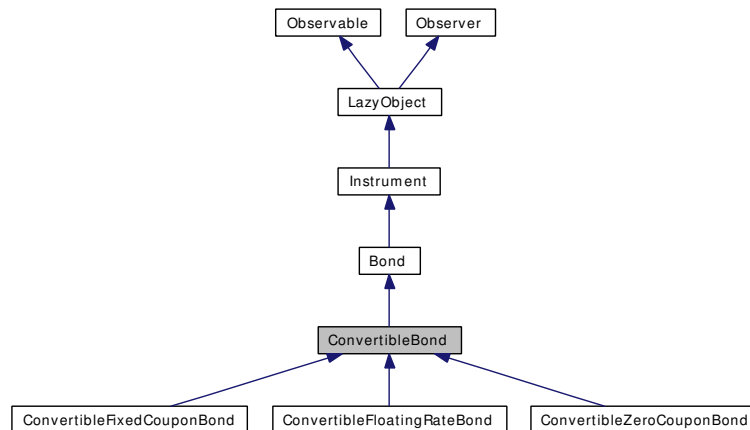
Public Member Functions

- **ConvergenceStatistics** (const T &stats, const U &rule=U())
- **ConvergenceStatistics** (const U &rule=U())
- void **add** (const value_type &value, Real weight=1.0)
- template<class DataIterator>
void **addSequence** (DataIterator begin, DataIterator end)
- template<class DataIterator, class WeightIterator>
void **addSequence** (DataIterator begin, DataIterator end, WeightIterator wbegin)
- void **reset** ()
- const std::vector< std::pair< Size, value_type > > & **convergenceTable** () const

9.164 ConvertibleBond Class Reference

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

Inheritance diagram for ConvertibleBond:



9.164.1 Detailed Description

base class for convertible bonds

Public Member Functions

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

Protected Member Functions

- **ConvertibleBond** (const boost::shared_ptr< [StochasticProcess](#) > &process, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine, Real conversionRatio, const DividendSchedule ÷nds, const CallabilitySchedule &callability, const [Handle](#)< [Quote](#) > &creditSpread, const [Date](#) &issueDate, Natural settlementDays, const [DayCounter](#) &dayCounter, const [Schedule](#) &schedule, Real redemption)
- void **performCalculations** () const

Protected Attributes

- Real **conversionRatio_**
- CallabilitySchedule **callability_**
- DividendSchedule **dividends_**
- [Handle](#)< [Quote](#) > **creditSpread_**
- boost::shared_ptr< option > **option_**

9.164.2 Member Function Documentation

9.164.2.1 void performCalculations () const [protected, virtual]

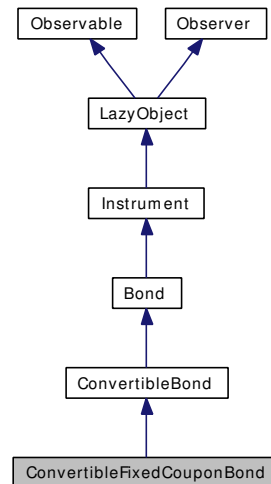
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Bond](#).

9.165 ConvertibleFixedCouponBond Class Reference

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

Inheritance diagram for ConvertibleFixedCouponBond:



9.165.1 Detailed Description

convertible fixed-coupon bond

Warning

Most methods inherited from [Bond](#) (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

Examples:

[ConvertibleBonds.cpp](#).

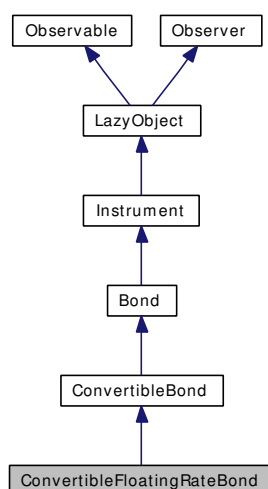
Public Member Functions

- **ConvertibleFixedCouponBond** (const boost::shared_ptr< [StochasticProcess](#) > &process, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine, Real conversionRatio, const DividendSchedule ÷nds, const CallabilitySchedule &callability, const [Handle](#)< [Quote](#) > &creditSpread, const [Date](#) &issueDate, Natural settlementDays, const std::vector< Rate > &coupons, const [DayCounter](#) &dayCounter, const [Schedule](#) &schedule, Real redemption=100)

9.166 ConvertibleFloatingRateBond Class Reference

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

Inheritance diagram for ConvertibleFloatingRateBond:



9.166.1 Detailed Description

convertible floating-rate bond

Warning

Most methods inherited from [Bond](#) (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

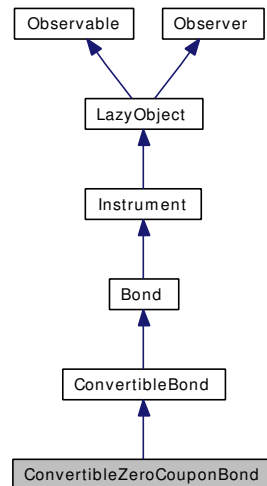
Public Member Functions

- **ConvertibleFloatingRateBond** (const boost::shared_ptr< [StochasticProcess](#) > &process, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine, Real conversionRatio, const DividendSchedule ÷nds, const CallabilitySchedule &callability, const [Handle](#)< [Quote](#) > &creditSpread, const [Date](#) &issueDate, Natural settlementDays, const boost::shared_ptr< [IborIndex](#) > &index, Natural fixingDays, const std::vector< Spread > &spreads, const [DayCounter](#) &dayCounter, const [Schedule](#) &schedule, Real redemption=100)

9.167 ConvertibleZeroCouponBond Class Reference

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

Inheritance diagram for ConvertibleZeroCouponBond:



9.167.1 Detailed Description

convertible zero-coupon bond

Warning

Most methods inherited from [Bond](#) (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

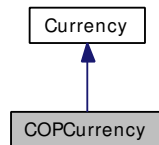
Public Member Functions

- **ConvertibleZeroCouponBond** (const boost::shared_ptr< [StochasticProcess](#) > &process, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine, Real conversionRatio, const DividendSchedule ÷nds, const CallabilitySchedule &callability, const [Handle](#)< [Quote](#) > &creditSpread, const [Date](#) &issueDate, Natural settlementDays, const [DayCounter](#) &dayCounter, const [Schedule](#) &schedule, Real redemption=100)

9.168 COPCurrency Class Reference

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

Inheritance diagram for COPCurrency:



9.168.1 Detailed Description

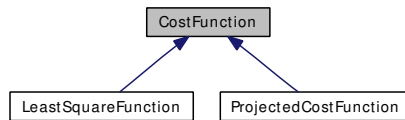
Colombian peso.

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

9.169 CostFunction Class Reference

```
#include <ql/math/optimization/costfunction.hpp>
```

Inheritance diagram for CostFunction:



9.169.1 Detailed Description

Cost function abstract class for optimization problem.

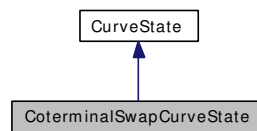
Public Member Functions

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

9.170 CoterminSwapCurveState Class Reference

```
#include <ql/models/marketmodels/curvestates/coterminswapcurvestate.hpp>
```

Inheritance diagram for CoterminSwapCurveState:



9.170.1 Detailed Description

Curve state for coterminal-swap market models

This class stores the state of the yield curve associated to the fixed calendar times within the simulation. This is the workhorse discounting object associated to the rate times of the simulation. It's important to pass the rates via an object like this to the product rather than directly to make it easier to switch to other engines such as a coterminal swap rate engine. Many products will not need expired rates and others will only require the first rate.

Public Member Functions

- **CoterminSwapCurveState** (const std::vector< Time > &rateTimes)
- std::auto_ptr< [CurveState](#) > **clone** () const

Modifiers

- void **setOnCoterminSwapRates** (const std::vector< Rate > &swapRates, Size firstValidIndex=0)

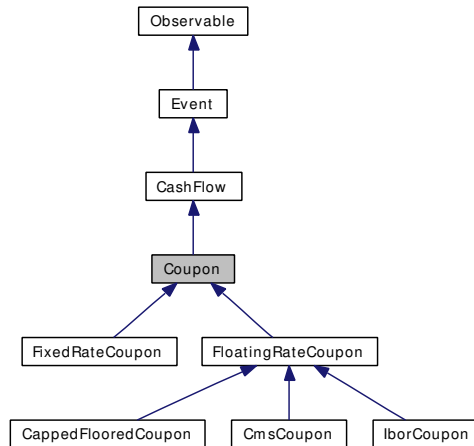
Inspectors

- Real **discountRatio** (Size i, Size j) const
- Rate **forwardRate** (Size i) const
- Rate **coterminSwapRate** (Size i) const
- Rate **coterminSwapAnnuity** (Size numeraire, Size i) const
- Rate **cmSwapRate** (Size i, Size spanningForwards) const
- Rate **cmSwapAnnuity** (Size numeraire, Size i, Size spanningForwards) const
- const std::vector< Rate > & **forwardRates** () const
- const std::vector< Rate > & **coterminSwapRates** () const
- const std::vector< Rate > & **cmSwapRates** (Size spanningForwards) const

9.171 Coupon Class Reference

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

Inheritance diagram for Coupon:



9.171.1 Detailed Description

coupon accruing over a fixed period

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

Public Member Functions

- [Coupon](#) (Real nominal, const [Date](#) &paymentDate, const [Date](#) &accrualStartDate, const [Date](#) &accrualEndDate, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)())

Partial CashFlow interface

- [Date](#) [date](#) () const

Note:

This is inherited from the event class

Inspectors

- Real **nominal** () const
- const [Date](#) & [accrualStartDate](#) () const
start of the accrual period
- const [Date](#) & [accrualEndDate](#) () const
end of the accrual period
- const [Date](#) & [referencePeriodStart](#) () const
start date of the reference period

- const [Date](#) & [referencePeriodEnd](#) () const
end date of the reference 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_](#)

9.171.2 Constructor & Destructor Documentation

- 9.171.2.1 [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.

9.172 CovarianceDecomposition Class Reference

```
#include <ql/math/matrixutilities/getcovariance.hpp>
```

9.172.1 Detailed Description

Covariance decomposition into correlation and variances.

Extracts the correlation matrix and the vector of variances 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

9.172.2 Constructor & Destructor Documentation

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

Precondition:

covarianceMatrix must be symmetric

9.172.3 Member Function Documentation

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

returns the variances [Array](#)

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

returns the standard deviations [Array](#)

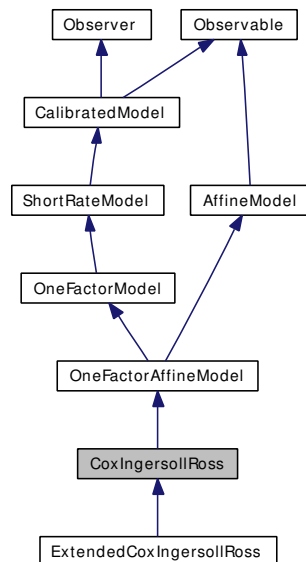
9.172.3.3 const [Matrix](#)& [correlationMatrix](#) () const

returns the correlation matrix

9.173 CoxIngersollRoss Class Reference

```
#include <ql/models/shortrate/onefactormodels/coxingersollross.hpp>
```

Inheritance diagram for CoxIngersollRoss:



9.173.1 Detailed Description

Cox-Ingersoll-Ross model class.

This class implements the Cox-Ingersoll-Ross model defined by

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

Bug

this class was not tested enough to guarantee its functionality.

Public Member Functions

- **CoxIngersollRoss** (Rate `r0=0.05`, Real `theta=0.1`, Real `k=0.1`, Real `sigma=0.1`)
- virtual Real **discountBondOption** (Option::Type `type`, Real `strike`, Time `maturity`, Time `bondMaturity`) const
- virtual boost::shared_ptr< ShortRateDynamics > **dynamics** () const
returns the short-rate dynamics
- boost::shared_ptr< 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

Classes

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

9.174 CoxIngersollRoss::Dynamics Class Reference

```
#include <ql/models/shortrate/onefactormodels/coxingersollross.hpp>
```

9.174.1 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

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

.

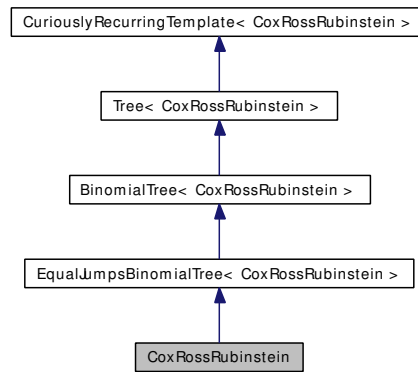
Public Member Functions

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

9.175 CoxRossRubinstein Class Reference

```
#include <ql/methods/lattices/binomialtree.hpp>
```

Inheritance diagram for CoxRossRubinstein:



9.175.1 Detailed Description

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

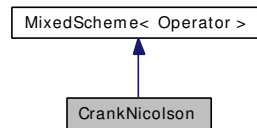
Public Member Functions

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

9.176 CrankNicolson Class Template Reference

```
#include <ql/methods/finitedifferences/cranknicolson.hpp>
```

Inheritance diagram for CrankNicolson:



9.176.1 Detailed Description

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

Crank-Nicolson scheme for finite difference methods.

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

```

typedef ... array_type;

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

// inspectors
Size size();

// modifiers
void setTime(Time t);

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

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

```

Warning

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

Public Types

- `typedef OperatorTraits< Operator > traits`
- `typedef traits::operator_type operator_type`
- `typedef traits::array_type array_type`
- `typedef traits::bc_set bc_set`
- `typedef traits::condition_type condition_type`

Public Member Functions

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

9.177 Cubic Class Reference

```
#include <ql/math/interpolations/cubicspline.hpp>
```

9.177.1 Detailed Description

cubic-spline interpolation factory and traits

Public Types

- enum { **global** = 1 }

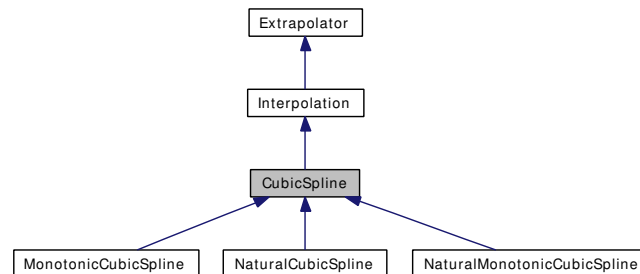
Public Member Functions

- **Cubic** ([CubicSpline::BoundaryCondition](#) leftCondition=CubicSpline::SecondDerivative, Real leftConditionValue=0.0, [CubicSpline::BoundaryCondition](#) rightCondition=CubicSpline::SecondDerivative, Real rightConditionValue=0.0, bool monotonicityConstraint=false)
- template<class I1, class I2>
[Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

9.178 CubicSpline Class Reference

```
#include <ql/math/interpolations/cubicspline.hpp>
```

Inheritance diagram for CubicSpline:



9.178.1 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 monotonicityConstraint)
- const std::vector< Real > & **aCoefficients** () const
- const std::vector< Real > & **bCoefficients** () const
- const std::vector< Real > & **cCoefficients** () const

9.178.2 Member Enumeration Documentation

9.178.2.1 enum BoundaryCondition

Enumerator:

NotAKnot Make second(-last) point an inactive knot.

FirstDerivative Match value of end-slope.

SecondDerivative Match value of second derivative at end.

Periodic Match first and second derivative at either end.

Lagrange Match end-slope to the slope of the cubic that matches the first four data at the respective end

9.178.3 Constructor & Destructor Documentation

9.178.3.1 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.

9.179 CumulativeBinomialDistribution Class Reference

```
#include <ql/math/distributions/binomialdistribution.hpp>
```

9.179.1 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

9.180 CumulativeNormalDistribution Class Reference

```
#include <ql/math/distributions/normaldistribution.hpp>
```

9.180.1 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

9.181 CumulativePoissonDistribution Class Reference

```
#include <ql/math/distributions/poissondistribution.hpp>
```

9.181.1 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.

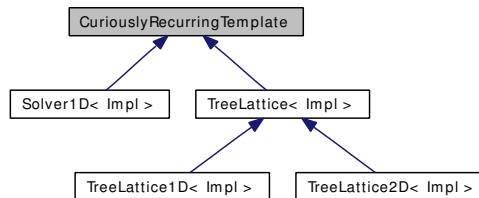
Public Member Functions

- **CumulativePoissonDistribution** (Real mu)
- Real **operator()** (BigNatural k) const

9.182 CuriouslyRecurringTemplate Class Template Reference

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

Inheritance diagram for CuriouslyRecurringTemplate:



9.182.1 Detailed Description

template<class Impl> class QuantLib::CuriouslyRecurringTemplate< Impl >

Support for the curiously recurring template pattern.

See James O. Coplien. A Curiously Recurring Template Pattern. In Stanley B. Lippman, editor, C++ Gems, 135-144. Cambridge University Press, New York, New York, 1996.

Protected Member Functions

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

9.183 Currency Class Reference

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

Inheritance diagram for Currency:



9.183.1 Detailed Description

Currency specification

Public Member Functions

- [Currency](#) ()
default constructor

Inspectors

- const std::string & [name](#) () const
currency name, e.g., "U.S. Dollar"
- const std::string & [code](#) () const
ISO 4217 three-letter code, e.g., "USD".
- Integer [numericCode](#) () const
ISO 4217 numeric code, e.g., "840".
- const std::string & [symbol](#) () const
symbol, e.g., "\$"
- const std::string & [fractionSymbol](#) () const
fraction symbol, e.g., "¢"
- Integer [fractionsPerUnit](#) () const
number of fractionary parts in a unit, e.g., 100
- const [Rounding](#) & [rounding](#) () const
rounding convention
- std::string [format](#) () const
output format

Other information

- bool [empty](#) () const
is this a usable instance?
- const [Currency](#) & [triangulationCurrency](#) () const
currency used for triangulated exchange when required

Protected Attributes

- `boost::shared_ptr< Data > data_`

Related Functions

(Note that these are not member functions.)

- `bool operator==` (`const Currency &`, `const Currency &`)
- `bool operator!=` (`const Currency &`, `const Currency &`)
- `std::ostream & operator<<` (`std::ostream &`, `const Currency &`)

9.183.2 Constructor & Destructor Documentation

9.183.2.1 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.

9.183.3 Member Function Documentation

9.183.3.1 std::string format () const

output format

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

9.183.4 Friends And Related Function Documentation

9.183.4.1 `bool operator==` (`const Currency &`, `const Currency &`) [related]

9.183.4.2 `bool operator!=` (`const Currency &`, `const Currency &`) [related]

9.183.4.3 `std::ostream & operator<<` (`std::ostream &`, `const Currency &`) [related]

9.184 Curve Class Reference

```
#include <ql/math/curve.hpp>
```

9.184.1 Detailed Description

abstract curve class

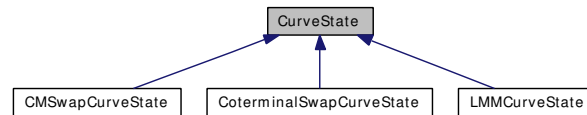
Public Member Functions

- virtual Real **operator()** (Real x) const=0

9.185 CurveState Class Reference

```
#include <ql/models/marketmodels/curvestate.hpp>
```

Inheritance diagram for CurveState:



9.185.1 Detailed Description

Curve state for market-model simulations

This class stores the state of the yield curve associated to the fixed calendar times within the simulation. This is the workhorse discounting object associated to the rate times of the simulation. It's important to pass the rates via an object like this to the product rather than directly to make it easier to switch to other engines such as a coterminal swap rate engine. Many products will not need expired rates and others will only require the first rate.

Public Member Functions

- **CurveState** (const std::vector< Time > &rateTimes)

Inspectors

- Size **numberOfRates** () const
- const std::vector< Time > & **rateTimes** () const
- const std::vector< Time > & **rateTaus** () const
- virtual Real **discountRatio** (Size i, Size j) const=0
- virtual Rate **forwardRate** (Size i) const=0
- virtual Rate **coterminalSwapAnnuity** (Size numeraire, Size i) const=0
- virtual Rate **coterminalSwapRate** (Size i) const=0
- virtual Rate **cmSwapAnnuity** (Size numeraire, Size i, Size spanningForwards) const=0
- virtual Rate **cmSwapRate** (Size i, Size spanningForwards) const =0
- virtual const std::vector< Rate > & **forwardRates** () const=0
- virtual const std::vector< Rate > & **coterminalSwapRates** () const=0
- virtual const std::vector< Rate > & **cmSwapRates** (Size spanningForwards) const=0
- Rate **swapRate** (Size begin, Size end) const
- virtual std::auto_ptr< [CurveState](#) > **clone** () const=0

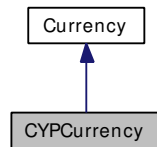
Protected Attributes

- Size **numberOfRates_**
- std::vector< Time > **rateTimes_**
- std::vector< Time > **rateTaus_**

9.186 CYPCurrency Class Reference

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

Inheritance diagram for CYPCurrency:



9.186.1 Detailed Description

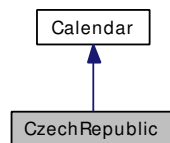
Cyprus pound.

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

9.187 CzechRepublic Class Reference

```
#include <ql/time/calendars/czechrepublic.hpp>
```

Inheritance diagram for CzechRepublic:



9.187.1 Detailed Description

Czech calendars.

Holidays for the Prague stock exchange (see <http://www.pse.cz/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Easter Monday
- Labour Day, May 1st
- Liberation Day, May 8th
- SS. Cyril and Methodius, July 5th
- Jan Hus Day, July 6th
- Czech Statehood Day, September 28th
- Independence Day, October 28th
- Struggle for Freedom and Democracy Day, November 17th
- Christmas Eve, December 24th
- Christmas, December 25th
- St. Stephen, December 26th

Public Types

- enum `Market` { `PSE` }

Public Member Functions

- `CzechRepublic` (`Market` m=PSE)

9.187.2 Member Enumeration Documentation

9.187.2.1 enum Market

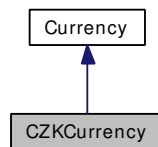
Enumerator:

PSE Prague stock exchange.

9.188 CZKCurrency Class Reference

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

Inheritance diagram for CZKCurrency:



9.188.1 Detailed Description

Czech koruna.

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

9.189 Date Class Reference

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

9.189.1 Detailed Description

Concrete date class.

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

Tests

self-consistency of dates, serial numbers, days of month, months, and weekdays is checked over the whole date range.

Examples:

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

Public Member Functions

constructors

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

inspectors

- [Weekday](#) weekday () const
- Day [dayOfMonth](#) () const
- Day [dayOfYear](#) () const
One-based (Jan 1st = 1).
- [Month](#) month () const
- Year [year](#) () const
- BigInteger [serialNumber](#) () const

date algebra

- [Date](#) & [operator+=](#) (BigInteger days)
increments date by the given number of days
- [Date](#) & [operator+=](#) (const [Period](#) &)
increments date by the given period

- `Date & operator-=` (BigInteger days)
decrement date by the given number of days
- `Date & operator-=` (const `Period` &)
decrements date by the given period
- `Date & operator++` ()
1-day pre-increment
- `Date operator++` (int)
1-day post-increment
- `Date & operator--` ()
1-day pre-decrement
- `Date operator--` (int)
1-day post-decrement
- `Date operator+` (BigInteger days) const
returns a new date incremented by the given number of days
- `Date operator+` (const `Period` &) const
returns a new date incremented by the given period
- `Date operator-` (BigInteger days) const
returns a new date decremented by the given number of days
- `Date operator-` (const `Period` &) const
returns a new date decremented by the given period

Static Public Member Functions

static methods

- static `Date todaysDate` ()
today's date.
- static `Date minDate` ()
earliest allowed date
- static `Date maxDate` ()
latest allowed date
- static bool `isLeap` (Year y)
whether the given year is a leap one
- static `Date endOfMonth` (const `Date` &d)
last day of the month to which the given date belongs
- static bool `isEndOfMonth` (const `Date` &d)

whether a date is the last day of its month

- static `Date nextWeekday` (const `Date` &*d*, `Weekday`)
next given weekday following or equal to the given date
- static `Date nthWeekday` (Size *n*, `Weekday`, `Month` *m*, Year *y*)
n-th given weekday in the given month and year

Related Functions

(Note that these are not member functions.)

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

9.189.2 Member Function Documentation

9.189.2.1 static `Date nextWeekday` (const `Date` & *d*, `Weekday`) [static]

next given weekday following or equal to the given date

E.g., the Friday following Tuesday, January 15th, 2002 was January 18th, 2002.

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

9.189.2.2 static `Date nthWeekday` (Size *n*, `Weekday`, `Month` *m*, Year *y*) [static]

n-th given weekday in the given month and year

E.g., the 4th Thursday of March, 1998 was March 26th, 1998.

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

9.189.3 Friends And Related Function Documentation

9.189.3.1 `bool operator==(const Date &, const Date &)` [related]

9.189.3.2 `bool operator!=(const Date &, const Date &)` [related]

9.189.3.3 `bool operator<(const Date &, const Date &)` [related]

9.189.3.4 `bool operator<=(const Date &, const Date &)` [related]

9.189.3.5 `bool operator>(const Date &, const Date &)` [related]

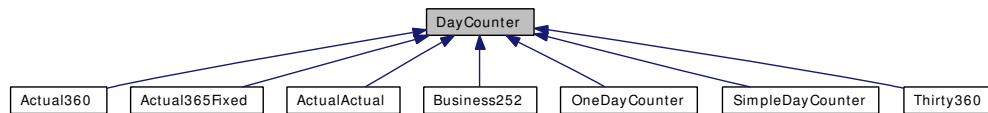
9.189.3.6 `bool operator>=(const Date &, const Date &)` [related]

9.189.3.7 `std::ostream & operator<<(std::ostream &, const Date &)` [related]

9.190 DayCounter Class Reference

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

Inheritance diagram for DayCounter:



9.190.1 Detailed Description

day counter class

This class provides methods for determining the length of a time period according to given market convention, both as a number of days and as a year fraction.

The Bridge pattern is used to provide the base behavior of the day counter.

Examples:

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

Public Member Functions

- [DayCounter](#) ()

DayCounter interface

- bool [empty](#) () const
Returns whether or not the day counter is initialized.
- 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< [Impl](#) > &impl)

Protected Attributes

- boost::shared_ptr< [Impl](#) > [impl_](#)

Related Functions

(Note that these are not member functions.)

- `bool operator==(const DayCounter &, const DayCounter &)`
- `bool operator!=(const DayCounter &, const DayCounter &)`
- `std::ostream & operator<< (std::ostream &, const DayCounter &)`

Classes

- class `Impl`
abstract base class for day counter implementations

9.190.2 Constructor & Destructor Documentation

9.190.2.1 DayCounter (const boost::shared_ptr< Impl > & impl) [protected]

This constructor can be invoked by derived classes which define a given implementation.

9.190.2.2 DayCounter ()

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

9.190.3 Member Function Documentation

9.190.3.1 std::string name () const

Returns the name of the day counter.

Warning

This method is used for output and comparison between day counters. It is **not** meant to be used for writing switch-on-type code.

9.190.4 Friends And Related Function Documentation

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

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

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

9.190.4.3 std::ostream & operator<< (std::ostream &, const DayCounter &) [related]

9.191 DayCounter::Impl Class Reference

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

9.191.1 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](#) &d1, const [Date](#) &d2, const [Date](#) &refPeriodStart, const [Date](#) &refPeriodEnd) const =0

9.192 DecInterpCapletVolStructure Class Reference

```
#include <ql/termstructures/volatilities/capletvolatilitiesstructures.hpp>
```

9.192.1 Detailed Description

this class is interpolating caplets volatilities linealy in two steps (instead of

Public Member Functions

- **DecInterpCapletVolStructure** (const [Date](#) &referenceDate, const [DayCounter](#) dayCounter, const CapMatrix &referenceCaps, const std::vector< Rate > &strikes)
- Volatility [volatilityImpl](#) (Time length, Rate strike) const
implements the actual volatility calculation in derived classes
- void **setClosestTenors** (Time time, Time &nextLowerTenor, Time &nextHigherTenor)
- Time **minTime** () const
- Time **maxTime** () const
the latest time for which the curve can return values
- Real & **volatilityParameter** (Size i, Size j) const
- [Matrix](#) & **volatilityParameters** () const
- void **update** ()

TermStructure interface

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

CapletVolatilityStructure interface

- 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

9.192.2 Member Function Documentation

9.192.2.1 void update () [virtual]

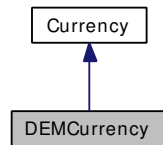
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

9.193 DEMCurrency Class Reference

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

Inheritance diagram for DEMCurrency:



9.193.1 Detailed Description

Deutsche mark.

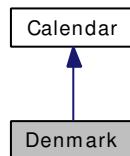
The ISO three-letter code was DEM; the numeric code was 276. It was divided into 100 pfennig.

Obsoleted by the Euro since 1999.

9.194 Denmark Class Reference

```
#include <ql/time/calendars/denmark.hpp>
```

Inheritance diagram for Denmark:



9.194.1 Detailed Description

Danish calendar.

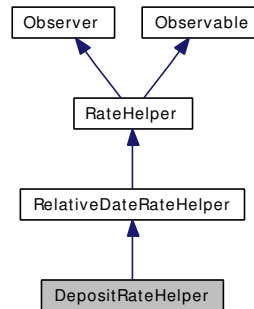
Holidays:

- Saturdays
- Sundays
- Maunday Thursday
- Good Friday
- Easter Monday
- General Prayer Day, 25 days after Easter Monday
- Ascension
- Whit (Pentecost) Monday
- New Year's Day, January 1st
- Constitution Day, June 5th
- Christmas, December 25th
- Boxing Day, December 26th

9.195 DepositRateHelper Class Reference

```
#include <ql/termstructures/yieldcurves/ratehelpers.hpp>
```

Inheritance diagram for DepositRateHelper:



9.195.1 Detailed Description

Rate helper for bootstrapping over deposit rates.

Examples:

[swapvaluation.cpp](#).

Public Member Functions

- **DepositRateHelper** (const [Handle](#)< [Quote](#) > &rate, const [Period](#) &tenor, Natural settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, bool endOfMonth, Natural fixingDays, const [DayCounter](#) &dayCounter)
- **DepositRateHelper** (Rate rate, const [Period](#) &tenor, Natural settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, bool endOfMonth, Natural fixingDays, const [DayCounter](#) &dayCounter)
- Real **impliedQuote** () const
- DiscountFactor **discountGuess** () const
- void **setTermStructure** ([YieldTermStructure](#) *)
sets the term structure to be used for pricing

9.195.2 Member Function Documentation

9.195.2.1 void setTermStructure (YieldTermStructure *) [virtual]

sets the term structure to be used for pricing

Warning

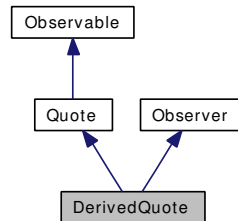
Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the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

Reimplemented from [RateHelper](#).

9.196 DerivedQuote Class Template Reference

```
#include <ql/quotes/derivedquote.hpp>
```

Inheritance diagram for DerivedQuote:



9.196.1 Detailed Description

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

market quote whose value depends on another quote

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](#) ()

9.196.2 Member Function Documentation

9.196.2.1 void update () [virtual]

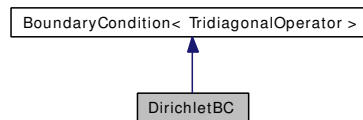
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.197 DirichletBC Class Reference

```
#include <ql/methods/finitedifferences/boundarycondition.hpp>
```

Inheritance diagram for DirichletBC:



9.197.1 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)

9.197.2 Member Function Documentation

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

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

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

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

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

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

9.197.2.4 void applyAfterSolving (Array &) const [virtual]

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

9.197.2.5 void setTime (Time t) [virtual]

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

9.198 Discount Struct Reference

```
#include <ql/termstructures/yieldcurves/bootstraptraits.hpp>
```

9.198.1 Detailed Description

Discount-curve traits.

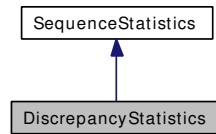
Static Public Member Functions

- static DiscountFactor **initialValue** ()
- static DiscountFactor **initialGuess** ()
- static DiscountFactor **guess** (const [YieldTermStructure](#) *c, const [Date](#) &d)
- static DiscountFactor **minValueAfter** (Size, const std::vector< Real > &)
- static DiscountFactor **maxValueAfter** (Size i, const std::vector< Real > &data)
- static void **updateGuess** (std::vector< DiscountFactor > &data, DiscountFactor discount, Size i)

9.199 DiscrepancyStatistics Class Reference

```
#include <ql/math/statistics/discrepancystatistics.hpp>
```

Inheritance diagram for DiscrepancyStatistics:



9.199.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

It inherit from SequenceStatistics<Statistics> and adds L^2 discrepancy calculation

Public Types

- typedef SequenceStatistics::value_type **value_type**

Public Member Functions

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

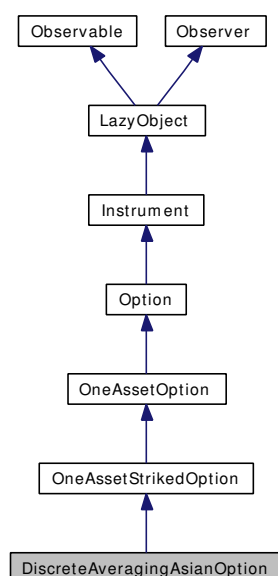
1-dimensional inspectors

- Real **discrepancy** () const

9.200 DiscreteAveragingAsianOption Class Reference

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

Inheritance diagram for DiscreteAveragingAsianOption:



9.200.1 Detailed Description

Discrete-averaging Asian option.

Public Member Functions

- **DiscreteAveragingAsianOption** (Average::Type averageType, Real runningAccumulator, Size pastFixings, std::vector< [Date](#) > fixingDates, const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void **setupArguments** ([PricingEngine::arguments](#) *) const

Protected Attributes

- Average::Type **averageType_**
- Real **runningAccumulator_**
- Size **pastFixings_**
- std::vector< [Date](#) > **fixingDates_**

Classes

- class [arguments](#)

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

- class [engine](#)

Discrete-averaging Asian engine base class.

9.201 DiscreteAveragingAsianOption::arguments Class Reference

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

9.201.1 Detailed Description

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

Public Member Functions

- void **validate** () const

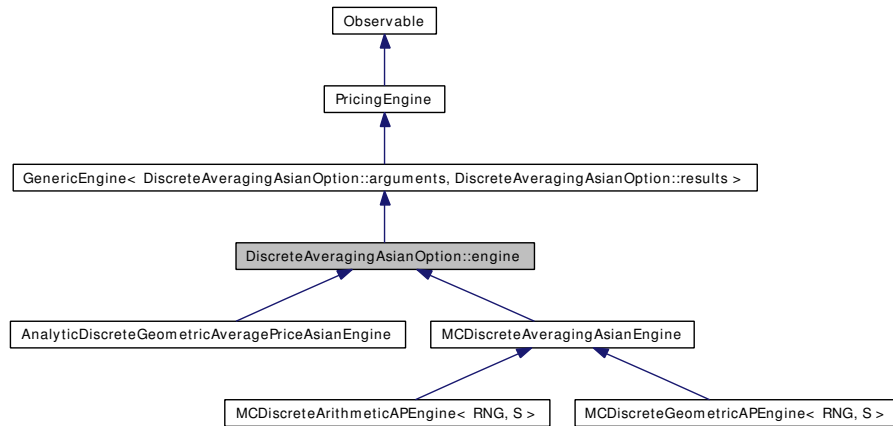
Public Attributes

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

9.202 DiscreteAveragingAsianOption::engine Class Reference

#include <ql/instruments/asianoption.hpp>

Inheritance diagram for DiscreteAveragingAsianOption::engine:



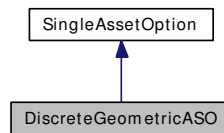
9.202.1 Detailed Description

Discrete-averaging Asian engine base class.

9.203 DiscreteGeometricASO Class Reference

```
#include <ql/legacy/pricers/discretegeometricaso.hpp>
```

Inheritance diagram for DiscreteGeometricASO:



9.203.1 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

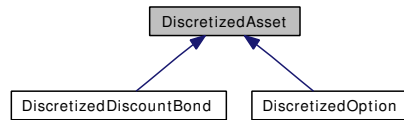
Public Member Functions

- **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

9.204 DiscretizedAsset Class Reference

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

Inheritance diagram for DiscretizedAsset:



9.204.1 Detailed Description

Discretized asset class used by numerical methods.

Public Member Functions

inspectors

- Time **time** () const
- Time & **time** ()
- const [Array](#) & **values** () const
- [Array](#) & **values** ()
- const boost::shared_ptr< [Lattice](#) > & **method** () const

High-level interface

Users of discretized assets should use these methods in order to initialize, evolve and take the present value of the assets. They call the corresponding methods in the Lattice interface, to which we refer for documentation.

- void **initialize** (const boost::shared_ptr< [Lattice](#) > &, Time t)
- void **rollback** (Time to)
- void **partialRollback** (Time to)
- Real **presentValue** ()

Low-level interface

These methods (that developers should override when deriving from DiscretizedAsset) are to be used by numerical methods and not directly by users, with the exception of adjustValues(), preAdjustValues() and postAdjustValues() that can be used together with partialRollback().

- virtual void **reset** (Size size)=0
- void **preAdjustValues** ()
- void **postAdjustValues** ()
- void **adjustValues** ()
- virtual std::vector< Time > **mandatoryTimes** () const=0

Protected Member Functions

- bool **isOnTime** (Time t) const
- virtual void **preAdjustValuesImpl** ()
- virtual void **postAdjustValuesImpl** ()

Protected Attributes

- Time `time_`
- Time `latestPreAdjustment_`
- Time `latestPostAdjustment_`
- [Array](#) `values_`

9.204.2 Member Function Documentation

9.204.2.1 `virtual void reset (Size size)` [pure virtual]

This method should initialize the asset values to an [Array](#) of the given size and with values depending on the particular asset.

Implemented in [DiscretizedDiscountBond](#), and [DiscretizedOption](#).

9.204.2.2 `void preAdjustValues ()`

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

This method is not virtual; derived classes must override the protected [preAdjustValuesImpl\(\)](#) method instead.

9.204.2.3 `void postAdjustValues ()`

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

This method is not virtual; derived classes must override the protected [postAdjustValuesImpl\(\)](#) method instead.

9.204.2.4 `void adjustValues ()`

This method performs both pre- and post-adjustment

9.204.2.5 `virtual std::vector<Time> mandatoryTimes () const` [pure virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

Note:

The returned values are not guaranteed to be sorted.

Implemented in [DiscretizedDiscountBond](#), and [DiscretizedOption](#).

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

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

9.204.2.7 `virtual void preAdjustValuesImpl ()` [protected, virtual]

This method performs the actual pre-adjustment

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

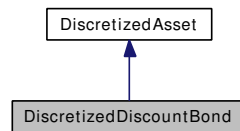
This method performs the actual post-adjustment

Reimplemented in [DiscretizedOption](#).

9.205 DiscretizedDiscountBond Class Reference

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

Inheritance diagram for DiscretizedDiscountBond:



9.205.1 Detailed Description

Useful discretized discount bond asset.

Public Member Functions

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

9.205.2 Member Function Documentation

9.205.2.1 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](#).

9.205.2.2 std::vector<Time> mandatoryTimes () const [virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

Note:

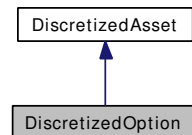
The returned values are not guaranteed to be sorted.

Implements [DiscretizedAsset](#).

9.206 DiscretizedOption Class Reference

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

Inheritance diagram for DiscretizedOption:



9.206.1 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_**

9.206.2 Member Function Documentation

9.206.2.1 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](#).

9.206.2.2 `std::vector< Time > mandatoryTimes () const` [virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

Note:

The returned values are not guaranteed to be sorted.

Implements [DiscretizedAsset](#).

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

This method performs the actual post-adjustment

Reimplemented from [DiscretizedAsset](#).

9.207 Disposable Class Template Reference

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

9.207.1 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;  
}
```

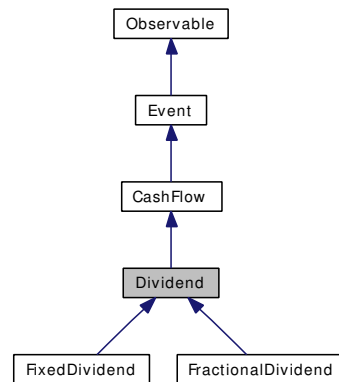
Public Member Functions

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

9.208 Dividend Class Reference

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

Inheritance diagram for Dividend:



9.208.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

Public Member Functions

- **Dividend** (const [Date](#) &date)
- virtual Real **amount** (Real underlying) const=0

CashFlow interface

- virtual [Date](#) **date** () const
Note:
This is inherited from the event class
- virtual Real [amount](#) () const=0
returns the amount of the cash flow

Visitability

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

Protected Attributes

- [Date](#) **date_**

9.208.2 Member Function Documentation

9.208.2.1 `virtual Real amount () const` [pure virtual]

returns the amount of the cash flow

Note:

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

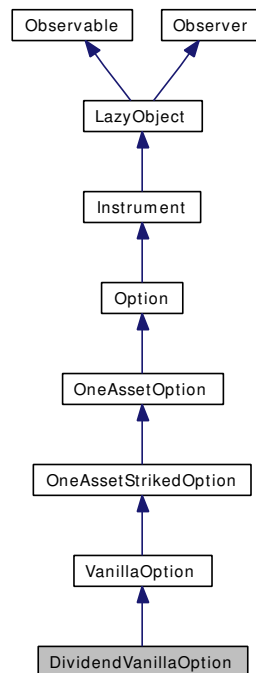
Implements [CashFlow](#).

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

9.209 DividendVanillaOption Class Reference

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

Inheritance diagram for DividendVanillaOption:



9.209.1 Detailed Description

Single-asset vanilla option (no barriers) with discrete dividends.

Public Member Functions

- **DividendVanillaOption** (const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const std::vector< [Date](#) > ÷ndDates, const std::vector< Real > ÷nds, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())

Protected Member Functions

- void **setupArguments** ([PricingEngine::arguments](#) *) const

Classes

- class [arguments](#)
Arguments for dividend vanilla option calculation
- class [engine](#)

Dividend-vanilla-option engine base class

9.210 DividendVanillaOption::arguments Class Reference

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

9.210.1 Detailed Description

Arguments for dividend vanilla option calculation

Public Member Functions

- void **validate** () const

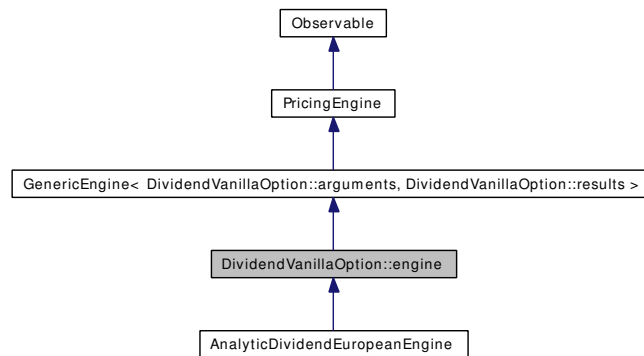
Public Attributes

- DividendSchedule **cashFlow**

9.211 DividendVanillaOption::engine Class Reference

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

Inheritance diagram for DividendVanillaOption::engine:



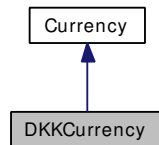
9.211.1 Detailed Description

Dividend-vanilla-option engine base class

9.212 DKKCurrency Class Reference

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

Inheritance diagram for DKKCurrency:



9.212.1 Detailed Description

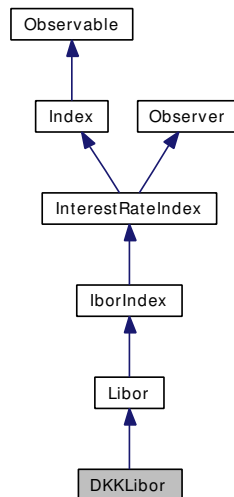
Danish krone.

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

9.213 DKKLibor Class Reference

```
#include <ql/indexes/ibor/dkklbor.hpp>
```

Inheritance diagram for DKKLibor:



9.213.1 Detailed Description

DKK LIBOR rate

Danish Krona LIBOR fixed by BBA.

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

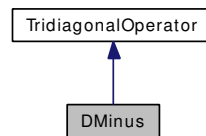
Public Member Functions

- **DKKLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), Natural settlementDays=2)

9.214 DMinus Class Reference

```
#include <ql/methods/finitedifferences/dminus.hpp>
```

Inheritance diagram for DMinus:



9.214.1 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)

9.215 Domain Class Reference

```
#include <ql/math/domain.hpp>
```

9.215.1 Detailed Description

domain abstract lclass

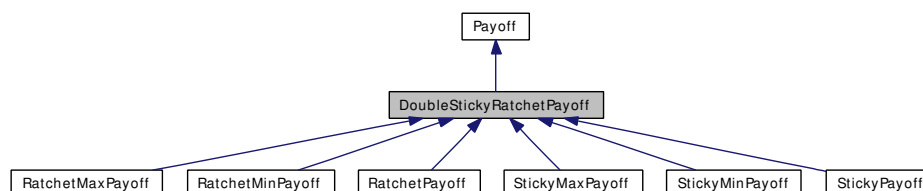
Public Member Functions

- virtual bool **includes** (Real x, Real y) const=0
- bool **operator()** (Real x, Real y)

9.216 DoubleStickyRatchetPayoff Class Reference

```
#include <ql/instruments/stickyratchet.hpp>
```

Inheritance diagram for DoubleStickyRatchetPayoff:



9.216.1 Detailed Description

Intermediate class for single/double sticky/ratchet payoffs.

Public Member Functions

- **DoubleStickyRatchetPayoff** (Real type1, Real type2, Real gearing1, Real gearing2, Real gearing3, Real spread1, Real spread2, Real spread3, Real initialValue1, Real initialValue2, Real accrualFactor)

Payoff interface

- std::string **name** () const
- Real **operator()** (Real forward) const
- std::string **description** () const
- virtual void **accept** (AcyclicVisitor &)

Protected Attributes

- Real **type1_**
- Real **type2_**
- Real **gearing1_**
- Real **gearing2_**
- Real **gearing3_**
- Real **spread1_**
- Real **spread2_**
- Real **spread3_**
- Real **initialValue1_**
- Real **initialValue2_**
- Real **accrualFactor_**

9.216.2 Member Function Documentation

9.216.2.1 `std::string name () const` [virtual]

Warning

This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

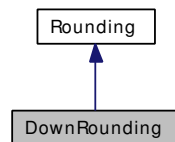
Implements [Payoff](#).

Reimplemented in [RatchetPayoff](#), [StickyPayoff](#), [RatchetMaxPayoff](#), [RatchetMinPayoff](#), [StickyMaxPayoff](#), and [StickyMinPayoff](#).

9.217 DownRounding Class Reference

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

Inheritance diagram for DownRounding:



9.217.1 Detailed Description

Down-rounding.

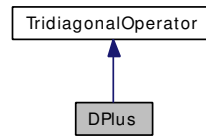
Public Member Functions

- **DownRounding** (Integer precision, Integer digit=5)

9.218 DPlus Class Reference

```
#include <ql/methods/finitedifferences/dplus.hpp>
```

Inheritance diagram for DPlus:



9.218.1 Detailed Description

D_+ matricial representation

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

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

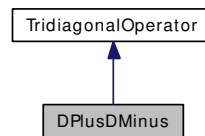
Public Member Functions

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

9.219 DPlusDMinus Class Reference

```
#include <ql/methods/finitedifferences/dplusplusminus.hpp>
```

Inheritance diagram for DPlusDMinus:



9.219.1 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.

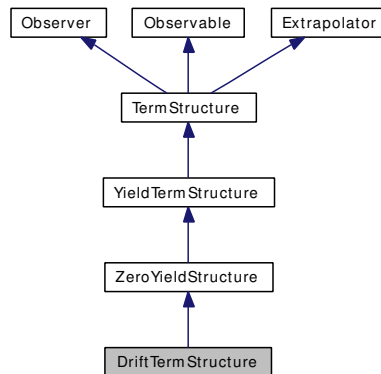
Public Member Functions

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

9.220 DriftTermStructure Class Reference

```
#include <ql/termstructures/yieldcurves/driftermstructure.hpp>
```

Inheritance diagram for DriftTermStructure:



9.220.1 Detailed Description

Drift term structure.

Drift term structure for modelling the common drift term: $\text{riskFreeRate} - \text{dividendYield} - 0.5 \cdot \text{vol} \cdot \text{vol}$

Note:

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

Public Member Functions

- **DriftTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > ÷ndTS, const [Handle](#)< [BlackVolTermStructure](#) > &blackVolTS)

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- const [Date](#) & [referenceDate](#) () const
the date at which discount = 1.0 and/or variance = 0.0
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return values

Protected Member Functions

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

9.221 Duration Struct Reference

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

9.221.1 Detailed Description

duration type

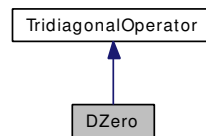
Public Types

- enum Type { Simple, Macaulay, Modified }

9.222 DZero Class Reference

```
#include <ql/methods/finitedifferences/dzero.hpp>
```

Inheritance diagram for DZero:



9.222.1 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.

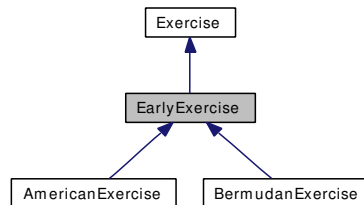
Public Member Functions

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

9.223 EarlyExercise Class Reference

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

Inheritance diagram for EarlyExercise:



9.223.1 Detailed Description

Early-exercise base class.

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

Public Member Functions

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

9.224 EarlyExercisePathPricer Class Template Reference

```
#include <ql/methods/montecarlo/earlyexercisepathpricer.hpp>
```

9.224.1 Detailed Description

```
template<class PathType, class TimeType = Size, class ValueType = Real> class  
QuantLib::EarlyExercisePathPricer< PathType, TimeType, ValueType >
```

base class for early exercise path pricers

Returns the value of an option on a given path and given time.

Public Types

- typedef EarlyExerciseTraits< PathType >::StateType **StateType**

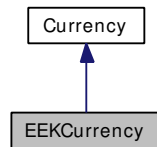
Public Member Functions

- virtual ValueType **operator()** (const PathType &path, TimeType t) const=0
- virtual StateType **state** (const PathType &path, TimeType t) const=0
- virtual std::vector< boost::function1< ValueType, StateType > > **basisSystem** () const=0

9.225 EEKCurrency Class Reference

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

Inheritance diagram for EEKCurrency:



9.225.1 Detailed Description

Estonian kroon.

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

9.226 EndCriteria Class Reference

```
#include <ql/math/optimization/endcriteria.hpp>
```

9.226.1 Detailed Description

Criteria to end optimization process:.

- maximum number of iterations AND minimum number of iterations around stationary point
 - x (independent variable) stationary point
 - $y=f(x)$ (dependent variable) stationary point
 - stationary gradient

Examples:

[BermudanSwaption.cpp](#).

Public Types

- enum `Type` {
None, **MaxIterations**, **StationaryPoint**, **StationaryFunctionValue**,
StationaryFunctionAccuracy, **ZeroGradientNorm**, **Unknown** }

Public Member Functions

- [EndCriteria](#) (Size **maxIterations**, Size **maxStationaryStateIterations**, Real **rootEpsilon**, Real **functionEpsilon**, Real **gradientNormEpsilon**)
Initialization constructor.
- Size **maxIterations** () const
- Size **maxStationaryStateIterations** () const
- Real **rootEpsilon** () const
- Real **functionEpsilon** () const
- Real **gradientNormEpsilon** () const
- bool [operator\(\)](#) (const Size iteration, Size &statState, const bool positiveOptimization, const Real fold, const Real normgold, const Real fnew, const Real normgnew, EndCriteria::Type &ecType) const
- bool [checkMaxIterations](#) (const Size iteration, EndCriteria::Type &ecType) const
- bool [checkStationaryPoint](#) (const Real xOld, const Real xNew, Size &statStateIterations, EndCriteria::Type &ecType) const
- bool [checkStationaryFunctionValue](#) (const Real fxOld, const Real fxNew, Size &statStateIterations, EndCriteria::Type &ecType) const
- bool [checkStationaryFunctionAccuracy](#) (const Real f, const bool positiveOptimization, EndCriteria::Type &ecType) const
- bool [checkZeroGradientNorm](#) (const Real gNorm, EndCriteria::Type &ecType) const

Protected Attributes

- Size `maxIterations_`
Maximum number of iterations.
- Size `maxStationaryStateIterations_`
Maximum number of iterations in stationary state.
- Real `rootEpsilon_`
root, function and gradient epsilons
- Real `functionEpsilon_`
- Real `gradientNormEpsilon_`

9.226.2 Member Function Documentation

9.226.2.1 `bool operator() (const Size iteration, Size & statState, const bool positiveOptimization, const Real fold, const Real normgold, const Real fnew, const Real normgnew, EndCriteria::Type & ecType) const`

Test if the number of iterations is not too big and if a minimum point is not reached

9.226.2.2 `bool checkMaxIterations (const Size iteration, EndCriteria::Type & ecType) const`

Test if the number of iteration is below MaxIterations

9.226.2.3 `bool checkStationaryPoint (const Real xOld, const Real xNew, Size & statStateIterations, EndCriteria::Type & ecType) const`

Test if the root variation is below rootEpsilon

9.226.2.4 `bool checkStationaryFunctionValue (const Real fxOld, const Real fxNew, Size & statStateIterations, EndCriteria::Type & ecType) const`

Test if the function variation is below functionEpsilon

9.226.2.5 `bool checkStationaryFunctionAccuracy (const Real f, const bool positiveOptimization, EndCriteria::Type & ecType) const`

Test if the function value is below functionEpsilon

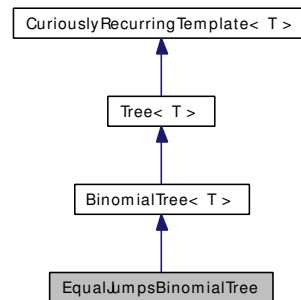
9.226.2.6 `bool checkZeroGradientNorm (const Real gNorm, EndCriteria::Type & ecType) const`

Test if the gradient norm value is below gradientNormEpsilon

9.227 EqualJumpsBinomialTree Class Template Reference

```
#include <ql/methods/lattices/binomialtree.hpp>
```

Inheritance diagram for EqualJumpsBinomialTree:



9.227.1 Detailed Description

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

Base class for equal jumps binomial tree.

Public Member Functions

- **EqualJumpsBinomialTree** (const boost::shared_ptr< [StochasticProcess1D](#) > &process, Time end, Size steps)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size branch) const

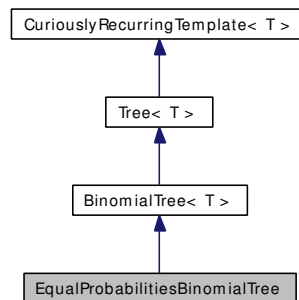
Protected Attributes

- Real **dx_**
- Real **pu_**
- Real **pd_**

9.228 EqualProbabilitiesBinomialTree Class Template Reference

```
#include <ql/methods/lattices/binomialtree.hpp>
```

Inheritance diagram for EqualProbabilitiesBinomialTree:



9.228.1 Detailed Description

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

Base class for equal probabilities binomial tree.

Public Member Functions

- **EqualProbabilitiesBinomialTree** (const boost::shared_ptr< [StochasticProcess1D](#) > &process, Time end, Size steps)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size) const

Protected Attributes

- Real **up_**

9.229 Error Class Reference

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

9.229.1 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.

9.229.2 Constructor & Destructor Documentation

9.229.2.1 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.

9.229.2.2 ~Error () throw ()

the automatically generated destructor would not have the throw specifier.

9.230 ErrorFunction Class Reference

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

9.230.1 Detailed Description

Error function

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

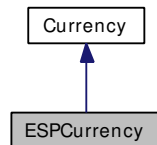
Public Member Functions

- Real **operator()** (Real x) const

9.231 ESPCurrency Class Reference

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

Inheritance diagram for ESPCurrency:



9.231.1 Detailed Description

Spanish peseta.

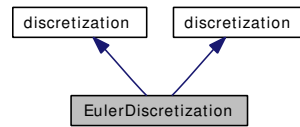
The ISO three-letter code was ESP; the numeric code was 724. It was divided in 100 centimos.

Obsoleted by the Euro since 1999.

9.232 EulerDiscretization Class Reference

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

Inheritance diagram for EulerDiscretization:



9.232.1 Detailed Description

Euler discretization for stochastic processes.

Public Member Functions

- `Disposable< Array > drift` (const [StochasticProcess](#) &, Time *t0*, const [Array](#) &*x0*, Time *dt*) const
- `Real drift` (const [StochasticProcess1D](#) &, Time *t0*, Real *x0*, Time *dt*) const
- `Disposable< Matrix > diffusion` (const [StochasticProcess](#) &, Time *t0*, const [Array](#) &*x0*, Time *dt*) const
- `Real diffusion` (const [StochasticProcess1D](#) &, Time *t0*, Real *x0*, Time *dt*) const
- `Disposable< Matrix > covariance` (const [StochasticProcess](#) &, Time *t0*, const [Array](#) &*x0*, Time *dt*) const
- `Real variance` (const [StochasticProcess1D](#) &, Time *t0*, Real *x0*, Time *dt*) const

9.232.2 Member Function Documentation

9.232.2.1 `Disposable<Array> drift` (const [StochasticProcess](#) &, Time *t0*, const [Array](#) & *x0*, Time *dt*) const [virtual]

Returns an approximation of the drift defined as $\mu(t_0, x_0)\Delta t$.

Implements [StochasticProcess::discretization](#).

9.232.2.2 `Real drift` (const [StochasticProcess1D](#) &, Time *t0*, Real *x0*, Time *dt*) const [virtual]

Returns an approximation of the drift defined as $\mu(t_0, x_0)\Delta t$.

Implements [StochasticProcess1D::discretization](#).

9.232.2.3 `Disposable<Matrix> diffusion` (const [StochasticProcess](#) &, Time *t0*, const [Array](#) & *x0*, Time *dt*) const [virtual]

Returns an approximation of the diffusion defined as $\sigma(t_0, x_0) \sqrt{\Delta t}$.

Implements [StochasticProcess::discretization](#).

9.232.2.4 Real diffusion (const StochasticProcess1D &, Time t_0 , Real x_0 , Time dt) const [virtual]

Returns an approximation of the diffusion defined as $\sigma(t_0, x_0) \sqrt{\Delta t}$.

Implements [StochasticProcess1D::discretization](#).

9.232.2.5 Disposable<Matrix> covariance (const StochasticProcess &, Time t_0 , const Array & x_0 , Time dt) const [virtual]

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

Implements [StochasticProcess::discretization](#).

9.232.2.6 Real variance (const StochasticProcess1D &, Time t_0 , Real x_0 , Time dt) const [virtual]

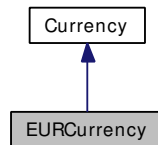
Returns an approximation of the variance defined as $\sigma(t_0, x_0)^2 \Delta t$.

Implements [StochasticProcess1D::discretization](#).

9.233 EURCurrency Class Reference

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

Inheritance diagram for EURCurrency:



9.233.1 Detailed Description

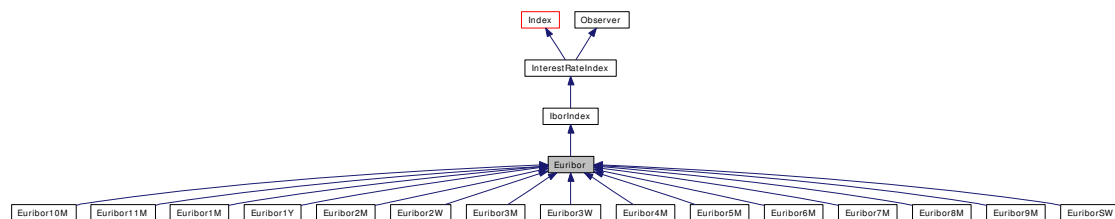
European Euro.

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

9.234 Euribor Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor:



9.234.1 Detailed Description

Euribor index

Euribor rate fixed by the ECB.

Warning

This is the rate fixed by the ECB. Use **EurLibor** if you're interested in the London fixing by BBA.

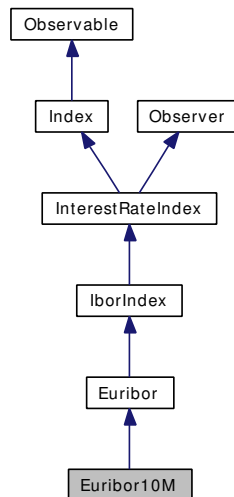
Public Member Functions

- **Euribor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.235 Euribor10M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor10M:



9.235.1 Detailed Description

10-months Euribor index

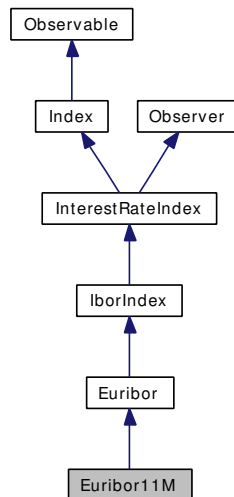
Public Member Functions

- **Euribor10M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.236 Euribor11M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor11M:



9.236.1 Detailed Description

11-months Euribor index

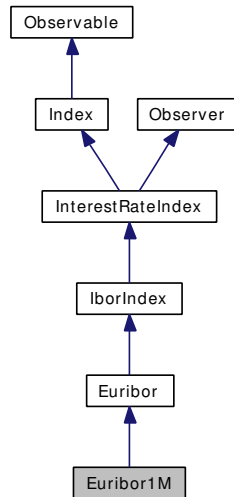
Public Member Functions

- **Euribor11M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.237 Euribor1M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor1M:



9.237.1 Detailed Description

1-month Euribor index

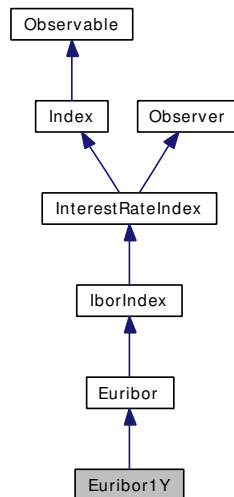
Public Member Functions

- **Euribor1M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.238 Euribor1Y Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor1Y:



9.238.1 Detailed Description

1-year Euribor index

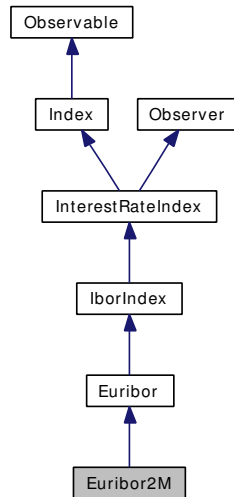
Public Member Functions

- **Euribor1Y** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.239 Euribor2M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor2M:



9.239.1 Detailed Description

2-months Euribor index

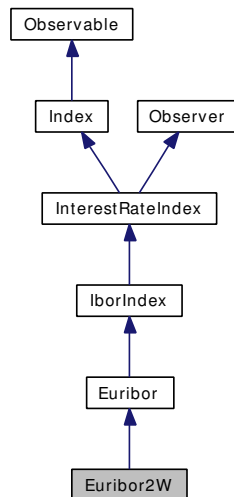
Public Member Functions

- **Euribor2M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.240 Euribor2W Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor2W:



9.240.1 Detailed Description

2-weeks Euribor index

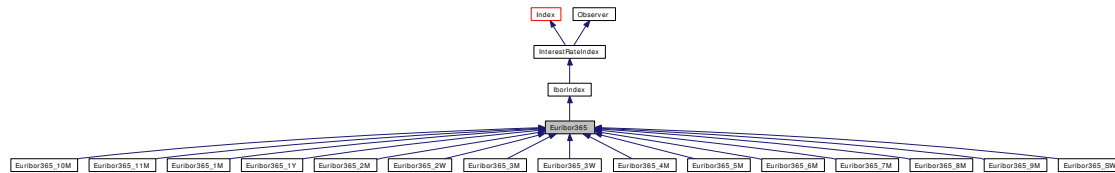
Public Member Functions

- **Euribor2W** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.241 Euribor365 Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365:



9.241.1 Detailed Description

Actual/365 Euribor index.

[Euribor](#) rate adjusted for the mismatch between the actual/360 convention used for [Euribor](#) and the actual/365 convention previously used by a few pre-EUR currencies.

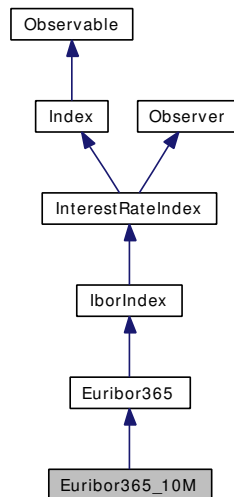
Public Member Functions

- **Euribor365** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.242 Euribor365_10M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_10M:



9.242.1 Detailed Description

10-months Euribor365 index

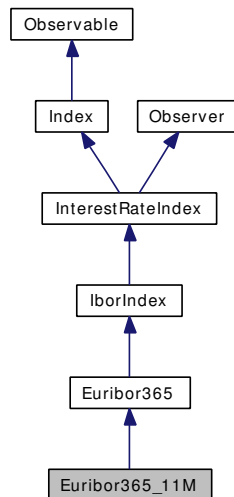
Public Member Functions

- **Euribor365_10M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.243 Euribor365_11M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_11M:



9.243.1 Detailed Description

11-months Euribor365 index

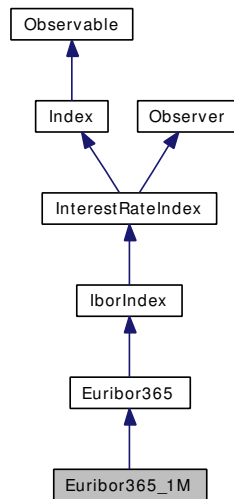
Public Member Functions

- **Euribor365_11M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.244 Euribor365_1M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_1M:



9.244.1 Detailed Description

1-month Euribor365 index

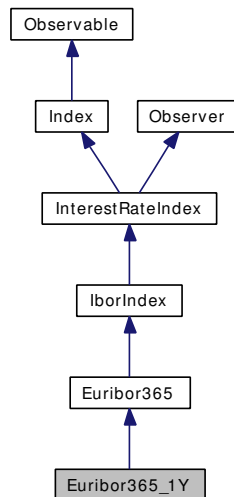
Public Member Functions

- **Euribor365_1M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.245 Euribor365_1Y Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_1Y:



9.245.1 Detailed Description

1-year Euribor365 index

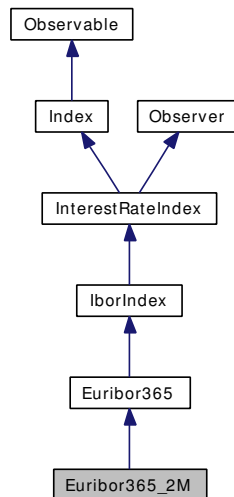
Public Member Functions

- `Euribor365_1Y(const Handle<YieldTermStructure> &h=Handle<YieldTermStructure>())`

9.246 Euribor365_2M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_2M:



9.246.1 Detailed Description

2-months Euribor365 index

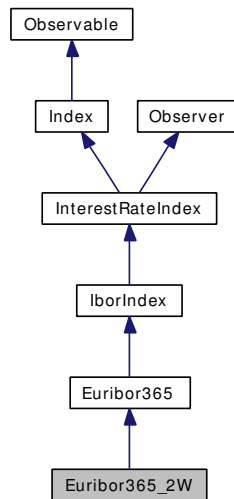
Public Member Functions

- **Euribor365_2M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.247 Euribor365_2W Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_2W:



9.247.1 Detailed Description

2-weeks Euribor365 index

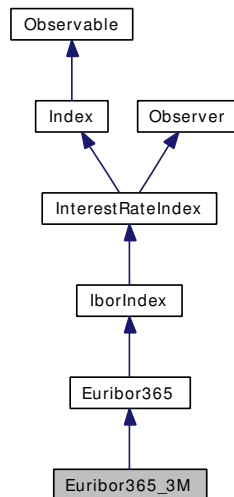
Public Member Functions

- **Euribor365_2W** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.248 Euribor365_3M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_3M:



9.248.1 Detailed Description

3-months Euribor365 index

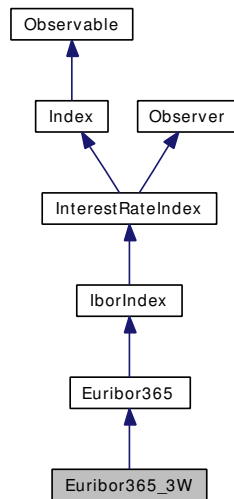
Public Member Functions

- **Euribor365_3M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.249 Euribor365_3W Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_3W:



9.249.1 Detailed Description

3-weeks Euribor365 index

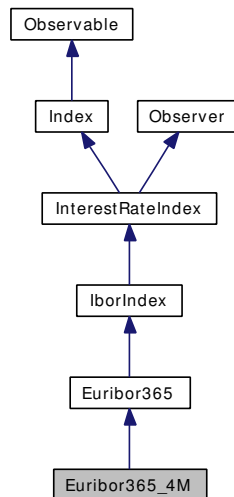
Public Member Functions

- **Euribor365_3W** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.250 Euribor365_4M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_4M:



9.250.1 Detailed Description

4-months Euribor365 index

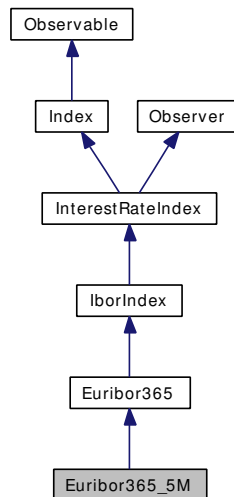
Public Member Functions

- **Euribor365_4M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.251 Euribor365_5M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_5M:



9.251.1 Detailed Description

5-months Euribor365 index

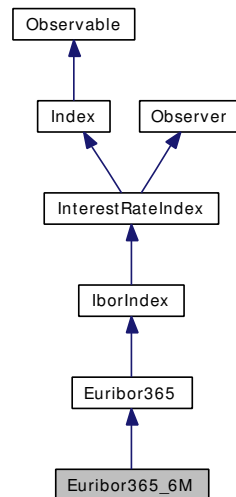
Public Member Functions

- **Euribor365_5M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.252 Euribor365_6M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_6M:



9.252.1 Detailed Description

6-months Euribor365 index

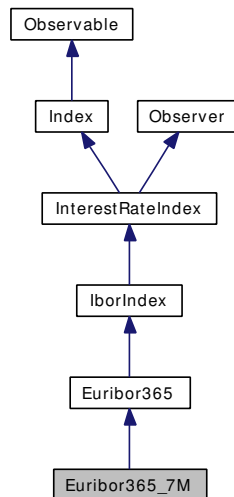
Public Member Functions

- **Euribor365_6M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.253 Euribor365_7M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_7M:



9.253.1 Detailed Description

7-months Euribor365 index

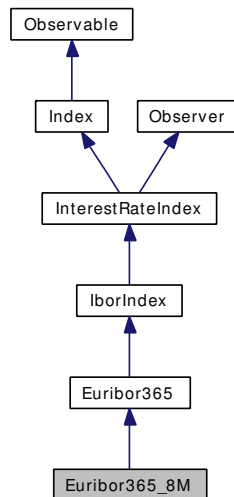
Public Member Functions

- **Euribor365_7M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.254 Euribor365_8M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_8M:



9.254.1 Detailed Description

8-months Euribor365 index

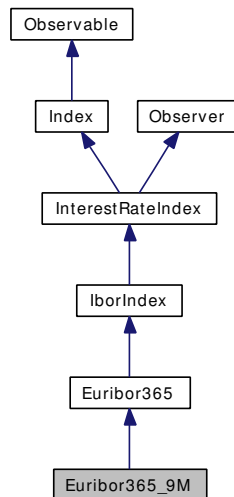
Public Member Functions

- **Euribor365_8M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.255 Euribor365_9M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_9M:



9.255.1 Detailed Description

9-months Euribor365 index

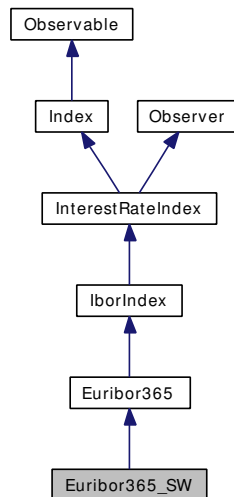
Public Member Functions

- **Euribor365_9M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.256 Euribor365_SW Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor365_SW:



9.256.1 Detailed Description

1-week Euribor365 index

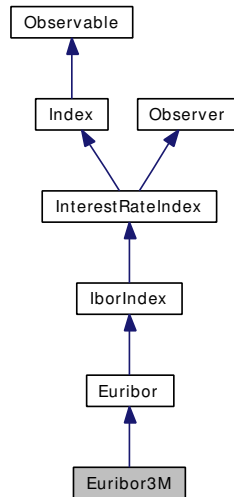
Public Member Functions

- **Euribor365_SW** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.257 Euribor3M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor3M:



9.257.1 Detailed Description

3-months Euribor index

Examples:

[FRA.cpp](#).

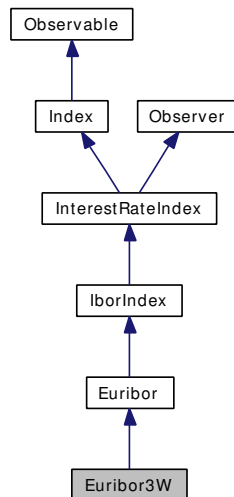
Public Member Functions

- **Euribor3M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.258 Euribor3W Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor3W:



9.258.1 Detailed Description

3-weeks Euribor index

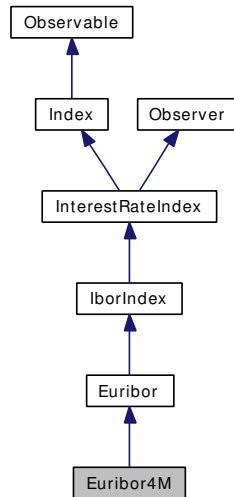
Public Member Functions

- **Euribor3W** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.259 Euribor4M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor4M:



9.259.1 Detailed Description

4-months Euribor index

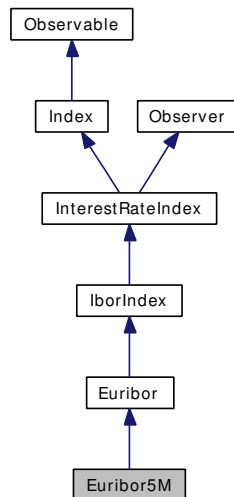
Public Member Functions

- **Euribor4M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.260 Euribor5M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor5M:



9.260.1 Detailed Description

5-months Euribor index

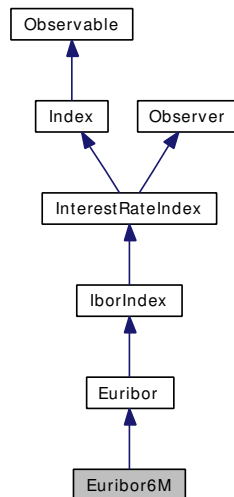
Public Member Functions

- **Euribor5M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.261 Euribor6M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor6M:



9.261.1 Detailed Description

6-months Euribor index

Examples:

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

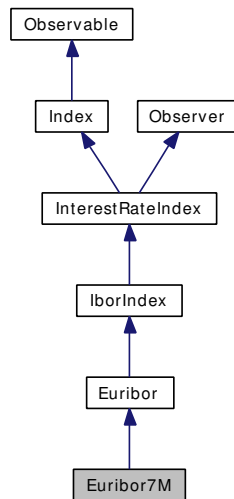
Public Member Functions

- **Euribor6M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.262 Euribor7M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor7M:



9.262.1 Detailed Description

7-months Euribor index

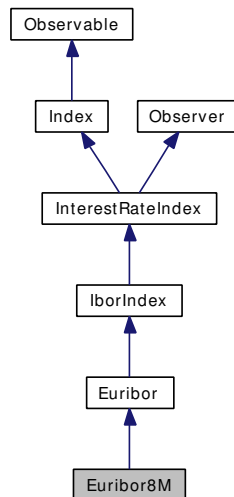
Public Member Functions

- **Euribor7M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.263 Euribor8M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor8M:



9.263.1 Detailed Description

8-months Euribor index

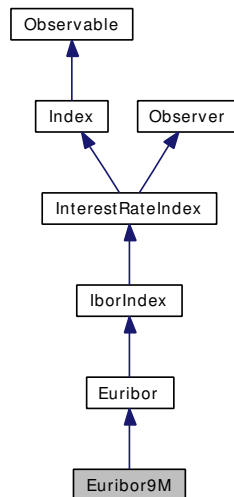
Public Member Functions

- **Euribor8M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.264 Euribor9M Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for Euribor9M:



9.264.1 Detailed Description

9-months Euribor index

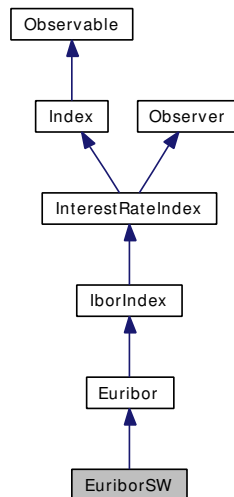
Public Member Functions

- **Euribor9M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.265 EuriborSW Class Reference

```
#include <ql/indexes/ibor/euribor.hpp>
```

Inheritance diagram for EuriborSW:



9.265.1 Detailed Description

1-week Euribor index

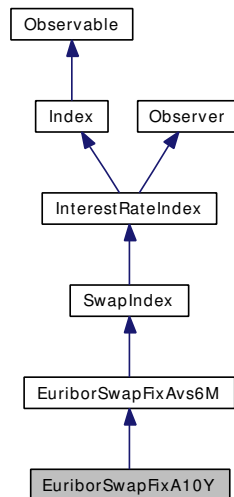
Public Member Functions

- **EuriborSW** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.266 EuriborSwapFixA10Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA10Y:



9.266.1 Detailed Description

10-year `EuriborSwapFixAvs6M` index

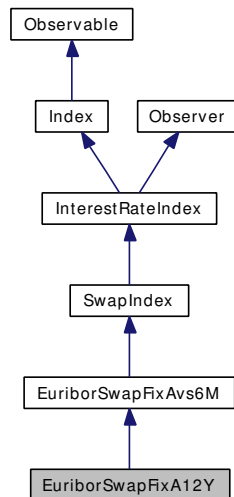
Public Member Functions

- `EuriborSwapFixA10Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.267 EuriborSwapFixA12Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA12Y:



9.267.1 Detailed Description

12-year `EuriborSwapFixAvs6M` index

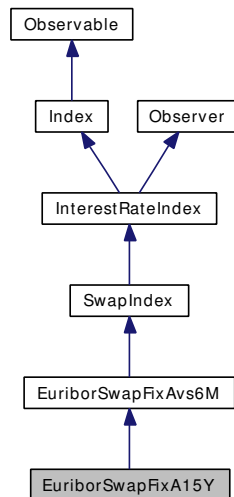
Public Member Functions

- `EuriborSwapFixA12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.268 EuriborSwapFixA15Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA15Y:



9.268.1 Detailed Description

15-year `EuriborSwapFixAvs6M` index

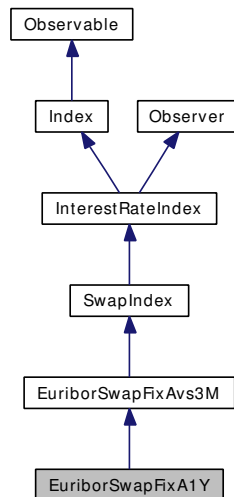
Public Member Functions

- `EuriborSwapFixA15Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.269 EuriborSwapFixA1Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA1Y:



9.269.1 Detailed Description

1-year EuriborSwapFixAvs3M index

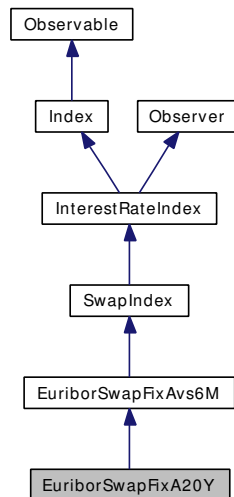
Public Member Functions

- `EuriborSwapFixA1Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.270 EuriborSwapFixA20Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA20Y:



9.270.1 Detailed Description

20-year `EuriborSwapFixAvs6M` index

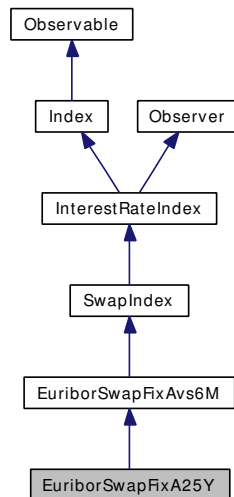
Public Member Functions

- `EuriborSwapFixA20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.271 EuriborSwapFixA25Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA25Y:



9.271.1 Detailed Description

25-year EuriborSwapFixAvs6M index

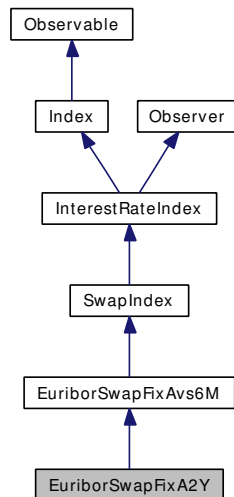
Public Member Functions

- EuriborSwapFixA25Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.272 EuriborSwapFixA2Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA2Y:



9.272.1 Detailed Description

2-year EuriborSwapFixAvs6M index

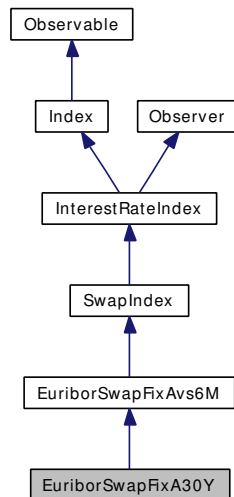
Public Member Functions

- `EuriborSwapFixA2Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.273 EuriborSwapFixA30Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA30Y:



9.273.1 Detailed Description

30-year EuriborSwapFixAvs6M index

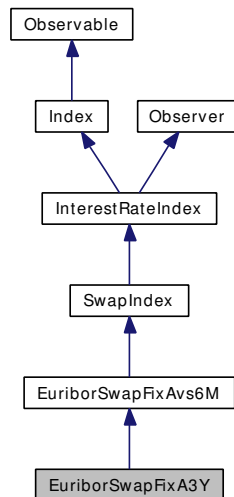
Public Member Functions

- EuriborSwapFixA30Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.274 EuriborSwapFixA3Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA3Y:



9.274.1 Detailed Description

3-year EuriborSwapFixAvs6M index

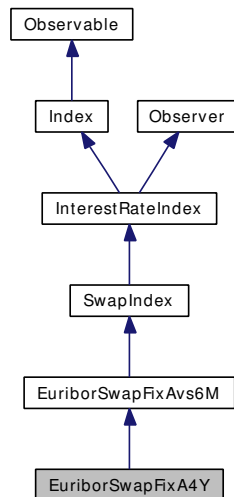
Public Member Functions

- `EuriborSwapFixA3Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.275 EuriborSwapFixA4Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA4Y:



9.275.1 Detailed Description

4-year EuriborSwapFixAvs6M index

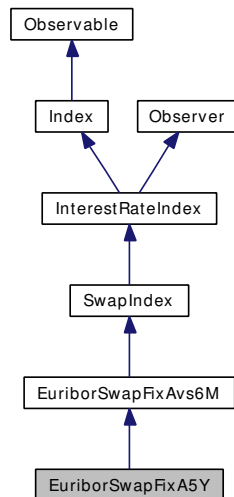
Public Member Functions

- EuriborSwapFixA4Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.276 EuriborSwapFixA5Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA5Y:



9.276.1 Detailed Description

5-year EuriborSwapFixAvs6M index

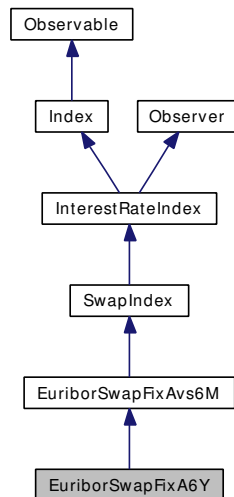
Public Member Functions

- `EuriborSwapFixA5Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.277 EuriborSwapFixA6Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA6Y:



9.277.1 Detailed Description

6-year EuriborSwapFixAvs6M index

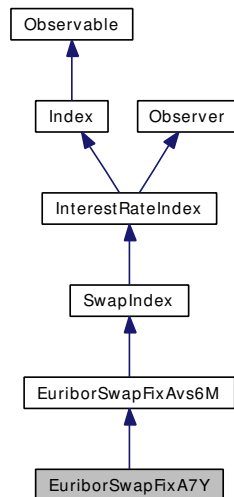
Public Member Functions

- `EuriborSwapFixA6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.278 EuriborSwapFixA7Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA7Y:



9.278.1 Detailed Description

7-year EuriborSwapFixAvs6M index

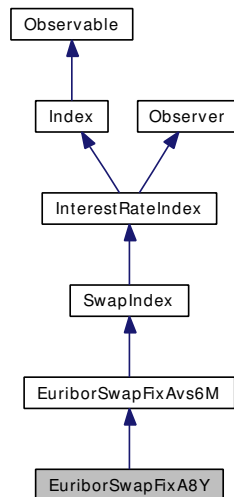
Public Member Functions

- `EuriborSwapFixA7Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.279 EuriborSwapFixA8Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA8Y:



9.279.1 Detailed Description

8-year EuriborSwapFixAvs6M index

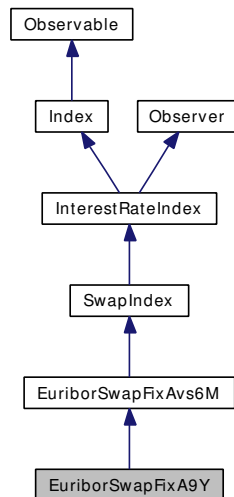
Public Member Functions

- EuriborSwapFixA8Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.280 EuriborSwapFixA9Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA9Y:



9.280.1 Detailed Description

9-year EuriborSwapFixAvs6M index

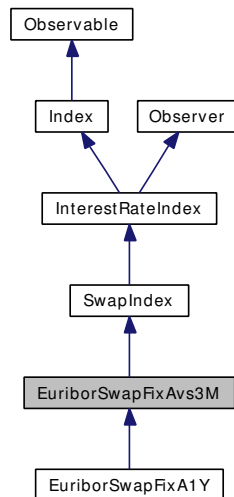
Public Member Functions

- `EuriborSwapFixA9Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.281 EuriborSwapFixAvs3M Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixAvs3M:



9.281.1 Detailed Description

EuriborSwapFixA vs 3M index base class

EuriborSwapFixA rate fixed by ISDA. The swap index is based on the [Euribor](#) 3M and is fixed at 11:00AM FRANKFURT. Reuters page ISDAFIX2 or EURSFIXA=.

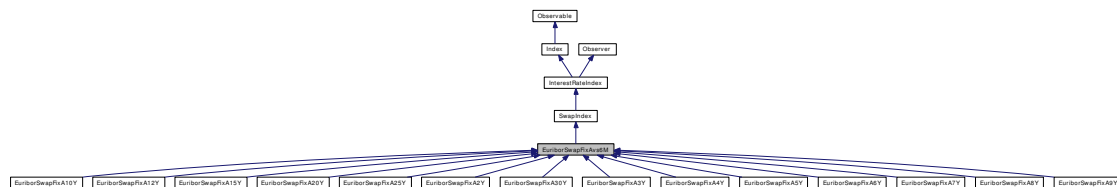
Public Member Functions

- `EuriborSwapFixAvs3M` (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.282 EuriborSwapFixAvs6M Class Reference

```
#include <ql/indexes/swap/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixAvs6M:



9.282.1 Detailed Description

EuriborSwapFixA vs 6M index base class

EuriborSwapFixA rate fixed by ISDA. The swap index is based on the [Euribor](#) 6M and is fixed at 11:00AM FRANKFURT. Reuters page ISDAFIX2 or EURSFXA=.

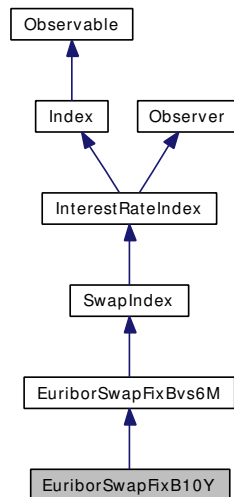
Public Member Functions

- **EuriborSwapFixAvs6M** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.283 EuriborSwapFixB10Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB10Y:



9.283.1 Detailed Description

10-year `EuriborSwapFixBvs6M` index

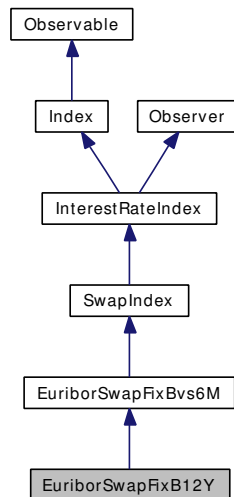
Public Member Functions

- `EuriborSwapFixB10Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.284 EuriborSwapFixB12Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB12Y:



9.284.1 Detailed Description

12-year `EuriborSwapFixBvs6M` index

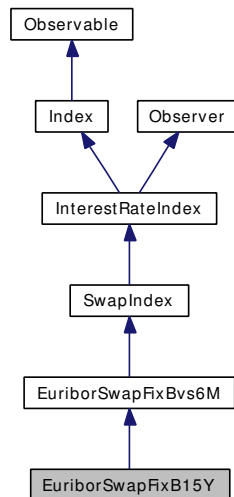
Public Member Functions

- `EuriborSwapFixB12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.285 EuriborSwapFixB15Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB15Y:



9.285.1 Detailed Description

15-year EuriborSwapFixBvs6M index

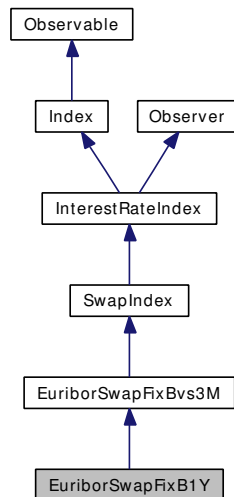
Public Member Functions

- EuriborSwapFixB15Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.286 EuriborSwapFixB1Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB1Y:



9.286.1 Detailed Description

1-year EuriborSwapFixBvs3M index

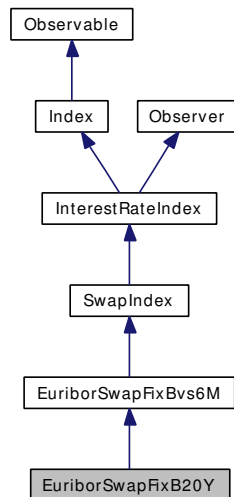
Public Member Functions

- `EuriborSwapFixB1Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.287 EuriborSwapFixB20Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB20Y:



9.287.1 Detailed Description

20-year `EuriborSwapFixBvs6M` index

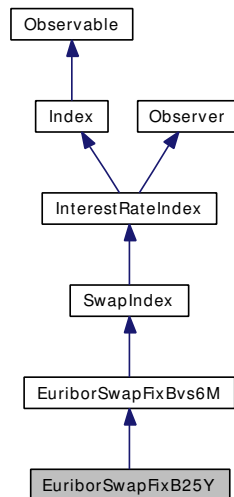
Public Member Functions

- `EuriborSwapFixB20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.288 EuriborSwapFixB25Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB25Y:



9.288.1 Detailed Description

25-year `EuriborSwapFixBvs6M` index

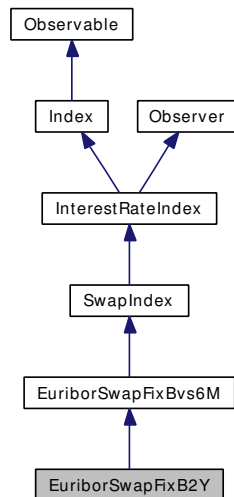
Public Member Functions

- `EuriborSwapFixB25Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.289 EuriborSwapFixB2Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB2Y:



9.289.1 Detailed Description

2-year EuriborSwapFixBvs6M index

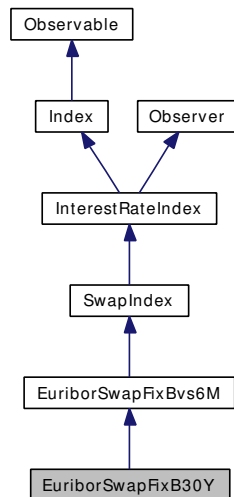
Public Member Functions

- `EuriborSwapFixB2Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.290 EuriborSwapFixB30Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB30Y:



9.290.1 Detailed Description

30-year `EuriborSwapFixBvs6M` index

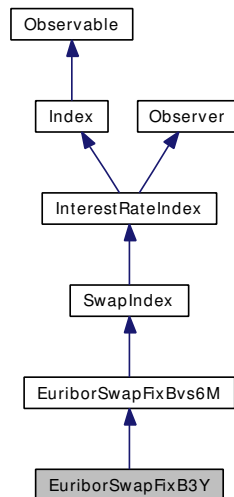
Public Member Functions

- `EuriborSwapFixB30Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.291 EuriborSwapFixB3Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB3Y:



9.291.1 Detailed Description

3-year EuriborSwapFixBvs6M index

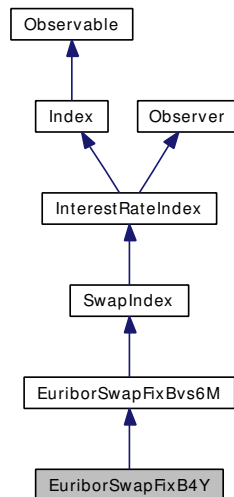
Public Member Functions

- `EuriborSwapFixB3Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.292 EuriborSwapFixB4Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB4Y:



9.292.1 Detailed Description

4-year EuriborSwapFixBvs6M index

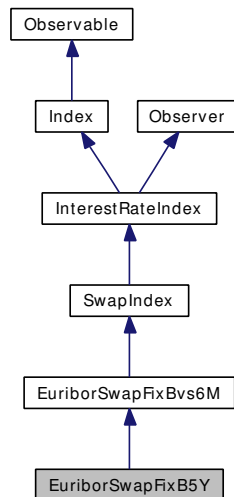
Public Member Functions

- `EuriborSwapFixB4Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.293 EuriborSwapFixB5Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB5Y:



9.293.1 Detailed Description

5-year EuriborSwapFixBvs6M index

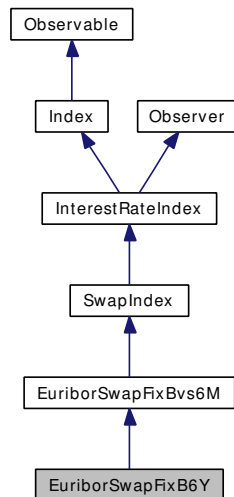
Public Member Functions

- EuriborSwapFixB5Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.294 EuriborSwapFixB6Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB6Y:



9.294.1 Detailed Description

6-year EuriborSwapFixBvs6M index

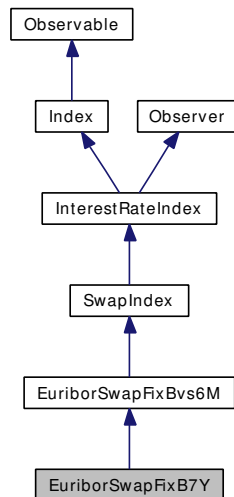
Public Member Functions

- `EuriborSwapFixB6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.295 EuriborSwapFixB7Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB7Y:



9.295.1 Detailed Description

7-year EuriborSwapFixBvs6M index

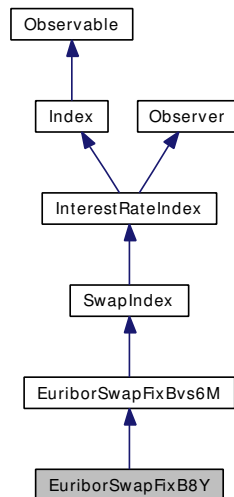
Public Member Functions

- `EuriborSwapFixB7Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.296 EuriborSwapFixB8Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB8Y:



9.296.1 Detailed Description

8-year EuriborSwapFixBvs6M index

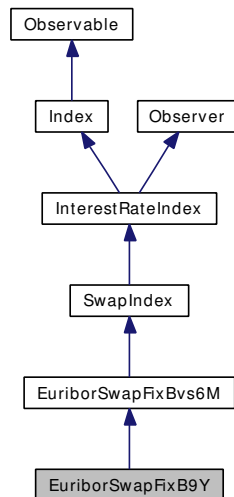
Public Member Functions

- `EuriborSwapFixB8Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.297 EuriborSwapFixB9Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixB9Y:



9.297.1 Detailed Description

9-year `EuriborSwapFixBvs6M` index

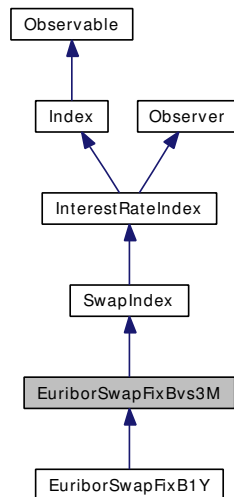
Public Member Functions

- `EuriborSwapFixB9Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.298 EuriborSwapFixBvs3M Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixBvs3M:



9.298.1 Detailed Description

EuriborSwapFixB vs 3M index base class

EuriborSwapFixB rate fixed by ISDA. The swap index is based on the [Euribor](#) 3M and is fixed at 12:00AM FRANKFURT. Reuters page ISDAFIX2 or EURSFXA=.

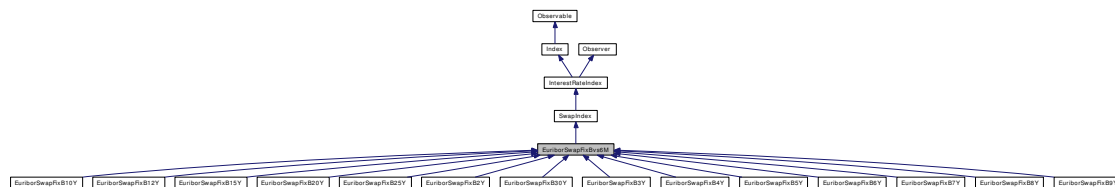
Public Member Functions

- `EuriborSwapFixBvs3M` (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.299 EuriborSwapFixBvs6M Class Reference

```
#include <ql/indexes/swap/euriborswapfixb.hpp>
```

Inheritance diagram for EuriborSwapFixBvs6M:



9.299.1 Detailed Description

EuriborSwapFixB vs 6M index base class

EuriborSwapFixB rate fixed by ISDA. The swap index is based on the [Euribor](#) 6M and is fixed at 12:00AM FRANKFURT. Reuters page ISDAFIX2 or EURSFXB=.

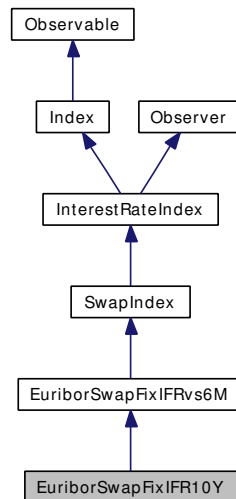
Public Member Functions

- **EuriborSwapFixBvs6M** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.300 EuriborSwapFixIFR10Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR10Y:



9.300.1 Detailed Description

10-year EuriborSwapFixIFRvs6M index

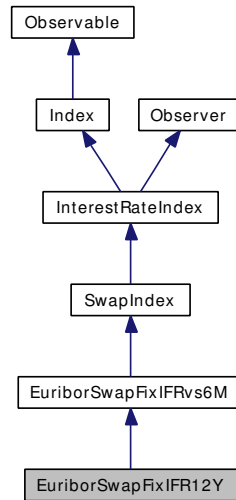
Public Member Functions

- `EuriborSwapFixIFR10Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.301 EuriborSwapFixIFR12Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR12Y:



9.301.1 Detailed Description

12-year `EuriborSwapFixIFRvs6M` index

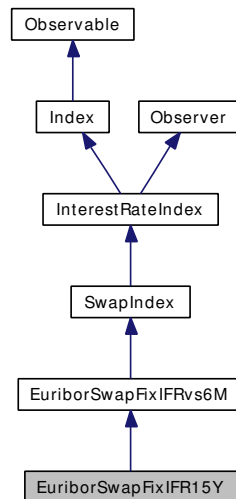
Public Member Functions

- `EuriborSwapFixIFR12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.302 EuriborSwapFixIFR15Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR15Y:



9.302.1 Detailed Description

15-year `EuriborSwapFixIFRvs6M` index

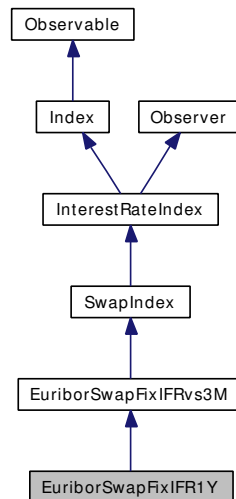
Public Member Functions

- `EuriborSwapFixIFR15Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.303 EuriborSwapFixIFR1Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR1Y:



9.303.1 Detailed Description

1-year EuriborSwapFixIFR3M index

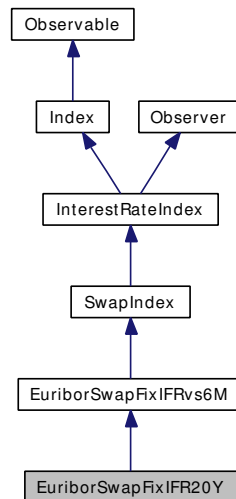
Public Member Functions

- **EuriborSwapFixIFR1Y** (const [Handle](#)< [YieldTermStructure](#) > &h)

9.304 EuriborSwapFixIFR20Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR20Y:



9.304.1 Detailed Description

20-year `EuriborSwapFixIFRvs6M` index

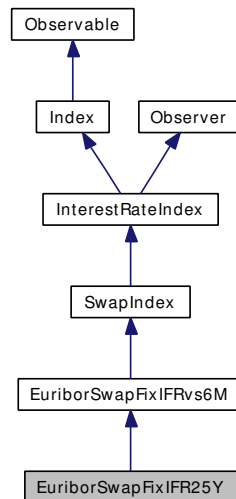
Public Member Functions

- `EuriborSwapFixIFR20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.305 EuriborSwapFixIFR25Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR25Y:



9.305.1 Detailed Description

25-year EuriborSwapFixIFRvs6M index

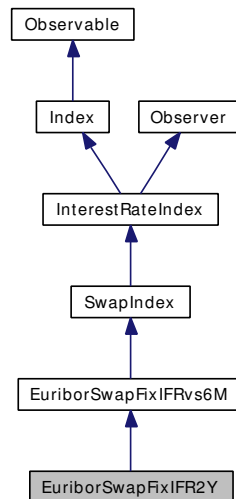
Public Member Functions

- **EuriborSwapFixIFR25Y** (const [Handle](#)< [YieldTermStructure](#) > &h)

9.306 EuriborSwapFixIFR2Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR2Y:



9.306.1 Detailed Description

2-year EuriborSwapFixIFRvs6M index

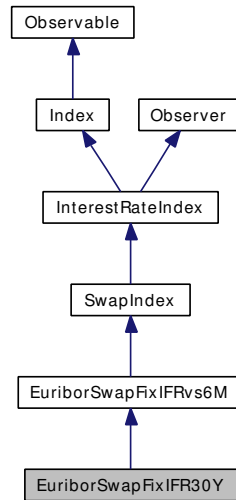
Public Member Functions

- `EuriborSwapFixIFR2Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.307 EuriborSwapFixIFR30Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR30Y:



9.307.1 Detailed Description

30-year EuriborSwapFixIFRvs6M index

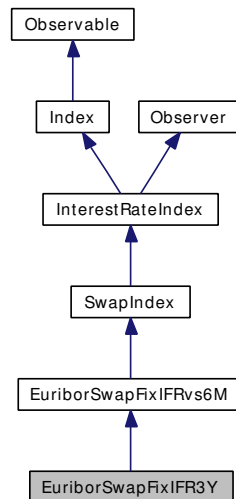
Public Member Functions

- **EuriborSwapFixIFR30Y** (const [Handle](#)< [YieldTermStructure](#) > &h)

9.308 EuriborSwapFixIFR3Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR3Y:



9.308.1 Detailed Description

3-year EuriborSwapFixIFRvs6M index

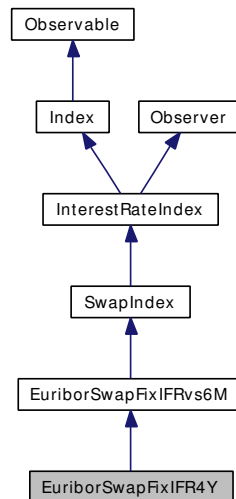
Public Member Functions

- `EuriborSwapFixIFR3Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.309 EuriborSwapFixIFR4Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR4Y:



9.309.1 Detailed Description

4-year EuriborSwapFixIFRvs6M index

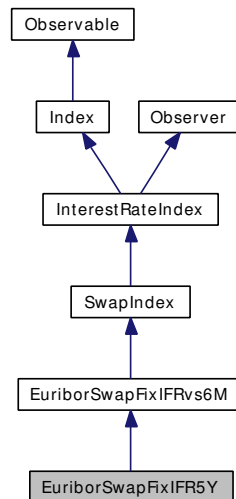
Public Member Functions

- EuriborSwapFixIFR4Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.310 EuriborSwapFixIFR5Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR5Y:



9.310.1 Detailed Description

5-year EuriborSwapFixIFRvs6M index

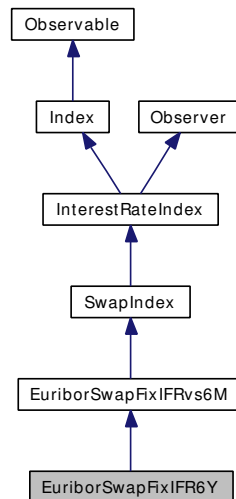
Public Member Functions

- `EuriborSwapFixIFR5Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.311 EuriborSwapFixIFR6Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR6Y:



9.311.1 Detailed Description

6-year EuriborSwapFixIFRvs6M index

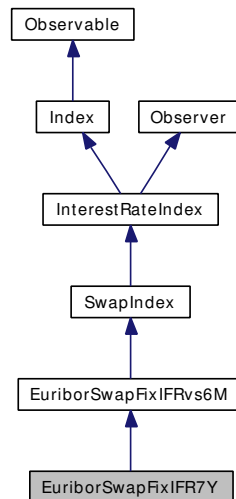
Public Member Functions

- `EuriborSwapFixIFR6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.312 EuriborSwapFixIFR7Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR7Y:



9.312.1 Detailed Description

7-year EuriborSwapFixIFRvs6M index

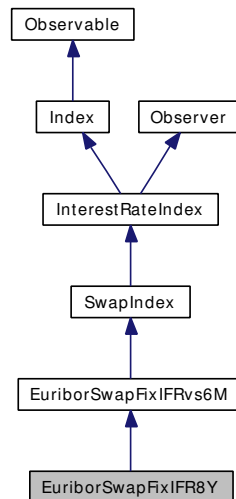
Public Member Functions

- `EuriborSwapFixIFR7Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.313 EuriborSwapFixIFR8Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR8Y:



9.313.1 Detailed Description

8-year EuriborSwapFixIFRvs6M index

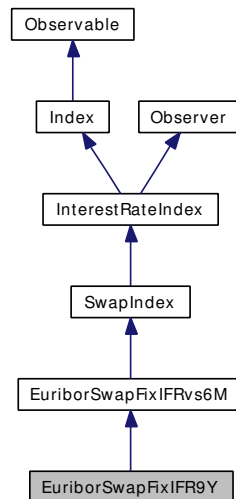
Public Member Functions

- `EuriborSwapFixIFR8Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.314 EuriborSwapFixIFR9Y Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR9Y:



9.314.1 Detailed Description

9-year EuriborSwapFixIFRvs6M index

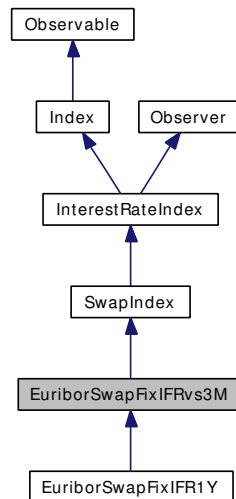
Public Member Functions

- `EuriborSwapFixIFR9Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.315 EuriborSwapFixIFRvs3M Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFRvs3M:



9.315.1 Detailed Description

EuriborSwapFixIFR vs 3M index base class

EuriborSwapFixIFR index published by IFR Markets and distributed by Reuters page TGM42281 and by Telerate. For more info see <<http://www.ifrmarkets.com>>.

Public Member Functions

- `EuriborSwapFixIFRvs3M` (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.316 EuriborSwapFixIFRvs6M Class Reference

```
#include <ql/indexes/swap/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFRvs6M:



9.316.1 Detailed Description

EuriborSwapFixIFR vs 6M index base class

EuriborSwapFixIFR index published by IFR Markets and distributed by Reuters page TGM42281 and by Telerate. For more info see <<http://www.ifrmarkets.com>>.

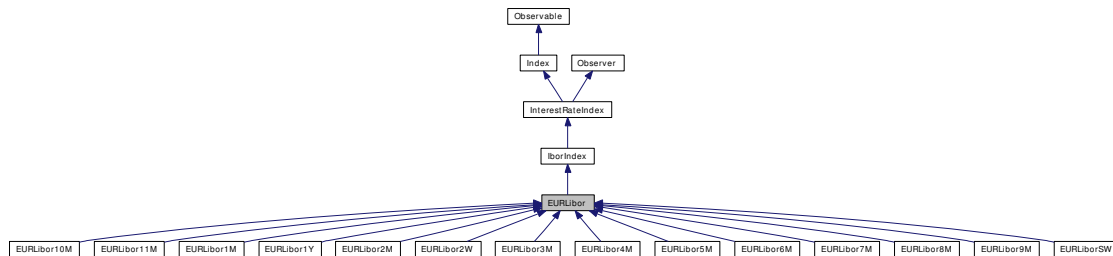
Public Member Functions

- **EuriborSwapFixIFRvs6M** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.317 EURLibor Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor:



9.317.1 Detailed Description

EUR LIBOR rate

Euro LIBOR fixed by BBA.

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

Warning

This is the rate fixed in London by BBA. Use [Euribor](#) if you're interested in the fixing by the ECB.

Warning

This is not a valid base class for the O/N index

Public Member Functions

- **EURLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h)

Date calculations

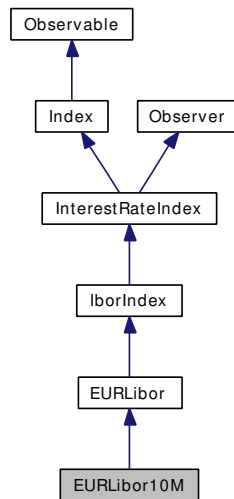
see <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1412>

- **Date valueDate** (const [Date](#) &fixingDate) const
- **Date maturityDate** (const [Date](#) &valueDate) const

9.318 EURLibor10M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor10M:



9.318.1 Detailed Description

10-months EURLibor index

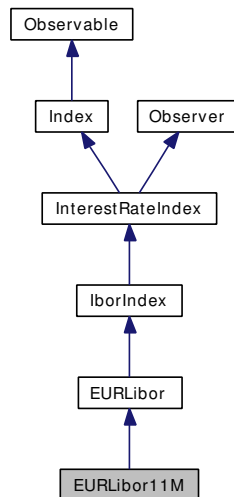
Public Member Functions

- `EURLibor10M(const Handle<YieldTermStructure> &h=Handle<YieldTermStructure>())`

9.319 EURLibor11M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor11M:



9.319.1 Detailed Description

11-months EURLibor index

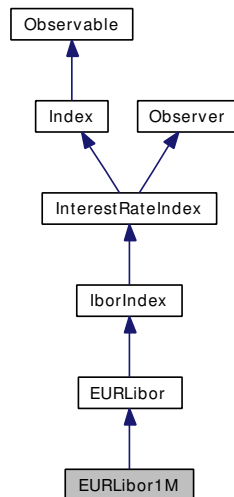
Public Member Functions

- `EURLibor11M(const Handle<YieldTermStructure> &h=Handle<YieldTermStructure>())`

9.320 EURLibor1M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor1M:



9.320.1 Detailed Description

1-month EURLibor index

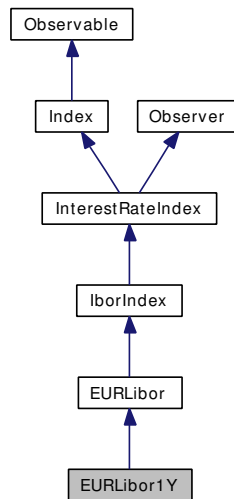
Public Member Functions

- EURLibor1M (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.321 EURLibor1Y Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor1Y:



9.321.1 Detailed Description

1-year EURLibor index

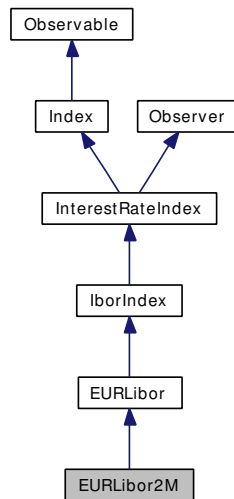
Public Member Functions

- EURLibor1Y (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.322 EURLibor2M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor2M:



9.322.1 Detailed Description

2-months EURLibor index

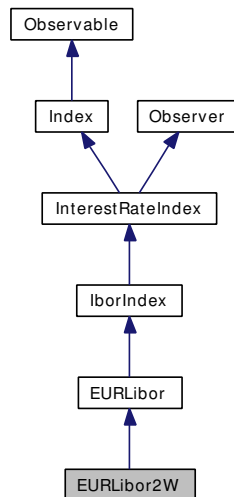
Public Member Functions

- EURLibor2M (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.323 EURLibor2W Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor2W:



9.323.1 Detailed Description

2-weeks Euribor index

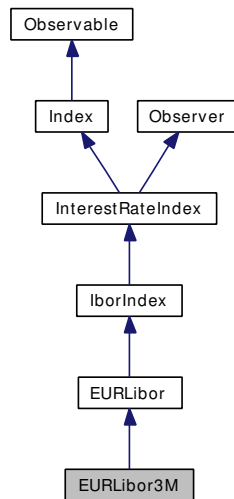
Public Member Functions

- EURLibor2W (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.324 EURLibor3M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor3M:



9.324.1 Detailed Description

3-months EURLibor index

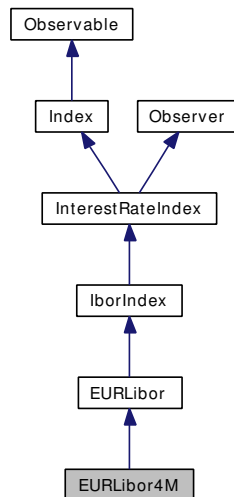
Public Member Functions

- EURLibor3M (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.325 EURLibor4M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor4M:



9.325.1 Detailed Description

4-months EURLibor index

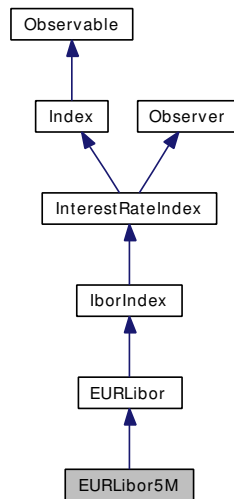
Public Member Functions

- EURLibor4M (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.326 EURLibor5M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor5M:



9.326.1 Detailed Description

5-months EURLibor index

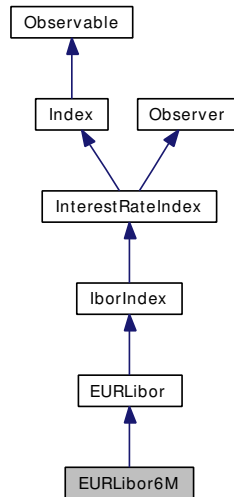
Public Member Functions

- EURLibor5M (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.327 EURLibor6M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor6M:



9.327.1 Detailed Description

6-months EURLibor index

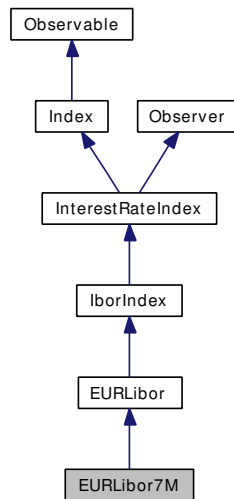
Public Member Functions

- **EURLibor6M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.328 EURLibor7M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor7M:



9.328.1 Detailed Description

7-months EURLibor index

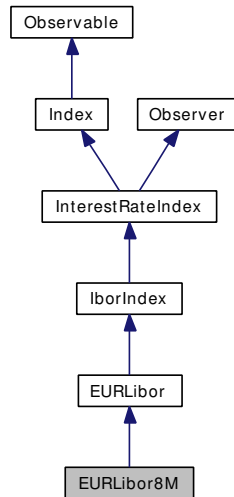
Public Member Functions

- EURLibor7M (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.329 EURLibor8M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor8M:



9.329.1 Detailed Description

8-months EURLibor index

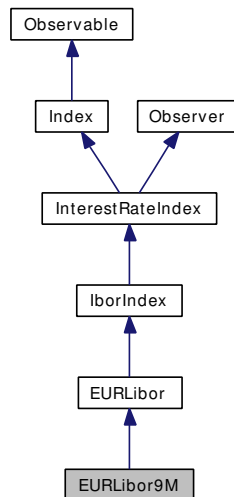
Public Member Functions

- EURLibor8M (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.330 EURLibor9M Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLibor9M:



9.330.1 Detailed Description

9-months EURLibor index

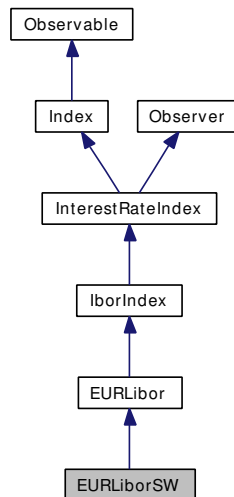
Public Member Functions

- EURLibor9M (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.331 EURLiborSW Class Reference

```
#include <ql/indexes/ibor/eurlibor.hpp>
```

Inheritance diagram for EURLiborSW:



9.331.1 Detailed Description

1-week EURLibor index

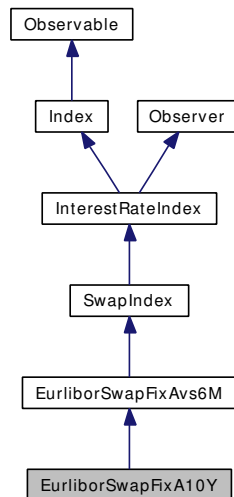
Public Member Functions

- EURLiborSW (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.332 EurliborSwapFixA10Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA10Y:



9.332.1 Detailed Description

10-year EurliborSwapFixAvs6M index

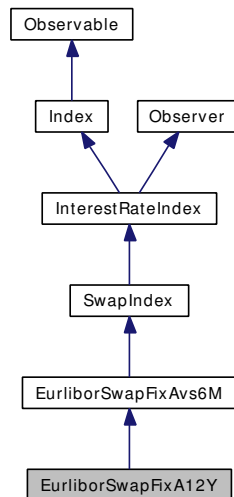
Public Member Functions

- `EurliborSwapFixA10Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.333 EurliborSwapFixA12Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA12Y:



9.333.1 Detailed Description

12-year EurliborSwapFixAvs6M index

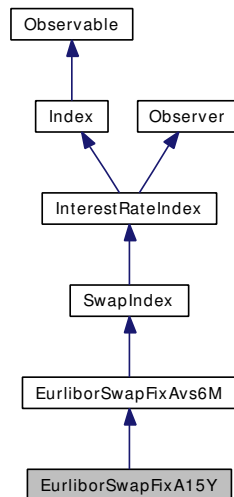
Public Member Functions

- `EurliborSwapFixA12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.334 EurliborSwapFixA15Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA15Y:



9.334.1 Detailed Description

15-year EurliborSwapFixAvs6M index

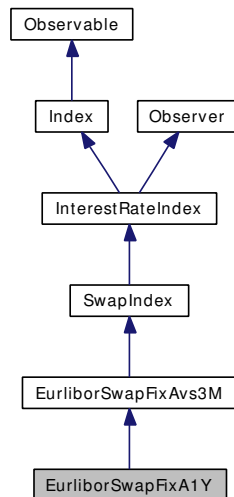
Public Member Functions

- `EurliborSwapFixA15Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.335 EurliborSwapFixA1Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA1Y:



9.335.1 Detailed Description

1-year EurliborSwapFixAvs3M index

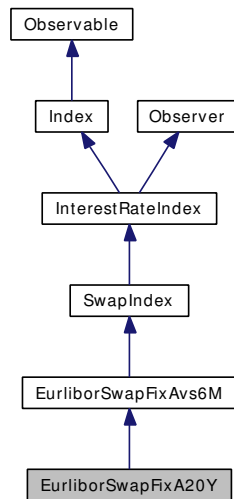
Public Member Functions

- EurliborSwapFixA1Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.336 EurliborSwapFixA20Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA20Y:



9.336.1 Detailed Description

20-year EurliborSwapFixAvs6M index

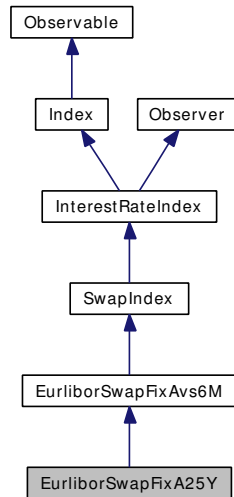
Public Member Functions

- `EurliborSwapFixA20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.337 EurliborSwapFixA25Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA25Y:



9.337.1 Detailed Description

25-year EurliborSwapFixAvs6M index

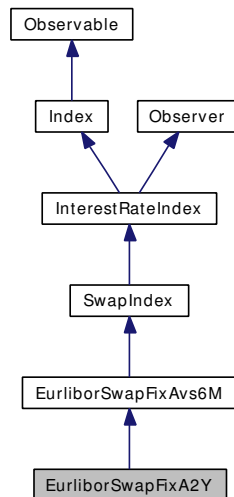
Public Member Functions

- `EurliborSwapFixA25Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.338 EurliborSwapFixA2Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA2Y:



9.338.1 Detailed Description

2-year EurliborSwapFixAvs6M index

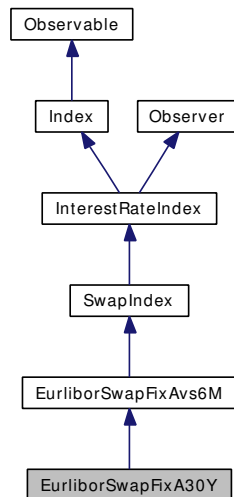
Public Member Functions

- `EurliborSwapFixA2Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.339 EurliborSwapFixA30Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA30Y:



9.339.1 Detailed Description

30-year EurliborSwapFixAvs6M index

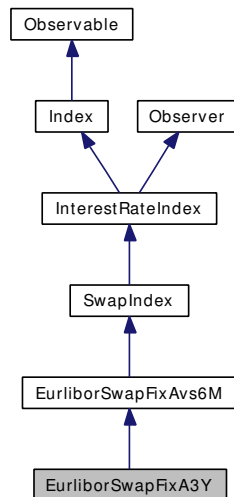
Public Member Functions

- `EurliborSwapFixA30Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.340 EurliborSwapFixA3Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA3Y:



9.340.1 Detailed Description

3-year EurliborSwapFixAvs6M index

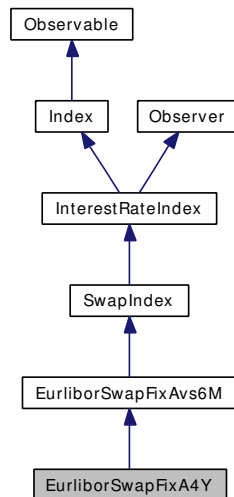
Public Member Functions

- `EurliborSwapFixA3Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.341 EurliborSwapFixA4Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA4Y:



9.341.1 Detailed Description

4-year EurliborSwapFixAvs6M index

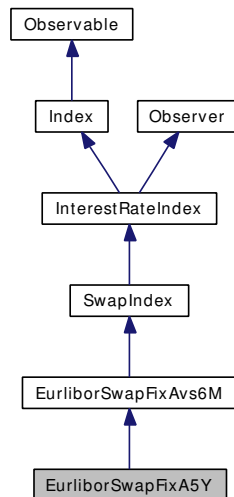
Public Member Functions

- `EurliborSwapFixA4Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.342 EurliborSwapFixA5Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA5Y:



9.342.1 Detailed Description

5-year EurliborSwapFixAvs6M index

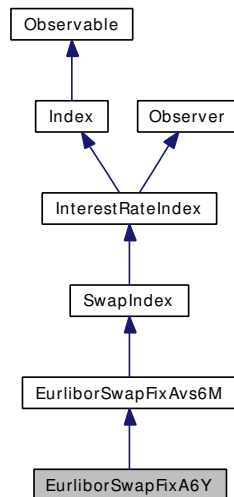
Public Member Functions

- `EurliborSwapFixA5Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.343 EurliborSwapFixA6Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA6Y:



9.343.1 Detailed Description

6-year EurliborSwapFixAvs6M index

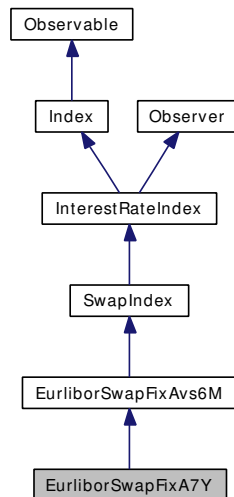
Public Member Functions

- `EurliborSwapFixA6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.344 EurliborSwapFixA7Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA7Y:



9.344.1 Detailed Description

7-year EurliborSwapFixAvs6M index

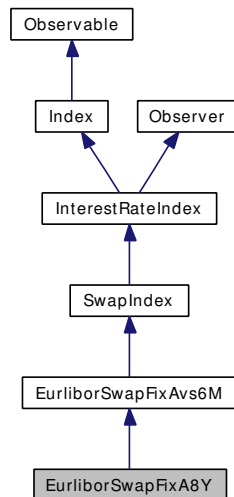
Public Member Functions

- `EurliborSwapFixA7Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.345 EurliborSwapFixA8Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA8Y:



9.345.1 Detailed Description

8-year EurliborSwapFixAvs6M index

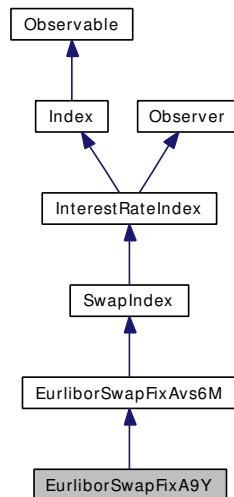
Public Member Functions

- EurliborSwapFixA8Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.346 EurliborSwapFixA9Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA9Y:



9.346.1 Detailed Description

9-year EurliborSwapFixAvs6M index

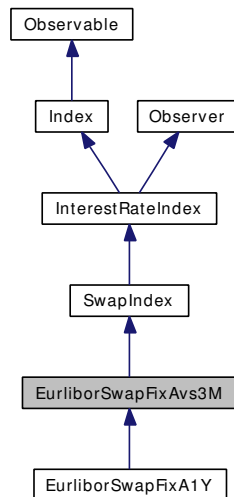
Public Member Functions

- `EurliborSwapFixA9Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.347 EurliborSwapFixAvs3M Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixAvs3M:



9.347.1 Detailed Description

EurliborSwapFixA vs 3M index base class

EurliborSwapFixA rate fixed by ISDA in cooperation with Reuters and Intercapital Brokers. The swap index is based on the EuroLibor 3M and is fixed at 10:00 AM London. Reuters page ISDAFIX2 or EURSFIXLA=. Further info can be found at: <http://www.isda.org/fix/isdafix.html>.

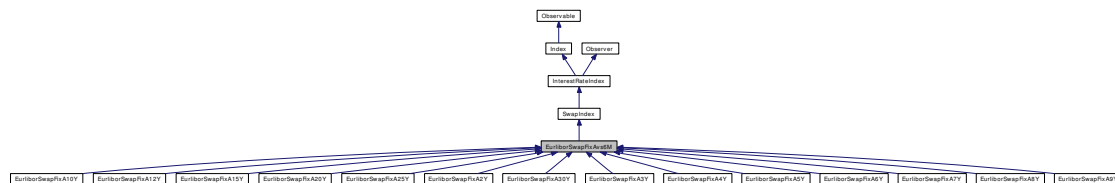
Public Member Functions

- `EurliborSwapFixAvs3M` (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.348 EurliborSwapFixAvs6M Class Reference

```
#include <ql/indexes/swap/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixAvs6M:



9.348.1 Detailed Description

EurliborSwapFixA vs 6M index base class

EurliborSwapFixA rate fixed by ISDA in cooperation with Reuters and Intercapital Brokers. The swap index is based on the EuroLibor 6M and is fixed at 10:00 AM London. Reuters page ISDAFIX2 or EURSFXLA=. Further info can be found at: <http://www.isda.org/fix/isdafix.html>.

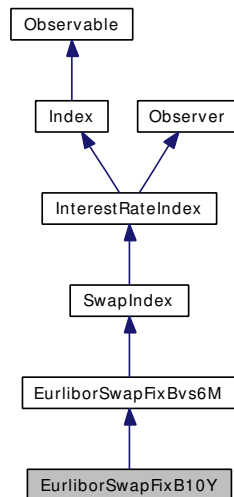
Public Member Functions

- **EurliborSwapFixAvs6M** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.349 EurliborSwapFixB10Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB10Y:



9.349.1 Detailed Description

10-year EurliborSwapFixBvs6M index

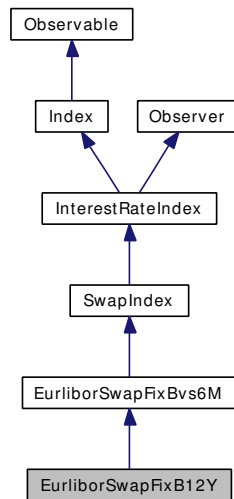
Public Member Functions

- EurliborSwapFixB10Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.350 EurliborSwapFixB12Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB12Y:



9.350.1 Detailed Description

12-year EurliborSwapFixBvs6M index

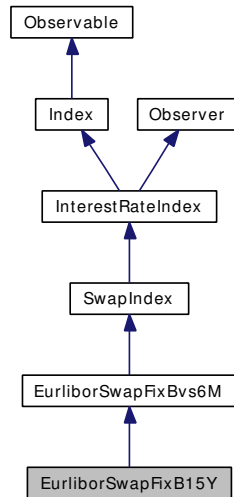
Public Member Functions

- `EurliborSwapFixB12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.351 EurliborSwapFixB15Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB15Y:



9.351.1 Detailed Description

15-year EurliborSwapFixBvs6M index

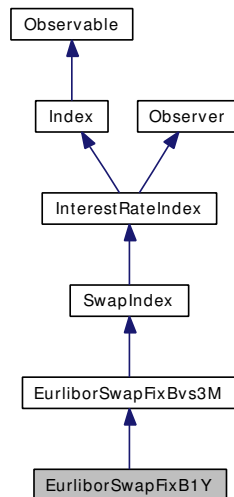
Public Member Functions

- EurliborSwapFixB15Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.352 EurliborSwapFixB1Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB1Y:



9.352.1 Detailed Description

1-year EurliborSwapFixBvs3M index

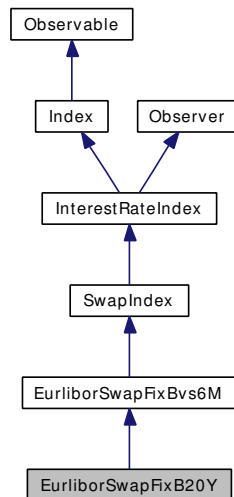
Public Member Functions

- `EurliborSwapFixB1Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.353 EurliborSwapFixB20Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB20Y:



9.353.1 Detailed Description

20-year EurliborSwapFixBvs6M index

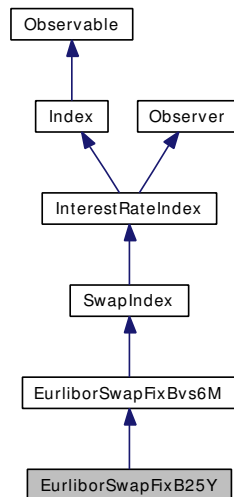
Public Member Functions

- `EurliborSwapFixB20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.354 EurliborSwapFixB25Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB25Y:



9.354.1 Detailed Description

25-year EurliborSwapFixBvs6M index

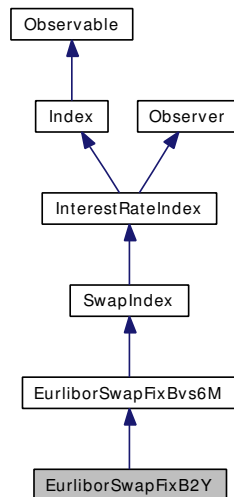
Public Member Functions

- `EurliborSwapFixB25Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.355 EurliborSwapFixB2Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB2Y:



9.355.1 Detailed Description

2-year EurliborSwapFixBvs6M index

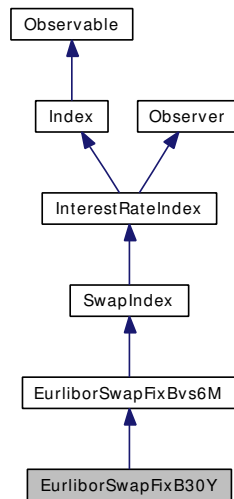
Public Member Functions

- EurliborSwapFixB2Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.356 EurliborSwapFixB30Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB30Y:



9.356.1 Detailed Description

30-year EurliborSwapFixBvs6M index

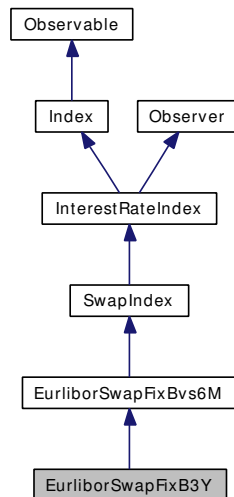
Public Member Functions

- `EurliborSwapFixB30Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.357 EurliborSwapFixB3Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB3Y:



9.357.1 Detailed Description

3-year EurliborSwapFixBvs6M index

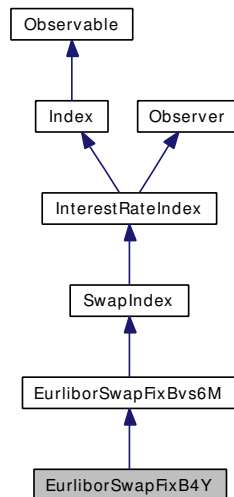
Public Member Functions

- `EurliborSwapFixB3Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.358 EurliborSwapFixB4Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB4Y:



9.358.1 Detailed Description

4-year EurliborSwapFixBvs6M index

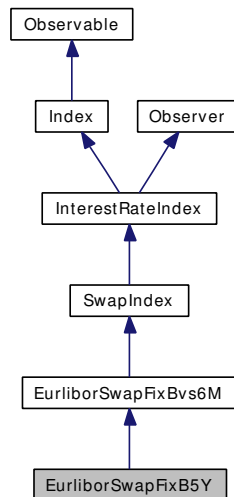
Public Member Functions

- `EurliborSwapFixB4Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.359 EurliborSwapFixB5Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB5Y:



9.359.1 Detailed Description

5-year EurliborSwapFixBvs6M index

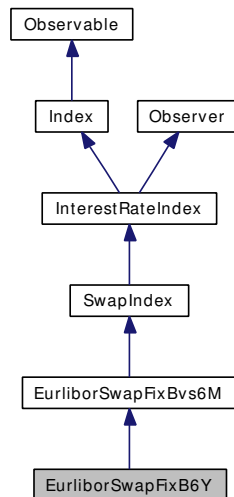
Public Member Functions

- EurliborSwapFixB5Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.360 EurliborSwapFixB6Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB6Y:



9.360.1 Detailed Description

6-year EurliborSwapFixBvs6M index

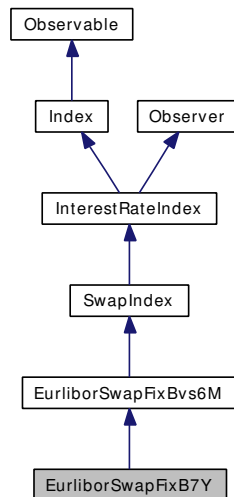
Public Member Functions

- `EurliborSwapFixB6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.361 EurliborSwapFixB7Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB7Y:



9.361.1 Detailed Description

7-year EurliborSwapFixBvs6M index

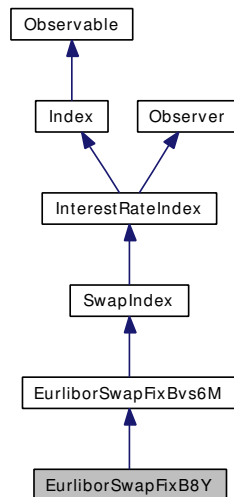
Public Member Functions

- EurliborSwapFixB7Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.362 EurliborSwapFixB8Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB8Y:



9.362.1 Detailed Description

8-year EurliborSwapFixBvs6M index

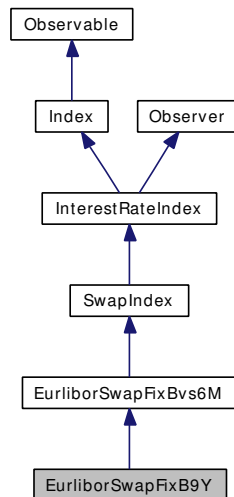
Public Member Functions

- `EurliborSwapFixB8Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.363 EurliborSwapFixB9Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB9Y:



9.363.1 Detailed Description

9-year EurliborSwapFixBvs6M index

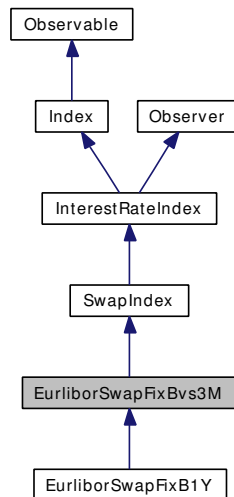
Public Member Functions

- `EurliborSwapFixB9Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.364 EurliborSwapFixBvs3M Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixBvs3M:



9.364.1 Detailed Description

EurliborSwapFixB vs 3M index base class

EurliborSwapFixB rate fixed by ISDA in cooperation with Reuters and Intercapital Brokers. The swap index is based on the EuroLibor 3M and is fixed at 11:00AM London. Reuters page ISDAFIX2 or EURSFXLB= Further info can be found at: <<http://www.isda.org/fix/isdafix.html>>.

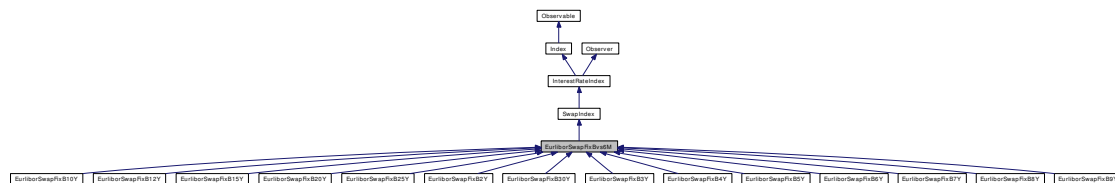
Public Member Functions

- `EurliborSwapFixBvs3M` (const `Period` &tenor, const `Handle`< `YieldTermStructure` > &h=`Handle`< `YieldTermStructure` >())

9.365 EurliborSwapFixBvs6M Class Reference

```
#include <ql/indexes/swap/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixBvs6M:



9.365.1 Detailed Description

EurliborSwapFixB vs 6M index base class

EurliborSwapFixB rate fixed by ISDA in cooperation with Reuters and Intercapital Brokers. The swap index is based on the EuroLibor 6M and is fixed at 11:00AM London. Reuters page ISDAFIX2 or EURSFIXLB= Further info can be found at: <http://www.isda.org/fix/isdafix.html>.

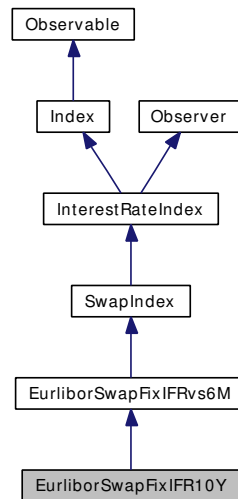
Public Member Functions

- **EurliborSwapFixBvs6M** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.366 EurliborSwapFixIFR10Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR10Y:



9.366.1 Detailed Description

10-year EurliborSwapFixIFRvs6M index

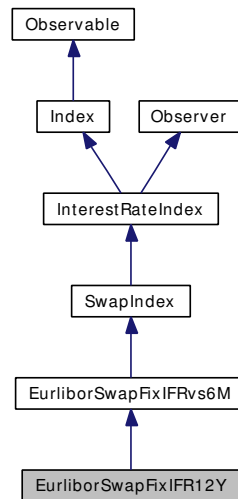
Public Member Functions

- `EurliborSwapFixIFR10Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.367 EurliborSwapFixIFR12Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR12Y:



9.367.1 Detailed Description

12-year EurliborSwapFixIFRvs6M index

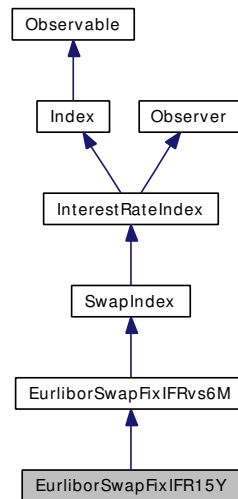
Public Member Functions

- `EurliborSwapFixIFR12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.368 EurliborSwapFixIFR15Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR15Y:



9.368.1 Detailed Description

15-year EurliborSwapFixIFRvs6M index

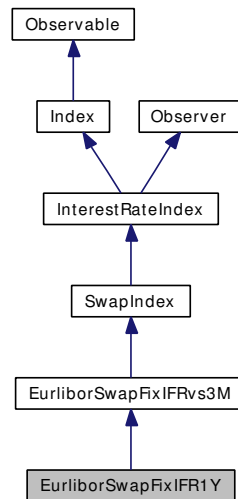
Public Member Functions

- `EurliborSwapFixIFR15Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.369 EurliborSwapFixIFR1Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR1Y:



9.369.1 Detailed Description

1-year EurliborSwapFixIFRvs3M index

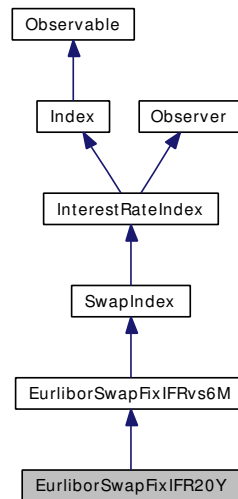
Public Member Functions

- `EurliborSwapFixIFR1Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.370 EurliborSwapFixIFR20Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR20Y:



9.370.1 Detailed Description

20-year EurliborSwapFixIFRvs6M index

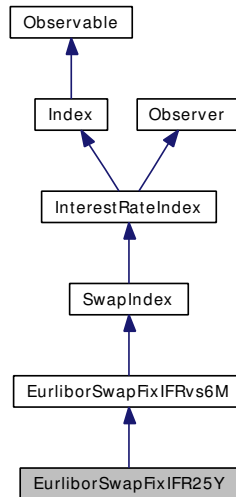
Public Member Functions

- `EurliborSwapFixIFR20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.371 EurliborSwapFixIFR25Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR25Y:



9.371.1 Detailed Description

25-year EurliborSwapFixIFRvs6M index

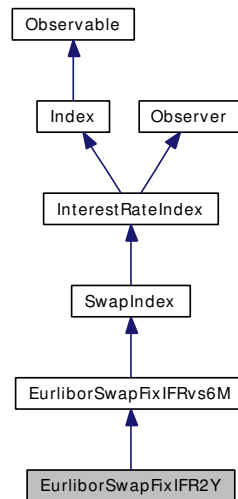
Public Member Functions

- `EurliborSwapFixIFR25Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.372 EurliborSwapFixIFR2Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR2Y:



9.372.1 Detailed Description

2-year EurliborSwapFixIFRvs6M index

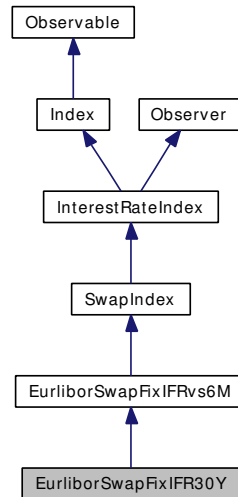
Public Member Functions

- `EurliborSwapFixIFR2Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.373 EurliborSwapFixIFR30Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR30Y:



9.373.1 Detailed Description

30-year EurliborSwapFixIFRvs6M index

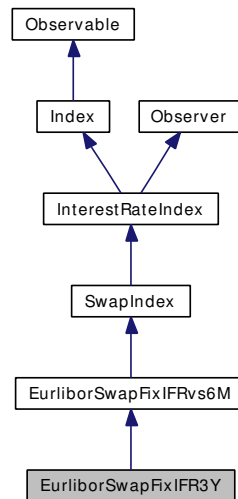
Public Member Functions

- `EurliborSwapFixIFR30Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.374 EurliborSwapFixIFR3Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR3Y:



9.374.1 Detailed Description

3-year EurliborSwapFixIFRvs6M index

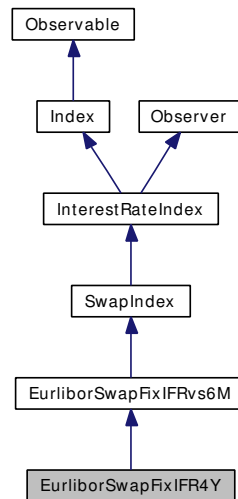
Public Member Functions

- `EurliborSwapFixIFR3Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.375 EurliborSwapFixIFR4Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR4Y:



9.375.1 Detailed Description

4-year EurliborSwapFixIFRvs6M index

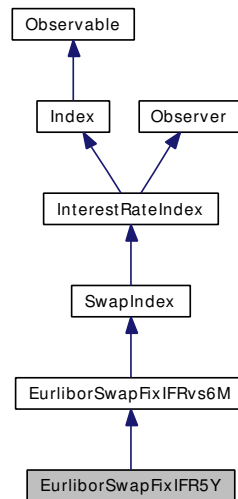
Public Member Functions

- `EurliborSwapFixIFR4Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.376 EurliborSwapFixIFR5Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR5Y:



9.376.1 Detailed Description

5-year EurliborSwapFixIFRvs6M index

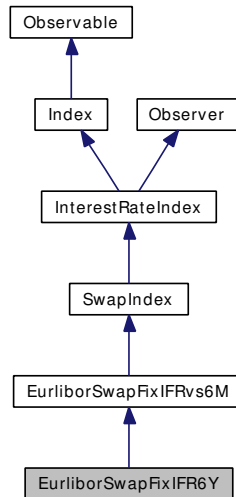
Public Member Functions

- `EurliborSwapFixIFR5Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.377 EurliborSwapFixIFR6Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR6Y:



9.377.1 Detailed Description

6-year EurliborSwapFixIFRvs6M index

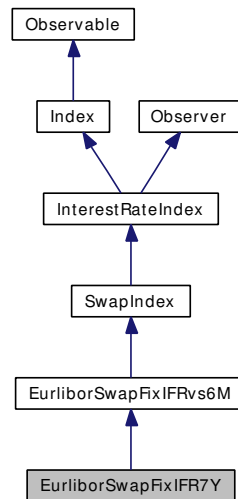
Public Member Functions

- `EurliborSwapFixIFR6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.378 EurliborSwapFixIFR7Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR7Y:



9.378.1 Detailed Description

7-year EurliborSwapFixIFRvs6M index

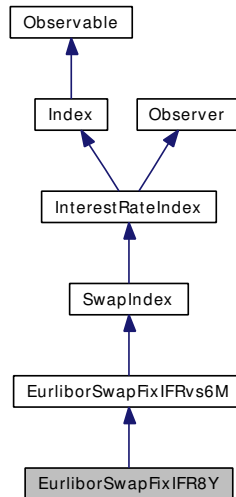
Public Member Functions

- `EurliborSwapFixIFR7Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.379 EurliborSwapFixIFR8Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR8Y:



9.379.1 Detailed Description

8-year EurliborSwapFixIFRvs6M index

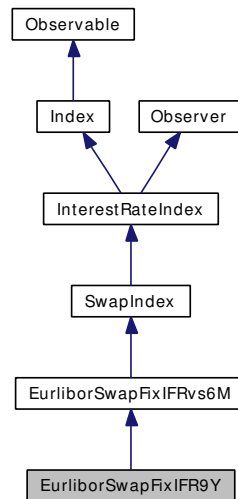
Public Member Functions

- EurliborSwapFixIFR8Y (const [Handle](#)< [YieldTermStructure](#) > &h)

9.380 EurliborSwapFixIFR9Y Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR9Y:



9.380.1 Detailed Description

9-year EurliborSwapFixIFRvs6M index

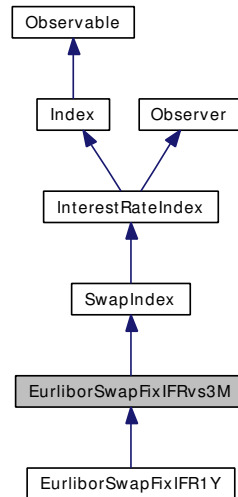
Public Member Functions

- `EurliborSwapFixIFR9Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

9.381 EurliborSwapFixIFRvs3M Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFRvs3M:



9.381.1 Detailed Description

EurliborSwapFixIFR vs 3M index base class

EuriborSwapFix index published by IFR Markets and distributed by Reuters page TGM42281 and by Telerate. For more info see <<http://www.ifrmarkets.com>>.

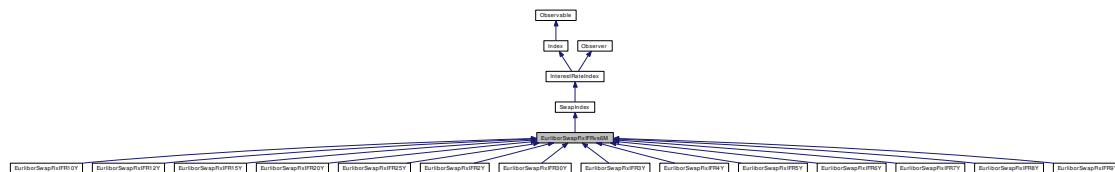
Public Member Functions

- **EurliborSwapFixIFRvs3M** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.382 EurliborSwapFixIFRvs6M Class Reference

```
#include <ql/indexes/swap/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFRvs6M:



9.382.1 Detailed Description

EurliborSwapFixIFR vs 6M index base class

EuriborSwapFix index published by IFR Markets and distributed by Reuters page TGM42281 and by Telerate. For more info see <<http://www.ifrmarkets.com>>.

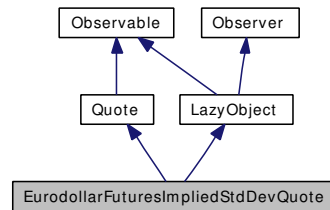
Public Member Functions

- **EurliborSwapFixIFRvs6M** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.383 EurodollarFuturesImpliedStdDevQuote Class Reference

```
#include <ql/quotes/eurodollarfuturesquote.hpp>
```

Inheritance diagram for EurodollarFuturesImpliedStdDevQuote:



9.383.1 Detailed Description

quote for the Eurodollar-future implied standard deviation

Public Member Functions

- **EurodollarFuturesImpliedStdDevQuote** (const [Handle](#)< [Quote](#) > &forward, const [Handle](#)< [Quote](#) > &callPrice, const [Handle](#)< [Quote](#) > &putPrice, Real strike, Real guess=.15, Real accuracy=1.0e-6)
- Real [value](#) () const
returns the current value

Protected Member Functions

- void [performCalculations](#) () const

Protected Attributes

- Real [impliedStdev_](#)
- Real [strike_](#)
- Real [accuracy_](#)
- [Handle](#)< [Quote](#) > [forward_](#)
- [Handle](#)< [Quote](#) > [callPrice_](#)
- [Handle](#)< [Quote](#) > [putPrice_](#)

9.383.2 Member Function Documentation

9.383.2.1 void performCalculations () const [protected, virtual]

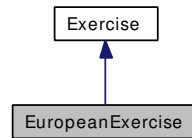
This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implements [LazyObject](#).

9.384 EuropeanExercise Class Reference

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

Inheritance diagram for EuropeanExercise:



9.384.1 Detailed Description

European exercise.

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

Examples:

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

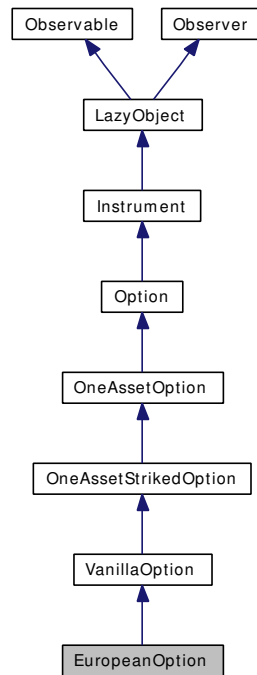
Public Member Functions

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

9.385 EuropeanOption Class Reference

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

Inheritance diagram for EuropeanOption:



9.385.1 Detailed Description

European option on a single asset.

Examples:

[Replication.cpp](#).

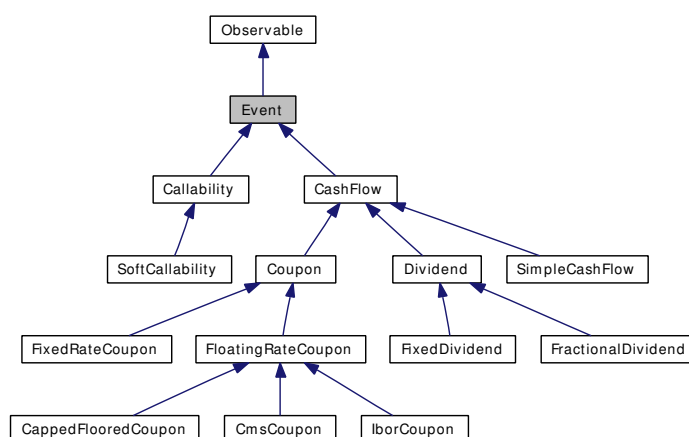
Public Member Functions

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

9.386 Event Class Reference

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

Inheritance diagram for Event:



9.386.1 Detailed Description

Base class for event.

This class acts as a base class for the actual event implementations.

Public Member Functions

Event interface

- virtual `Date date () const=0`
returns the date at which the event occurs
- bool `hasOccurred (const Date &d, bool includeToday=false) const`
returns true if an event has already occurred before a date

Visitability

- virtual void `accept (AcyclicVisitor &)`

9.386.2 Member Function Documentation

9.386.2.1 bool hasOccurred (const Date & d, bool includeToday = false) const

returns true if an event has already occurred before a date

If `QL_TODAYS_PAYMENT` is true, then a payment event has not occurred if the input date is the same as the event date, and so `includeToday` should be defaulted to true.

This should be the only place in the code that is affected directly by `QL_TODAYS_PAYMENT`

Todo

make QL_TODAYS_PAYMENT dynamically configurable?

9.387 EvolutionDescription Class Reference

```
#include <ql/models/marketmodels/evolutiondescription.hpp>
```

9.387.1 Detailed Description

Market-model evolution description.

This class stores:

1. `evolutionTimes` = the times defining the rates that are to be evolved,
2. `rateTimes` = the times at which the rates need to be known,
3. `relevanceRates` = which rates need to be known at each time.

This class is really just a tuple of evolution and rate times;

- there will be $n+1$ rate times expressing payment and reset times of forward rates.
- there will be any number of evolution times.
- we also store which part of the rates are relevant for pricing via relevance rates. The important part for the i -th step will then range from `relevanceRates[i].first` to `relevanceRates[i].second`. Default values for relevance rates will be 0 and n .
- example for $n = 5$:

----- ----- ----- ----- -----	(size = 6)
t0 t1 t2 t3 t4 t5	rateTimes
f0 f1 f2 f3 f4	forwardRates
d0 d1 d2 d3 d4 d5	discountBonds
d0/d0 d1/d0 d2/d0 d3/d0 d4/d0 d5/d0	discountRatios
sr0 sr1 sr2 sr3 sr4	coterminalSwaps

Public Member Functions

- **EvolutionDescription** (const std::vector< Time > &rateTimes, const std::vector< Time > &evolutionTimes=std::vector< Time >(), const std::vector< std::pair< Size, Size > > &relevanceRates=std::vector< range >())
- const std::vector< Time > & **rateTimes** () const
- const std::vector< Time > & **rateTaus** () const
- const std::vector< Time > & **evolutionTimes** () const
- const std::vector< Size > & **firstAliveRate** () const
- const std::vector< std::pair< Size, Size > > & **relevanceRates** () const
- Size **numberOfRates** () const
- Size **numberOfSteps** () const

9.388 ExchangeRate Class Reference

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

9.388.1 Detailed Description

exchange rate between two currencies

Tests

application of direct and derived exchange rate is tested against calculations.

Utility methods

- [Money exchange](#) (const [Money](#) &amount) const
apply the exchange rate to a cash amount
- static [ExchangeRate chain](#) (const [ExchangeRate](#) &r1, const [ExchangeRate](#) &r2)
chain two exchange rates

Public Types

- enum [Type](#) { [Direct](#), [Derived](#) }

Public Member Functions

Constructors

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

Inspectors

- const [Currency](#) & [source](#) () const
the source currency.
- const [Currency](#) & [target](#) () const
the target currency.
- [Type](#) [type](#) () const
the type
- Decimal [rate](#) () const
the exchange rate (when available)

9.388.2 Member Enumeration Documentation

9.388.2.1 enum Type

Enumerator:

Direct given directly by the user

Derived derived from exchange rates between other currencies

9.388.3 Constructor & Destructor Documentation

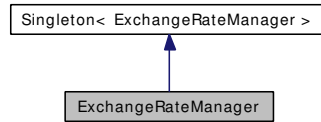
9.388.3.1 ExchangeRate (const Currency & *source*, const Currency & *target*, Decimal *rate*)

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

9.389 ExchangeRateManager Class Reference

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

Inheritance diagram for ExchangeRateManager:



9.389.1 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](#) >

9.389.2 Member Function Documentation

9.389.2.1 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.

9.389.2.2 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.

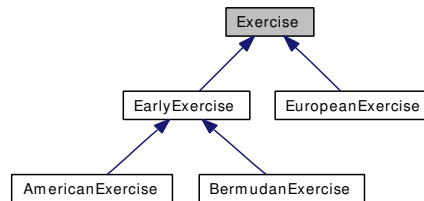
Warning

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

9.390 Exercise Class Reference

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

Inheritance diagram for Exercise:



9.390.1 Detailed Description

Base exercise class.

Public Types

- enum Type { American, Bermudan, European }

Public Member Functions

- Exercise (Type type)
- Type type () const
- Date date (Size index) const
- const std::vector< Date > & dates () const
Returns all exercise dates.
- Date lastDate () const

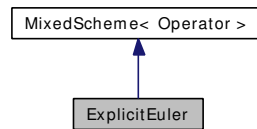
Protected Attributes

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

9.391 ExplicitEuler Class Template Reference

```
#include <ql/methods/finitedifferences/expliciteuler.hpp>
```

Inheritance diagram for ExplicitEuler:



9.391.1 Detailed Description

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

Forward Euler scheme for finite difference methods

See sect. [Finite-differences framework](#) for details on the method.

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

```

typedef ... array_type;

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

// inspectors
Size size();

// modifiers
void setTime(Time t);

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

// operator algebra
Operator operator*(Real, const Operator&);
Operator operator-(const Operator&, const Operator&);

```

Todo

add Richardson extrapolation

Public Types

- `typedef OperatorTraits< Operator > traits`
- `typedef traits::operator_type operator_type`
- `typedef traits::array_type array_type`
- `typedef traits::bc_type bc_type`
- `typedef traits::bc_set bc_set`
- `typedef traits::condition_type condition_type`

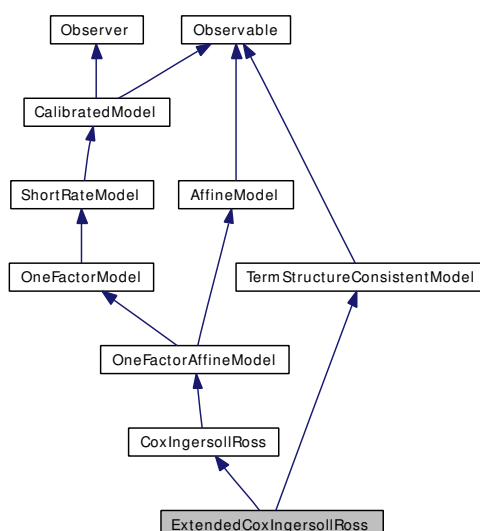
Public Member Functions

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

9.392 ExtendedCoxIngersollRoss Class Reference

```
#include <ql/models/shortrate/onefactormodels/extendedcoxingersollross.hpp>
```

Inheritance diagram for ExtendedCoxIngersollRoss:



9.392.1 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 bondMaturity) const

Protected Member Functions

- void **generateArguments** ()
- Real **A** (Time t, Time T) const

Classes

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

9.393 ExtendedCoxIngersollRoss::Dynamics Class Reference

```
#include <ql/models/shortrate/onefactormodels/extendedcoxingersollross.hpp>
```

9.393.1 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.

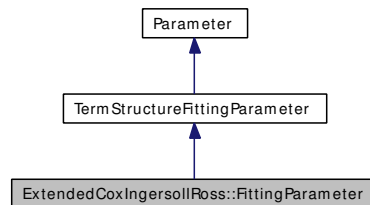
Public Member Functions

- **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

9.394 ExtendedCoxIngersollRoss::FittingParameter Class Reference

```
#include <ql/models/shortrate/onefactormodels/extendedcoxingersollross.hpp>
```

Inheritance diagram for ExtendedCoxIngersollRoss::FittingParameter:



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

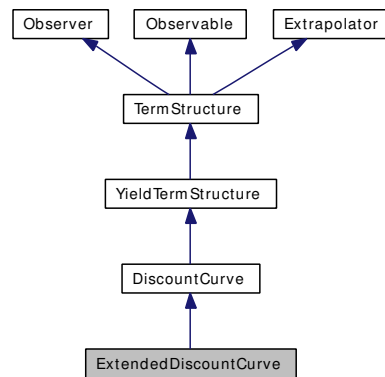
Public Member Functions

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

9.395 ExtendedDiscountCurve Class Reference

```
#include <ql/termstructures/yieldcurves/extendeddiscountcurve.hpp>
```

Inheritance diagram for ExtendedDiscountCurve:



9.395.1 Detailed Description

Term structure based on loglinear interpolation of discount factors.

Loglinear interpolation guarantees piecewise constant forward rates.

Rates are assumed to be annual continuous compounding.

Public Member Functions

- **ExtendedDiscountCurve** (const std::vector< [Date](#) > &dates, const std::vector< [DiscountFactor](#) > &dfs, const [Calendar](#) &calendar, const [BusinessDayConvention](#) conv, const [DayCounter](#) &dayCounter)
- [BusinessDayConvention](#) **businessDayConvention** () const
- void **update** ()
- Rate **compoundForward** (const [Date](#) &d1, Integer f, bool extrapolate=false) const
- Rate **compoundForward** (Time t1, Integer f, bool extrapolate=false) const

Protected Member Functions

- Rate **compoundForwardImpl** (Time, Integer) const
- Rate **zeroYieldImpl** (Time) const
- void **calibrateNodes** () const
- boost::shared_ptr< [CompoundForward](#) > **reversebootstrap** (Integer) const
- boost::shared_ptr< [CompoundForward](#) > **forwardCurve** (Integer) const

9.395.2 Member Function Documentation

9.395.2.1 void update () [virtual]

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

they need to notify any changes.

Reimplemented from [TermStructure](#).

9.395.2.2 Rate compoundForwardImpl (Time, Integer) const [protected]

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

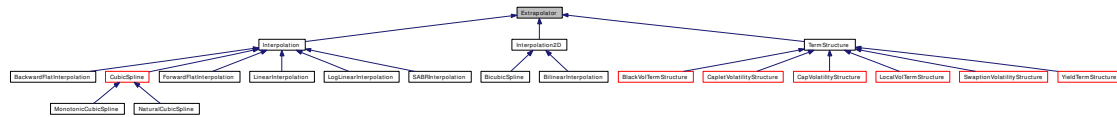
9.395.2.3 Rate zeroYieldImpl (Time) const [protected]

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

9.396 Extrapolator Class Reference

```
#include <ql/math/interpolations/extrapolation.hpp>
```

Inheritance diagram for Extrapolator:



9.396.1 Detailed Description

base class for classes possibly allowing extrapolation

Public Member Functions

modifiers

- void [enableExtrapolation](#) (bool b=true)
enable extrapolation in subsequent calls
- void [disableExtrapolation](#) (bool b=true)
disable extrapolation in subsequent calls

inspectors

- bool [allowsExtrapolation](#) () const
tells whether extrapolation is enabled

9.397 Factorial Class Reference

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

9.397.1 Detailed Description

Factorial numbers calculator

Tests

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

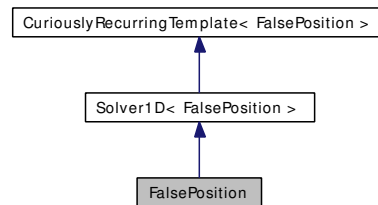
Static Public Member Functions

- static Real **get** (Natural n)
- static Real **ln** (Natural n)

9.398 FalsePosition Class Reference

```
#include <ql/math/solvers1d/falseposition.hpp>
```

Inheritance diagram for FalsePosition:



9.398.1 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`

9.399 FaureRsg Class Reference

```
#include <ql/math/randomnumbers/faurersg.hpp>
```

9.399.1 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](#)< std::vector< Real > > **sample_type**

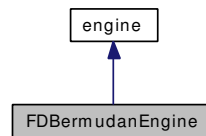
Public Member Functions

- **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

9.400 FDBermudanEngine Class Reference

```
#include <ql/pricingengines/vanilla/fdbermudanengine.hpp>
```

Inheritance diagram for FDBermudanEngine:



9.400.1 Detailed Description

Finite-differences Bermudan engine.

Examples:

[EquityOption.cpp](#).

Public Member Functions

- `FDBermudanEngine` (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)
- `void calculate () const`

Protected Member Functions

- `void initializeStepCondition () const`
- `void executeIntermediateStep (Size) const`

Protected Attributes

- Real `extraTermInBermudan`

9.401 FDDividendEngineMerton73 Class Reference

```
#include <ql/pricingengines/vanilla/fddividendengine.hpp>
```

9.401.1 Detailed Description

Finite-differences pricing engine for dividend options using.

Public Member Functions

- **FDDividendEngineMerton73** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)

9.402 FDDividendEngineShiftScale Class Reference

```
#include <ql/pricingengines/vanilla/fddividendengine.hpp>
```

9.402.1 Detailed Description

Finite-differences pricing engine for dividend options using.

Public Member Functions

- **FDDividendEngineShiftScale** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)

9.403 FDEuropeanEngine Class Reference

```
#include <ql/pricingengines/vanilla/fdeuropeanengine.hpp>
```

9.403.1 Detailed Description

Pricing engine for European options using finite-differences.

Tests

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

Examples:

[EquityOption.cpp](#).

Public Member Functions

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

9.404 FDStepConditionEngine Class Reference

```
#include <ql/pricingengines/vanilla/fdstepconditionengine.hpp>
```

9.404.1 Detailed Description

Finite-differences pricing engine for American-style vanilla options.

Public Member Functions

- **FDStepConditionEngine** (Size timeSteps, Size gridPoints, bool timeDependent=false)

Protected Member Functions

- virtual void **initializeStepCondition** () const=0
- virtual void **calculate** (PricingEngine::results *) const

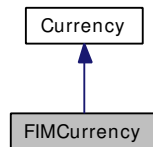
Protected Attributes

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

9.405 FIMCurrency Class Reference

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

Inheritance diagram for FIMCurrency:



9.405.1 Detailed Description

Finnish markka.

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

Obsoleted by the Euro since 1999.

9.406 FiniteDifferenceModel Class Template Reference

```
#include <ql/methods/finitedifferences/finitedifferencemodel.hpp>
```

9.406.1 Detailed Description

```
template<class Evolver> class QuantLib::FiniteDifferenceModel< Evolver >
```

Generic finite difference model.

Public Types

- typedef Evolver::traits **traits**
- typedef traits::operator_type **operator_type**
- typedef traits::array_type **array_type**
- typedef traits::bc_set **bc_set**
- typedef traits::condition_type **condition_type**

Public Member Functions

- **FiniteDifferenceModel** (const operator_type &L, const bc_set &bcs, const std::vector< Time > &stoppingTimes=std::vector< Time >())
- **FiniteDifferenceModel** (const Evolver &evolver, const std::vector< Time > &stoppingTimes=std::vector< Time >())
- const Evolver & **evolver** () const
- void **rollback** (array_type &a, Time from, Time to, Size steps)
- void **rollback** (array_type &a, Time from, Time to, Size steps, const condition_type &condition)

9.406.2 Member Function Documentation

9.406.2.1 void rollback (array_type & a, Time from, Time to, Size steps)

solves the problem between the given times.

Warning

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

9.406.2.2 void rollback (array_type & a, Time from, Time to, Size steps, const condition_type & condition)

solves the problem between the given times, applying a condition at every step.

Warning

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

9.407 Finland Class Reference

```
#include <ql/time/calendars/finland.hpp>
```

Inheritance diagram for Finland:



9.407.1 Detailed Description

Finnish calendar.

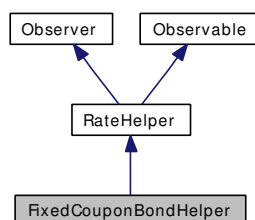
Holidays:

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

9.408 FixedCouponBondHelper Class Reference

```
#include <ql/termstructures/yieldcurves/bondhelpers.hpp>
```

Inheritance diagram for FixedCouponBondHelper:



9.408.1 Detailed Description

fixed-coupon bond helper

Warning

This class assumes that the reference date does not change between calls of [setTermStructure\(\)](#).

Public Member Functions

- **FixedCouponBondHelper** (const [Handle](#)< [Quote](#) > &cleanPrice, Natural settlementDays, const [Schedule](#) &schedule, const std::vector< Rate > &coupons, const [DayCounter](#) &paymentDayCounter, [BusinessDayConvention](#) paymentConvention=Following, Real redemption=100.0, const [Date](#) &issueDate=[Date](#)())
- Real **impliedQuote** () const
- void **setTermStructure** ([YieldTermStructure](#) *)

sets the term structure to be used for pricing

Protected Attributes

- Natural **settlementDays_**
- [Schedule](#) **schedule_**
- std::vector< Rate > **coupons_**
- [DayCounter](#) **paymentDayCounter_**
- [BusinessDayConvention](#) **paymentConvention_**
- Real **redemption_**
- [Date](#) **issueDate_**
- boost::shared_ptr< [FixedRateBond](#) > **bond_**
- [RelinkableHandle](#)< [YieldTermStructure](#) > **termStructureHandle_**

9.408.2 Member Function Documentation

9.408.2.1 void setTermStructure (YieldTermStructure *) [virtual]

sets the term structure to be used for pricing

Warning

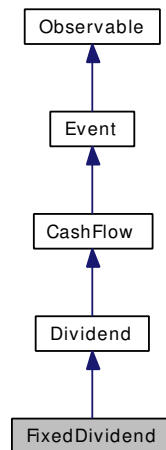
Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the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

Reimplemented from [RateHelper](#).

9.409 FixedDividend Class Reference

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

Inheritance diagram for FixedDividend:



9.409.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

Examples:

[ConvertibleBonds.cpp](#).

Public Member Functions

- **FixedDividend** (Real amount, const [Date](#) &date)

Dividend interface

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

Protected Attributes

- Real **amount_**

9.409.2 Member Function Documentation

9.409.2.1 virtual Real amount () const [virtual]

returns the amount of the cash flow

Note:

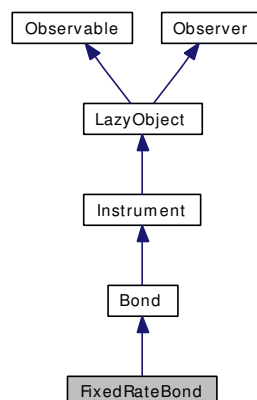
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [Dividend](#).

9.410 FixedRateBond Class Reference

```
#include <ql/instruments/fixedratebond.hpp>
```

Inheritance diagram for FixedRateBond:



9.410.1 Detailed Description

fixed-rate bond

Tests

calculations are tested by checking results against cached values.

Examples:

[Repo.cpp](#).

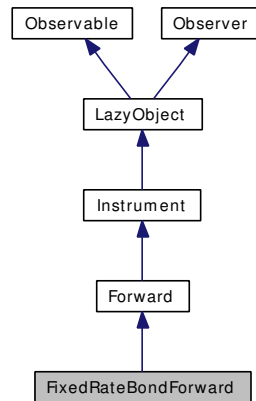
Public Member Functions

- **FixedRateBond** (Natural settlementDays, Real faceAmount, const [Schedule](#) &schedule, const std::vector< Rate > &coupons, const [DayCounter](#) &accrualDayCounter, [BusinessDayConvention](#) paymentConvention=Following, Real redemption=100.0, const [Date](#) &issueDate=[Date](#)(), const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())
- **FixedRateBond** (Natural settlementDays, Real faceAmount, const [Date](#) &startDate, const [Date](#) &maturityDate, [Frequency](#) couponFrequency, const [Calendar](#) &calendar, const std::vector< Rate > &coupons, const [DayCounter](#) &accrualDayCounter, [BusinessDayConvention](#) accrualConvention=Following, [BusinessDayConvention](#) paymentConvention=Following, Real redemption=100.0, const [Date](#) &issueDate=[Date](#)(), const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >(), const [Date](#) &stubDate=[Date](#)(), bool fromEnd=true)

9.411 FixedRateBondForward Class Reference

```
#include <ql/instruments/fixedratebondforward.hpp>
```

Inheritance diagram for FixedRateBondForward:



9.411.1 Detailed Description

Forward contract on a fixed-rate bond

1. `valueDate` refers to the settlement date of the bond forward contract. `maturityDate` is the delivery (or repurchase) date for the underlying bond (not the bond's maturity date).

2. Relevant formulas used in the calculations (P refers to a price):

a. $P_{CleanFwd}(t) = P_{DirtyFwd}(t) - AI(t = deliveryDate)$ where AI refers to the accrued interest on the underlying bond.

b. $P_{DirtyFwd}(t) = \frac{P_{DirtySpot}(t) - SpotIncome(t)}{discountCurve \rightarrow discount(t=deliveryDate)}$

c. $SpotIncome(t) = \sum_i (CF_i \times incomeDiscountCurve \rightarrow discount(t_i))$ where CF_i represents the i th bond cash flow (coupon payment) associated with the underlying bond falling between the `settlementDate` and the `deliveryDate`. (Note the two different discount curves used in b. and c.)

Example: [valuation of a repo on a fixed-rate bond](#)

Todo

Add preconditions and tests

Todo

Create switch- if coupon goes to seller is toggled on, don't consider income in the $P_{DirtyFwd}(t)$ calculation.

Todo

Verify this works when the underlying is paper (in which case ignore all AI.)

Warning

This class still needs to be rigorously tested

Examples:

[Repo.cpp](#).

Public Member Functions

Constructors

- **FixedRateBondForward** (const [Date](#) &valueDate, const [Date](#) &maturityDate, [Position::Type](#) type, Real strike, Natural settlementDays, const [DayCounter](#) &dayCounter, const [Calendar](#) &calendar, [BusinessDayConvention](#) businessDayConvention, const boost::shared_ptr< [FixedRateBond](#) > &fixedCouponBond, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >(), const [Handle](#)< [YieldTermStructure](#) > &incomeDiscountCurve=[Handle](#)< [YieldTermStructure](#) >())

Calculations

- Real [forwardPrice](#) () const
(dirty) forward bond price
- Real [cleanForwardPrice](#) () const
(dirty) forward bond price minus accrued on bond at delivery
- Real [spotIncome](#) (const [Handle](#)< [YieldTermStructure](#) > &incomeDiscountCurve) const
NPV of bond coupons discounted using incomeDiscountCurve.
- Real [spotValue](#) () const
NPV of underlying bond.

Protected Member Functions

- void [performCalculations](#) () const

Protected Attributes

- boost::shared_ptr< [FixedRateBond](#) > fixedCouponBond_

9.411.2 Constructor & Destructor Documentation

9.411.2.1 FixedRateBondForward (const *Date* & valueDate, const *Date* & maturityDate, *Position::Type* type, Real strike, Natural settlementDays, const *DayCounter* & dayCounter, const *Calendar* & calendar, *BusinessDayConvention* businessDayConvention, const boost::shared_ptr< *FixedRateBond* > & fixedCouponBond, const *Handle*< *YieldTermStructure* > & discountCurve = *Handle*< *YieldTermStructure* >(), const *Handle*< *YieldTermStructure* > & incomeDiscountCurve = *Handle*< *YieldTermStructure* >())

If strike is given in the constructor, can calculate the NPV of the contract via [NPV\(\)](#).

If strike/forward price is desired, it can be obtained via [forwardPrice\(\)](#). In this case, the strike variable in the constructor is irrelevant and will be ignored.

9.411.3 Member Function Documentation

9.411.3.1 Real `spotIncome` (`const Handle< YieldTermStructure > & incomeDiscountCurve`) `const` [virtual]

NPV of bond coupons discounted using `incomeDiscountCurve`.

Here only coupons between `max(evaluation date, settlement date)` and maturity date of bond forward contract are considered income.

Implements [Forward](#).

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

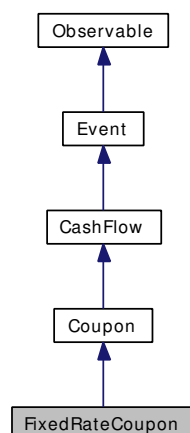
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Forward](#).

9.412 FixedRateCoupon Class Reference

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

Inheritance diagram for FixedRateCoupon:



9.412.1 Detailed Description

Coupon paying a fixed interest rate

Public Member Functions

- **FixedRateCoupon** (Real nominal, const [Date](#) &paymentDate, Rate rate, const [DayCounter](#) &dayCounter, const [Date](#) &accrualStartDate, const [Date](#) &accrualEndDate, 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](#) &)

9.412.2 Member Function Documentation

9.412.2.1 Real amount () const [virtual]

returns the amount of the cash flow

Note:

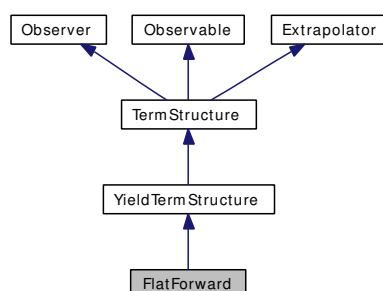
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

9.413 FlatForward Class Reference

```
#include <ql/termstructures/yieldcurves/flatforward.hpp>
```

Inheritance diagram for FlatForward:



9.413.1 Detailed Description

Flat interest-rate curve.

Examples:

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

Public Member Functions

- **FlatForward** (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &forward, const [DayCounter](#) &dayCounter, Compounding compounding=Continuous, [Frequency](#) frequency=Annual)
- **FlatForward** (const [Date](#) &referenceDate, Rate forward, const [DayCounter](#) &dayCounter, Compounding compounding=Continuous, [Frequency](#) frequency=Annual)
- **FlatForward** (Natural settlementDays, const [Calendar](#) &calendar, const [Handle](#)< [Quote](#) > &forward, const [DayCounter](#) &dayCounter, Compounding compounding=Continuous, [Frequency](#) frequency=Annual)
- **FlatForward** (Natural settlementDays, const [Calendar](#) &calendar, Rate forward, const [DayCounter](#) &dayCounter, Compounding compounding=Continuous, [Frequency](#) frequency=Annual)
- Compounding **compounding** () const
- [Frequency](#) **compoundingFrequency** () const
- [Date](#) **maxDate** () const
the latest date for which the curve can return values
- void **update** ()

9.413.2 Member Function Documentation

9.413.2.1 `void update ()` [virtual]

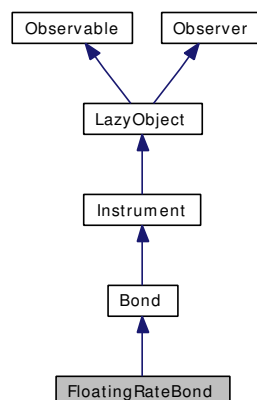
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

9.414 FloatingRateBond Class Reference

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

Inheritance diagram for FloatingRateBond:



9.414.1 Detailed Description

floating-rate bond (possibly capped and/or floored)

Tests

calculations are tested by checking results against cached values.

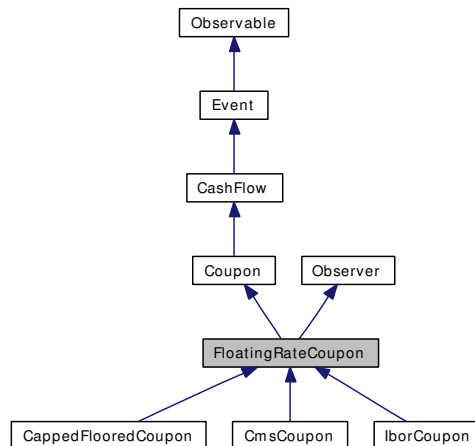
Public Member Functions

- **FloatingRateBond** (Natural settlementDays, Real faceAmount, const [Schedule](#) &sched-
ule, const boost::shared_ptr< [IborIndex](#) > &index, const [DayCounter](#) &accrualDayCounter,
[BusinessDayConvention](#) paymentConvention=Following, Natural fixingDays=[Null](#)< Nat-
ural >(), const std::vector< Real > &gearings=std::vector< Real >(1, 1.0), const
std::vector< Spread > &spreads=std::vector< Spread >(1, 0.0), const std::vector< Rate >
&caps=std::vector< Rate >(), const std::vector< Rate > &floors=std::vector< Rate >(), bool
inArrears=false, Real redemption=100.0, const [Date](#) &issueDate=[Date](#)(), const [Handle](#)<
[YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())
- **FloatingRateBond** (Natural settlementDays, Real faceAmount, const [Date](#) &startDate,
const [Date](#) &maturityDate, [Frequency](#) couponFrequency, const [Calendar](#) &calendar, const
boost::shared_ptr< [IborIndex](#) > &index, const [DayCounter](#) &accrualDayCounter, [Busi-
nessDayConvention](#) accrualConvention=Following, [BusinessDayConvention](#) payment-
Convention=Following, Natural fixingDays=[Null](#)< Natural >(), const std::vector< Real >
&gearings=std::vector< Real >(1, 1.0), const std::vector< Spread > &spreads=std::vector<
Spread >(1, 0.0), const std::vector< Rate > &caps=std::vector< Rate >(), const std::vector<
Rate > &floors=std::vector< Rate >(), bool inArrears=false, Real redemption=100.0, const
[Date](#) &issueDate=[Date](#)(), const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)<
[YieldTermStructure](#) >(), const [Date](#) &stubDate=[Date](#)(), bool fromEnd=true)

9.415 FloatingRateCoupon Class Reference

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

Inheritance diagram for FloatingRateCoupon:



9.415.1 Detailed Description

base floating-rate coupon class

Public Member Functions

- **FloatingRateCoupon** (const [Date](#) &paymentDate, const Real nominal, const [Date](#) &startDate, const [Date](#) &endDate, const Natural fixingDays, const boost::shared_ptr< [InterestRateIndex](#) > &index, const Real gearing=1.0, const Spread spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)(), bool isInArrears=false)
- void **setPricer** (const boost::shared_ptr< [FloatingRateCouponPricer](#) > &pricer)
- boost::shared_ptr< [FloatingRateCouponPricer](#) > **pricer** () const

CashFlow interface

- Real **amount** () const
returns the amount of the cash flow

Coupon interface

- Rate **rate** () const
accrued rate
- Real **price** (const [Handle](#)< [YieldTermStructure](#) > &discountingCurve) const
- [DayCounter](#) **dayCounter** () const
day counter for accrual calculation

- Real `accruedAmount` (const `Date` &) const
accrued amount at the given date

Inspectors

- const boost::shared_ptr< `InterestRateIndex` > & `index` () const
floating index
- Natural `fixingDays` () const
fixing days
- virtual `Date` `fixingDate` () const
fixing date
- Real `gearing` () const
index gearing, i.e. multiplicative coefficient for the index
- Spread `spread` () const
spread paid over the fixing of the underlying index
- Rate `indexFixing` () const
fixing of the underlying index
- Rate `convexityAdjustment` () const
convexity adjustment
- Rate `adjustedFixing` () const
convexity-adjusted fixing
- bool `isInArrears` () const

Observer interface

- void `update` ()

Visitability

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

Protected Member Functions

- Rate `convexityAdjustmentImpl` (Rate fixing) const
convexity adjustment for the given index fixing

Protected Attributes

- boost::shared_ptr< `InterestRateIndex` > `index_`
- `DayCounter` `dayCounter_`
- Natural `fixingDays_`
- Real `gearing_`
- Spread `spread_`
- bool `isInArrears_`
- boost::shared_ptr< `FloatingRateCouponPricer` > `pricer_`

9.415.2 Member Function Documentation

9.415.2.1 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](#).

9.415.2.2 void update () [virtual]

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

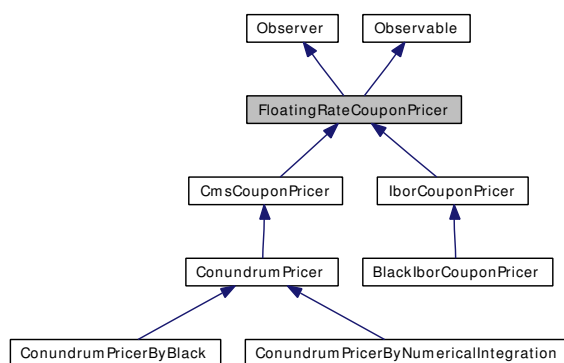
Implements [Observer](#).

Reimplemented in [CappedFlooredCoupon](#).

9.416 FloatingRateCouponPricer Class Reference

#include <ql/cashflows/couponpricer.hpp>

Inheritance diagram for FloatingRateCouponPricer:



9.416.1 Detailed Description

generic pricer for floating-rate coupons

Public Member Functions

required interface

- virtual Real **swapletPrice** () const=0
- virtual Rate **swapletRate** () const=0
- virtual Real **capletPrice** (Rate effectiveCap) const=0
- virtual Rate **capletRate** (Rate effectiveCap) const=0
- virtual Real **floorletPrice** (Rate effectiveFloor) const =0
- virtual Rate **floorletRate** (Rate effectiveFloor) const =0
- virtual void **initialize** (const [FloatingRateCoupon](#) &coupon)=0

Observer interface

- void [update](#) ()

9.416.2 Member Function Documentation

9.416.2.1 void update () [virtual]

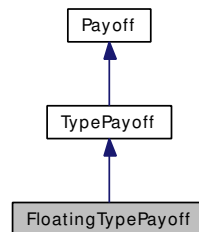
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.417 FloatingTypePayoff Class Reference

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

Inheritance diagram for FloatingTypePayoff:



9.417.1 Detailed Description

Payoff based on a floating strike

Public Member Functions

- **FloatingTypePayoff** (Option::Type type)

Payoff interface

- `std::string name () const`
- `Real operator() (Real price) const`
- `virtual void accept (AcyclicVisitor &)`

9.417.2 Member Function Documentation

9.417.2.1 `std::string name () const` [virtual]

Warning

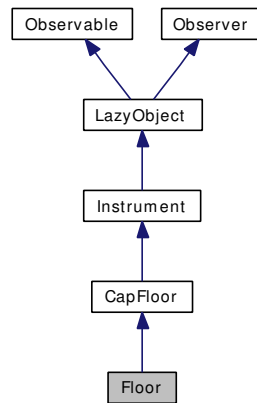
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implements [Payoff](#).

9.418 Floor Class Reference

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

Inheritance diagram for Floor:



9.418.1 Detailed Description

Concrete floor class.

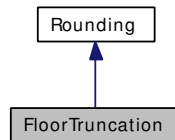
Public Member Functions

- **Floor** (const Leg &floatingLeg, const std::vector< Rate > &exerciseRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared_ptr< [PricingEngine](#) > &engine)

9.419 FloorTruncation Class Reference

```
#include <ql/math/rounding.hpp>
```

Inheritance diagram for FloorTruncation:



9.419.1 Detailed Description

Floor truncation.

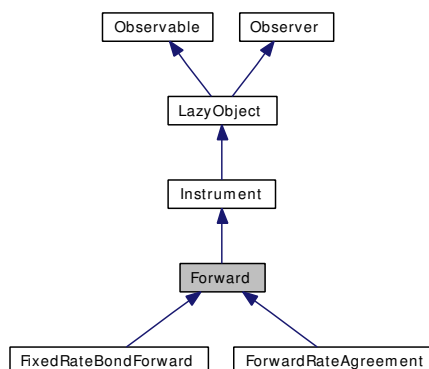
Public Member Functions

- **FloorTruncation** (Integer precision, Integer digit=5)

9.420 Forward Class Reference

```
#include <ql/instruments/forward.hpp>
```

Inheritance diagram for Forward:



9.420.1 Detailed Description

Abstract base forward class.

Derived classes must implement the virtual functions `spotValue()` (NPV or spot price) and `spotIncome()` associated with the specific relevant underlying (e.g. bond, stock, commodity, loan/deposit). These functions must be used to set the protected member variables `underlyingSpotValue_` and `underlyingIncome_` within `performCalculations()` in the derived class before the base-class implementation is called.

`spotIncome()` refers generically to the present value of coupons, dividends or storage costs.

`discountCurve_` is the curve used to discount forward contract cash flows back to the evaluation day, as well as to obtain forward values for spot values/prices.

`incomeDiscountCurve_`, which for generality is not automatically set to the `discountCurve_`, is the curve used to discount future income/dividends/storage-costs etc back to the evaluation date.

Todo

Add preconditions and tests

Warning

This class still needs to be rigorously tested

Public Member Functions

- virtual Real `spotValue()` const=0
returns spot value/price of an underlying financial instrument
- virtual Real `spotIncome` (const `Handle< YieldTermStructure >` &incomeDiscountCurve) const=0
NPV of income/dividends/storage-costs etc. of underlying instrument.

Inspectors

- virtual [Date](#) **settlementDate** () const
- const [Calendar](#) & **calendar** () const
- [BusinessDayConvention](#) **businessDayConvention** () const
- const [DayCounter](#) & **dayCounter** () const
- [Handle](#)< [YieldTermStructure](#) > **discountCurve** () const
term structure relevant to the contract (e.g. repo curve)
- [Handle](#)< [YieldTermStructure](#) > **incomeDiscountCurve** () const
term structure that discounts the underlying's income cash flows
- bool **isExpired** () const
returns whether the instrument is still tradable.

Calculations

- virtual Real **forwardValue** () const
forward value/price of underlying, discounting income/dividends
- [InterestRate](#) **impliedYield** (Real underlyingSpotValue, Real forwardValue, [Date](#) settlementDate, Compounding compoundingConvention, [DayCounter](#) dayCounter)

Protected Member Functions

- **Forward** (const [DayCounter](#) &dayCounter, const [Calendar](#) &calendar, [BusinessDayConvention](#) businessDayConvention, Natural settlementDays, const boost::shared_ptr< [Payoff](#) > &payoff, const [Date](#) &valueDate, const [Date](#) &maturityDate, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())
- void **performCalculations** () const

Protected Attributes

- Real **underlyingIncome_**
- Real **underlyingSpotValue_**
- [DayCounter](#) **dayCounter_**
- [Calendar](#) **calendar_**
- [BusinessDayConvention](#) **businessDayConvention_**
- Natural **settlementDays_**
- boost::shared_ptr< [Payoff](#) > **payoff_**
- [Date](#) **valueDate_**
- [Date](#) **maturityDate_**
maturityDate of the forward contract or delivery date of underlying
- [Handle](#)< [YieldTermStructure](#) > **discountCurve_**
- [Handle](#)< [YieldTermStructure](#) > **incomeDiscountCurve_**

9.420.2 Member Function Documentation

9.420.2.1 virtual Real forwardValue () const [virtual]

forward value/price of underlying, discounting income/dividends

Note:

if this is a bond forward price, it must be a dirty forward price.

9.420.2.2 InterestRate impliedYield (Real *underlyingSpotValue*, Real *forwardValue*, Date *settlementDate*, Compounding *compoundingConvention*, DayCounter *dayCounter*)

Simple yield calculation based on underlying spot and forward values, taking into account underlying income. When $t > 0$, call with: `underlyingSpotValue=spotValue(t)`, `forwardValue=strikePrice`, to get current yield. For a repo, if $t = 0$, `impliedYield` should reproduce the spot repo rate. For FRA's, this should reproduce the relevant zero rate at the FRA's maturityDate_;

9.420.2.3 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

Reimplemented in [FixedRateBondForward](#), and [ForwardRateAgreement](#).

9.420.3 Member Data Documentation

9.420.3.1 Real underlyingIncome_ [mutable, protected]

derived classes must set this, typically via [spotIncome\(\)](#)

9.420.3.2 Real underlyingSpotValue_ [mutable, protected]

derived classes must set this, typically via [spotValue\(\)](#)

9.420.3.3 Date valueDate_ [protected]

valueDate = settlement date (date the fwd contract starts accruing)

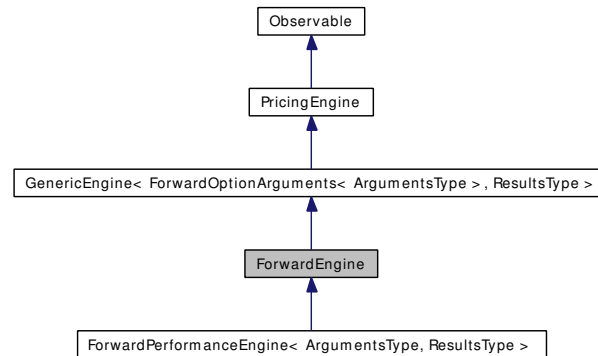
9.420.3.4 Handle<YieldTermStructure> incomeDiscountCurve_ [protected]

must set this in derived classes, based on particular underlying

9.421 ForwardEngine Class Template Reference

```
#include <ql/pricingengines/forward/forwardengine.hpp>
```

Inheritance diagram for ForwardEngine:



9.421.1 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_**

9.422 ForwardFlat Class Reference

```
#include <ql/math/interpolations/forwardflatinterpolation.hpp>
```

9.422.1 Detailed Description

Forward-flat interpolation factory and traits.

Public Types

- enum { **global** = 0 }

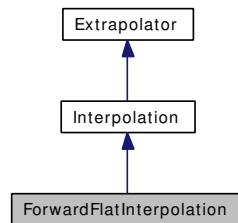
Public Member Functions

- template<class I1, class I2>
[Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

9.423 ForwardFlatInterpolation Class Reference

#include <ql/math/interpolations/forwardflatinterpolation.hpp>

Inheritance diagram for ForwardFlatInterpolation:



9.423.1 Detailed Description

Forward-flat interpolation between discrete points.

Public Member Functions

- `template<class I1, class I2>`
`ForwardFlatInterpolation` (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

9.423.2 Constructor & Destructor Documentation

9.423.2.1 ForwardFlatInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

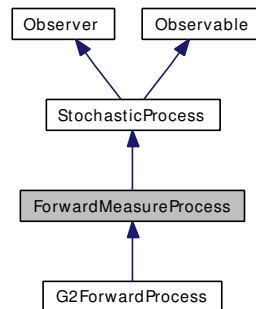
Precondition:

the *x* values must be sorted.

9.424 ForwardMeasureProcess Class Reference

```
#include <ql/processes/forwardmeasureprocess.hpp>
```

Inheritance diagram for ForwardMeasureProcess:



9.424.1 Detailed Description

forward-measure stochastic process

stochastic process whose dynamics are expressed in the forward measure.

Public Member Functions

- void **setForwardMeasureTime** (Time)

Protected Member Functions

- **ForwardMeasureProcess** (Time T)
- **ForwardMeasureProcess** (const boost::shared_ptr< discretization > &)

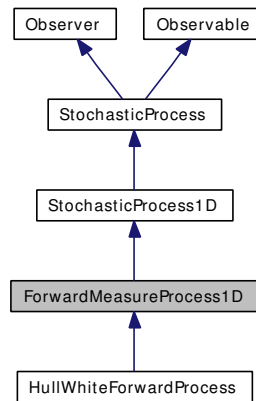
Protected Attributes

- Time T_

9.425 ForwardMeasureProcess1D Class Reference

```
#include <ql/processes/forwardmeasureprocess.hpp>
```

Inheritance diagram for ForwardMeasureProcess1D:



9.425.1 Detailed Description

forward-measure 1-D stochastic process

1-D stochastic process whose dynamics are expressed in the forward measure.

Public Member Functions

- void **setForwardMeasureTime** (Time)

Protected Member Functions

- **ForwardMeasureProcess1D** (Time T)
- **ForwardMeasureProcess1D** (const boost::shared_ptr< discretization > &)

Protected Attributes

- Time T_

9.426 ForwardOptionArguments Class Template Reference

```
#include <ql/pricingengines/forward/forwardengine.hpp>
```

9.426.1 Detailed Description

```
template<class ArgumentsType> class QuantLib::ForwardOptionArguments< ArgumentsType >
```

Arguments for forward (strike-resetting) option calculation

Public Member Functions

- void `validate ()` const

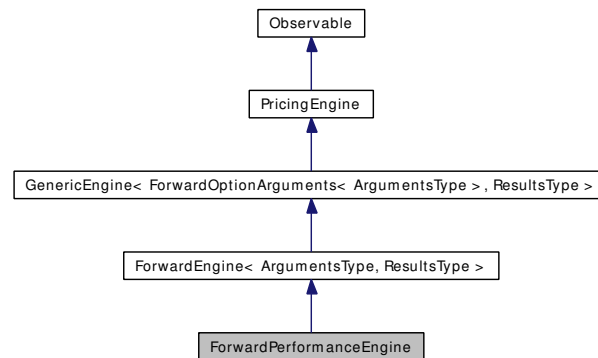
Public Attributes

- Real `moneyness`
- [Date](#) `resetDate`

9.427 ForwardPerformanceEngine Class Template Reference

```
#include <ql/pricingengines/forward/forwardperformanceengine.hpp>
```

Inheritance diagram for ForwardPerformanceEngine:



9.427.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class
QuantLib::ForwardPerformanceEngine< ArgumentsType, ResultsType >
```

Forward performance engine

Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **ForwardPerformanceEngine** (const boost::shared_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > > &)
- void **calculate** () const
- void **getOriginalResults** () const

9.428 ForwardRate Struct Reference

```
#include <ql/termstructures/yieldcurves/bootstraptraits.hpp>
```

9.428.1 Detailed Description

Forward-curve traits.

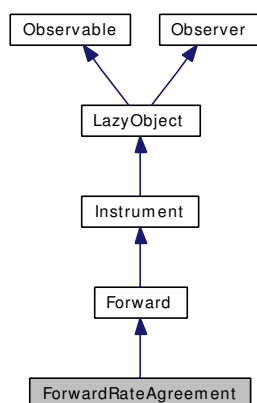
Static Public Member Functions

- static Rate **initialValue** ()
- static Rate **initialGuess** ()
- static Rate **guess** (const [YieldTermStructure](#) *c, const [Date](#) &d)
- static Rate **minValueAfter** (Size, const std::vector< Real > &)
- static Rate **maxValueAfter** (Size, const std::vector< Real > &)
- static void **updateGuess** (std::vector< Rate > &data, Rate forward, Size i)

9.429 ForwardRateAgreement Class Reference

```
#include <ql/instruments/forwardrateagreement.hpp>
```

Inheritance diagram for ForwardRateAgreement:



9.429.1 Detailed Description

Forward rate agreement (FRA) class

1. Unlike the forward contract conventions on carryable financial assets (stocks, bonds, commodities), the valueDate for a FRA is taken to be the day when the forward loan or deposit begins and when full settlement takes place (based on the NPV of the contract on that date). maturityDate is the date when the forward loan or deposit ends. In fact, the FRA settles and expires on the valueDate, not on the (later) maturityDate. It follows that (maturityDate - valueDate) is the tenor/term of the underlying loan or deposit
2. Choose position type = Long for an "FRA purchase" (future long loan, short deposit [borrower])
3. Choose position type = Short for an "FRA sale" (future short loan, long deposit [lender])
4. If strike is given in the constructor, can calculate the NPV of the contract via [NPV\(\)](#).
5. If forward rate is desired/unknown, it can be obtained via [forwardRate\(\)](#). In this case, the strike variable in the constructor is irrelevant and will be ignored.

Example: [valuation of a forward-rate agreement](#)

Todo

Add preconditions and tests

Todo

Should put an instance of [ForwardRateAgreement](#) in the [FraRateHelper](#) to ensure consistency with the piecewise yield curve.

Todo

Differentiate between BBA (British)/AFB (French) [assumed here] and ABA (Australian) banker conventions in the calculations.

Warning

This class still needs to be rigorously tested

Examples:

[FRA.cpp](#).

Public Member Functions

- **ForwardRateAgreement** (const [Date](#) &valueDate, const [Date](#) &maturityDate, Position::Type type, Rate strikeForwardRate, Real notionalAmount, const boost::shared_ptr<[IborIndex](#)> &index, const [Handle](#)<[YieldTermStructure](#)> &discountCurve=[Handle](#)<[YieldTermStructure](#)>())

Calculations

- bool [isExpired](#) () const
- [Date](#) [settlementDate](#) () const
- Real [spotIncome](#) (const [Handle](#)<[YieldTermStructure](#)> &incomeDiscountCurve) const
- Real [spotValue](#) () const
Spot value (NPV) of the underlying loan.
- [InterestRate](#) [forwardRate](#) () const
Returns the relevant forward rate associated with the FRA term.

Protected Member Functions

- void [performCalculations](#) () const

Protected Attributes

- Position::Type [fraType_](#)
- [InterestRate](#) [forwardRate_](#)
aka FRA rate (the market forward rate)
- [InterestRate](#) [strikeForwardRate_](#)
aka FRA fixing rate, contract rate
- Real [notionalAmount_](#)
- boost::shared_ptr<[IborIndex](#)> [index_](#)

9.429.2 Member Function Documentation

9.429.2.1 bool isExpired () const [virtual]

A FRA expires/settles on the valueDate

Reimplemented from [Forward](#).

9.429.2.2 Date settlementDate () const [virtual]

This returns evaluationDate + settlementDays (not FRA valueDate).

Reimplemented from [Forward](#).

9.429.2.3 Real spotIncome (const Handle< YieldTermStructure > & incomeDiscountCurve) const [virtual]

Income is zero for a FRA

Implements [Forward](#).

9.429.2.4 Real spotValue () const [virtual]

Spot value (NPV) of the underlying loan.

This has always a positive value (asset), even if short the FRA

Implements [Forward](#).

9.429.2.5 void performCalculations () const [protected, virtual]

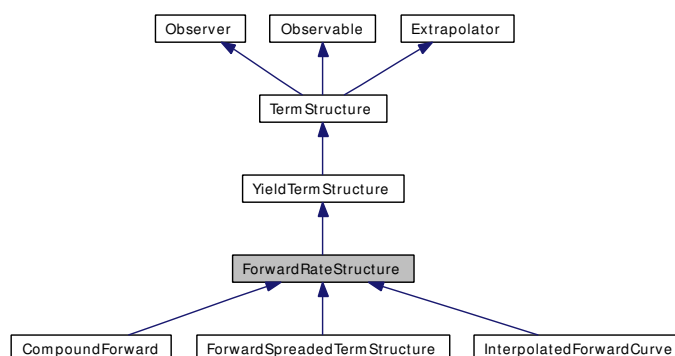
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Forward](#).

9.430 ForwardRateStructure Class Reference

```
#include <ql/termstructures/yieldcurves/forwardstructure.hpp>
```

Inheritance diagram for ForwardRateStructure:



9.430.1 Detailed Description

Forward-rate term structure

This abstract class acts as an adapter to [TermStructure](#) allowing the programmer to implement only the `forwardImpl(const Date&, bool)` method in derived classes. Zero yields and discounts are calculated from forwards.

Rates are assumed to be annual continuous compounding.

Public Member Functions

Constructors

See the [TermStructure](#) documentation for issues regarding constructors.

- **ForwardRateStructure** (const [DayCounter](#) &dayCounter=[Actual365Fixed](#)())
- **ForwardRateStructure** (const [Date](#) &referenceDate, const [Calendar](#) &cal=[Calendar](#)(), const [DayCounter](#) &dayCounter=[Actual365Fixed](#)())
- **ForwardRateStructure** (Natural settlementDays, const [Calendar](#) &, const [DayCounter](#) &dayCounter=[Actual365Fixed](#)())

Protected Member Functions

YieldTermStructure implementation

- DiscountFactor [discountImpl](#) (Time) const
- virtual Rate [forwardImpl](#) (Time) const=0
instantaneous forward-rate calculation
- virtual Rate [zeroYieldImpl](#) (Time) const

9.430.2 Member Function Documentation

9.430.2.1 DiscountFactor discountImpl (Time) const [protected, virtual]

Returns the discount factor for the given date calculating it from the instantaneous forward rate.

Implements [YieldTermStructure](#).

Reimplemented in [CompoundForward](#).

9.430.2.2 Rate zeroYieldImpl (Time) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

Warning

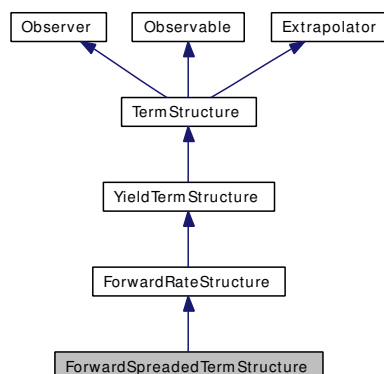
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

Reimplemented in [CompoundForward](#), [InterpolatedForwardCurve](#), and [ForwardSpread-edTermStructure](#).

9.431 ForwardSpreadedTermStructure Class Reference

```
#include <ql/termstructures/yieldcurves/forwardspreadedtermstructure.hpp>
```

Inheritance diagram for ForwardSpreadedTermStructure:



9.431.1 Detailed Description

Term structure with added spread on the instantaneous forward rate.

Note:

This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Tests

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

Public Member Functions

- **ForwardSpreadedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Handle](#)< [Quote](#) > &spread)

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- const [Date](#) & [referenceDate](#) () const
the date at which discount = 1.0 and/or variance = 0.0

- [Date](#) `maxDate` () const
the latest date for which the curve can return values
- [Time](#) `maxTime` () const
the latest time for which the curve can return values

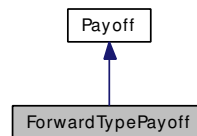
Protected Member Functions

- [Rate](#) `forwardImpl` (Time) const
returns the spreaded forward rate
- [Rate](#) `zeroYieldImpl` (Time) const
returns the spreaded zero yield rate

9.432 ForwardTypePayoff Class Reference

```
#include <ql/instruments/forward.hpp>
```

Inheritance diagram for ForwardTypePayoff:



9.432.1 Detailed Description

Class for forward type payoffs.

Public Member Functions

- **ForwardTypePayoff** (Position::Type type, Real strike)
- Position::Type **forwardType** () const
- Real **strike** () const

Payoff interface

- std::string **name** () const
- std::string **description** () const
- Real **operator()** (Real price) const

Protected Attributes

- Position::Type **type_**
- Real **strike_**

9.432.2 Member Function Documentation

9.432.2.1 std::string name () const [virtual]

Warning

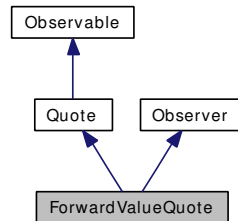
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implements [Payoff](#).

9.433 ForwardValueQuote Class Reference

```
#include <ql/quotes/forwardvaluequote.hpp>
```

Inheritance diagram for ForwardValueQuote:



9.433.1 Detailed Description

quote for the forward value of an index

Public Member Functions

- **ForwardValueQuote** (const boost::shared_ptr< [Index](#) > &index, const [Date](#) &fixingDate)
- Real [value](#) () const
returns the current value
- void [update](#) ()

9.433.2 Member Function Documentation

9.433.2.1 void update () [virtual]

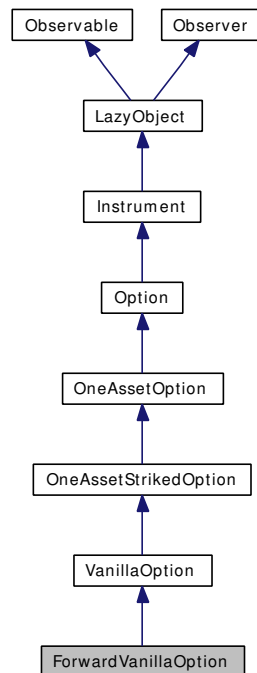
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.434 ForwardVanillaOption Class Reference

```
#include <ql/instruments/forwardvanillaoption.hpp>
```

Inheritance diagram for ForwardVanillaOption:



9.434.1 Detailed Description

Forward version of a vanilla option

Public Types

- typedef [ForwardOptionArguments](#)< [VanillaOption::arguments](#) > **arguments**
- typedef [VanillaOption::results](#) **results**
- typedef [ForwardEngine](#)< [VanillaOption::arguments](#), [VanillaOption::results](#) > **engine**

Public Member Functions

- **ForwardVanillaOption** (Real moneyness, [Date](#) resetDate, const boost::shared_ptr< [StochasticProcess](#) > &stochProc, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine)
- void **setupArguments** ([PricingEngine::arguments](#) *) const
- void **fetchResults** (const [PricingEngine::results](#) *) const

9.434.2 Member Function Documentation

9.434.2.1 `void fetchResults (const PricingEngine::results *) const` [virtual]

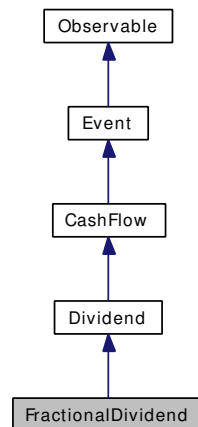
When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetStrikedOption](#).

9.435 FractionalDividend Class Reference

```
#include <ql/cashflows/dividend.hpp>
```

Inheritance diagram for FractionalDividend:



9.435.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

Public Member Functions

- **FractionalDividend** (Real rate, const [Date](#) &date)
- **FractionalDividend** (Real rate, Real nominal, const [Date](#) &date)

Dividend interface

- virtual Real [amount](#) () const
returns the amount of the cash flow
- virtual Real **amount** (Real underlying) const

Inspectors

- Real **rate** () const
- Real **nominal** () const

Protected Attributes

- Real **rate_**
- Real **nominal_**

9.435.2 Member Function Documentation

9.435.2.1 `virtual Real amount () const` [virtual]

returns the amount of the cash flow

Note:

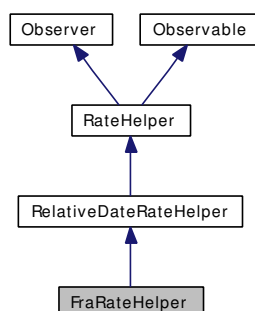
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [Dividend](#).

9.436 FraRateHelper Class Reference

```
#include <ql/termstructures/yieldcurves/ratehelpers.hpp>
```

Inheritance diagram for FraRateHelper:



9.436.1 Detailed Description

Rate helper for bootstrapping over FRA rates.

Examples:

[FRA.cpp](#), and [swapvaluation.cpp](#).

Public Member Functions

- **FraRateHelper** (const [Handle](#)< [Quote](#) > &rate, Natural monthsToStart, Natural monthsToEnd, Natural settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, bool endOfMonth, Natural fixingDays, const [DayCounter](#) &dayCounter)
- **FraRateHelper** (Rate rate, Natural monthsToStart, Natural monthsToEnd, Natural settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, bool endOfMonth, Natural fixingDays, const [DayCounter](#) &dayCounter)
- Real **impliedQuote** () const
- DiscountFactor **discountGuess** () const
- void **setTermStructure** ([YieldTermStructure](#) *)
sets the term structure to be used for pricing

9.436.2 Member Function Documentation

9.436.2.1 void setTermStructure (YieldTermStructure *) [virtual]

sets the term structure to be used for pricing

Warning

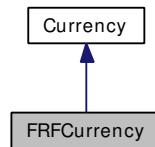
Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

9.437 FRFCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for FRFCurrency:



9.437.1 Detailed Description

French franc.

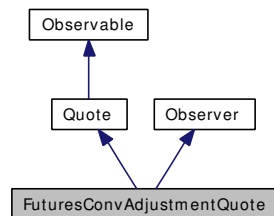
The ISO three-letter code was FRF; the numeric code was 250. It was divided in 100 centimes.

Obsoleted by the Euro since 1999.

9.438 FuturesConvAdjustmentQuote Class Reference

```
#include <ql/quotes/futuresconvadjustmentquote.hpp>
```

Inheritance diagram for FuturesConvAdjustmentQuote:



9.438.1 Detailed Description

quote for the futures-convexity adjustment of an index

Public Member Functions

- **FuturesConvAdjustmentQuote** (const boost::shared_ptr< [IborIndex](#) > &index, const [Date](#) &futuresDate, const [Handle](#)< [Quote](#) > &futuresQuote, const [Handle](#)< [Quote](#) > &volatility, const [Handle](#)< [Quote](#) > &meanReversion)
- **FuturesConvAdjustmentQuote** (const boost::shared_ptr< [IborIndex](#) > &index, const std::string &immCode, const [Handle](#)< [Quote](#) > &futuresQuote, const [Handle](#)< [Quote](#) > &volatility, const [Handle](#)< [Quote](#) > &meanReversion)
- Real [value](#) () const
returns the current value
- void [update](#) ()

Inspectors

- Real **futuresValue** () const
- Real **volatility** () const
- Real **meanReversion** () const
- [Date](#) **immDate** () const

Protected Attributes

- [DayCounter](#) dc_
- const [Date](#) futuresDate_
- const [Date](#) indexMaturityDate_
- [Handle](#)< [Quote](#) > futuresQuote_
- [Handle](#)< [Quote](#) > volatility_
- [Handle](#)< [Quote](#) > meanReversion_

9.438.2 Member Function Documentation

9.438.2.1 void update () [virtual]

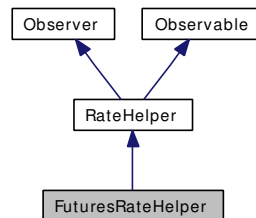
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.439 FuturesRateHelper Class Reference

```
#include <ql/termstructures/yieldcurves/ratehelpers.hpp>
```

Inheritance diagram for FuturesRateHelper:



9.439.1 Detailed Description

Rate helper for bootstrapping over interest-rate futures prices.

Todo

implement/refactor constructors with: [Index](#) instead of (nMonths, calendar, convention, dayCounter), [IMM](#) code

Examples:

[swapvaluation.cpp](#).

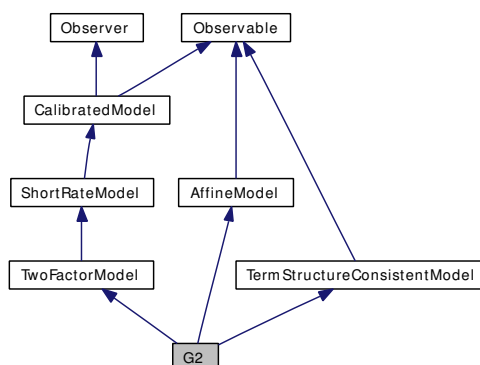
Public Member Functions

- **FuturesRateHelper** (const [Handle](#)< [Quote](#) > &price, const [Date](#) &immDate, Size nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter, const [Handle](#)< [Quote](#) > &convexityAdjustment)
- **FuturesRateHelper** (const [Handle](#)< [Quote](#) > &price, const [Date](#) &immDate, Size nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter, Rate convexityAdjustment=0.0)
- **FuturesRateHelper** (Real price, const [Date](#) &immDate, Size nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter, Rate convexityAdjustment=0.0)
- Real **impliedQuote** () const
- DiscountFactor **discountGuess** () const
- Real **convexityAdjustment** () const

9.440 G2 Class Reference

```
#include <ql/models/shortrate/twofactormodels/g2.hpp>
```

Inheritance diagram for G2:



9.440.1 Detailed Description

Two-additive-factor gaussian model class.

This class implements a two-additive-factor model defined by

$$dr_t = \varphi(t) + x_t + y_t$$

where x_t and y_t are defined by

$$dx_t = -ax_t dt + \sigma dW_t^1, x_0 = 0$$

$$dy_t = -by_t dt + \sigma dW_t^2, y_0 = 0$$

and $dW_t^1 dW_t^2 = \rho dt$.

Bug

This class was not tested enough to guarantee its functionality.

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

- **G2** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, Real a=0.1, Real sigma=0.01, Real b=0.1, Real eta=0.01, Real rho=-0.75)
- [boost::shared_ptr](#)< [ShortRateDynamics](#) > [dynamics](#) () const
Returns the short-rate dynamics.
- virtual Real **discountBond** (Time now, Time maturity, [Array](#) factors) const
- Real **discountBond** (Time, Time, Rate, Rate) const

- Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const
- Real **swaption** (const [Swaption::arguments](#) &arguments, Real range, Size intervals) const
- DiscountFactor [discount](#) (Time t) const

Implied discount curve.

Protected Member Functions

- void **generateArguments** ()
- Real **A** (Time t, Time T) const
- Real **B** (Real x, Time t) const

Friends

- class **SwaptionPricingFunction**

Classes

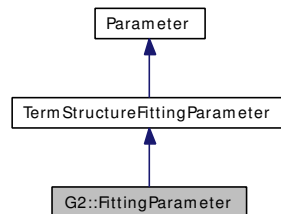
- class [FittingParameter](#)

Analytical term-structure fitting parameter $\varphi(t)$.

9.441 G2::FittingParameter Class Reference

```
#include <ql/models/shortrate/twofactormodels/g2.hpp>
```

Inheritance diagram for G2::FittingParameter:



9.441.1 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 .

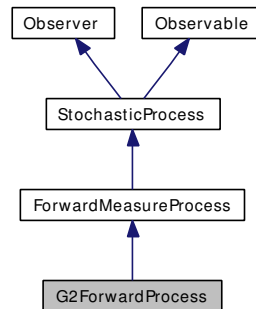
Public Member Functions

- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, Real a, Real sigma, Real b, Real eta, Real rho)

9.442 G2ForwardProcess Class Reference

```
#include <ql/processes/g2process.hpp>
```

Inheritance diagram for G2ForwardProcess:



9.442.1 Detailed Description

Forward G2 stochastic process

Public Member Functions

- **G2ForwardProcess** (Real a, Real sigma, Real b, Real eta, Real rho)

StochasticProcess interface

- **Size** [size](#) () const
returns the number of dimensions of the stochastic process
- **Disposable**< [Array](#) > **initialValues** () const
returns the initial values of the state variables
- **Disposable**< [Array](#) > **drift** (Time t, const [Array](#) &x) const
returns the drift part of the equation, i.e., $\mu(t, x_t)$
- **Disposable**< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- **Disposable**< [Array](#) > **expectation** (Time t0, const [Array](#) &x0, Time dt) const
- **Disposable**< [Matrix](#) > **stdDeviation** (Time t0, const [Array](#) &x0, Time dt) const
- **Disposable**< [Matrix](#) > **covariance** (Time t0, const [Array](#) &x0, Time dt) const

Protected Member Functions

- Real **xForwardDrift** (Time t, Time T) const
- Real **yForwardDrift** (Time t, Time T) const
- Real **Mx_T** (Real s, Real t, Real T) const
- Real **My_T** (Real s, Real t, Real T) const

Protected Attributes

- Real `x0_`
- Real `y0_`
- Real `a_`
- Real `sigma_`
- Real `b_`
- Real `eta_`
- Real `rho_`
- `boost::shared_ptr< QuantLib::OrnsteinUhlenbeckProcess > xProcess_`
- `boost::shared_ptr< QuantLib::OrnsteinUhlenbeckProcess > yProcess_`

9.442.2 Member Function Documentation

9.442.2.1 Disposable<Array> expectation (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

9.442.2.2 Disposable<Matrix> stdDeviation (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

9.442.2.3 Disposable<Matrix> covariance (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

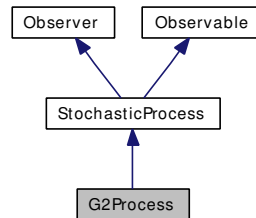
returns the covariance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

9.443 G2Process Class Reference

```
#include <ql/processes/g2process.hpp>
```

Inheritance diagram for G2Process:



9.443.1 Detailed Description

G2 stochastic process

Public Member Functions

- **G2Process** (Real a, Real sigma, Real b, Real eta, Real rho)
- Real **x0** () const
- Real **y0** () const
- Real **a** () const
- Real **sigma** () const
- Real **b** () const
- Real **eta** () const
- Real **rho** () const

StochasticProcess interface

- Size **size** () const
returns the number of dimensions of the stochastic process
- Disposable< Array > **initialValues** () const
returns the initial values of the state variables
- Disposable< Array > **drift** (Time t, const Array &x) const
returns the drift part of the equation, i.e., $\mu(t, x_t)$
- Disposable< Matrix > **diffusion** (Time t, const Array &x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- Disposable< Array > **expectation** (Time t0, const Array &x0, Time dt) const
- Disposable< Matrix > **stdDeviation** (Time t0, const Array &x0, Time dt) const
- Disposable< Matrix > **covariance** (Time t0, const Array &x0, Time dt) const

9.443.2 Member Function Documentation

9.443.2.1 Disposable<Array> expectation (Time t_0 , const Array & x_0 , Time dt) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval 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](#).

9.443.2.2 Disposable<Matrix> stdDeviation (Time t_0 , const Array & x_0 , Time dt) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval 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](#).

9.443.2.3 Disposable<Matrix> covariance (Time t_0 , const Array & x_0 , Time dt) const [virtual]

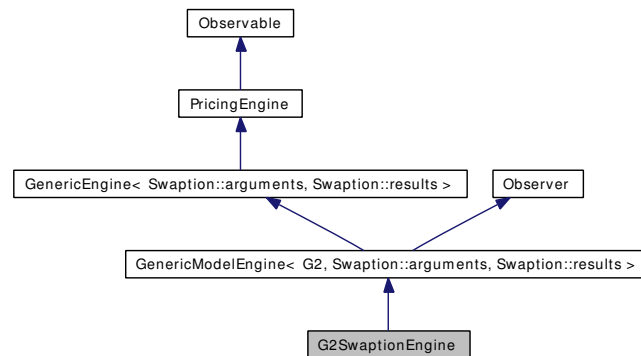
returns the covariance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

9.444 G2SwaptionEngine Class Reference

```
#include <ql/pricingengines/swaption/g2swaptionengine.hpp>
```

Inheritance diagram for G2SwaptionEngine:



9.444.1 Detailed Description

Swaption priced by means of the Black formula

Warning

The engine assumes that the exercise date equals the start date of the passed swap.

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

- **G2SwaptionEngine** (const boost::shared_ptr< [G2](#) > &mod, Real range, Size intervals)
- void **calculate** () const

9.445 GammaFunction Class Reference

```
#include <ql/math/distributions/gammadistribution.hpp>
```

9.445.1 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.

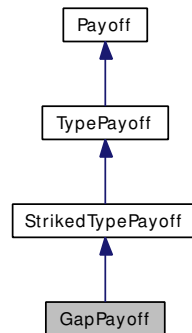
Public Member Functions

- Real **logValue** (Real x) const

9.446 GapPayoff Class Reference

```
#include <ql/instruments/payoffs.hpp>
```

Inheritance diagram for GapPayoff:



9.446.1 Detailed Description

Binary gap payoff.

This payoff is equivalent to being a) long a [PlainVanillaPayoff](#) at the first strike (same Call/Put type) and b) short a [CashOrNothingPayoff](#) at the first strike (same Call/Put type) with cash payoff equal to the difference between the second and the first strike.

Warning

this payoff can be negative depending on the strikes

Public Member Functions

- **GapPayoff** (Option::Type type, Real strike, Real secondStrike)
- Real **secondStrike** () const

Payoff interface

- std::string **name** () const
- std::string **description** () const
- Real **operator()** (Real price) const
- virtual void **accept** ([AcyclicVisitor](#) &)

Protected Attributes

- Real **secondStrike_**

9.446.2 Member Function Documentation

9.446.2.1 `std::string name () const` [virtual]

Warning

This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implements [Payoff](#).

9.447 Garch11 Class Reference

```
#include <ql/models/volatility/garch.hpp>
```

9.447.1 Detailed Description

GARCH volatility model.

Volatilities are assumed to be expressed on an annual basis.

Public Member Functions

- **Garch11** (Real a, Real b, Real vl)
- **Garch11** (const [TimeSeries](#)< Volatility > &qs)
- [TimeSeries](#)< Volatility > **calculate** (const [TimeSeries](#)< Volatility > "eSeries)
- [TimeSeries](#)< Volatility > **calculate** (const [TimeSeries](#)< Volatility > "eSeries, Real, Real, Real)
- void **calibrate** (const [TimeSeries](#)< Volatility > "eSeries)

9.448 GarmanKlassAbstract Class Reference

```
#include <ql/models/volatility/garmanklass.hpp>
```

9.448.1 Detailed Description

Garman-Klass volatility model.

This class implements a concrete volatility model based on high low formulas using the method of Garman and Klass in their paper "On the Estimation of the Security Price from Historical Data" at http://www.fea.com/resources/pdf/a_estimation_of_security_price.pdf

Volatilities are assumed to be expressed on an annual basis.

Public Member Functions

- **GarmanKlassAbstract** (Real y)
- **TimeSeries**< Volatility > **calculate** (const **TimeSeries**< **IntervalPrice** > "eSeries)

Protected Member Functions

- virtual Real **calculatePoint** (const **IntervalPrice** &p)=0

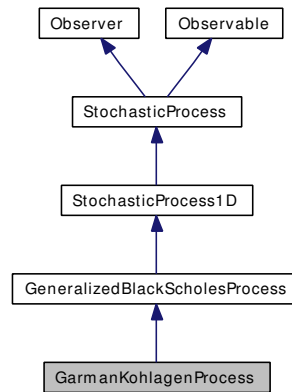
Protected Attributes

- Real **yearFraction_**

9.449 GarmanKohlagenProcess Class Reference

```
#include <ql/processes/blackscholesprocess.hpp>
```

Inheritance diagram for GarmanKohlagenProcess:



9.449.1 Detailed Description

Garman-Kohlhagen (1983) stochastic process.

This class describes the stochastic process for an exchange rate given by

$$dS(t, S) = (r(t) - r_f(t) - \frac{\sigma(t, S)^2}{2})dt + \sigma dW_t.$$

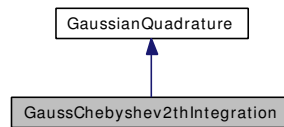
Public Member Functions

- **GarmanKohlagenProcess** (const [Handle](#)< [Quote](#) > &x0, const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [YieldTermStructure](#) > &domesticRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &blackVolTS, const boost::shared_ptr< discretization > &d=boost::shared_ptr< discretization >(new [EulerDiscretization](#)))

9.450 GaussChebyshev2thIntegration Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

Inheritance diagram for GaussChebyshev2thIntegration:



9.450.1 Detailed Description

Gauss-Chebyshev integration (second kind).

This class performs a 1-dimensional Gauss-Chebyshev integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{1/2}$$

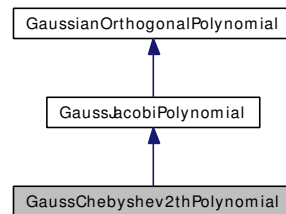
Public Member Functions

- `GaussChebyshev2thIntegration` (Size n)

9.451 GaussChebyshev2thPolynomial Class Reference

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussChebyshev2thPolynomial:



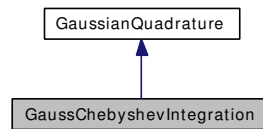
9.451.1 Detailed Description

Gauss-Chebyshev polynomial (second kind).

9.452 GaussChebyshevIntegration Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

Inheritance diagram for GaussChebyshevIntegration:



9.452.1 Detailed Description

Gauss-Chebyshev integration.

This class performs a 1-dimensional Gauss-Chebyshev integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{-1/2}$$

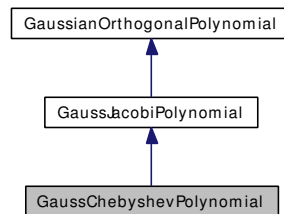
Public Member Functions

- `GaussChebyshevIntegration` (Size n)

9.453 GaussChebyshevPolynomial Class Reference

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussChebyshevPolynomial:



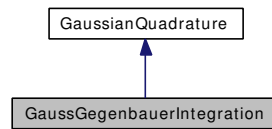
9.453.1 Detailed Description

Gauss-Chebyshev polynomial.

9.454 GaussGegenbauerIntegration Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

Inheritance diagram for GaussGegenbauerIntegration:



9.454.1 Detailed Description

Gauss-Gegenbauer integration.

This class performs a 1-dimensional Gauss-Gegenbauer integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{\lambda-1/2}$$

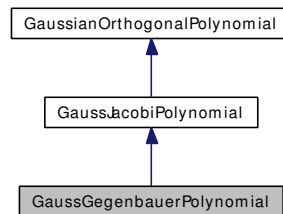
Public Member Functions

- `GaussGegenbauerIntegration` (Size n, Real lambda)

9.455 GaussGegenbauerPolynomial Class Reference

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussGegenbauerPolynomial:



9.455.1 Detailed Description

Gauss-Gegenbauer polynomial.

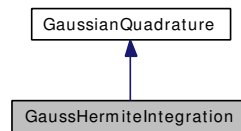
Public Member Functions

- **GaussGegenbauerPolynomial** (Real lambda)

9.456 GaussHermiteIntegration Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

Inheritance diagram for GaussHermiteIntegration:



9.456.1 Detailed Description

generalized Gauss-Hermite integration

This class performs a 1-dimensional Gauss-Hermite integration.

$$\int_{-\infty}^{\infty} f(x) dx$$

The weighting function is

$$w(x; \mu) = |x|^{2\mu} \exp -x * x$$

and

$$\mu > -0.5$$

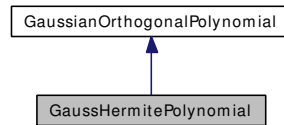
Public Member Functions

- **GaussHermiteIntegration** (Size n, Real mu=0.0)

9.457 GaussHermitePolynomial Class Reference

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussHermitePolynomial:



9.457.1 Detailed Description

Gauss-Hermite polynomial.

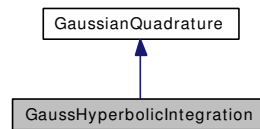
Public Member Functions

- **GaussHermitePolynomial** (Real mu=0.0)
- Real **mu_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

9.458 GaussHyperbolicIntegration Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

Inheritance diagram for GaussHyperbolicIntegration:



9.458.1 Detailed Description

Gauss-Hyperbolic integration.

This class performs a 1-dimensional Gauss-Hyperbolic integration.

$$\int_{-\infty}^{\infty} f(x) dx$$

The weighting function is

$$w(x) = 1/\cosh(x)$$

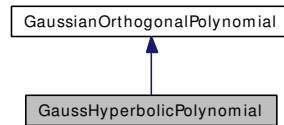
Public Member Functions

- `GaussHyperbolicIntegration` (Size n)

9.459 GaussHyperbolicPolynomial Class Reference

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussHyperbolicPolynomial:



9.459.1 Detailed Description

Gauss hyperbolic polynomial.

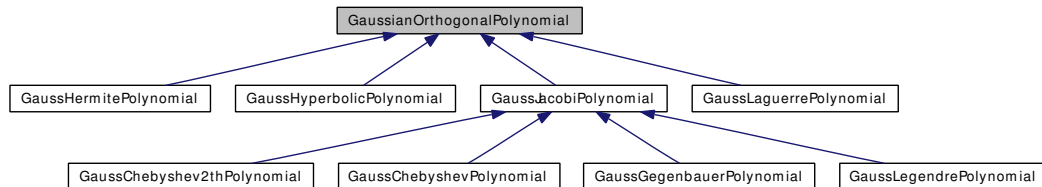
Public Member Functions

- Real **mu_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

9.460 GaussianOrthogonalPolynomial Class Reference

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussianOrthogonalPolynomial:



9.460.1 Detailed Description

orthogonal polynomial for Gaussian quadratures

References: Gauss quadratures and orthogonal polynomials

G.H. Gloub and J.H. Welsch: Calculation of Gauss quadrature rule. Math. Comput. 23 (1986), 221-230

"Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

The polynomials are defined by the three-term recurrence relation

$$P_{k+1}(x) = (x - \alpha_k)P_k(x) - \beta_k P_{k-1}(x)$$

and

$$\mu_0 = \int w(x)dx$$

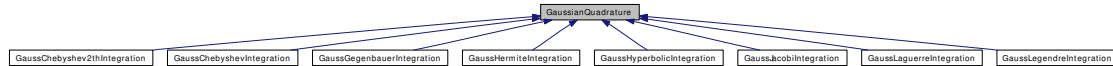
Public Member Functions

- virtual Real **mu_0** () const=0
- virtual Real **alpha** (Size i) const=0
- virtual Real **beta** (Size i) const=0
- virtual Real **w** (Real x) const=0
- Real **value** (Size i, Real x) const
- Real **weightedValue** (Size i, Real x) const

9.461 GaussianQuadrature Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

Inheritance diagram for GaussianQuadrature:



9.461.1 Detailed Description

Integral of a 1-dimensional function using the Gauss quadratures method.

References: Gauss quadratures and orthogonal polynomials

G.H. Gloub and J.H. Welsch: Calculation of Gauss quadrature rule. Math. Comput. 23 (1986), 221-230

"Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

Tests

the correctness of the result is tested by checking it against known good values.

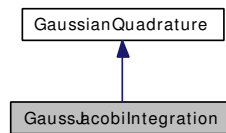
Public Member Functions

- **GaussianQuadrature** (Size n, const [GaussianOrthogonalPolynomial](#) &p)
- template<class F>
Real **operator()** (const F &f) const
- Size **order** () const

9.462 GaussJacobiIntegration Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

Inheritance diagram for GaussJacobiIntegration:



9.462.1 Detailed Description

Gauss-Jacobi integration.

This class performs a 1-dimensional Gauss-Jacobi integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x; \alpha, \beta) = (1 - x)^\alpha (1 + x)^\beta$$

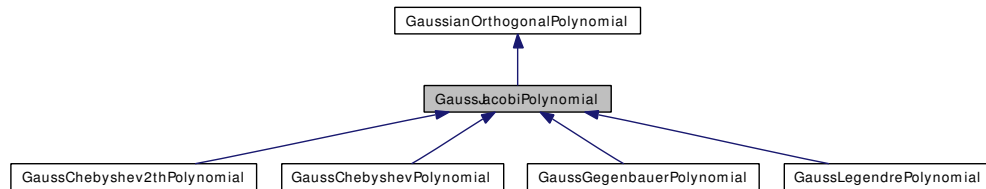
Public Member Functions

- **GaussJacobiIntegration** (Size n, Real alpha, Real beta)

9.463 GaussJacobiPolynomial Class Reference

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussJacobiPolynomial:



9.463.1 Detailed Description

Gauss-Jacobi polynomial.

Public Member Functions

- **GaussJacobiPolynomial** (Real alpha, Real beta)
- Real **mu_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

9.464 GaussKronrodAdaptive Class Reference

```
#include <ql/math/integrals/kronrodintegral.hpp>
```

9.464.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod methods.

This class provide an adaptive integration procedure using 15 points Gauss-Kronrod integration rule. This is more robust in that it allows to integrate less smooth functions (though singular functions should be integrated using dedicated algorithms) but less efficient beacuse it does not reuse precedently computed points during computation steps.

References:

Gauss-Kronrod Integration <<http://mathcssun1.emporia.edu/~oneilcat/ExperimentApplet3/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

- **GaussKronrodAdaptive** (Real tolerance, Size maxFunctionEvaluations=[Null](#)< Size >())

Protected Member Functions

- Real **integrate** (const boost::function< Real(Real)> &f, Real a, Real b) const

9.465 GaussKronrodNonAdaptive Class Reference

```
#include <ql/math/integrals/kronrodintegral.hpp>
```

9.465.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod methods.

This class provide a non-adaptive integration procedure which uses fixed Gauss-Kronrod abscissae to sample the integrand at a maximum of 87 points. It is provided for fast integration of smooth functions.

This function applies the Gauss-Kronrod 10-point, 21-point, 43-point and 87-point integration rules in succession until an estimate of the integral of f over (a, b) is achieved within the desired absolute and relative error limits, `epsabs` and `epsrel`. The function returns the final approximation, `result`, an estimate of the absolute error, `abserr` and the number of function evaluations used, `neval`. The Gauss-Kronrod rules are designed in such a way that each rule uses all the results of its predecessors, in order to minimize the total number of function evaluations.

Public Member Functions

- **GaussKronrodNonAdaptive** (Real `absoluteAccuracy`, Size `maxEvaluations`, Real `relativeAccuracy`)
- void **setRelativeAccuracy** (Real)
- Real **relativeAccuracy** () const

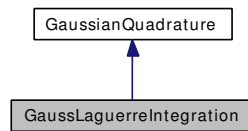
Protected Member Functions

- Real **integrate** (const boost::function< Real(Real)> &`f`, Real `a`, Real `b`) const

9.466 GaussLaguerreIntegration Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

Inheritance diagram for GaussLaguerreIntegration:



9.466.1 Detailed Description

generalized Gauss-Laguerre integration

This class performs a 1-dimensional Gauss-Laguerre integration.

$$\int_0^{\infty} f(x) dx$$

The weighting function is

$$w(x; s) = x^s \exp -x$$

and

$$s > -1$$

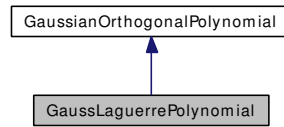
Public Member Functions

- **GaussLaguerreIntegration** (Size n, Real s=0.0)

9.467 GaussLaguerrePolynomial Class Reference

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussLaguerrePolynomial:



9.467.1 Detailed Description

Gauss-Laguerre polynomial.

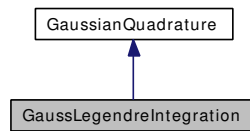
Public Member Functions

- **GaussLaguerrePolynomial** (Real s=0.0)
- Real **mu_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

9.468 GaussLegendreIntegration Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

Inheritance diagram for GaussLegendreIntegration:



9.468.1 Detailed Description

Gauss-Legendre integration.

This class performs a 1-dimensional Gauss-Legendre integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = 1$$

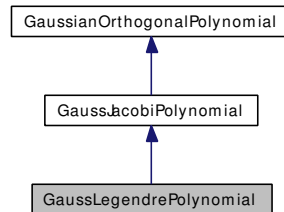
Public Member Functions

- `GaussLegendreIntegration` (Size n)

9.469 GaussLegendrePolynomial Class Reference

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussLegendrePolynomial:



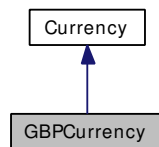
9.469.1 Detailed Description

Gauss-Legendre polynomial.

9.470 GBPCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for GBPCurrency:



9.470.1 Detailed Description

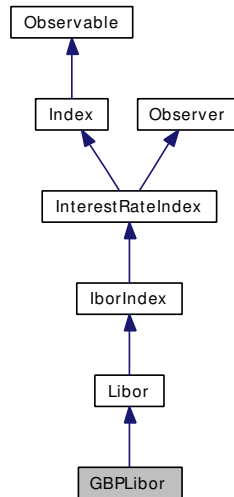
British pound sterling.

The ISO three-letter code is GBP; the numeric code is 826. It is divided into 100 pence.

9.471 GBPLibor Class Reference

```
#include <ql/indexes/ibor/gbplibor.hpp>
```

Inheritance diagram for GBPLibor:



9.471.1 Detailed Description

GBP LIBOR rate

Pound Sterling LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

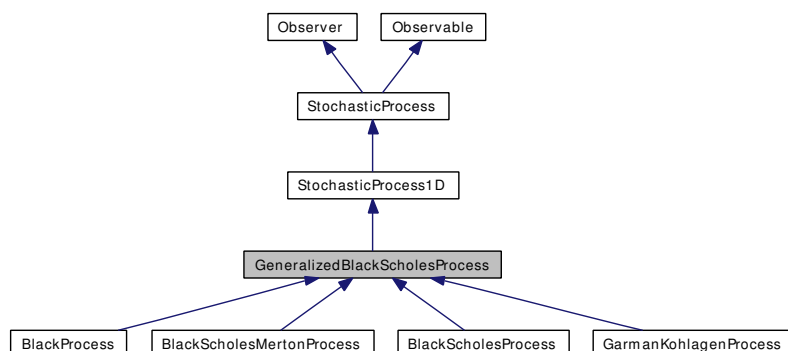
Public Member Functions

- **GBPLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.472 GeneralizedBlackScholesProcess Class Reference

```
#include <ql/processes/blackscholesprocess.hpp>
```

Inheritance diagram for GeneralizedBlackScholesProcess:



9.472.1 Detailed Description

Generalized Black-Scholes stochastic process.

This class describes the stochastic process governed by

$$dS(t, S) = (r(t) - q(t) - \frac{\sigma(t, S)^2}{2})dt + \sigma dW_t.$$

Public Member Functions

- **GeneralizedBlackScholesProcess** (const [Handle< Quote >](#) &x0, const [Handle< YieldTermStructure >](#) ÷ndTS, const [Handle< YieldTermStructure >](#) &riskFreeTS, const [Handle< BlackVolTermStructure >](#) &blackVolTS, const boost::shared_ptr< discretization > &d=boost::shared_ptr< discretization >(new [EulerDiscretization](#)))
- Time [time](#) (const [Date](#) &) const

StochasticProcess1D interface

- Real [x0](#) () const
returns the initial value of the state variable
- Real [drift](#) (Time t, Real x) const
- Real [diffusion](#) (Time t, Real x) const
- Real [apply](#) (Real x0, Real dx) const

Observer interface

- void [update](#) ()

Inspectors

- const [Handle< Quote >](#) & [stateVariable](#) () const
- const [Handle< YieldTermStructure >](#) & [dividendYield](#) () const
- const [Handle< YieldTermStructure >](#) & [riskFreeRate](#) () const
- const [Handle< BlackVolTermStructure >](#) & [blackVolatility](#) () const
- const [Handle< LocalVolTermStructure >](#) & [localVolatility](#) () const

9.472.2 Member Function Documentation

9.472.2.1 Real drift (Time t , Real x) const [virtual]

Todo

revise extrapolation

Implements [StochasticProcess1D](#).

9.472.2.2 Real diffusion (Time t , Real x) const [virtual]

Todo

revise extrapolation

Implements [StochasticProcess1D](#).

9.472.2.3 Real apply (Real x_0 , Real dx) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented from [StochasticProcess1D](#).

9.472.2.4 Time time (const Date &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:

As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

9.472.2.5 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [StochasticProcess](#).

9.473 GeneralStatistics Class Reference

```
#include <ql/math/statistics/generalstatistics.hpp>
```

9.473.1 Detailed Description

Statistics tool.

This class accumulates a set of data and returns their statistics (e.g: mean, variance, skewness, kurtosis, error estimation, percentile, etc.) based on the empirical distribution (no gaussian assumption)

It doesn't suffer the numerical instability problem of [IncrementalStatistics](#). The downside is that it stores all samples, thus increasing the memory requirements.

Public Types

- typedef Real **value_type**

Public Member Functions

Inspectors

- Size **samples** () const
number of samples collected
- const std::vector< std::pair< Real, Real > > & **data** () const
collected data
- Real **weightSum** () const
sum of data weights
- Real **mean** () const
- Real **variance** () const
- Real **standardDeviation** () const
- Real **errorEstimate** () const
- Real **skewness** () const
- Real **kurtosis** () const
- Real **min** () const
- Real **max** () const
- template<class Func, class Predicate>
std::pair< Real, Size > **expectationValue** (const Func &f, const Predicate &inRange) const
- Real **percentile** (Real y) const
- Real **topPercentile** (Real y) const

Modifiers

- void **add** (Real value, Real weight=1.0)
adds a datum to the set, possibly with a weight
- template<class DataIterator>
void **addSequence** (DataIterator begin, DataIterator end)
adds a sequence of data to the set, with default weight

- `template<class DataIterator, class WeightIterator>`
`void addSequence (DataIterator begin, DataIterator end, WeightIterator wbegin)`
adds a sequence of data to the set, each with its weight
- `void reset ()`
resets the data to a null set
- `void sort () const`
sort the data set in increasing order

9.473.2 Member Function Documentation

9.473.2.1 Real mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

9.473.2.2 Real variance () const

returns the variance, defined as

$$\sigma^2 = \frac{N}{N-1} \left\langle (x - \langle x \rangle)^2 \right\rangle.$$

9.473.2.3 Real standardDeviation () const

returns the standard deviation σ , defined as the square root of the variance.

9.473.2.4 Real errorEstimate () const

returns the error estimate on the mean value, defined as $\epsilon = \sigma / \sqrt{N}$.

9.473.2.5 Real skewness () const

returns the skewness, defined as

$$\frac{N^2}{(N-1)(N-2)} \frac{\left\langle (x - \langle x \rangle)^3 \right\rangle}{\sigma^3}.$$

The above evaluates to 0 for a Gaussian distribution.

9.473.2.6 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.

9.473.2.7 Real min () const

returns the minimum sample value

9.473.2.8 Real max () const

returns the maximum sample value

9.473.2.9 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.,

$$\mathbb{E}[f | \mathcal{R}] = \frac{\sum_{x_i \in \mathcal{R}} f(x_i) w_i}{\sum_{x_i \in \mathcal{R}} w_i}.$$

The range is passed as a boolean function returning `true` if the argument belongs to the range or `false` otherwise.

The function returns a pair made of the result and the number of observations in the given range.

9.473.2.10 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 $(0 - 1]$.

9.473.2.11 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 $(0 - 1]$.

9.473.2.12 void add (Real *value*, Real *weight* = 1.0)

adds a datum to the set, possibly with a weight

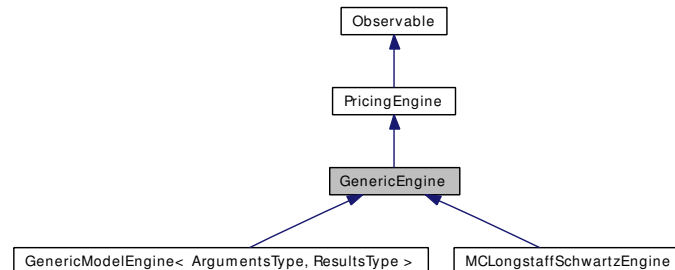
Precondition:

weights must be positive or null

9.474 GenericEngine Class Template Reference

```
#include <ql/pricingengine.hpp>
```

Inheritance diagram for GenericEngine:



9.474.1 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

- PricingEngine::arguments * **getArguments** () const
- const PricingEngine::results * **getResults** () const
- void **reset** ()

Protected Attributes

- ArgumentsType **arguments_**
- ResultsType **results_**

9.475 GenericGaussianStatistics Class Template Reference

```
#include <ql/math/statistics/gaussianstatistics.hpp>
```

9.475.1 Detailed Description

```
template<class Stat> class QuantLib::GenericGaussianStatistics< Stat >
```

Statistics tool for gaussian-assumption risk measures.

This class wraps a somewhat generic statistic tool and adds a number of gaussian risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the mean and variance provided by the underlying statistic tool.

Public Types

- typedef Stat::value_type **value_type**

Public Member Functions

- **GenericGaussianStatistics** (const Stat &s)

Gaussian risk measures

- Real [gaussianDownsideVariance](#) () const
- Real [gaussianDownsideDeviation](#) () const
- Real [gaussianRegret](#) (Real target) const
- Real [gaussianPercentile](#) (Real percentile) const
- Real [gaussianTopPercentile](#) (Real percentile) const
- Real [gaussianPotentialUpside](#) (Real percentile) const
gaussian-assumption Potential-Upside at a given percentile
- Real [gaussianValueAtRisk](#) (Real percentile) const
gaussian-assumption Value-At-Risk at a given percentile
- Real [gaussianExpectedShortfall](#) (Real percentile) const
gaussian-assumption Expected Shortfall at a given percentile
- Real [gaussianShortfall](#) (Real target) const
gaussian-assumption Shortfall (observations below target)
- Real [gaussianAverageShortfall](#) (Real target) const
gaussian-assumption [Average](#) Shortfall (averaged shortfallness)

9.475.2 Member Function Documentation

9.475.2.1 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 $x < 0$

9.475.2.2 Real gaussianDownsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

9.475.2.3 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)

9.475.2.4 Real gaussianPercentile (Real *percentile*) const

gaussian-assumption y-th percentile, defined as the value x such that

$$y = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp(-u^2/2) du$$

9.475.2.5 Real gaussianTopPercentile (Real *percentile*) const

Precondition:

percentile must be in range (0-100%) extremes excluded

9.475.2.6 Real gaussianPotentialUpside (Real *percentile*) const

gaussian-assumption Potential-Upside at a given percentile

Precondition:

percentile must be in range [90-100%)

9.475.2.7 Real gaussianValueAtRisk (Real *percentile*) const

gaussian-assumption Value-At-Risk at a given percentile

Precondition:

percentile must be in range [90-100%)

9.475.2.8 Real gaussianExpectedShortfall (Real *percentile*) const

gaussian-assumption Expected Shortfall at a given percentile

Assuming a gaussian distribution it returns the expected loss in case that the loss exceeded a VaR threshold,

$$E[x \mid x < \text{VaR}(p)],$$

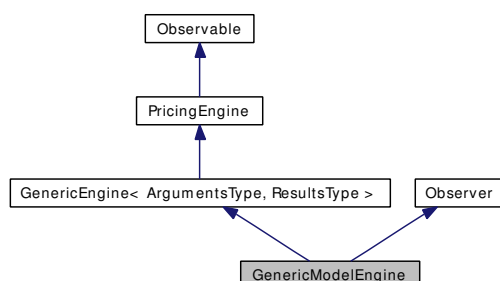
that is the average of observations below the given percentile p . Also know as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

9.476 GenericModelEngine Class Template Reference

```
#include <ql/pricingengines/genericmodelengine.hpp>
```

Inheritance diagram for GenericModelEngine:



9.476.1 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_**

9.476.2 Member Function Documentation

9.476.2.1 virtual void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

Reimplemented in [LatticeShortRateModelEngine](#), [LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >](#), [LatticeShortRateModelEngine< VanillaSwap::arguments, VanillaSwap::results >](#), and [LatticeShortRateModelEngine< Swap::arguments, Swap::results >](#).

9.477 GenericRiskStatistics Class Template Reference

```
#include <ql/math/statistics/riskstatistics.hpp>
```

9.477.1 Detailed Description

```
template<class S> class QuantLib::GenericRiskStatistics< S >
```

empirical-distribution risk measures

This class wraps a somewhat generic statistic tool and adds a number of risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the data distribution as reported by the underlying statistic tool.

Todo

add historical annualized volatility

Examples:

[DiscreteHedging.cpp](#).

Public Types

- typedef S::value_type **value_type**

Public Member Functions

- Real [semiVariance](#) () const
- Real [semiDeviation](#) () const
- Real [downsideVariance](#) () const
- Real [downsideDeviation](#) () const
- Real [regret](#) (Real target) const
- Real [potentialUpside](#) (Real percentile) const
potential upside (the reciprocal of VAR) at a given percentile
- Real [valueAtRisk](#) (Real percentile) const
value-at-risk at a given percentile
- Real [expectedShortfall](#) (Real percentile) const
expected shortfall at a given percentile
- Real [shortfall](#) (Real target) const
- Real [averageShortfall](#) (Real target) const

9.477.2 Member Function Documentation

9.477.2.1 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).

9.477.2.2 Real semiDeviation () const

returns the semi deviation, defined as the square root of the semi variance.

9.477.2.3 Real downsideVariance () const

returns the variance of observations below 0.0,

$$\frac{N}{N-1} \mathbb{E} \left[x^2 \mid x < 0 \right].$$

9.477.2.4 Real downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

9.477.2.5 Real regret (Real *target*) const

returns the variance of observations below target,

$$\frac{N}{N-1} \mathbb{E} \left[(x - t)^2 \mid x < t \right].$$

See Dembo and Freeman, "The Rules Of Risk", Wiley (2001).

9.477.2.6 Real potentialUpside (Real *centile*) const

potential upside (the reciprocal of VAR) at a given percentile

Precondition:

percentile must be in range [90-100%)

9.477.2.7 Real valueAtRisk (Real *centile*) const

value-at-risk at a given percentile

Precondition:

percentile must be in range [90-100%)

9.477.2.8 Real expectedShortfall (Real *percentile*) const

expected shortfall at a given percentile

returns the expected loss in case that the loss exceeded a VaR threshold,

$$E[x \mid x < \text{VaR}(p)],$$

that is the average of observations below the given percentile p . Also known as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

9.477.2.9 Real shortfall (Real *target*) const

probability of missing the given target, defined as

$$E[\Theta \mid (-\infty, \infty)]$$

where

$$\Theta(x) = \begin{cases} 1 & x < t \\ 0 & x \geq t \end{cases}$$

9.477.2.10 Real averageShortfall (Real *target*) const

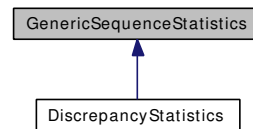
averaged shortfallness, defined as

$$E[t - x \mid x < t]$$

9.478 GenericSequenceStatistics Class Template Reference

```
#include <ql/math/statistics/sequencestatistics.hpp>
```

Inheritance diagram for GenericSequenceStatistics:



9.478.1 Detailed Description

```
template<class StatisticsType = Statistics> class QuantLib::GenericSequenceStatistics< StatisticsType >
```

Statistics analysis of N-dimensional (sequence) data.

It provides 1-dimensional statistics as discrepancy plus N-dimensional (sequence) statistics (e.g. mean, variance, skewness, kurtosis, etc.) with one component for each dimension of the sample space.

For most of the statistics this class relies on the StatisticsType underlying class to provide 1-D methods that will be iterated for all the components of the N-D data. These lifted methods are the union of all the methods that might be requested to the 1-D underlying StatisticsType class, with the usual compile-time checks provided by the template approach.

Tests

the correctness of the returned values is tested by checking them against numerical calculations.

Public Types

- typedef StatisticsType **statistics_type**
- typedef std::vector< typename StatisticsType::value_type > **value_type**

Public Member Functions

- **GenericSequenceStatistics** (Size dimension)

inspectors

- Size **size** () const

covariance and correlation

- [Disposable](#)< [Matrix](#) > **covariance** () const
returns the covariance [Matrix](#)
- [Disposable](#)< [Matrix](#) > **correlation** () const

returns the correlation [Matrix](#)

1-D inspectors lifted from underlying statistics class

- Size **samples** () const
- Real **weightSum** () const

N-D inspectors lifted from underlying statistics class

- std::vector< Real > **mean** () const
- std::vector< Real > **variance** () const
- std::vector< Real > **standardDeviation** () const
- std::vector< Real > **downsideVariance** () const
- std::vector< Real > **downsideDeviation** () const
- std::vector< Real > **semiVariance** () const
- std::vector< Real > **semiDeviation** () const
- std::vector< Real > **errorEstimate** () const
- std::vector< Real > **skewness** () const
- std::vector< Real > **kurtosis** () const
- std::vector< Real > **min** () const
- std::vector< Real > **max** () const
- std::vector< Real > **gaussianPercentile** (Real y) const
- std::vector< Real > **percentile** (Real y) const
- std::vector< Real > **gaussianPotentialUpside** (Real percentile) const
- std::vector< Real > **potentialUpside** (Real percentile) const
- std::vector< Real > **gaussianValueAtRisk** (Real percentile) const
- std::vector< Real > **valueAtRisk** (Real percentile) const
- std::vector< Real > **gaussianExpectedShortfall** (Real percentile) const
- std::vector< Real > **expectedShortfall** (Real percentile) const
- std::vector< Real > **regret** (Real target) const
- std::vector< Real > **gaussianShortfall** (Real target) const
- std::vector< Real > **shortfall** (Real target) const
- std::vector< Real > **gaussianAverageShortfall** (Real target) const
- std::vector< Real > **averageShortfall** (Real target) const

Modifiers

- void **reset** (Size dimension=0)
- template<class Sequence>
void **add** (const Sequence &sample, Real weight=1.0)
- template<class Iterator>
void **add** (Iterator begin, Iterator end, Real weight=1.0)

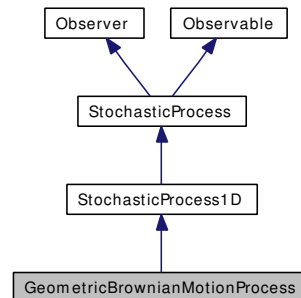
Protected Attributes

- Size **dimension_**
- std::vector< statistics_type > **stats_**
- std::vector< Real > **results_**
- [Matrix](#) **quadraticSum_**

9.479 GeometricBrownianMotionProcess Class Reference

```
#include <ql/processes/geometricbrownianprocess.hpp>
```

Inheritance diagram for GeometricBrownianMotionProcess:



9.479.1 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. $\sigma(t, x_t)$

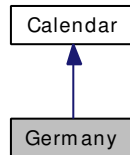
Protected Attributes

- double **initialValue_**
- double **mue_**
- double **sigma_**

9.480 Germany Class Reference

```
#include <ql/time/calendars/germany.hpp>
```

Inheritance diagram for Germany:



9.480.1 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)

9.480.2 Member Enumeration Documentation

9.480.2.1 enum Market

German calendars.

Enumerator:

Settlement generic settlement calendar

FrankfurtStockExchange Frankfurt stock-exchange.

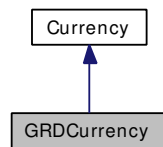
Xetra Xetra.

Eurex Eurex.

9.481 GRDCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for GRDCurrency:



9.481.1 Detailed Description

Greek drachma.

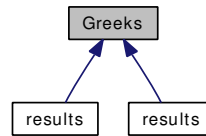
The ISO three-letter code was GRD; the numeric code was 300. It was divided in 100 lepta.

Obsoleted by the Euro since 2001.

9.482 Greeks Class Reference

```
#include <ql/option.hpp>
```

Inheritance diagram for Greeks:



9.482.1 Detailed Description

additional option results

Public Member Functions

- void **reset** ()

Public Attributes

- Real **delta**
- Real **gamma**
- Real **theta**
- Real **vega**
- Real **rho**
- Real **dividendRho**

9.483 HaltonRsg Class Reference

```
#include <ql/math/randomnumbers/haltonrsg.hpp>
```

9.483.1 Detailed Description

Halton low-discrepancy sequence generator.

Halton algorithm for low-discrepancy sequence. For more details see chapter 8, paragraph 2 of "Monte Carlo Methods in Finance", by Peter Jäckel

Tests

- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

Public Types

- typedef [Sample](#)< std::vector< Real > > **sample_type**

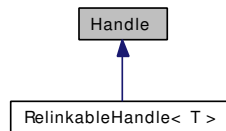
Public Member Functions

- **HaltonRsg** (Size dimensionality, unsigned long seed=0, bool randomStart=true, bool randomShift=false)
- const [sample_type](#) & **nextSequence** () const
- const [sample_type](#) & **lastSequence** () const
- Size **dimension** () const

9.484 Handle Class Template Reference

```
#include <ql/handle.hpp>
```

Inheritance diagram for Handle:



9.484.1 Detailed Description

```
template<class T> class QuantLib::Handle< T >
```

Shared handle to an observable.

All copies of an instance of this class refer to the same observable by means of a relinkable smart pointer. When such pointer is relinked to another observable, the change will be propagated to all the copies.

Precondition:

Class T must inherit from [Observable](#)

Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), [Replication.cpp](#), and [swapvaluation.cpp](#).

Public Member Functions

- [Handle](#) (const boost::shared_ptr< T > &h=boost::shared_ptr< T >(), bool registerAsObserver=true)
- const boost::shared_ptr< T > & [currentLink](#) () const
dereferencing
- const boost::shared_ptr< T > & [operator](#) → () const
- bool [empty](#) () const
checks if the contained shared pointer points to anything
- [operator boost::shared_ptr](#) () const
allows registration as observable
- template<class U>
bool [operator==](#) (const [Handle](#)< U > &other)
equality test
- template<class U>
bool [operator!=](#) (const [Handle](#)< U > &other)

disequality test

- `template<class U>`
`bool operator< (const Handle< U > &other)`
strict weak ordering

Protected Attributes

- `boost::shared_ptr< Link > link_`

9.484.2 Constructor & Destructor Documentation

- 9.484.2.1 `Handle (const boost::shared_ptr< T > & h = boost::shared_ptr< T >(), bool registerAsObserver = true) [explicit]`

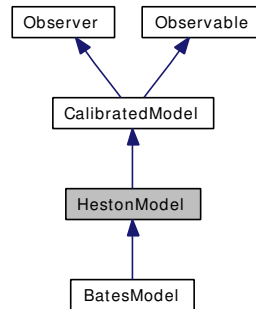
Warning

`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 does not own the pointee (this 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.

9.485 HestonModel Class Reference

```
#include <ql/models/equity/hestonmodel.hpp>
```

Inheritance diagram for HestonModel:



9.485.1 Detailed Description

Heston model for the stochastic volatility of an asset.

References:

Heston, Steven L., 1993. A Closed-Form Solution for Options with Stochastic Volatility with Applications to [Bond](#) and [Currency](#) Options. The review of Financial Studies, Volume 6, Issue 2, 327-343.

Tests

calibration is tested against known good values.

Public Member Functions

- **HestonModel** (const boost::shared_ptr< [HestonProcess](#) > &process)
- Real **theta** () const
- Real **kappa** () const
- Real **sigma** () const
- Real **rho** () const
- Real **v0** () const

Protected Member Functions

- void **generateArguments** ()

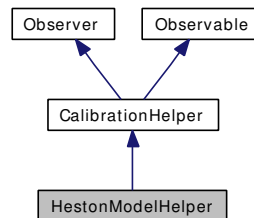
Protected Attributes

- [RelinkableHandle](#)< [Quote](#) > **v0_**
- [RelinkableHandle](#)< [Quote](#) > **kappa_**
- [RelinkableHandle](#)< [Quote](#) > **theta_**
- [RelinkableHandle](#)< [Quote](#) > **sigma_**
- [RelinkableHandle](#)< [Quote](#) > **rho_**

9.486 HestonModelHelper Class Reference

```
#include <ql/models/equity/hestonmodelhelper.hpp>
```

Inheritance diagram for HestonModelHelper:



9.486.1 Detailed Description

calibration helper for Heston model

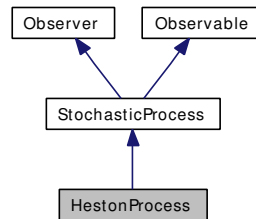
Public Member Functions

- **HestonModelHelper** (const [Period](#) &maturity, const [Calendar](#) &calendar, const Real s0, const Real strikePrice, const [Handle](#)< [Quote](#) > &volatility, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [YieldTermStructure](#) > ÷ndYield, bool calibrateVolatility=false)
- void **addTimesTo** (std::list< Time > &) const
- Real **modelValue** () const
returns the price of the instrument according to the model
- Real **blackPrice** (Real volatility) const
- Time **maturity** () const

9.487 HestonProcess Class Reference

```
#include <ql/processes/hestonprocess.hpp>
```

Inheritance diagram for HestonProcess:



9.487.1 Detailed Description

Square-root stochastic-volatility Heston process.

This class describes the square root stochastic volatility process governed by

$$\begin{aligned}
 dS(t, S) &= \mu S dt + \sqrt{v} S dW_1 \\
 dv(t, S) &= \kappa(\theta - v) dt + \sigma \sqrt{v} dW_2 \\
 dW_1 dW_2 &= \rho dt
 \end{aligned}$$

Public Types

- enum **Discretization** { **PartialTruncation**, **FullTruncation**, **Reflection**, **ExactVariance** }

Public Member Functions

- **HestonProcess** (const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [YieldTermStructure](#) > ÷ndYield, const [Handle](#)< [Quote](#) > &s0, double v0, double kappa, double theta, double sigma, double rho, Discretization d=FullTruncation)
- **Size** [size](#) () const
returns the number of dimensions of the stochastic process
- **Disposable**< [Array](#) > **initialValues** () const
returns the initial values of the state variables
- **Disposable**< [Array](#) > **drift** (Time t, const [Array](#) &x) const
returns the drift part of the equation, i.e., $\mu(t, x_t)$
- **Disposable**< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- **Disposable**< [Array](#) > **apply** (const [Array](#) &x0, const [Array](#) &dx) const
- **Disposable**< [Array](#) > **evolve** (Time t0, const [Array](#) &x0, Time dt, const [Array](#) &dw) const
- const [RelinkableHandle](#)< [Quote](#) > & **v0** () const
- const [RelinkableHandle](#)< [Quote](#) > & **rho** () const

- const [RelinkableHandle](#)< [Quote](#) > & **kappa** () const
- const [RelinkableHandle](#)< [Quote](#) > & **theta** () const
- const [RelinkableHandle](#)< [Quote](#) > & **sigma** () const
- const [Handle](#)< [Quote](#) > & **s0** () const
- const [Handle](#)< [YieldTermStructure](#) > & **dividendYield** () const
- const [Handle](#)< [YieldTermStructure](#) > & **riskFreeRate** () const
- void **update** ()
- Time **time** (const [Date](#) &) const

9.487.2 Member Function Documentation

9.487.2.1 [Disposable](#)<[Array](#)> **apply** (const [Array](#) & *x0*, const [Array](#) & *dx*) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented from [StochasticProcess](#).

9.487.2.2 [Disposable](#)<[Array](#)> **evolve** (Time *t0*, const [Array](#) & *x0*, Time *dt*, const [Array](#) & *dw*) const [virtual]

returns the asset value after a time interval Δt according to the given discretization. By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where E is the expectation and S the standard deviation.

Reimplemented from [StochasticProcess](#).

9.487.2.3 **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.

Reimplemented from [StochasticProcess](#).

9.487.2.4 **Time time** (const [Date](#) &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:

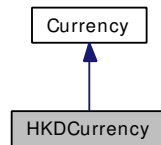
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

9.488 HKDCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for HKDCurrency:



9.488.1 Detailed Description

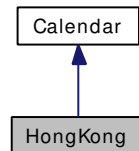
Honk Kong dollar.

The ISO three-letter code is HKD; the numeric code is 344. It is divided in 100 cents.

9.489 HongKong Class Reference

```
#include <ql/time/calendars/hongkong.hpp>
```

Inheritance diagram for HongKong:



9.489.1 Detailed Description

Hong Kong calendars.

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Ching Ming Festival, April 5th
- Good Friday
- Easter Monday
- Labor Day, May 1st
- SAR Establishment Day, July 1st (possibly moved to Monday)
- National Day, October 1st (possibly moved to Monday)
- Christmas, December 25th
- Boxing Day, December 26th (possibly moved to Monday)

Other holidays for which no rule is given (data available for 2004-2007 only:)

- Lunar New Year
- Chinese New Year
- Buddha's birthday
- Tuen NG Festival
- Mid-autumn Festival
- Chung Yeung Festival

Data from <http://www.hkex.com.hk>

Public Types

- enum [Market](#) { [HKEx](#) }

Public Member Functions

- [HongKong](#) ([Market](#) m=[HKEx](#))

9.489.2 Member Enumeration Documentation

9.489.2.1 enum [Market](#)

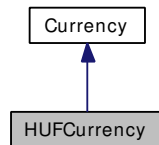
Enumerator:

[HKEx](#) Hong Kong stock exchange.

9.490 HUFCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for HUFCurrency:



9.490.1 Detailed Description

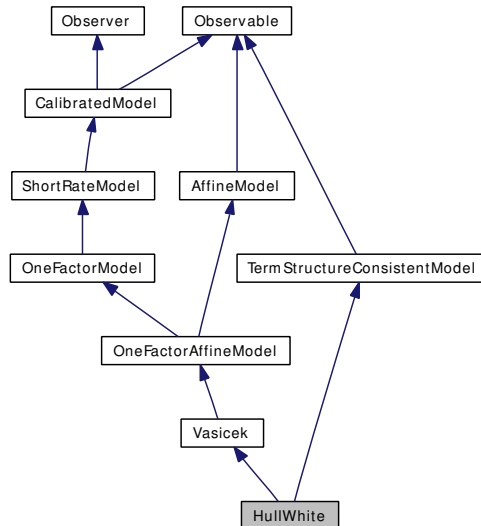
Hungarian forint.

The ISO three-letter code is HUF; the numeric code is 348. It has no subdivisions.

9.491 HullWhite Class Reference

```
#include <ql/models/shortrate/onefactormodels/hullwhite.hpp>
```

Inheritance diagram for HullWhite:



9.491.1 Detailed Description

Single-factor Hull-White (extended Vasicek) model class.

This class implements the standard single-factor Hull-White model defined by

$$dr_t = (\theta(t) - \alpha r_t)dt + \sigma dW_t$$

where α and σ are constants.

Tests

calibration results are tested against cached values

Bug

When the term structure is relinked, the `r0` parameter of the underlying [Vasicek](#) model is not updated.

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

- **HullWhite** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, Real a=0.1, Real sigma=0.01)
- `boost::shared_ptr< 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 bondMaturity) const

Static Public Member Functions

- static Rate `convexityBias` (Real futurePrice, Time t, Time T, Real sigma, Real a)

Protected Member Functions

- void `generateArguments` ()
- Real A (Time t, Time T) const

Classes

- class `Dynamics`
Short-rate dynamics in the Hull-White model.
- class `FittingParameter`
Analytical term-structure fitting parameter $\varphi(t)$.

9.491.2 Member Function Documentation

9.491.2.1 static Rate `convexityBias` (Real *futurePrice*, Time *t*, Time *T*, Real *sigma*, Real *a*) [static]

Futures convexity bias (i.e., the difference between futures implied rate and forward rate) calculated as in G. Kirikos, D. Novak, "Convexity Conundrums", Risk Magazine, March 1997.

Note:

t and *T* should be expressed in yearfraction using deposit day counter, *F_quoted* is futures' market price.

9.492 HullWhite::Dynamics Class Reference

```
#include <ql/models/shortrate/onefactormodels/hullwhite.hpp>
```

9.492.1 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.

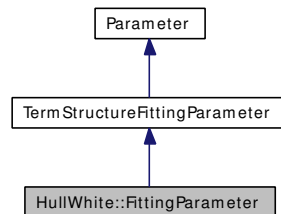
Public Member Functions

- **Dynamics** (const [Parameter](#) &fitting, Real a, Real sigma)
- Real **variable** (Time t, Rate r) const
- Real **shortRate** (Time t, Real x) const

9.493 HullWhite::FittingParameter Class Reference

```
#include <ql/models/shortrate/onefactormodels/hullwhite.hpp>
```

Inheritance diagram for HullWhite::FittingParameter:



9.493.1 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 .

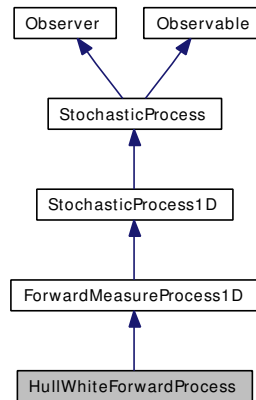
Public Member Functions

- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, Real a, Real sigma)

9.494 HullWhiteForwardProcess Class Reference

```
#include <ql/processes/hullwhiteprocess.hpp>
```

Inheritance diagram for HullWhiteForwardProcess:



9.494.1 Detailed Description

Forward Hull-White stochastic process

Public Member Functions

- **HullWhiteForwardProcess** (const [Handle](#)< [YieldTermStructure](#) > &h, Real a, Real sigma)

StochasticProcess1D interface

- Real [x0](#) () const
returns the initial value of the state variable
- Real [drift](#) (Time t, Real x) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- Real [diffusion](#) (Time t, Real x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- Real [expectation](#) (Time t0, Real x0, Time dt) const
- Real [stdDeviation](#) (Time t0, Real x0, Time dt) const
- Real [variance](#) (Time t0, Real x0, Time dt) const

Protected Member Functions

- Real [alpha](#) (Time t) const
- Real [M_T](#) (Real s, Real t, Real T) const
- Real [B](#) (Time t, Time T) const

Protected Attributes

- `boost::shared_ptr< QuantLib::OrnsteinUhlenbeckProcess > process_`
- `Handle< YieldTermStructure > h_`
- Real `a_`
- Real `sigma_`

9.494.2 Member Function Documentation

9.494.2.1 Real expectation (Time t_0 , Real x_0 , Time dt) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

9.494.2.2 Real stdDeviation (Time t_0 , Real x_0 , Time dt) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

9.494.2.3 Real variance (Time t_0 , Real x_0 , Time dt) const [virtual]

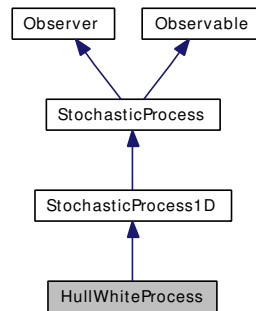
returns the variance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

9.495 HullWhiteProcess Class Reference

```
#include <ql/processes/hullwhiteprocess.hpp>
```

Inheritance diagram for HullWhiteProcess:



9.495.1 Detailed Description

Hull-White stochastic process.

Public Member Functions

- **HullWhiteProcess** (const [Handle](#)< [YieldTermStructure](#) > &h, Real a, Real sigma)

StochasticProcess1D interface

- Real [x0](#) () const
returns the initial value of the state variable
- Real [drift](#) (Time t, Real x) const
returns the drift part of the equation, i.e. $\mu(t, x_i)$
- Real [diffusion](#) (Time t, Real x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_i)$
- Real [expectation](#) (Time t0, Real x0, Time dt) const
- Real [stdDeviation](#) (Time t0, Real x0, Time dt) const
- Real [variance](#) (Time t0, Real x0, Time dt) const

Protected Member Functions

- Real [alpha](#) (Time t) const

Protected Attributes

- boost::shared_ptr< [QuantLib::OrnsteinUhlenbeckProcess](#) > **process_**
- [Handle](#)< [YieldTermStructure](#) > **h_**
- Real **a_**
- Real **sigma_**

9.495.2 Member Function Documentation

9.495.2.1 Real expectation (Time t_0 , Real x_0 , Time dt) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

9.495.2.2 Real stdDeviation (Time t_0 , Real x_0 , Time dt) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

9.495.2.3 Real variance (Time t_0 , Real x_0 , Time dt) const [virtual]

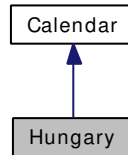
returns the variance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

9.496 Hungary Class Reference

```
#include <ql/time/calendars/hungary.hpp>
```

Inheritance diagram for Hungary:



9.496.1 Detailed Description

Hungarian calendar.

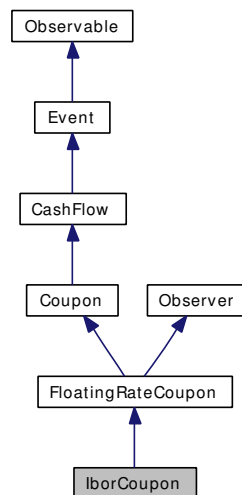
Holidays:

- Saturdays
- Sundays
- Easter Monday
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- National Day, March 15th
- Labour Day, May 1st
- Constitution Day, August 20th
- Republic Day, October 23rd
- All Saints Day, November 1st
- Christmas, December 25th
- 2nd Day of Christmas, December 26th

9.497 IborCoupon Class Reference

```
#include <ql/cashflows/iborcoupon.hpp>
```

Inheritance diagram for IborCoupon:



9.497.1 Detailed Description

Coupon paying a Libor-type index

Public Member Functions

- **IborCoupon** (const [Date](#) &paymentDate, const Real nominal, const [Date](#) &startDate, const [Date](#) &endDate, const Natural fixingDays, const boost::shared_ptr< [InterestRateIndex](#) > &index, const Real gearing=1.0, const Spread spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)(), bool isInArrears=false)
- Rate [indexFixing](#) () const
fixing of the underlying index

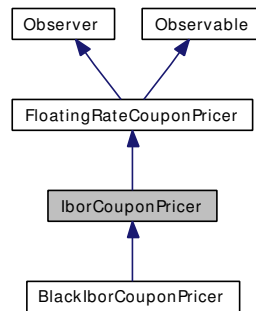
Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

9.498 IborCouponPricer Class Reference

```
#include <ql/cashflows/couponpricer.hpp>
```

Inheritance diagram for IborCouponPricer:



9.498.1 Detailed Description

base pricer for capped/floored Ibor coupons

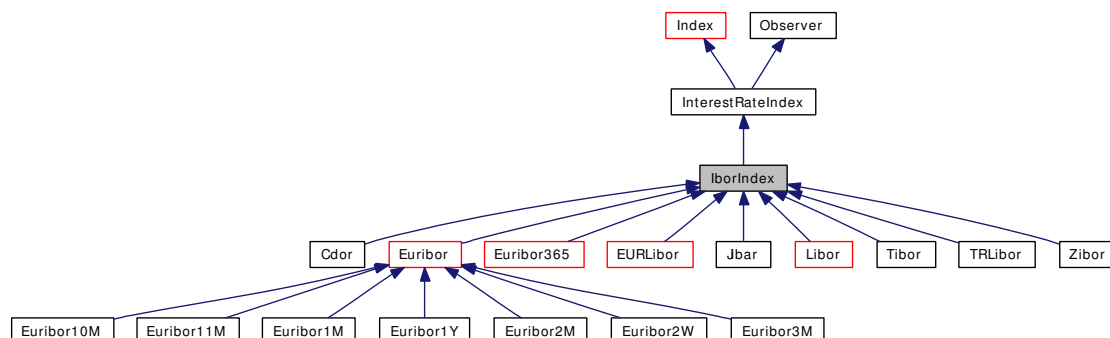
Public Member Functions

- **IborCouponPricer** (const [Handle](#)< [CapletVolatilityStructure](#) > &capletVol)
- [Handle](#)< [CapletVolatilityStructure](#) > **capletVolatility** () const
- void **setCapletVolatility** (const [Handle](#)< [CapletVolatilityStructure](#) > &capletVol)

9.499 IborIndex Class Reference

```
#include <ql/indexes/iborindex.hpp>
```

Inheritance diagram for IborIndex:



9.499.1 Detailed Description

base class for Inter-Bank-Offered-Rate indexes (e.g. Libor, etc.)

Todo

add methods returning [InterestRate](#)

Public Member Functions

- **IborIndex** (const std::string &familyName, const [Period](#) &tenor, Natural settlement-Days, const [Currency](#) ¤cy, const [Calendar](#) &fixingCalendar, [BusinessDayConvention](#) convention, bool endOfMonth, const [DayCounter](#) &dayCounter, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

InterestRateIndex interface

- Rate **forecastFixing** (const [Date](#) &fixingDate) const
- [Handle](#)< [YieldTermStructure](#) > **termStructure** () const

Inspectors

- [BusinessDayConvention](#) **businessDayConvention** () const
- bool **endOfMonth** () const

Date calculations

- [Date](#) **maturityDate** (const [Date](#) &valueDate) const

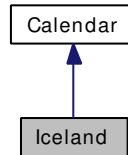
Protected Attributes

- [BusinessDayConvention](#) **convention_**
- [Handle](#)< [YieldTermStructure](#) > **termStructure_**
- bool **endOfMonth_**

9.500 Iceland Class Reference

```
#include <ql/time/calendars/iceland.hpp>
```

Inheritance diagram for Iceland:



9.500.1 Detailed Description

Icelandic calendars.

Holidays for the [Iceland](http://www.icex.is/is/calendar?languageID=1) stock exchange (data from [<http://www.icex.is/is/calendar?languageID=1>](http://www.icex.is/is/calendar?languageID=1)):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Holy Thursday
- Good Friday
- Easter Monday
- First day of Summer (third or fourth Thursday in April)
- Labour Day, May 1st
- Ascension Thursday
- Pentecost Monday
- Independence Day, June 17th
- Commerce Day, first Monday in August
- Christmas, December 25th
- Boxing Day, December 26th

Public Types

- enum [Market](#) { [ICEX](#) }

Public Member Functions

- `Iceland` ([Market](#) m=[ICEX](#))

9.500.2 Member Enumeration Documentation

9.500.2.1 enum Market

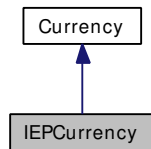
Enumerator:

ICEX [Iceland](#) stock exchange.

9.501 IEPCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for IEPCurrency:



9.501.1 Detailed Description

Irish punt.

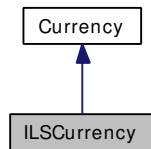
The ISO three-letter code was IEP; the numeric code was 372. It was divided in 100 pence.

Obsoleted by the Euro since 1999.

9.502 ILSCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for ILSCurrency:



9.502.1 Detailed Description

Israeli shekel.

The ISO three-letter code is ILS; the numeric code is 376. It is divided in 100 agorot.

9.503 IMM Struct Reference

```
#include <ql/time/imm.hpp>
```

9.503.1 Detailed Description

Main cycle of the International Money Market (a.k.a. IMM) months.

Public Types

- enum **Month** {
 F = 1, **G** = 2, **H** = 3, **J** = 4,
 K = 5, **M** = 6, **N** = 7, **Q** = 8,
 U = 9, **V** = 10, **X** = 11, **Z** = 12 }

Static Public Member Functions

- static bool **isIMMdate** (const **Date** &d, bool mainCycle=true)
 returns whether or not the given date is an IMM date
- static bool **isIMMcode** (const std::string &in, bool mainCycle=true)
 returns whether or not the given string is an IMM code
- static std::string **code** (const **Date** &immDate)
- static **Date** **date** (const std::string &immCode, const **Date** &referenceDate=**Date**())
- static **Date** **nextDate** (const **Date** &d=**Date**(), bool mainCycle=true)
 next IMM date following the given date
- static **Date** **nextDate** (const std::string &immCode, bool mainCycle=true, const **Date** &referenceDate=**Date**())
 next IMM date following the given IMM code
- static std::string **nextCode** (const **Date** &d=**Date**(), bool mainCycle=true)
- static std::string **nextCode** (const std::string &immCode, bool mainCycle=true, const **Date** &referenceDate=**Date**())

9.503.2 Member Function Documentation

9.503.2.1 static std::string code (const **Date** & *immDate*) [static]

returns the IMM code for the given date (e.g. H3 for March 20th, 2013).

Warning

It raises an exception if the input date is not an IMM date

9.503.2.2 `static Date date (const std::string & immCode, const Date & referenceDate = Date())`
[static]

returns the IMM date for the given IMM code (e.g. March 20th, 2013 for H3).

Warning

It raises an exception if the input string is not an IMM code

9.503.2.3 `static Date nextDate (const Date & d = Date(), bool mainCycle = true)` [static]

next IMM date following the given date

returns the 1st delivery date for next contract listed in the International Money Market section of the Chicago Mercantile Exchange.

9.503.2.4 `static Date nextDate (const std::string & immCode, bool mainCycle = true, const Date & referenceDate = Date())` [static]

next IMM date following the given IMM code

returns the 1st delivery date for next contract listed in the International Money Market section of the Chicago Mercantile Exchange.

9.503.2.5 `static std::string nextCode (const Date & d = Date(), bool mainCycle = true)`
[static]

returns the IMM code for next contract listed in the International Money Market section of the Chicago Mercantile Exchange.

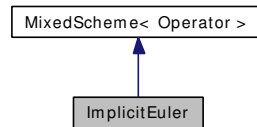
9.503.2.6 `static std::string nextCode (const std::string & immCode, bool mainCycle = true, const Date & referenceDate = Date())` [static]

returns the IMM code for next contract listed in the International Money Market section of the Chicago Mercantile Exchange.

9.504 ImplicitEuler Class Template Reference

```
#include <ql/methods/finitedifferences/impliciteuler.hpp>
```

Inheritance diagram for ImplicitEuler:



9.504.1 Detailed Description

```
template<class Operator> class QuantLib::ImplicitEuler< Operator >
```

Backward Euler scheme for finite difference methods.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```

typedef ... array_type;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
array_type solveFor(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, const Operator&);
Operator operator+(const Operator&, const Operator&);

```

Public Types

- typedef OperatorTraits< Operator > **traits**
- typedef traits::operator_type **operator_type**
- typedef traits::array_type **array_type**
- typedef traits::bc_set **bc_set**
- typedef [traits::condition_type](#) **condition_type**

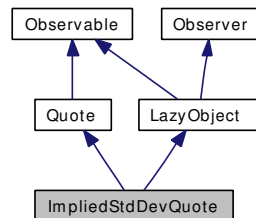
Public Member Functions

- **ImplicitEuler** (const operator_type &L, const bc_set &bcs)

9.505 ImpliedStdDevQuote Class Reference

```
#include <ql/quotes/impliedstddevquote.hpp>
```

Inheritance diagram for ImpliedStdDevQuote:



9.505.1 Detailed Description

quote for the implied standard deviation of an underlying

Public Member Functions

- **ImpliedStdDevQuote** (Option::Type optionType, const [Handle< Quote >](#) &forward, const [Handle< Quote >](#) &price, Real strike, Real guess=.15, Real accuracy=1.0e-6)
- Real [value](#) () const
returns the current value

Protected Member Functions

- void [performCalculations](#) () const

Protected Attributes

- Real **impliedStdev_**
- Option::Type **optionType_**
- Real **strike_**
- Real **accuracy_**
- [Handle< Quote >](#) **forward_**
- [Handle< Quote >](#) **price_**

9.505.2 Member Function Documentation

9.505.2.1 void performCalculations () const [protected, virtual]

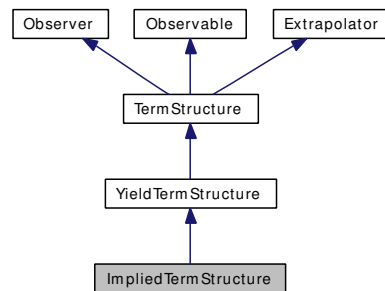
This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implements [LazyObject](#).

9.506 ImpliedTermStructure Class Reference

```
#include <ql/termstructures/yieldcurves/impliedtermstructure.hpp>
```

Inheritance diagram for ImpliedTermStructure:



9.506.1 Detailed Description

Implied term structure at a given date in the future.

The given date will be the implied reference date.

Note:

This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Tests

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure is checked.

Public Member Functions

- **ImpliedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Date](#) &referenceDate)

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return values

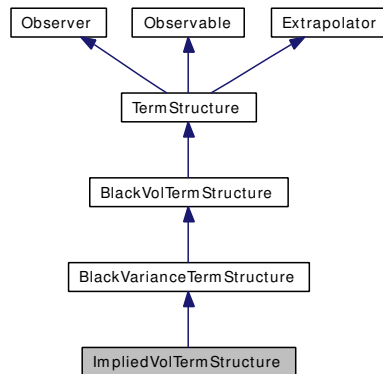
Protected Member Functions

- DiscountFactor [discountImpl](#) (Time) const
returns the discount factor as seen from the evaluation date

9.507 ImpliedVolTermStructure Class Reference

```
#include <ql/termstructures/volatilities/impliedvoltermstructure.hpp>
```

Inheritance diagram for ImpliedVolTermStructure:



9.507.1 Detailed Description

Implied vol term structure at a given date in the future.

The given date will be the implied reference date.

Note:

This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Warning

It doesn't make financial sense to have an asset-dependant implied Vol Term Structure. This class should be used with term structures that are time dependant only.

Public Member Functions

- **ImpliedVolTermStructure** (const [Handle](#)< [BlackVolTermStructure](#) > &originalTS, const [Date](#) &referenceDate)

BlackVolTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return values
- Real [minStrike](#) () const
the minimum strike for which the term structure can return vols
- Real [maxStrike](#) () const

the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

Protected Member Functions

- virtual Real [blackVarianceImpl](#) (Time t, Real strike) const
Black variance calculation.

9.508 IncrementalStatistics Class Reference

```
#include <ql/math/statistics/incrementalstatistics.hpp>
```

9.508.1 Detailed Description

Statistics tool based on incremental accumulation.

It can accumulate a set of data and return statistics (e.g: mean, variance, skewness, kurtosis, error estimation, etc.)

Warning

high moments are numerically unstable for high average/standardDeviation ratios.

Public Types

- typedef Real **value_type**

Public Member Functions

Inspectors

- Size **samples** () const
number of samples collected
- Real **weightSum** () const
sum of data weights
- Real **mean** () const
- Real **variance** () const
- Real **standardDeviation** () const
- Real **errorEstimate** () const
- Real **skewness** () const
- Real **kurtosis** () const
- Real **min** () const
- Real **max** () const
- Real **downsideVariance** () const
- Real **downsideDeviation** () const

Modifiers

- void **add** (Real value, Real weight=1.0)
adds a datum to the set, possibly with a weight
- template<class DataIterator>
void **addSequence** (DataIterator begin, DataIterator end)
adds a sequence of data to the set, with default weight
- template<class DataIterator, class WeightIterator>
void **addSequence** (DataIterator begin, DataIterator end, WeightIterator wbegin)
adds a sequence of data to the set, each with its weight
- void **reset** ()
resets the data to a null set

Protected Attributes

- Size `sampleNumber_`
- Size `downsideSampleNumber_`
- Real `sampleWeight_`
- Real `downsideSampleWeight_`
- Real `sum_`
- Real `quadraticSum_`
- Real `downsideQuadraticSum_`
- Real `cubicSum_`
- Real `fourthPowerSum_`
- Real `min_`
- Real `max_`

9.508.2 Member Function Documentation

9.508.2.1 Real `mean () const`

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

9.508.2.2 Real `variance () const`

returns the variance, defined as

$$\frac{N}{N-1} \langle (x - \langle x \rangle)^2 \rangle.$$

9.508.2.3 Real `standardDeviation () const`

returns the standard deviation σ , defined as the square root of the variance.

9.508.2.4 Real `errorEstimate () const`

returns the error estimate ϵ , defined as the square root of the ratio of the variance to the number of samples.

9.508.2.5 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.

9.508.2.6 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.

9.508.2.7 Real min () const

returns the minimum sample value

9.508.2.8 Real max () const

returns the maximum sample value

9.508.2.9 Real downsideVariance () const

returns the downside variance, defined as

$$\frac{N}{N-1} \times \frac{\sum_{i=1}^N \theta \times x_i^2}{\sum_{i=1}^N w_i}$$

, where $\theta = 0$ if $x > 0$ and $\theta = 1$ if $x < 0$

9.508.2.10 Real downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

9.508.2.11 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

9.508.2.12 void addSequence (DataIterator *begin*, DataIterator *end*, WeightIterator *wbegin*)

adds a sequence of data to the set, each with its weight

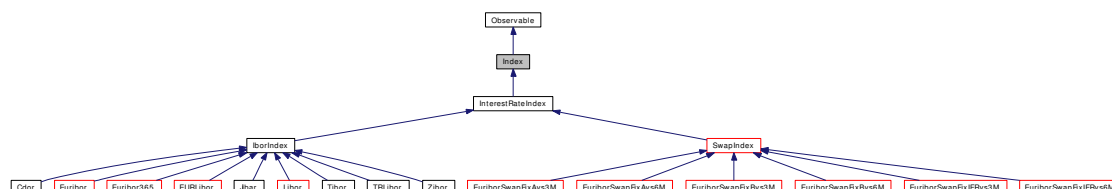
Precondition:

weights must be positive or null

9.509 Index Class Reference

```
#include <ql/index.hpp>
```

Inheritance diagram for Index:



9.509.1 Detailed Description

purely virtual base class for indexes

Public Member Functions

- virtual `std::string name () const=0`
Returns the name of the index.
- `Calendar fixingCalendar () const`
returns the calendar defining valid fixing dates
- `bool isValidFixingDate (const Date &fixingDate) const`
returns TRUE if the fixing date is a valid one
- virtual `Real fixing (const Date &fixingDate, bool forecastTodaysFixing=false) const=0`
returns the fixing at the given date
- `void addFixing (const Date &fixingDate, Real fixing)`
stores the historical fixing at the given date
- `template<class DateIterator, class ValueIterator>`
`void addFixings (DateIterator dBegin, DateIterator dEnd, ValueIterator vBegin)`
stores historical fixings at the given dates
- `void clearFixings ()`
clears all stored historical fixings

Protected Attributes

- `Calendar fixingCalendar_`

9.509.2 Member Function Documentation

9.509.2.1 `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 [InterestRateIndex](#).

9.509.2.2 `virtual Real fixing (const Date & fixingDate, bool forecastTodaysFixing = false) const` [pure virtual]

returns the fixing at the given date

the date passed as arguments must be the actual calendar date of the fixing; no settlement days must be used.

Implemented in [InterestRateIndex](#).

9.509.2.3 `void addFixing (const Date & fixingDate, Real fixing)`

stores the historical fixing at the given date

the date passed as arguments must be the actual calendar date of the fixing; no settlement days must be used.

9.509.2.4 `void addFixings (DateIterator dBegin, DateIterator dEnd, ValueIterator vBegin)`

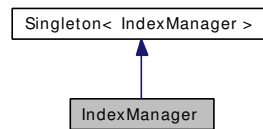
stores historical fixings at the given dates

the dates passed as arguments must be the actual calendar dates of the fixings; no settlement days must be used.

9.510 IndexManager Class Reference

```
#include <ql/indexes/indexmanager.hpp>
```

Inheritance diagram for IndexManager:



9.510.1 Detailed Description

global repository for past index fixings

Note:

index names are case insensitive

Public Member Functions

- bool `hasHistory` (const std::string &name) const
returns whether historical fixings were stored for the index
- const `TimeSeries`< Real > & `getHistory` (const std::string &name) const
returns the (possibly empty) history of the index fixings
- void `setHistory` (const std::string &name, const `TimeSeries`< Real > &)
stores the historical fixings of the index
- boost::shared_ptr< `Observable` > `notifier` (const std::string &name) const
observer notifying of changes in the index fixings
- std::vector< std::string > `histories` () const
returns all names of the indexes for which fixings were stored
- void `clearHistory` (const std::string &name)
clears the historical fixings of the index
- void `clearHistories` ()
clears all stored fixings

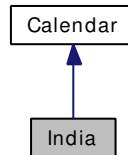
Friends

- class `Singleton`< `IndexManager` >

9.511 India Class Reference

```
#include <ql/time/calendars/india.hpp>
```

Inheritance diagram for India:



9.511.1 Detailed Description

Indian calendars.

Holidays for the National [Stock Exchange](http://www.nse-india.com/) (data from <http://www.nse-india.com/>):

- Saturdays
- Sundays
- Republic Day, January 26th
- Good Friday
- Ambedkar Jayanti, April 14th
- Independence Day, August 15th
- Gandhi Jayanti, October 2nd
- Christmas, December 25th

Other holidays for which no rule is given (data available for 2005-2007 only:)

- Bakri Id
- Moharram
- Mahashivratri
- Holi
- Ram Navami
- Mahavir Jayanti
- Id-E-Milad
- Maharashtra Day
- Buddha Pournima
- Ganesh Chaturthi
- Dasara

- Laxmi Puja
- Bhaubeej
- Ramzan Id
- Guru Nanak Jayanti

Public Types

- enum [Market](#) { [NSE](#) }

Public Member Functions

- [India](#) ([Market](#) m=NSE)

9.511.2 Member Enumeration Documentation

9.511.2.1 enum Market

Enumerator:

[NSE](#) National [Stock](#) Exchange.

9.512 Indonesia Class Reference

```
#include <ql/time/calendars/indonesia.hpp>
```

Inheritance diagram for Indonesia:



9.512.1 Detailed Description

Indonesian calendars

Holidays for the Jakarta stock exchange (data from <http://www.jsx.co.id/trading.asp?cmd=menu3>):

- Saturdays
- Sundays
- Good Friday
- New Year's Day, January 1st
- Ascension of Jesus Christ
- Independence Day, August 17th
- Christmas, December 25th

Other holidays for which no rule is given (data available for 2005-2007 only:)

- Idul Adha
- Ied Adha
- Imlek
- Moslem's New Year Day
- Nyepi (Saka's New Year)
- Birthday of Prophet Muhammad SAW
- Waisak
- Ascension of Prophet Muhammad SAW
- Idul Fitri
- Ied Fitri
- Other national leaves

Public Types

- enum [Market](#) { [BEJ](#), [JSX](#) }

Public Member Functions

- [Indonesia](#) ([Market](#) m=[BEJ](#))

9.512.2 Member Enumeration Documentation

9.512.2.1 enum Market

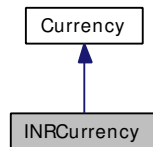
Enumerator:

- BEJ* Jakarta stock exchange.
- JSX* Jakarta stock exchange.

9.513 INRCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for INRCurrency:



9.513.1 Detailed Description

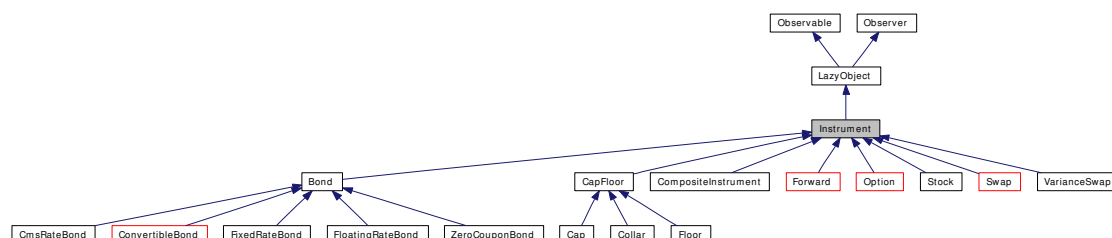
Indian rupee.

The ISO three-letter code is INR; the numeric code is 356. It is divided in 100 paise.

9.514 Instrument Class Reference

```
#include <ql/instrument.hpp>
```

Inheritance diagram for Instrument:



9.514.1 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](#) (PricingEngine::arguments *) const
- virtual void [fetchResults](#) (const PricingEngine::results *) const

Inspectors

- Real [NPV](#) () const
returns the net present value of the instrument.
- Real [errorEstimate](#) () const
returns the error estimate on the NPV when available.
- template<typename T>
T [result](#) (const std::string &tag) const
returns any additional result returned by the pricing engine.
- const std::map< std::string, boost::any > & [additionalResults](#) () const
returns all additional result returned by the pricing engine.
- 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_**
- std::map< std::string, boost::any > **additionalResults_**

9.514.2 Member Function Documentation

9.514.2.1 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.

9.514.2.2 void setupArguments (PricingEngine::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.

9.514.2.3 void fetchResults (const PricingEngine::results *) const [virtual]

When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented in [AssetSwap](#), [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoVanillaOption](#), and [VarianceSwap](#).

9.514.2.4 void calculate () const [protected, virtual]

This method performs all needed calculations by calling the **performCalculations** method.

Warning

Objects cache the results of the previous calculation. Such results will be returned upon later invocations of **calculate**. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.

Warning

Should this method be redefined in derived classes, [LazyObject::calculate\(\)](#) should be called in the overriding method.

Reimplemented from [LazyObject](#).

9.514.2.5 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented in [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoVanillaOption](#), [Swap](#), and [VarianceSwap](#).

9.514.2.6 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Implements [LazyObject](#).

Reimplemented in [Bond](#), [CompositeInstrument](#), [ConvertibleBond](#), [FixedRateBondForward](#), [Forward](#), [ForwardRateAgreement](#), [Stock](#), [Swap](#), and [VarianceSwap](#).

9.515 IntegralEngine Class Reference

```
#include <ql/pricingengines/vanilla/integralengine.hpp>
```

9.515.1 Detailed Description

Pricing engine for European vanilla options using integral approach.

Todo

- define tolerance for calculate()

Examples:

[EquityOption.cpp](#).

Public Member Functions

- void **calculate** () const

9.516 InterestRate Class Reference

```
#include <ql/interestrate.hpp>
```

9.516.1 Detailed Description

Concrete interest rate class.

This class encapsulate the interest rate compounding algebra. It manages day-counting conventions, compounding conventions, conversion between different conventions, discount/compound factor calculations, and implied/equivalent rate calculations.

Tests

Converted rates are checked against known good results

Public Member Functions

constructors

- [InterestRate](#) ()
Default constructor returning a null interest rate.
- [InterestRate](#) (Rate r, const [DayCounter](#) &dc, Compounding comp, [Frequency](#) freq=Annual)
Standard constructor.

conversions

- [operator Rate](#) () const

inspectors

- Rate [rate](#) () const
- const [DayCounter](#) & [dayCounter](#) () const
- Compounding [compounding](#) () const
- [Frequency](#) [frequency](#) () const

discount/compound factor calculations

- DiscountFactor [discountFactor](#) (Time t) const
discount factor implied by the rate compounded at time t.
- DiscountFactor [discountFactor](#) (const [Date](#) &d1, const [Date](#) &d2, const [Date](#) &refStart=[Date](#)(), const [Date](#) &refEnd=[Date](#)()) const
discount factor implied by the rate compounded between two dates
- Real [compoundFactor](#) (Time t) const
compound factor implied by the rate compounded at time t.
- Real [compoundFactor](#) (const [Date](#) &d1, const [Date](#) &d2, const [Date](#) &refStart=[Date](#)(), const [Date](#) &refEnd=[Date](#)()) const

compound factor implied by the rate compounded between two dates

equivalent rate calculations

- [InterestRate](#) [equivalentRate](#) (Time *t*, Compounding *comp*, [Frequency](#) *freq*=Annual) const
equivalent interest rate for a compounding period t.
- [InterestRate](#) [equivalentRate](#) (Date *d1*, Date *d2*, const [DayCounter](#) &*resultDC*, Compounding *comp*, [Frequency](#) *freq*=Annual) const
equivalent rate for a compounding period between two dates

Static Public Member Functions

implied rate calculations

- static [InterestRate](#) [impliedRate](#) (Real *compound*, Time *t*, const [DayCounter](#) &*resultDC*, Compounding *comp*, [Frequency](#) *freq*=Annual)
implied interest rate for a given compound factor at a given time.
- static [InterestRate](#) [impliedRate](#) (Real *compound*, const Date &*d1*, const Date &*d2*, const [DayCounter](#) &*resultDC*, Compounding *comp*, [Frequency](#) *freq*=Annual)
implied rate for a given compound factor between two dates.

Related Functions

(Note that these are not member functions.)

- `std::ostream & operator<< (std::ostream &, const InterestRate &)`

9.516.2 Member Function Documentation

9.516.2.1 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.

9.516.2.2 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.

9.516.2.3 Real compoundFactor (const Date & d1, const Date & d2, const Date & refStart = Date(), const Date & refEnd = Date()) const

compound factor implied by the rate compounded between two dates

returns the compound (a.k.a capitalization) factor implied by the rate compounded between two dates.

9.516.2.4 static InterestRate impliedRate (Real compound, Time t, const DayCounter & resultDC, Compounding comp, Frequency freq = Annual) [static]

implied interest rate for a given compound factor at a given time.

The resulting [InterestRate](#) has the day-counter provided as input.

Warning

Time must be measured using the day-counter provided as input.

9.516.2.5 static InterestRate impliedRate (Real compound, const Date & d1, const Date & d2, const DayCounter & resultDC, Compounding comp, Frequency freq = Annual) [static]

implied rate for a given compound factor between two dates.

The resulting rate is calculated taking the required day-counting rule into account.

9.516.2.6 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.

9.516.2.7 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.

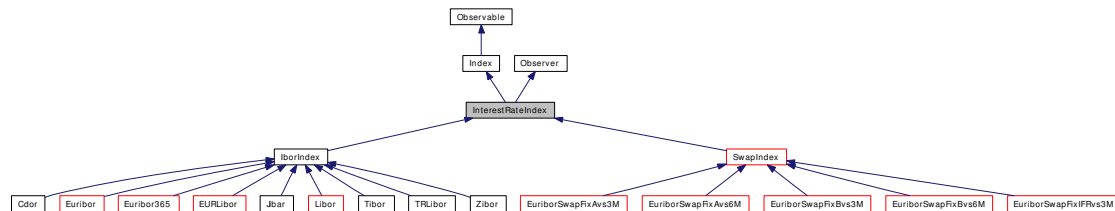
9.516.3 Friends And Related Function Documentation

9.516.3.1 std::ostream & operator<< (std::ostream &, const InterestRate &) [related]

9.517 InterestRateIndex Class Reference

```
#include <ql/indexes/interestrateindex.hpp>
```

Inheritance diagram for InterestRateIndex:



9.517.1 Detailed Description

base class for interest rate indexes

Todo

add methods returning [InterestRate](#)

Public Member Functions

- **InterestRateIndex** (const std::string &familyName, const [Period](#) &tenor, Natural settlementDays, const [Currency](#) ¤cy, const [Calendar](#) &fixingCalendar, const [DayCounter](#) &dayCounter)

Index interface

- std::string [name](#) () const
Returns the name of the index.
- Rate [fixing](#) (const [Date](#) &fixingDate, bool forecastTodaysFixing=false) const
returns the fixing at the given date

Observer interface

- void [update](#) ()

Inspectors

- std::string [familyName](#) () const
- [Period](#) [tenor](#) () const
- Natural [fixingDays](#) () const
- [Date](#) [fixingDate](#) (const [Date](#) &valueDate) const
- const [Currency](#) & [currency](#) () const
- const [DayCounter](#) & [dayCounter](#) () const
- virtual Rate [forecastFixing](#) (const [Date](#) &fixingDate) const=0
- virtual [Handle](#)< [YieldTermStructure](#) > [termStructure](#) () const=0
- virtual [Date](#) [maturityDate](#) (const [Date](#) &valueDate) const=0

Date calculations

These method can be overridden to implement particular conventions (e.g. EurLibor)

- virtual [Date](#) valueDate (const [Date](#) &fixingDate) const

Protected Attributes

- std::string familyName_
- [Period](#) tenor_
- Natural fixingDays_
- [Currency](#) currency_
- [DayCounter](#) dayCounter_

9.517.2 Member Function Documentation

9.517.2.1 std::string name () const [virtual]

Returns the name of the index.

Warning

This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implements [Index](#).

9.517.2.2 Rate fixing (const Date &fixingDate, bool forecastTodaysFixing = false) const [virtual]

returns the fixing at the given date

the date passed as arguments must be the actual calendar date of the fixing; no settlement days must be used.

Implements [Index](#).

9.517.2.3 void update () [virtual]

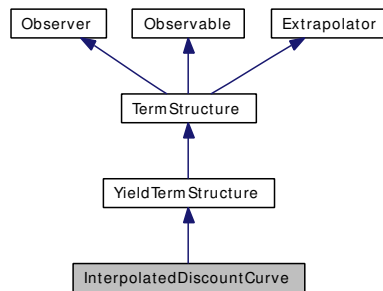
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.518 InterpolatedDiscountCurve Class Template Reference

```
#include <ql/termstructures/yieldcurves/discountcurve.hpp>
```

Inheritance diagram for InterpolatedDiscountCurve:



9.518.1 Detailed Description

```
template<class Interpolator> class QuantLib::InterpolatedDiscountCurve< Interpolator >
```

Term structure based on interpolation of discount factors.

Public Member Functions

- **InterpolatedDiscountCurve** (const std::vector< [Date](#) > &dates, const std::vector< DiscountFactor > &dfs, const [DayCounter](#) &dayCounter, const [Calendar](#) &cal=[Calendar](#)(), const Interpolator &interpolator=[Interpolator](#)())

Inspectors

- [Date](#) **maxDate** () const
the latest date for which the curve can return values
- const std::vector< Time > & **times** () const
- const std::vector< [Date](#) > & **dates** () const
- const std::vector< DiscountFactor > & **discounts** () const
- std::vector< std::pair< [Date](#), DiscountFactor > > **nodes** () const

Protected Member Functions

- **InterpolatedDiscountCurve** (const [DayCounter](#) &, const Interpolator &interpolator=[Interpolator](#)())
- **InterpolatedDiscountCurve** (const [Date](#) &referenceDate, const [DayCounter](#) &, const Interpolator &interpolator=[Interpolator](#)())
- **InterpolatedDiscountCurve** (Natural settlementDays, const [Calendar](#) &, const [DayCounter](#) &, const Interpolator &interpolator=[Interpolator](#)())
- DiscountFactor **discountImpl** (Time) const
discount calculation

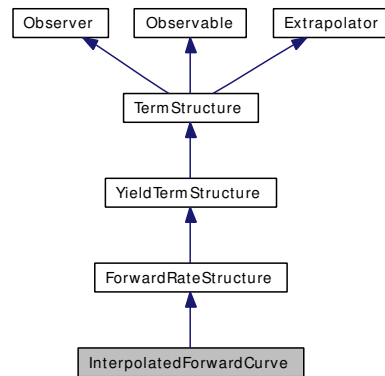
Protected Attributes

- `std::vector< Date > dates_`
- `std::vector< Time > times_`
- `std::vector< DiscountFactor > data_`
- `Interpolation interpolation_`
- `Interpolator interpolator_`

9.519 InterpolatedForwardCurve Class Template Reference

```
#include <ql/termstructures/yieldcurves/forwardcurve.hpp>
```

Inheritance diagram for InterpolatedForwardCurve:



9.519.1 Detailed Description

```
template<class Interpolator> class QuantLib::InterpolatedForwardCurve< Interpolator >
```

Term structure based on interpolation of forward rates.

Inspectors

- `std::vector< Date > dates_`
- `std::vector< Time > times_`
- `std::vector< Rate > data_`
- `Interpolation interpolation_`
- `Interpolator interpolator_`
- `Date maxDate () const`
the latest date for which the curve can return values
- `const std::vector< Time > & times () const`
- `const std::vector< Date > & dates () const`
- `const std::vector< Rate > & forwards () const`
- `std::vector< std::pair< Date, Rate > > nodes () const`
- `InterpolatedForwardCurve (const DayCounter &, const Interpolator &interpolator=Interpolator())`
- `InterpolatedForwardCurve (const Date &referenceDate, const DayCounter &, const Interpolator &interpolator=Interpolator())`
- `InterpolatedForwardCurve (Natural settlementDays, const Calendar &, const DayCounter &, const Interpolator &interpolator=Interpolator())`
- `Rate forwardImpl (Time t) const`
instantaneous forward-rate calculation
- `Rate zeroYieldImpl (Time t) const`

Public Member Functions

- **InterpolatedForwardCurve** (const std::vector< [Date](#) > &dates, const std::vector< Rate > &forwards, const [DayCounter](#) &dayCounter, const Interpolator &interpolator=Interpolator())

9.519.2 Member Function Documentation

9.519.2.1 Rate zeroYieldImpl (Time) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

[Warning](#)

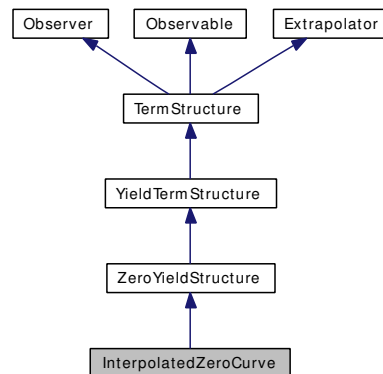
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

Reimplemented from [ForwardRateStructure](#).

9.520 InterpolatedZeroCurve Class Template Reference

```
#include <ql/termstructures/yieldcurves/zerocurve.hpp>
```

Inheritance diagram for InterpolatedZeroCurve:



9.520.1 Detailed Description

```
template<class Interpolator> class QuantLib::InterpolatedZeroCurve< Interpolator >
```

Term structure based on interpolation of zero yields.

Inspectors

- `std::vector< Date > dates_`
- `std::vector< Time > times_`
- `std::vector< Rate > data_`
- `Interpolation interpolation_`
- `Interpolator interpolator_`
- `Date maxDate () const`
the latest date for which the curve can return values
- `const std::vector< Time > & times () const`
- `const std::vector< Date > & dates () const`
- `const std::vector< Rate > & zeroRates () const`
- `std::vector< std::pair< Date, Rate > > nodes () const`
- `InterpolatedZeroCurve (const DayCounter &, const Interpolator & interpolator=Interpolator())`
- `InterpolatedZeroCurve (const Date &referenceDate, const DayCounter &, const Interpolator & interpolator=Interpolator())`
- `InterpolatedZeroCurve (Natural settlementDays, const Calendar &, const DayCounter &, const Interpolator & interpolator=Interpolator())`
- `Rate zeroYieldImpl (Time t) const`
zero-yield calculation

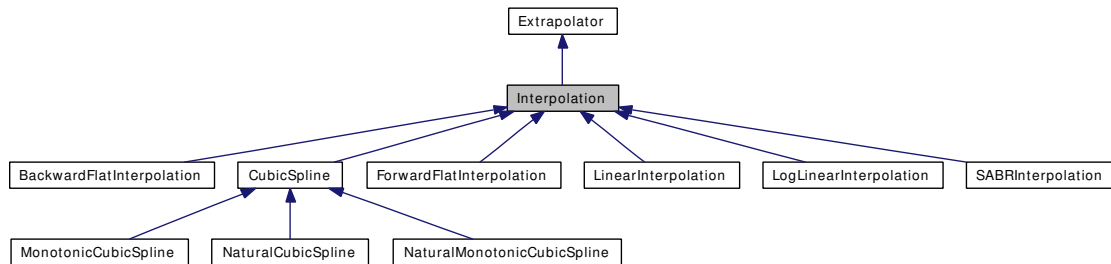
Public Member Functions

- **InterpolatedZeroCurve** (const std::vector< [Date](#) > &dates, const std::vector< Rate > &yields, const [DayCounter](#) &dayCounter, const Interpolator &interpolator=Interpolator())

9.521 Interpolation Class Reference

```
#include <ql/math/interpolation.hpp>
```

Inheritance diagram for Interpolation:



9.521.1 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

- bool **empty** () const
- Real **operator()** (Real x, bool allowExtrapolation=false) const
- Real **primitive** (Real x, bool allowExtrapolation=false) const
- Real **derivative** (Real x, bool allowExtrapolation=false) const
- Real **secondDerivative** (Real x, bool allowExtrapolation=false) const
- Real **xMin** () const
- Real **xMax** () const
- bool **isInRange** (Real x) const
- void **update** ()

Protected Member Functions

- void **checkRange** (Real x, bool extrapolate) const

Protected Attributes

- boost::shared_ptr< [Impl](#) > **impl_**

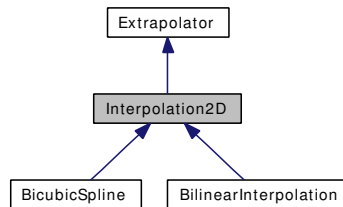
Classes

- class [Impl](#)
abstract base class for interpolation implementations
- class [templateImpl](#)
basic template implementation

9.522 Interpolation2D Class Reference

```
#include <ql/math/interpolations/interpolation2d.hpp>
```

Inheritance diagram for Interpolation2D:



9.522.1 Detailed Description

base class for 2-D interpolations.

Classes derived from this class will provide interpolated values from two sequences of length N and M , representing the discretized values of the x and y variables, and a $N \times M$ matrix representing the tabulated function values.

Public Types

- typedef Real **first_argument_type**
- typedef Real **second_argument_type**
- typedef Real **result_type**

Public Member Functions

- Real **operator()** (Real x , Real y , bool allowExtrapolation=false) const
- Real **xMin** () const
- Real **xMax** () const
- std::vector< Real > **xValues** () const
- Size **locateX** (Real x) const
- Real **yMin** () const
- Real **yMax** () const
- std::vector< Real > **yValues** () const
- Size **locateY** (Real y) const
- const [Matrix](#) & **zData** () const
- bool **isInRange** (Real x , Real y) const
- void **update** ()

Protected Member Functions

- void **checkRange** (Real x , Real y , bool extrapolate) const

Protected Attributes

- `boost::shared_ptr< Impl > impl_`

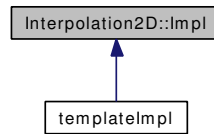
Classes

- class [Impl](#)
abstract base class for 2-D interpolation implementations
- class [templateImpl](#)
basic template implementation

9.523 Interpolation2D::Impl Class Reference

```
#include <ql/math/interpolations/interpolation2d.hpp>
```

Inheritance diagram for Interpolation2D::Impl:



9.523.1 Detailed Description

abstract base class for 2-D interpolation implementations

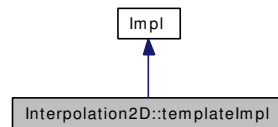
Public Member Functions

- virtual void **calculate** ()=0
- virtual Real **xMin** () const=0
- virtual Real **xMax** () const=0
- virtual std::vector< Real > **xValues** () const=0
- virtual Size **locateX** (Real x) const=0
- virtual Real **yMin** () const=0
- virtual Real **yMax** () const=0
- virtual std::vector< Real > **yValues** () const=0
- virtual Size **locateY** (Real y) const=0
- virtual const [Matrix](#) & **zData** () const=0
- virtual bool **isInRange** (Real x, Real y) const=0
- virtual Real **value** (Real x, Real y) const=0

9.524 Interpolation2D::templateImpl Class Template Reference

```
#include <ql/math/interpolations/interpolation2d.hpp>
```

Inheritance diagram for Interpolation2D::templateImpl:



9.524.1 Detailed Description

```
template<class I1, class I2, class M> class QuantLib::Interpolation2D::templateImpl< I1, I2, M
>
```

basic template implementation

Public Member Functions

- **templateImpl** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)
- Real **xMin** () const
- Real **xMax** () const
- std::vector< Real > **xValues** () const
- Real **yMin** () const
- Real **yMax** () const
- std::vector< Real > **yValues** () const
- const [Matrix](#) & **zData** () const
- bool **isInRange** (Real x, Real y) const

Protected Member Functions

- Size **locateX** (Real x) const
- Size **locateY** (Real y) const

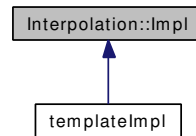
Protected Attributes

- I1 **xBegin_**
- I1 **xEnd_**
- I2 **yBegin_**
- I2 **yEnd_**
- const M & **zData_**

9.525 Interpolation::Impl Class Reference

```
#include <ql/math/interpolation.hpp>
```

Inheritance diagram for Interpolation::Impl:



9.525.1 Detailed Description

abstract base class for interpolation implementations

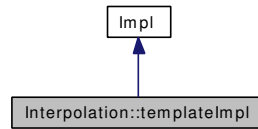
Public Member Functions

- virtual void **update** ()=0
- virtual Real **xMin** () const=0
- virtual Real **xMax** () const=0
- virtual std::vector< Real > **xValues** () const=0
- virtual std::vector< Real > **yValues** () 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

9.526 Interpolation::templateImpl Class Template Reference

```
#include <ql/math/interpolation.hpp>
```

Inheritance diagram for Interpolation::templateImpl:



9.526.1 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
- std::vector< Real > **xValues** () const
- std::vector< Real > **yValues** () const
- bool **isInRange** (Real x) const

Protected Member Functions

- Size **locate** (Real x) const

Protected Attributes

- I1 **xBegin_**
- I1 **xEnd_**
- I2 **yBegin_**

9.527 IntervalPrice Class Reference

```
#include <ql/prices.hpp>
```

9.527.1 Detailed Description

interval price

Public Types

- enum **Type** { **Open**, **Close**, **High**, **Low** }

Public Member Functions

- **IntervalPrice** (Real open, Real close, Real high, Real low)

Inspectors

- Real **open** () const
- Real **close** () const
- Real **high** () const
- Real **low** () const
- Real **value** (IntervalPrice::Type) const

Modifiers

- void **setValue** (Real value, IntervalPrice::Type)
- void **setValues** (Real open, Real close, Real high, Real low)

Static Public Member Functions

Helper functions

- static [TimeSeries](#)< [IntervalPrice](#) > **makeSeries** (const std::vector< [Date](#) > &d, const std::vector< Real > &open, const std::vector< Real > &close, const std::vector< Real > &high, const std::vector< Real > &low)
- static std::vector< Real > **extractValues** (const [TimeSeries](#)< [IntervalPrice](#) > &, IntervalPrice::Type)
- static [TimeSeries](#)< Real > **extractComponent** (const [TimeSeries](#)< [IntervalPrice](#) > &, enum IntervalPrice::Type)

9.528 InverseCumulativeNormal Class Reference

```
#include <ql/math/distributions/normaldistribution.hpp>
```

9.528.1 Detailed Description

Inverse cumulative normal distribution function.

Given x between zero and one as the integral value of a gaussian normal distribution this class provides the value y such that formula here ...

It use Acklam's approximation: by Peter J. Acklam, University of Oslo, Statistics Division. URL: <http://home.online.no/~pjacklam/notes/invnorm/index.html>

This class can also be used to generate a gaussian normal distribution from a uniform distribution. This is especially useful when a gaussian normal distribution is generated from a low discrepancy uniform distribution: in this case the traditional Box-Muller approach and its variants would not preserve the sequence's low-discrepancy.

Public Member Functions

- **InverseCumulativeNormal** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real x) const

9.529 InverseCumulativePoisson Class Reference

```
#include <ql/math/distributions/poissondistribution.hpp>
```

9.529.1 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

9.530 InverseCumulativeRng Class Template Reference

```
#include <ql/math/randomnumbers/inversecumulativerng.hpp>
```

9.530.1 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

9.531 InverseCumulativeRsg Class Template Reference

```
#include <ql/math/randomnumbers/inversecumulativrsg.hpp>
```

9.531.1 Detailed Description

```
template<class USG, class IC> class QuantLib::InverseCumulativeRsg< USG, IC >
```

Inverse cumulative random sequence generator.

It uses a sequence of uniform deviate in (0, 1) as the source of cumulative distribution values. Then an inverse cumulative distribution is used to calculate the distribution deviate.

The uniform deviate sequence is supplied by USG.

Class USG must implement the following interface:

```
USG::sample_type USG::nextSequence() const;
Size USG::dimension() const;
```

The inverse cumulative distribution is supplied by IC.

Class IC must implement the following interface:

```
IC::IC();
Real IC::operator() const;
```

Examples:

[DiscreteHedging.cpp](#).

Public Types

- typedef [Sample](#)< std::vector< Real > > **sample_type**

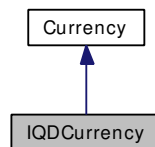
Public Member Functions

- **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

9.532 IQDCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for IQDCurrency:



9.532.1 Detailed Description

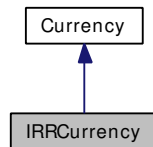
Iraqi dinar.

The ISO three-letter code is IQD; the numeric code is 368. It is divided in 1000 fils.

9.533 IRRCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for IRRCurrency:



9.533.1 Detailed Description

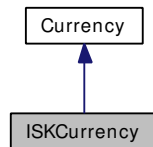
Iranian rial.

The ISO three-letter code is IRR; the numeric code is 364. It has no subdivisions.

9.534 ISKCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for ISKCurrency:



9.534.1 Detailed Description

Icelandic krona.

The ISO three-letter code is ISK; the numeric code is 352. It is divided in 100 aurar.

9.535 Italy Class Reference

```
#include <ql/time/calendars/italy.hpp>
```

Inheritance diagram for Italy:



9.535.1 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](#))

9.535.2 Member Enumeration Documentation

9.535.2.1 enum Market

Italian calendars.

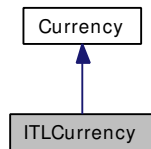
Enumerator:

Settlement generic settlement calendar
Exchange Milan stock-exchange calendar.

9.536 ITLCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for ITLCurrency:



9.536.1 Detailed Description

Italian lira.

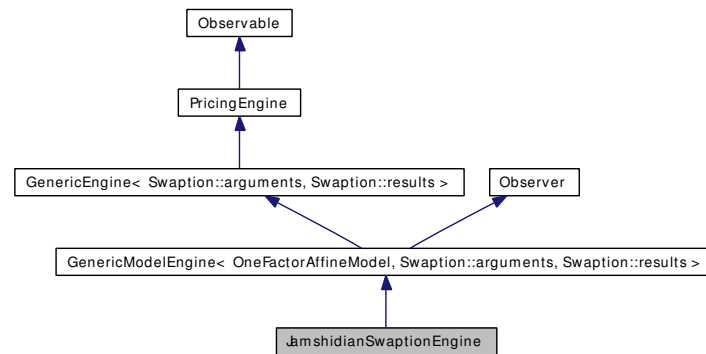
The ISO three-letter code was ITL; the numeric code was 380. It had no subdivisions.

Obsoleted by the Euro since 1999.

9.537 JamshidianSwaptionEngine Class Reference

```
#include <ql/pricingengines/swaption/jamshidianswaptionengine.hpp>
```

Inheritance diagram for JamshidianSwaptionEngine:



9.537.1 Detailed Description

Jamshidian swaption engine.

Warning

The engine assumes that the exercise date equals the start date of the passed swap.

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

- **JamshidianSwaptionEngine** (const boost::shared_ptr< [OneFactorAffineModel](#) > &model)
- void **calculate** () const

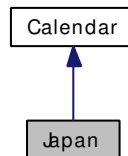
Friends

- class **rStarFinder**

9.538 Japan Class Reference

```
#include <ql/time/calendars/japan.hpp>
```

Inheritance diagram for Japan:



9.538.1 Detailed Description

Japanese calendar.

Holidays:

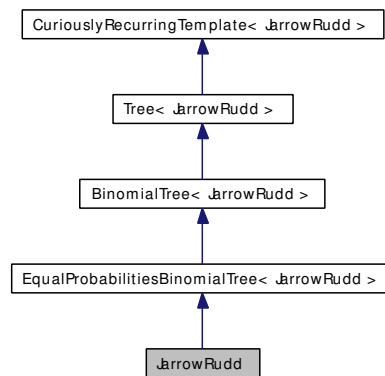
- Saturdays
- Sundays
- New Year's Day, January 1st
- Bank Holiday, January 2nd
- Bank Holiday, January 3rd
- Coming of Age Day, 2nd Monday in January
- National Foundation Day, February 11th
- Vernal Equinox
- Greenery Day, April 29th
- Constitution Memorial Day, May 3rd
- Holiday for a Nation, May 4th
- Children's Day, May 5th
- Marine Day, 3rd Monday in July
- Respect for the Aged Day, 3rd Monday in September
- Autumnal Equinox
- Health and Sports Day, 2nd Monday in October
- National Culture Day, November 3rd
- Labor Thanksgiving Day, November 23rd
- Emperor's Birthday, December 23rd
- Bank Holiday, December 31st
- a few one-shot holidays

Holidays falling on a Sunday are observed on the Monday following except for the bank holidays associated with the new year.

9.539 JarrowRudd Class Reference

```
#include <ql/methods/lattices/binomialtree.hpp>
```

Inheritance diagram for JarrowRudd:



9.539.1 Detailed Description

Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

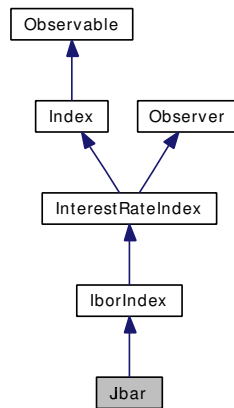
Public Member Functions

- **JarrowRudd** (const boost::shared_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)

9.540 Jibar Class Reference

```
#include <ql/indexes/ibor/jibar.hpp>
```

Inheritance diagram for Jibar:



9.540.1 Detailed Description

JIBAR rate

Johannesburg Interbank Agreed Rate

Todo

check settlement days and day-count convention.

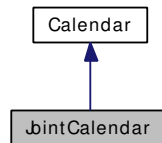
Public Member Functions

- **Jibar** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.541 JointCalendar Class Reference

```
#include <ql/time/calendars/jointcalendar.hpp>
```

Inheritance diagram for JointCalendar:



9.541.1 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.

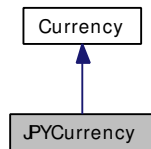
Public Member Functions

- **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)

9.542 JPYCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for JPYCurrency:



9.542.1 Detailed Description

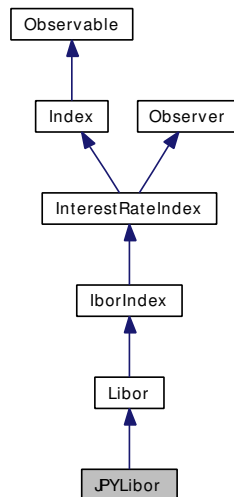
Japanese yen.

The ISO three-letter code is JPY; the numeric code is 392. It is divided into 100 sen.

9.543 JPYLibor Class Reference

```
#include <ql/indexes/ibor/jpylibor.hpp>
```

Inheritance diagram for JPYLibor:



9.543.1 Detailed Description

JPY LIBOR rate

Japanese Yen LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

Warning

This is the rate fixed in London by BBA. Use TIBOR if you're interested in the Tokio fixing.

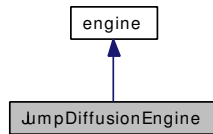
Public Member Functions

- **JPYLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), Natural settlementDays=2)

9.544 JumpDiffusionEngine Class Reference

```
#include <ql/pricingengines/vanilla/jumpdiffusionengine.hpp>
```

Inheritance diagram for JumpDiffusionEngine:



9.544.1 Detailed Description

Jump-diffusion engine for vanilla options.

Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

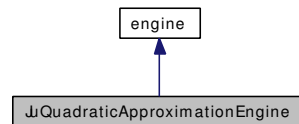
Public Member Functions

- **JumpDiffusionEngine** (const boost::shared_ptr< VanillaOption::engine > &, Real relativeAccuracy_=1e-4, Size maxIterations=100)
- void **calculate** () const

9.545 JuQuadraticApproximationEngine Class Reference

```
#include <ql/pricingengines/vanilla/juquadraticengine.hpp>
```

Inheritance diagram for JuQuadraticApproximationEngine:



9.545.1 Detailed Description

Pricing engine for American options with Ju quadratic approximation.

Reference: 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.

Public Member Functions

- void **calculate** () const

9.546 KnuthUniformRng Class Reference

```
#include <ql/math/randomnumbers/knuthuniformrng.hpp>
```

9.546.1 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

9.546.2 Constructor & Destructor Documentation

9.546.2.1 KnuthUniformRng (long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

9.546.3 Member Function Documentation

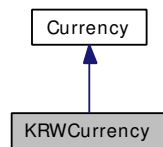
9.546.3.1 KnuthUniformRng::sample_type next () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

9.547 KRWCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for KRWCurrency:



9.547.1 Detailed Description

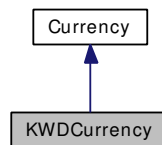
South-Korean won.

The ISO three-letter code is KRW; the numeric code is 410. It is divided in 100 chon.

9.548 KWDCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for KWDCurrency:



9.548.1 Detailed Description

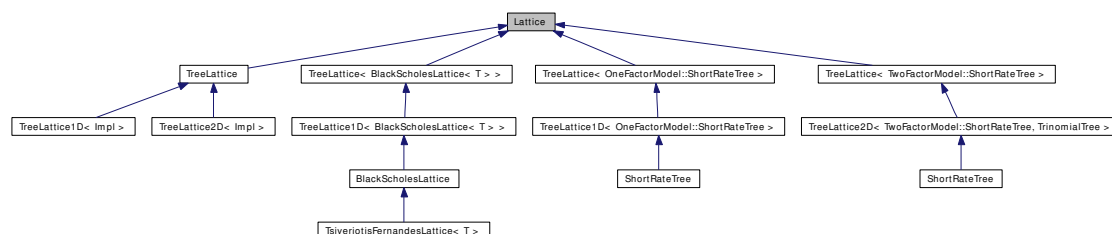
Kuwaiti dinar.

The ISO three-letter code is KWD; the numeric code is 414. It is divided in 1000 fils.

9.549 Lattice Class Reference

#include <ql/numericalmethod.hpp>

Inheritance diagram for Lattice:



9.549.1 Detailed Description

Lattice (tree, finite-differences) base class

Public Member Functions

- **Lattice** (const [TimeGrid](#) &timeGrid)
- virtual [Disposable](#)< [Array](#) > **grid** (Time) const=0

Inspectors

- const [TimeGrid](#) & **timeGrid** () const

Numerical method interface

These methods are to be used by discretized assets and must be overridden by developers implementing numerical methods. Users are advised to use the corresponding methods of [DiscretizedAsset](#) instead.

- virtual void [initialize](#) ([DiscretizedAsset](#) &, Time time) const =0
initialize an asset at the given time.
- virtual void [rollback](#) ([DiscretizedAsset](#) &, Time to) const=0
- virtual void [partialRollback](#) ([DiscretizedAsset](#) &, Time to) const=0
- virtual Real [presentValue](#) ([DiscretizedAsset](#) &) const=0
computes the present value of an asset.

Protected Attributes

- [TimeGrid](#) t_

9.549.2 Member Function Documentation

9.549.2.1 virtual void rollback ([DiscretizedAsset](#) &, Time to) const [pure virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implemented in [TreeLattice](#), [TsiveriotisFernandesLattice](#), [TreeLattice< OneFactorModel::ShortRateTree >](#), [TreeLattice< TwoFactorModel::ShortRateTree >](#), and [TreeLattice< BlackScholesLattice< T > >](#).

9.549.2.2 virtual void partialRollback (DiscretizedAsset &, Time *to*) const [pure virtual]

Roll back an asset until the given time, but do not perform the final adjustment.

Warning

In version 0.3.7 and earlier, this method was called `rollAlmostBack` method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

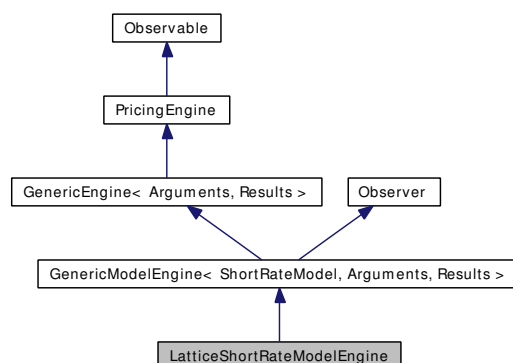
```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

Implemented in [TreeLattice](#), [TsiveriotisFernandesLattice](#), [TreeLattice< OneFactorModel::ShortRateTree >](#), [TreeLattice< TwoFactorModel::ShortRateTree >](#), and [TreeLattice< BlackScholesLattice< T > >](#).

9.550 LatticeShortRateModelEngine Class Template Reference

```
#include <ql/pricingengines/latticeshortratemodelengine.hpp>
```

Inheritance diagram for LatticeShortRateModelEngine:



9.550.1 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_

9.550.2 Member Function Documentation

9.550.2.1 void update () [virtual]

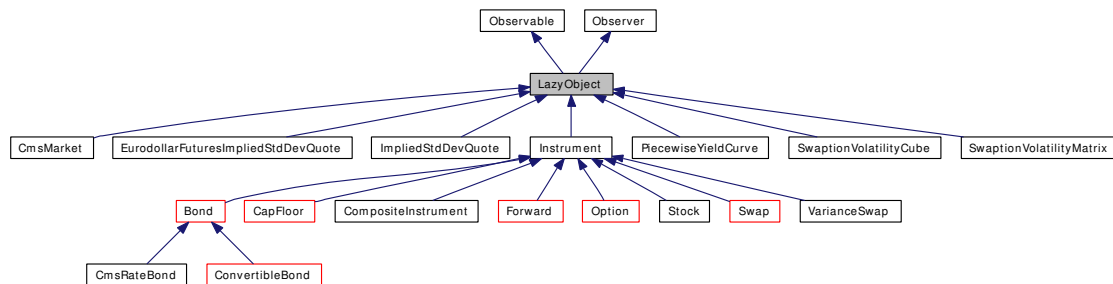
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [GenericModelEngine< ShortRateModel, Arguments, Results >](#).

9.551 LazyObject Class Reference

```
#include <ql/patterns/lazyobject.hpp>
```

Inheritance diagram for LazyObject:



9.551.1 Detailed Description

Framework for calculation on demand and result caching.

Calculations

These methods do not modify the structure of the object and are therefore declared as `const`. Data members which will be calculated on demand need to be declared as mutable.

- void [recalculate](#) ()
- void [freeze](#) ()
- void [unfreeze](#) ()
- virtual void [calculate](#) () const
- virtual void [performCalculations](#) () const=0

Public Member Functions

Observer interface

- void [update](#) ()

Protected Attributes

- bool `calculated_`
- bool `frozen_`

9.551.2 Member Function Documentation

9.551.2.1 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 [CmsMarket](#), [SwaptionVolatilityCube](#), [SwaptionVolatilityMatrix](#), and [PiecewiseYieldCurve](#).

9.551.2.2 void recalculate ()

This method force the recalculation of any results which would otherwise be cached. It is not declared as `const` since it needs to call the non-`const` `notifyObservers` method.

Note:

Explicit invocation of this method is **not** necessary if the object registered itself as observer with the structures on which such results depend. It is strongly advised to follow this policy when possible.

9.551.2.3 void freeze ()

This method constrains the object to return the presently cached results on successive invocations, even if arguments upon which they depend should change.

9.551.2.4 void unfreeze ()

This method reverts the effect of the `freeze` method, thus re-enabling recalculations.

9.551.2.5 void calculate () const [protected, virtual]

This method performs all needed calculations by calling the `performCalculations` method.

Warning

Objects cache the results of the previous calculation. Such results will be returned upon later invocations of `calculate`. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.

Warning

Should this method be redefined in derived classes, `LazyObject::calculate()` should be called in the overriding method.

Reimplemented in [Instrument](#).

9.551.2.6 virtual void performCalculations () const [protected, pure virtual]

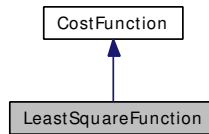
This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implemented in [Instrument](#), [Bond](#), [CompositeInstrument](#), [ConvertibleBond](#), [FixedRateBondForward](#), [Forward](#), [ForwardRateAgreement](#), [Stock](#), [Swap](#), [VarianceSwap](#), [EurodollarFuturesImpliedStdDevQuote](#), [ImpliedStdDevQuote](#), and [SwaptionVolatilityMatrix](#).

9.552 LeastSquareFunction Class Reference

```
#include <ql/math/optimization/leastsquare.hpp>
```

Inheritance diagram for LeastSquareFunction:



9.552.1 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 [Disposable](#)< [Array](#) > [values](#) (const [Array](#) &x) const
method to overload to compute the cost function values in x
- 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

9.553 LeastSquareProblem Class Reference

```
#include <ql/math/optimization/leastsquare.hpp>
```

9.553.1 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

9.553.2 Member Function Documentation

9.553.2.1 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

9.554 LecuyerUniformRng Class Reference

```
#include <ql/math/randomnumbers/lecuyeruniformrng.hpp>
```

9.554.1 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

9.554.2 Constructor & Destructor Documentation

9.554.2.1 LecuyerUniformRng (long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

9.554.3 Member Function Documentation

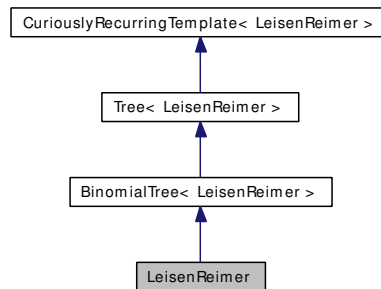
9.554.3.1 sample_type next () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

9.555 LeisenReimer Class Reference

```
#include <ql/methods/lattices/binomialtree.hpp>
```

Inheritance diagram for LeisenReimer:



9.555.1 Detailed Description

Leisen & Reimer tree: multiplicative approach.

Public Member Functions

- **LeisenReimer** (const boost::shared_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size branch) const

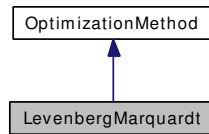
Protected Attributes

- Real **up_**
- Real **down_**
- Real **pu_**
- Real **pd_**

9.556 LevenbergMarquardt Class Reference

```
#include <ql/math/optimization/levenbergmarquardt.hpp>
```

Inheritance diagram for LevenbergMarquardt:



9.556.1 Detailed Description

Levenberg-Marquardt optimization method.

This implementation is based on MINPACK (<http://www.netlib.org/minpack>), <http://www.netlib.org/cephes/linalg.tgz>)

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

- **LevenbergMarquardt** (Real epsfcn=1.0e-8, Real xtol=1.0e-8, Real gtol=1.0e-8)
- virtual EndCriteria::Type **minimize** ([Problem](#) &P, const [EndCriteria](#) &endCriteria)
minimize the optimization problem P
- virtual Integer **getInfo** () const

Static Public Member Functions

- static void **fcn** (int m, int n, double *x, double *fvec, int *iflag)

9.557 LexicographicalView Class Template Reference

```
#include <ql/math/lexicographicalview.hpp>
```

9.557.1 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)

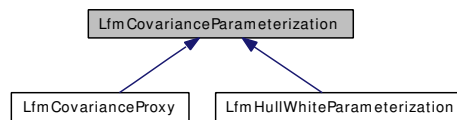
Inspectors

- Size `xSize ()` const
dimension of the array along x
- Size `ySize ()` const
dimension of the array along y

9.558 LfmCovarianceParameterization Class Reference

```
#include <ql/processes/lfmcovarParams.hpp>
```

Inheritance diagram for LfmCovarianceParameterization:



9.558.1 Detailed Description

Libor market model parameterization

Brigo, Damiano, Mercurio, Fabio, Morini, Massimo, 2003, Different Covariance Parameterizations of the [Libor](#) Market Model and Joint Caps/Swaptions Calibration (<http://www.exoticderivatives.com/Files/Papers/brigomercuriomorini.pdf>)

Public Member Functions

- **LfmCovarianceParameterization** (Size size, Size factors)
- Size **size** () const
- Size **factors** () const
- virtual [Disposable](#)< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const=0
- virtual [Disposable](#)< [Matrix](#) > **covariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual [Disposable](#)< [Matrix](#) > **integratedCovariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const

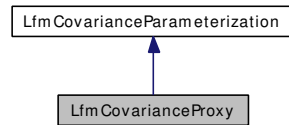
Protected Attributes

- const Size **size_**
- const Size **factors_**

9.559 LfmCovarianceProxy Class Reference

```
#include <ql/legacy/libormarketmodels/lfmcovarproxy.hpp>
```

Inheritance diagram for LfmCovarianceProxy:



9.559.1 Detailed Description

proxy for a libor forward model covariance parameterization

Public Member Functions

- **LfmCovarianceProxy** (const boost::shared_ptr< [LmVolatilityModel](#) > &volaModel, const boost::shared_ptr< [LmCorrelationModel](#) > &corrModel)
- boost::shared_ptr< [LmVolatilityModel](#) > **volatilityModel** () const
- boost::shared_ptr< [LmCorrelationModel](#) > **correlationModel** () const
- [Disposable](#)< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Disposable](#)< [Matrix](#) > **covariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual Real **integratedCovariance** (Size i, Size j, Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const

Protected Attributes

- const boost::shared_ptr< [LmVolatilityModel](#) > **volaModel_**
- const boost::shared_ptr< [LmCorrelationModel](#) > **corrModel_**

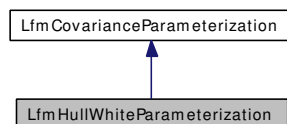
Friends

- class **Var_Helper**

9.560 LfmHullWhiteParameterization Class Reference

```
#include <ql/processes/lfmhullwhiteparam.hpp>
```

Inheritance diagram for LfmHullWhiteParameterization:



9.560.1 Detailed Description

Libor market model parameterization based on Hull White paper

Hull, John, White, Alan, 1999, [Forward Rate Volatilities, Swap Rate Volatilities and the Implementation of the Libor Market Model](http://www.rotman.utoronto.ca/~amackay/fin/libormktmodel2.pdf) (<http://www.rotman.utoronto.ca/~amackay/fin/libormktmodel2.pdf>)

Tests

the correctness is tested by Monte-Carlo reproduction of caplet & ratchet npvs and comparison with Black pricing.

Public Member Functions

- **LfmHullWhiteParameterization** (const boost::shared_ptr< [LiborForwardModelProcess](#) > &process, const boost::shared_ptr< [CapletVolatilityStructure](#) > &capletVol, const [Matrix](#) &correlation=[Matrix](#)(), Size factors=1)
- [Disposable](#)< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Disposable](#)< [Matrix](#) > **covariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Disposable](#)< [Matrix](#) > **integratedCovariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const

Protected Member Functions

- Size **nextIndexReset** (Time t) const

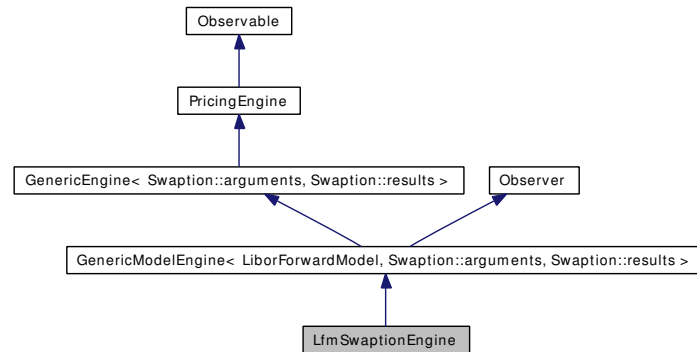
Protected Attributes

- [Matrix](#) **diffusion_**
- [Matrix](#) **covariance_**
- std::vector< Time > **fixingTimes_**

9.561 LfmSwaptionEngine Class Reference

```
#include <ql/pricingengines/swaption/lfmwaptionengine.hpp>
```

Inheritance diagram for LfmSwaptionEngine:



9.561.1 Detailed Description

Libor forward model swaption engine based on Black formula

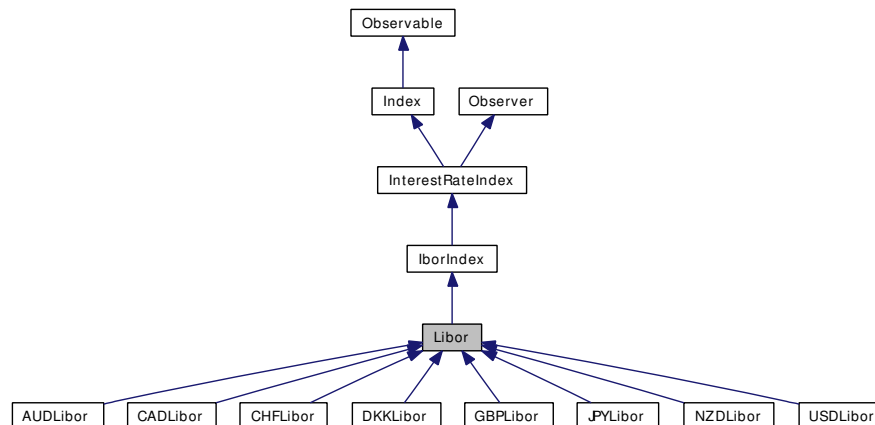
Public Member Functions

- **LfmSwaptionEngine** (const boost::shared_ptr< [LiborForwardModel](#) > &model)
- void **calculate** () const

9.562 Libor Class Reference

```
#include <ql/indexes/ibor/libor.hpp>
```

Inheritance diagram for Libor:



9.562.1 Detailed Description

base class for all BBA LIBOR indexes but the EUR ones

LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

Warning

This is not a valid base class for the O/N, S/N index

Public Member Functions

- **Libor** (const std::string &familyName, const [Period](#) &tenor, Natural settlementDays, const [Currency](#) ¤cy, const [Calendar](#) &financialCenterCalendar, const [DayCounter](#) &dayCounter, const [Handle](#)< [YieldTermStructure](#) > &h)

Date calculations

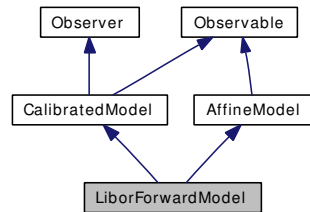
see <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1412>

- **Date** [valueDate](#) (const [Date](#) &fixingDate) const
- **Date** [maturityDate](#) (const [Date](#) &valueDate) const

9.563 LiborForwardModel Class Reference

```
#include <ql/legacy/libormarketmodels/liborforwardmodel.hpp>
```

Inheritance diagram for LiborForwardModel:



9.563.1 Detailed Description

Libor forward model

References:

Stefan Weber, 2005, Efficient Calibration for [Libor](http://workshop.mathfinance.de/2005/papers/weber/slides.pdf) Market Models, (<<http://workshop.mathfinance.de/2005/papers/weber/slides.pdf>>)

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of [Libor](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf) Market Model and Joint Caps/Swaptions Calibration, (<http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf>)

Tests

the correctness is tested using Monte-Carlo Simulation to reproduce swaption npvs, model calibration and exact cap pricing

Public Member Functions

- **LiborForwardModel** (const boost::shared_ptr< [LiborForwardModelProcess](#) > &process, const boost::shared_ptr< [LmVolatilityModel](#) > &volaModel, const boost::shared_ptr< [LmCorrelationModel](#) > &corrModel)
- Rate **S_0** (Size alpha, Size beta) const
- virtual boost::shared_ptr< [SwaptionVolatilityMatrix](#) > **getSwaptionVolatilityMatrix** () const
- DiscountFactor **discount** (Time t) const
Implied discount curve.
- Real **discountBond** (Time now, Time maturity, [Array](#) factors) const
- Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const
- void **setParams** (const [Array](#) ¶ms)

Protected Member Functions

- **Disposable**< [Array](#) > **w_0** (Size alpha, Size beta) const

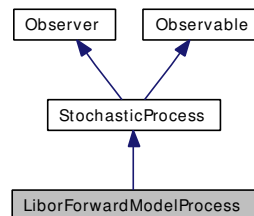
Protected Attributes

- `std::vector< Real > f_`
- `std::vector< Time > accrualPeriod_`
- `const boost::shared_ptr< LfmCovarianceProxy > covarProxy_`
- `const boost::shared_ptr< LiborForwardModelProcess > process_`
- `boost::shared_ptr< SwaptionVolatilityMatrix > swaptionVola`

9.564 LiborForwardModelProcess Class Reference

```
#include <ql/processes/lfmprocess.hpp>
```

Inheritance diagram for LiborForwardModelProcess:



9.564.1 Detailed Description

libor-forward-model process

stochastic process of a libor forward model using the rolling forward measure incl. predictor-corrector step

References:

Glasserman, Paul, 2004, Monte Carlo Methods in Financial Engineering, Springer, Section 3.7

Antoon Pelsser, 2000, Efficient Methods for Valuing Interest Rate Derivatives, Springer, 8

Hull, John, White, Alan, 1999, [Forward](#) Rate Volatilities, [Swap](#) Rate Volatilities and the Implementation of the [Libor](#) Market Model (<http://www.rotman.utoronto.ca/~amackay/fin/libormktmodel2.pdf>)

Tests

the correctness is tested by Monte-Carlo reproduction of caplet & ratchet NPVs and comparison with Black pricing.

Warning

this class does not work correctly with Visual C++ 6.

Public Member Functions

- **LiborForwardModelProcess** (Size size, const boost::shared_ptr< [IborIndex](#) > &index)
- **Disposable**< [Array](#) > **initialValues** () const
returns the initial values of the state variables
- **Disposable**< [Array](#) > **drift** (Time t, const [Array](#) &x) const
returns the drift part of the equation, i.e., $\mu(t, x_t)$
- **Disposable**< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- **Disposable**< [Matrix](#) > **covariance** (Time t0, const [Array](#) &x0, Time dt) const

- `Disposable< Array > apply (const Array &x0, const Array &dx) const`
- `Disposable< Array > evolve (Time t0, const Array &x0, Time dt, const Array &dw) const`
- `Size size () const`
returns the number of dimensions of the stochastic process
- `Size factors () const`
returns the number of independent factors of the process
- `boost::shared_ptr< LiborIndex > index () const`
- `Leg cashFlows (Real amount=1.0) const`
- `void setCovarParam (const boost::shared_ptr< LfmCovarianceParameterization > ¶m)`
- `boost::shared_ptr< LfmCovarianceParameterization > covarParam () const`
- `Size nextIndexReset (Time t) const`
- `const std::vector< Time > & fixingTimes () const`
- `const std::vector< Date > & fixingDates () const`
- `const std::vector< Time > & accrualStartTimes () const`
- `const std::vector< Time > & accrualEndTimes () const`
- `std::vector< DiscountFactor > discountBond (const std::vector< Rate > &rates) const`

9.564.2 Member Function Documentation

9.564.2.1 Disposable<Matrix> covariance (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the covariance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval 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](#).

9.564.2.2 Disposable<Array> apply (const Array & *x0*, const Array & *dx*) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented from [StochasticProcess](#).

9.564.2.3 Disposable<Array> evolve (Time *t0*, const Array & *x0*, Time *dt*, const Array & *dw*) const [virtual]

returns the asset value after a time interval Δt according to the given discretization. By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where E is the expectation and S the standard deviation.

Reimplemented from [StochasticProcess](#).

9.565 Linear Class Reference

```
#include <ql/math/interpolations/linearinterpolation.hpp>
```

9.565.1 Detailed Description

Linear-interpolation factory and traits

Public Types

- enum { **global** = 0 }

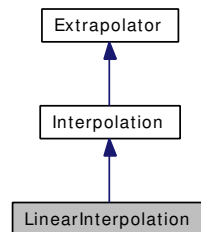
Public Member Functions

- template<class I1, class I2>
[Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

9.566 LinearInterpolation Class Reference

```
#include <ql/math/interpolations/linearinterpolation.hpp>
```

Inheritance diagram for LinearInterpolation:



9.566.1 Detailed Description

Linear interpolation between discrete points

Public Member Functions

- `template<class I1, class I2>`
[LinearInterpolation](#) (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

9.566.2 Constructor & Destructor Documentation

9.566.2.1 LinearInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

Precondition:

the *x* values must be sorted.

9.567 LinearLeastSquaresRegression Class Template Reference

```
#include <ql/math/linearleastsquaresregression.hpp>
```

9.567.1 Detailed Description

template<class ArgumentType = Real> class QuantLib::LinearLeastSquaresRegression< ArgumentType >

general linear least squares regression

References: "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

Tests

the correctness of the returned values is tested by checking their properties.

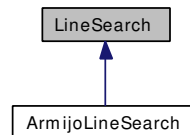
Public Member Functions

- **LinearLeastSquaresRegression** (const std::vector< ArgumentType > &x, const std::vector< Real > &y, const std::vector< boost::function1< Real, ArgumentType > > &v)
- const [Array](#) & **a** () const
- const [Array](#) & **err** () const

9.568 LineSearch Class Reference

```
#include <ql/math/optimization/lineSearch.hpp>
```

Inheritance diagram for LineSearch:



9.568.1 Detailed Description

Base class for line search.

Public Member Functions

- [LineSearch](#) (Real=0.0)
Default constructor.
- virtual [~LineSearch](#) ()
Destructor.
- const [Array](#) & [lastX](#) ()
return last x value
- Real [lastFunctionValue](#) ()
return last cost function value
- const [Array](#) & [lastGradient](#) ()
return last gradient
- Real [lastGradientNorm2](#) ()
return square norm of last gradient
- bool [succeed](#) ()
- virtual Real [operator\(\)](#) ([Problem](#) &P, EndCriteria::Type &ecType, const [EndCriteria](#) &, const Real t_ini)=0
Perform line search.
- Real [update](#) ([Array](#) ¶ms, const [Array](#) &direction, Real beta, const [Constraint](#) &constraint)
- const [Array](#) & [searchDirection](#) () const
current value of the search direction
- [Array](#) & [searchDirection](#) ()

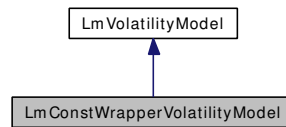
Protected Attributes

- [Array](#) `searchDirection_`
current values of the search direction
- [Array](#) `xtd_`
new x and its gradient
- [Array](#) `gradient_`
- Real `qt_`
cost function value and gradient norm corresponding to `xtd_`
- Real `qpt_`
- bool `succeed_`
flag to know if linesearch succeed

9.569 LmConstWrapperVolatilityModel Class Reference

```
#include <ql/legacy/libormarketmodels/lmconstwrappervolmodel.hpp>
```

Inheritance diagram for LmConstWrapperVolatilityModel:



9.569.1 Detailed Description

caplet const volatility model

Public Member Functions

- **LmConstWrapperVolatilityModel** (const boost::shared_ptr< [LmVolatilityModel](#) > &volaModel)
- [Disposable](#)< [Array](#) > **volatility** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- Volatility **volatility** (Size i, Time t, const [Array](#) &x=[Null](#)< [Array](#) >())
- Real **integratedVariance** (Size i, Size j, Time u, const [Array](#) &x=[Null](#)< [Array](#) >()) const

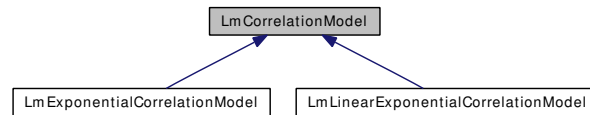
Protected Attributes

- const boost::shared_ptr< [LmVolatilityModel](#) > **volaModel_**

9.570 LmCorrelationModel Class Reference

```
#include <ql/legacy/libormarketmodels/lmcorrmodel.hpp>
```

Inheritance diagram for LmCorrelationModel:



9.570.1 Detailed Description

libor forward correlation model

Public Member Functions

- **LmCorrelationModel** (Size size, Size nArguments)
- virtual Size **size** () const
- virtual Size **factors** () const
- std::vector< [Parameter](#) > & **params** ()
- void **setParams** (const std::vector< [Parameter](#) > &arguments)
- virtual [Disposable](#)< [Matrix](#) > **correlation** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const=0
- virtual [Disposable](#)< [Matrix](#) > **pseudoSqrt** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual Real **correlation** (Size i, Size j, Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual bool **isTimeIndependent** () const

Protected Member Functions

- virtual void **generateArguments** ()=0

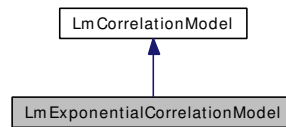
Protected Attributes

- const Size **size_**
- std::vector< [Parameter](#) > **arguments_**

9.571 LmExponentialCorrelationModel Class Reference

```
#include <ql/legacy/libormarketmodels/lmexpcorrmodel.hpp>
```

Inheritance diagram for LmExponentialCorrelationModel:



9.571.1 Detailed Description

exponential correlation model

This class describes a exponential correlation model

$$\rho_{i,j} = e^{(-\beta\|i-j\|)}$$

References:

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of [Libor](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf) Market Model and Joint Caps/Swaptions Calibration, (http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf)

Public Member Functions

- **LmExponentialCorrelationModel** (Size size, Real rho)
- **Disposable**< **Matrix** > **correlation** (Time t, const **Array** &x=**Null**< **Array** >()) const
- **Disposable**< **Matrix** > **pseudoSqrt** (Time t, const **Array** &x=**Null**< **Array** >()) const
- Real **correlation** (Size i, Size j, Time t, const **Array** &x) const
- bool **isTimeIndependent** () const

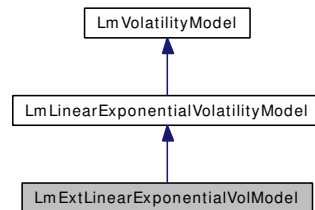
Protected Member Functions

- void **generateArguments** ()

9.572 LmExtLinearExponentialVolModel Class Reference

```
#include <ql/legacy/libormarketmodels/lmextlinexpvolmodel.hpp>
```

Inheritance diagram for LmExtLinearExponentialVolModel:



9.572.1 Detailed Description

extended linear exponential volatility model

This class describes an extended linear-exponential volatility model

$$\sigma_i(t) = k_i * ((a * (T_i - t) + d) * e^{-b(T_i - t)} + c)$$

References:

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of Libor Market Model and Joint Caps/Swaptions Calibration, (<http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf>)

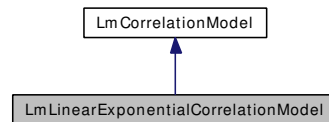
Public Member Functions

- **LmExtLinearExponentialVolModel** (const std::vector< Time > &fixingTimes, Real a, Real b, Real c, Real d)
- **Disposable**< Array > **volatility** (Time t, const Array &x=NULL< Array >()) const
- Volatility **volatility** (Size i, Time t, const Array &x=NULL< Array >()) const
- Real **integratedVariance** (Size i, Size j, Time u, const Array &x=NULL< Array >()) const

9.573 LmLinearExponentialCorrelationModel Class Reference

```
#include <ql/legacy/libormarketmodels/lmlinexpcorrmodel.hpp>
```

Inheritance diagram for LmLinearExponentialCorrelationModel:



9.573.1 Detailed Description

linear exponential correlation model

This class describes a exponential correlation model

$$\rho_{i,j} = rho + (1 - rho) * e^{(-\beta||i-j||)}$$

References:

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of [Libor](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf) Market Model and Joint Caps/Swaptions Calibration, (<http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf>)

Public Member Functions

- **LmLinearExponentialCorrelationModel** (Size size, Real rho, Real beta, Size factors=[Null](#)<Size >())
- [Disposable](#)< [Matrix](#) > **correlation** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Disposable](#)< [Matrix](#) > **pseudoSqrt** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- Real **correlation** (Size i, Size j, Time t, const [Array](#) &x) const
- Size **factors** () const
- bool **isTimeIndependent** () const

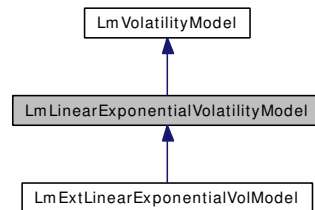
Protected Member Functions

- void **generateArguments** ()

9.574 LmLinearExponentialVolatilityModel Class Reference

```
#include <ql/legacy/libormarketmodels/lmlinepvolmodel.hpp>
```

Inheritance diagram for LmLinearExponentialVolatilityModel:



9.574.1 Detailed Description

linear exponential volatility model

This class describes a linear-exponential volatility model

$$\sigma_i(t) = (a * (T_i - t) + d) * e^{-b(T_i - t)} + c$$

References:

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of [Libor](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf) Market Model and Joint Caps/Swaptions Calibration, (<http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf>)

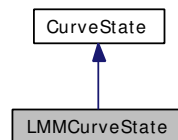
Public Member Functions

- **LmLinearExponentialVolatilityModel** (const std::vector< Time > &fixingTimes, Real a, Real b, Real c, Real d)
- **Disposable**< **Array** > **volatility** (Time t, const **Array** &x=**Null**< **Array** >()) const
- Volatility **volatility** (Size i, Time t, const **Array** &x=**Null**< **Array** >()) const
- Real **integratedVariance** (Size i, Size j, Time u, const **Array** &x=**Null**< **Array** >()) const

9.575 LMMCurveState Class Reference

```
#include <ql/models/marketmodels/curvestates/lmmcurvestate.hpp>
```

Inheritance diagram for LMMCurveState:



9.575.1 Detailed Description

Curve state for Libor market models

This class stores the state of the yield curve associated to the fixed calendar times within the simulation. This is the workhorse discounting object associated to the rate times of the simulation. It's important to pass the rates via an object like this to the product rather than directly to make it easier to switch to other engines such as a coterminal swap rate engine. Many products will not need expired rates and others will only require the first rate.

Public Member Functions

- **LMMCurveState** (const std::vector< Time > &rateTimes)
- std::auto_ptr< [CurveState](#) > **clone** () const

Modifiers

- void **setOnForwardRates** (const std::vector< Rate > &fwdRates, Size firstValidIndex=0)
- void **setOnDiscountRatios** (const std::vector< DiscountFactor > &discRatios, Size firstValidIndex=0)

Inspectors

- Real **discountRatio** (Size i, Size j) const
- Rate **forwardRate** (Size i) const
- Rate **coterminalSwapRate** (Size i) const
- Rate **coterminalSwapAnnuity** (Size numeraire, Size i) const
- Rate **cmSwapRate** (Size i, Size spanningForwards) const
- Rate **cmSwapAnnuity** (Size numeraire, Size i, Size spanningForwards) const
- const std::vector< Rate > & **forwardRates** () const
- const std::vector< Rate > & **coterminalSwapRates** () const
- const std::vector< Rate > & **cmSwapRates** (Size spanningForwards) const

9.576 LMMDriftCalculator Class Reference

```
#include <ql/models/marketmodels/driftcomputation/lmmdriftcalculator.hpp>
```

9.576.1 Detailed Description

Drift computation for log-normal Libor market models.

Returns the drift $\mu\Delta t$. See Mark Joshi, *Rapid Computation of Drifts in a Reduced Factor Libor Market Model*, Wilmott Magazine, May 2003.

Public Member Functions

- **LMMDriftCalculator** (const [Matrix](#) &pseudo, const std::vector< Spread > &displacements, const std::vector< Time > &taus, Size numeraire, Size alive)
- void [compute](#) (const [LMMCurveState](#) &cs, std::vector< Real > &drifts) const
Computes the drifts.
- void **compute** (const std::vector< Rate > &fwds, std::vector< Real > &drifts) const
- void [computePlain](#) (const [LMMCurveState](#) &cs, std::vector< Real > &drifts) const
- void **computePlain** (const std::vector< Rate > &fwds, std::vector< Real > &drifts) const
- void [computeReduced](#) (const [LMMCurveState](#) &cs, std::vector< Real > &drifts) const
- void **computeReduced** (const std::vector< Rate > &fwds, std::vector< Real > &drifts) const

9.576.2 Member Function Documentation

9.576.2.1 void computePlain (const LMMCurveState & cs, std::vector< Real > & drifts) const

Computes the drifts without factor reduction as in eqs. 2, 4 of ref. [1] (uses the covariance matrix directly).

9.576.2.2 void computeReduced (const LMMCurveState & cs, std::vector< Real > & drifts) const

Computes the drifts with factor reduction as in eq. 7 of ref. [1] (uses pseudo square root of the covariance matrix).

9.577 LMMNormalDriftCalculator Class Reference

```
#include <ql/models/marketmodels/driftcomputation/lmmnormaldriftcalculator.hpp>
```

9.577.1 Detailed Description

Drift computation for normal Libor market models.

Returns the drift $\mu\Delta t$. See Mark Joshi, *Rapid Computation of Drifts in a Reduced Factor Libor Market Model*, Wilmott Magazine, May 2003.

Public Member Functions

- **LMMNormalDriftCalculator** (const [Matrix](#) &pseudo, const std::vector< Time > &taus, Size numeraire, Size alive)
- void **compute** (const [LMMCurveState](#) &cs, std::vector< Real > &drifts) const
Computes the drifts.
- void **compute** (const std::vector< Rate > &fwds, std::vector< Real > &drifts) const
- void **computePlain** (const [LMMCurveState](#) &cs, std::vector< Real > &drifts) const
- void **computePlain** (const std::vector< Rate > &fwds, std::vector< Real > &drifts) const
- void **computeReduced** (const [LMMCurveState](#) &cs, std::vector< Real > &drifts) const
- void **computeReduced** (const std::vector< Rate > &fwds, std::vector< Real > &drifts) const

9.577.2 Member Function Documentation

9.577.2.1 void computePlain (const LMMCurveState & cs, std::vector< Real > & drifts) const

Computes the drifts without factor reduction as in eqs. 2, 4 of ref. [1], modified for normal forward rates dynamic (uses the covariance matrix directly).

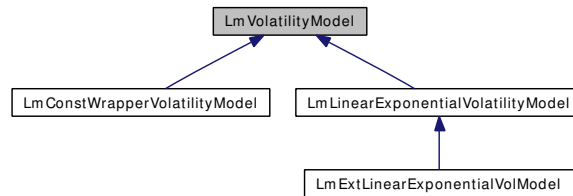
9.577.2.2 void computeReduced (const LMMCurveState & cs, std::vector< Real > & drifts) const

Computes the drifts with factor reduction as in eq. 7 of ref. [1], modified for normal forward rates dynamic (uses pseudo square root of the covariance matrix).

9.578 LmVolatilityModel Class Reference

```
#include <ql/legacy/libormarketmodels/lmvolmodel.hpp>
```

Inheritance diagram for LmVolatilityModel:



9.578.1 Detailed Description

caplet volatility model

Public Member Functions

- **LmVolatilityModel** (Size size, Size nArguments)
- Size **size** () const
- std::vector< [Parameter](#) > & **params** ()
- void **setParams** (const std::vector< [Parameter](#) > &arguments)
- virtual [Disposable](#)< [Array](#) > **volatility** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const=0
- virtual Volatility **volatility** (Size i, Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual Real **integratedVariance** (Size i, Size j, Time u, const [Array](#) &x=[Null](#)< [Array](#) >()) const

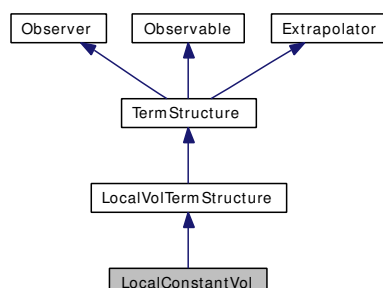
Protected Attributes

- const Size **size_**
- std::vector< [Parameter](#) > **arguments_**

9.579 LocalConstantVol Class Reference

```
#include <ql/termstructures/volatilities/localconstantvol.hpp>
```

Inheritance diagram for LocalConstantVol:



9.579.1 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** (Natural settlementDays, const [Calendar](#) &, Volatility volatility, const [DayCounter](#) &dayCounter)
- **LocalConstantVol** (Natural settlementDays, const [Calendar](#) &, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)

LocalVolTermStructure interface

- [DayCounter](#) dayCounter () const
the day counter used for date/time conversion
- [Date](#) maxDate () const
the latest date for which the curve can return values
- 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

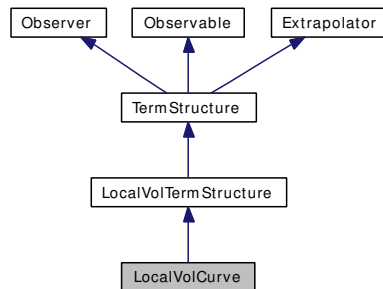
Visitability

- virtual void accept ([AcyclicVisitor](#) &)

9.580 LocalVolCurve Class Reference

```
#include <ql/termstructures/volatilities/localvolcurve.hpp>
```

Inheritance diagram for LocalVolCurve:



9.580.1 Detailed Description

Local volatility curve derived from a Black curve.

Public Member Functions

- **LocalVolCurve** (const [Handle](#)< [BlackVarianceCurve](#) > &curve)

LocalVolTermStructure interface

- const [Date](#) & [referenceDate](#) () const
the date at which discount = 1.0 and/or variance = 0.0
- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return values
- Real [minStrike](#) () const
the minimum strike for which the term structure can return vols
- Real [maxStrike](#) () const
the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

Protected Member Functions

- Volatility [localVolImpl](#) (Time, Real) const

9.580.2 Member Function Documentation

9.580.2.1 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

$$\sigma_L(t) = \sqrt{\frac{d}{dt} \sigma_B^2(t) t}$$

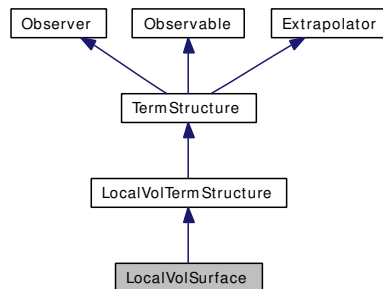
can be deduced which is here implemented.

Implements [LocalVolTermStructure](#).

9.581 LocalVolSurface Class Reference

```
#include <ql/termstructures/volatilities/localvolsurface.hpp>
```

Inheritance diagram for LocalVolSurface:



9.581.1 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 date at which discount = 1.0 and/or variance = 0.0
- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return values
- Real [minStrike](#) () const
the minimum strike for which the term structure can return vols

- Real [maxStrike](#) () const
the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

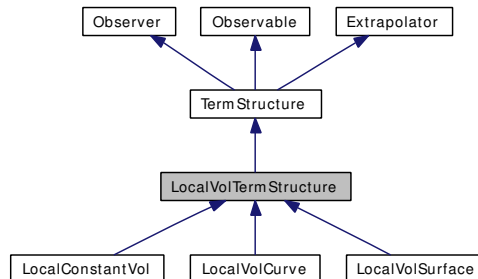
Protected Member Functions

- Volatility [localVolImpl](#) (Time, Real) const
local vol calculation

9.582 LocalVolTermStructure Class Reference

```
#include <ql/voltermstructure.hpp>
```

Inheritance diagram for LocalVolTermStructure:



9.582.1 Detailed Description

Local-volatility term structure.

This abstract class defines the interface of concrete local-volatility term structures which will be derived from this one.

Volatilities are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

See the *TermStructure* documentation for issues regarding constructors.

- **LocalVolTermStructure** (const [DayCounter](#) &dc=[Actual365Fixed](#)())
default constructor
- **LocalVolTermStructure** (const [Date](#) &referenceDate, const [Calendar](#) &cal=[Calendar](#)(), const [DayCounter](#) &dc=[Actual365Fixed](#)())
initialize with a fixed reference date
- **LocalVolTermStructure** (Natural settlementDays, const [Calendar](#) &, const [DayCounter](#) &dc=[Actual365Fixed](#)())
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 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

9.582.2 Constructor & Destructor Documentation

9.582.2.1 LocalVolTermStructure (const DayCounter & dc = Actual365Fixed())

default constructor

Warning

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

9.583 LogLinear Class Reference

```
#include <ql/math/interpolations/loglinearinterpolation.hpp>
```

9.583.1 Detailed Description

log-linear interpolation factory and traits

Public Types

- enum { **global** = 0 }

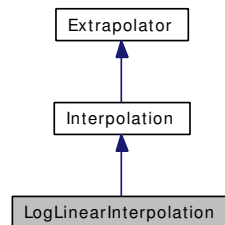
Public Member Functions

- template<class I1, class I2>
[Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

9.584 LogLinearInterpolation Class Reference

```
#include <ql/math/interpolations/loglinearinterpolation.hpp>
```

Inheritance diagram for LogLinearInterpolation:



9.584.1 Detailed Description

log-linear interpolation between discrete points

Todo

implement primitive, derivative, and secondDerivative functions.

Public Member Functions

- `template<class I1, class I2>`
`LogLinearInterpolation` (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

9.584.2 Constructor & Destructor Documentation

9.584.2.1 LogLinearInterpolation (const I1 & xBegin, const I1 & xEnd, const I2 & yBegin)

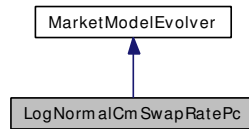
Precondition:

the x values must be sorted.

9.585 LogNormalCmSwapRatePc Class Reference

```
#include <ql/models/marketmodels/evolvers/lognormalcmswapratepc.hpp>
```

Inheritance diagram for LogNormalCmSwapRatePc:



9.585.1 Detailed Description

Predictor-Corrector.

Public Member Functions

- **LogNormalCmSwapRatePc** (const Size spanningForwards, const boost::shared_ptr< [MarketModel](#) > &, const BrownianGeneratorFactory &, const std::vector< Size > &numeraires, Size initialStep=0)

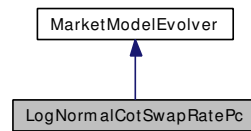
MarketModel interface

- const std::vector< Size > & **numeraires** () const
- Real **startNewPath** ()
- Real **advanceStep** ()
- Size **currentStep** () const
- const [CurveState](#) & **currentState** () const
- void **setInitialState** (const [CurveState](#) &)

9.586 LogNormalCotSwapRatePc Class Reference

```
#include <ql/models/marketmodels/evolvers/lognormalcotswapratepc.hpp>
```

Inheritance diagram for LogNormalCotSwapRatePc:



9.586.1 Detailed Description

Predictor-Corrector.

Public Member Functions

- **LogNormalCotSwapRatePc** (const boost::shared_ptr< [MarketModel](#) > &, const Brownian-GeneratorFactory &, const std::vector< Size > &numeraires, Size initialStep=0)

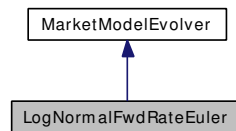
MarketModel interface

- const std::vector< Size > & **numeraires** () const
- Real **startNewPath** ()
- Real **advanceStep** ()
- Size **currentStep** () const
- const [CurveState](#) & **currentState** () const
- void **setInitialState** (const [CurveState](#) &)

9.587 LogNormalFwdRateEuler Class Reference

```
#include <ql/models/marketmodels/evolvers/lognormalfwdrateeuler.hpp>
```

Inheritance diagram for LogNormalFwdRateEuler:



9.587.1 Detailed Description

Euler.

Public Member Functions

- **LogNormalFwdRateEuler** (const boost::shared_ptr< [MarketModel](#) > &, const Brownian-GeneratorFactory &, const std::vector< Size > &numeraires, Size initialStep=0)

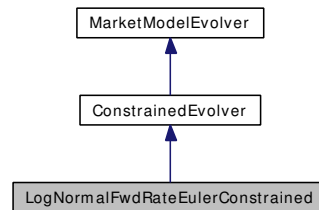
MarketModel interface

- const std::vector< Size > & **numeraires** () const
- Real **startNewPath** ()
- Real **advanceStep** ()
- Size **currentStep** () const
- const [CurveState](#) & **currentState** () const
- void **setInitialState** (const [CurveState](#) &)

9.588 LogNormalFwdRateEulerConstrained Class Reference

```
#include <ql/models/marketmodels/evolvers/lognormalfwdrateeulerconstrained.hpp>
```

Inheritance diagram for LogNormalFwdRateEulerConstrained:



9.588.1 Detailed Description

euler stepping

Public Member Functions

- **LogNormalFwdRateEulerConstrained** (const boost::shared_ptr< [MarketModel](#) > &, const BrownianGeneratorFactory &, const std::vector< Size > &numeraires, Size initialStep=0)

MarketModelConstrained interface

- virtual void [setConstraintType](#) (const std::vector< Size > &startIndexOfSwapRate, const std::vector< Size > &endIndexOfSwapRate)
call once
- virtual void [setThisConstraint](#) (const std::vector< Rate > &rateConstraints, const std::vector< bool > &isConstraintActive)
call before each path

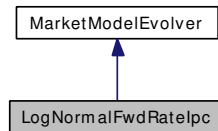
MarketModel interface

- const std::vector< Size > & **numeraires** () const
- Real **startNewPath** ()
- Real **advanceStep** ()
- Size **currentStep** () const
- const [CurveState](#) & **currentState** () const
- void **setInitialState** (const [CurveState](#) &)

9.589 LogNormalFwdRateIpc Class Reference

```
#include <ql/models/marketmodels/evolvers/lognormalfwdrateipc.hpp>
```

Inheritance diagram for LogNormalFwdRateIpc:



9.589.1 Detailed Description

Iterative Predictor-Corrector.

Public Member Functions

- **LogNormalFwdRateIpc** (const boost::shared_ptr< [MarketModel](#) > &, const Brownian-GeneratorFactory &, const std::vector< Size > &numeraires, Size initialStep=0)

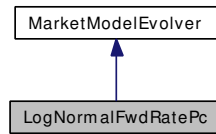
MarketModel interface

- const std::vector< Size > & **numeraires** () const
- Real **startNewPath** ()
- Real **advanceStep** ()
- Size **currentStep** () const
- const [CurveState](#) & **currentState** () const
- void **setInitialState** (const [CurveState](#) &)

9.590 LogNormalFwdRatePc Class Reference

```
#include <ql/models/marketmodels/evolvers/lognormalfwdratepc.hpp>
```

Inheritance diagram for LogNormalFwdRatePc:



9.590.1 Detailed Description

Predictor-Corrector.

Public Member Functions

- `LogNormalFwdRatePc` (const boost::shared_ptr< [MarketModel](#) > &, const BrownianGeneratorFactory &, const std::vector< Size > &numeraires, Size initialStep=0)

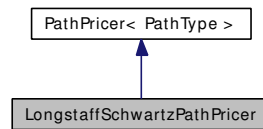
MarketModel interface

- const std::vector< Size > & `numeraires` () const
- Real `startNewPath` ()
- Real `advanceStep` ()
- Size `currentStep` () const
- const [CurveState](#) & `currentState` () const
- void `setInitialState` (const [CurveState](#) &)

9.591 LongstaffSchwartzPathPricer Class Template Reference

```
#include <ql/methods/montecarlo/longstaffschwartzpathpricer.hpp>
```

Inheritance diagram for LongstaffSchwartzPathPricer:



9.591.1 Detailed Description

```
template<class PathType> class QuantLib::LongstaffSchwartzPathPricer< PathType >
```

Longstaff-Schwarz path pricer for early exercise options.

References:

Francis Longstaff, Eduardo Schwartz, 2001. Valuing American Options by Simulation: A Simple Least-Squares Approach, The Review of Financial Studies, Volume 14, No. 1, 113-147

Tests

the correctness of the returned value is tested by reproducing results available in web/literature

Public Types

- typedef EarlyExerciseTraits< PathType >::StateType **StateType**

Public Member Functions

- **LongstaffSchwartzPathPricer** (const [TimeGrid](#) ×, const boost::shared_ptr< [EarlyExercisePathPricer](#)< PathType > > &, const boost::shared_ptr< [YieldTermStructure](#) > &termStructure)
- Real **operator()** (const PathType &path) const
- virtual void **calibrate** ()

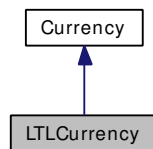
Protected Attributes

- bool **calibrationPhase_**
- const boost::shared_ptr< [EarlyExercisePathPricer](#)< PathType > > **pathPricer_**
- boost::scoped_array< [Array](#) > **coeff_**
- boost::scoped_array< DiscountFactor > **dF_**
- std::vector< PathType > **paths_**
- const std::vector< boost::function1< Real, StateType > > **v_**

9.592 LTLCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for LTLCurrency:



9.592.1 Detailed Description

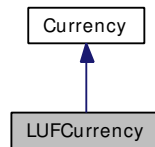
Lithuanian litas.

The ISO three-letter code is LTL; the numeric code is 440. It is divided in 100 centu.

9.593 LUFCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for LUFCurrency:



9.593.1 Detailed Description

Luxembourg franc.

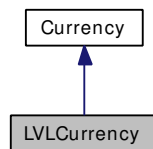
The ISO three-letter code was LUF; the numeric code was 442. It was divided in 100 centimes.

Obsoleted by the Euro since 1999.

9.594 LVLCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for LVLCurrency:



9.594.1 Detailed Description

Latvian lat.

The ISO three-letter code is `LVL`; the numeric code is 428. It is divided in 100 santims.

9.595 MakeCapFloor Class Reference

```
#include <ql/instruments/makecapfloor.hpp>
```

9.595.1 Detailed Description

helper class

This class provides a more comfortable way to instantiate standard market cap and floor.

Public Member Functions

- **MakeCapFloor** (CapFloor::Type capFloorType, const [Period](#) &capFloorTenor, const boost::shared_ptr< [IborIndex](#) > &index, Rate strike=[Null](#)< Rate >(), const [Period](#) &forwardStart=0*Days, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- **operator CapFloor** () const
- **operator boost::shared_ptr** () const
- **MakeCapFloor** & **withNominal** (Real n)
- **MakeCapFloor** & **withEffectiveDate** (const [Date](#) &effectiveDate, bool firstCapletExcluded)
- **MakeCapFloor** & **withDiscountingTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &discountingTermStructure)
- **MakeCapFloor** & **withTenor** (const [Period](#) &t)
- **MakeCapFloor** & **withCalendar** (const [Calendar](#) &cal)
- **MakeCapFloor** & **withConvention** ([BusinessDayConvention](#) bdc)
- **MakeCapFloor** & **withTerminationDateConvention** ([BusinessDayConvention](#) bdc)
- **MakeCapFloor** & **withForward** (bool flag=true)
- **MakeCapFloor** & **withEndOfMonth** (bool flag=true)
- **MakeCapFloor** & **withFirstDate** (const [Date](#) &d)
- **MakeCapFloor** & **withNextToLastDate** (const [Date](#) &d)
- **MakeCapFloor** & **withDayCount** (const [DayCounter](#) &dc)

9.596 MakeCms Class Reference

```
#include <ql/instruments/makecms.hpp>
```

9.596.1 Detailed Description

helper class

This class provides a more comfortable way to instantiate standard market constant maturity swap.

Public Member Functions

- **MakeCms** (const [Period](#) &swapTenor, const boost::shared_ptr< [SwapIndex](#) > &swapIndex, Spread iborSpread, const [Period](#) &forwardStart=0 *Days)
- **operator Swap** () const
- **operator boost::shared_ptr** () const
- **MakeCms** & **receiveCms** (bool flag=true)
- **MakeCms** & **withNominal** (Real n)
- **MakeCms** & **withEffectiveDate** (const [Date](#) &)
- **MakeCms** & **withDiscountingTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &discountingTermStructure)
- **MakeCms** & **withCmsLegTenor** (const [Period](#) &t)
- **MakeCms** & **withCmsLegCalendar** (const [Calendar](#) &cal)
- **MakeCms** & **withCmsLegConvention** ([BusinessDayConvention](#) bdc)
- **MakeCms** & **withCmsLegTerminationDateConvention** ([BusinessDayConvention](#) bdc)
- **MakeCms** & **withCmsLegForward** (bool flag=true)
- **MakeCms** & **withCmsLegEndOfMonth** (bool flag=true)
- **MakeCms** & **withCmsLegFirstDate** (const [Date](#) &d)
- **MakeCms** & **withCmsLegNextToLastDate** (const [Date](#) &d)
- **MakeCms** & **withCmsLegDayCount** (const [DayCounter](#) &dc)
- **MakeCms** & **withFloatingLegTenor** (const [Period](#) &t)
- **MakeCms** & **withFloatingLegCalendar** (const [Calendar](#) &cal)
- **MakeCms** & **withFloatingLegConvention** ([BusinessDayConvention](#) bdc)
- **MakeCms** & **withFloatingLegTerminationDateConvention** ([BusinessDayConvention](#) bdc)
- **MakeCms** & **withFloatingLegForward** (bool flag=true)
- **MakeCms** & **withFloatingLegEndOfMonth** (bool flag=true)
- **MakeCms** & **withFloatingLegFirstDate** (const [Date](#) &d)
- **MakeCms** & **withFloatingLegNextToLastDate** (const [Date](#) &d)
- **MakeCms** & **withFloatingLegDayCount** (const [DayCounter](#) &dc)

9.597 MakeMCAmericanEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/mcamericanengine.hpp>
```

9.597.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class  
QuantLib::MakeMCAmericanEngine< RNG, S >
```

Monte Carlo American engine factory.

Examples:

[EquityOption.cpp](#).

Public Member Functions

- [MakeMCAmericanEngine](#) & **withSteps** (Size steps)
- [MakeMCAmericanEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCAmericanEngine](#) & **withSamples** (Size samples)
- [MakeMCAmericanEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCAmericanEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCAmericanEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCAmericanEngine](#) & **withAntitheticVariate** (bool b=true)
- [MakeMCAmericanEngine](#) & **withControlVariate** (bool b=true)
- [MakeMCAmericanEngine](#) & **withPolynomOrder** (Size polynomOrder)
- [MakeMCAmericanEngine](#) & **withBasisSystem** (LsmBasisSystem::PolynomType)
- [MakeMCAmericanEngine](#) & **withCalibrationSamples** (Size calibrationSamples)
- **operator boost::shared_ptr () const**

9.598 MakeMCDigitalEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/mcdigitalengine.hpp>
```

9.598.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class  
QuantLib::MakeMCDigitalEngine< RNG, S >
```

Monte Carlo digital engine factory.

Public Member Functions

- [MakeMCDigitalEngine](#) & **withSteps** (Size steps)
- [MakeMCDigitalEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCDigitalEngine](#) & **withBrownianBridge** (bool b=true)
- [MakeMCDigitalEngine](#) & **withSamples** (Size samples)
- [MakeMCDigitalEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCDigitalEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCDigitalEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCDigitalEngine](#) & **withAntitheticVariate** (bool b=true)
- [MakeMCDigitalEngine](#) & **withControlVariate** (bool b=true)
- **operator boost::shared_ptr ()** const

9.599 MakeMCEuropeanEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/mceuropeanengine.hpp>
```

9.599.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MakeMCEuropeanEngine< RNG, S >
```

Monte Carlo European engine factory.

Examples:

[EquityOption.cpp](#).

Public Member Functions

- [MakeMCEuropeanEngine](#) & **withSteps** (Size steps)
- [MakeMCEuropeanEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCEuropeanEngine](#) & **withBrownianBridge** (bool b=true)
- [MakeMCEuropeanEngine](#) & **withSamples** (Size samples)
- [MakeMCEuropeanEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCEuropeanEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCEuropeanEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCEuropeanEngine](#) & **withAntitheticVariate** (bool b=true)
- [MakeMCEuropeanEngine](#) & **withControlVariate** (bool b=true)
- **operator boost::shared_ptr ()** const

9.600 MakeMCEuropeanHestonEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/mceuropeanhestonengine.hpp>
```

9.600.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class  
QuantLib::MakeMCEuropeanHestonEngine< RNG, S >
```

Monte Carlo Heston European engine factory.

Public Member Functions

- [MakeMCEuropeanHestonEngine](#) & **withSteps** (Size steps)
- [MakeMCEuropeanHestonEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCEuropeanHestonEngine](#) & **withSamples** (Size samples)
- [MakeMCEuropeanHestonEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCEuropeanHestonEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCEuropeanHestonEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCEuropeanHestonEngine](#) & **withAntitheticVariate** (bool b=true)
- **operator boost::shared_ptr ()** const

9.601 MakeMCHullWhiteCapFloorEngine Class Template Reference

```
#include <ql/pricingengines/capfloor/mchullwhiteengine.hpp>
```

9.601.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MakeMCHullWhiteCapFloorEngine< RNG, S >
```

Monte Carlo Hull-White cap-floor engine factory.

Public Member Functions

- [MakeMCHullWhiteCapFloorEngine](#) (const boost::shared_ptr< [HullWhite](#) > &)
- [MakeMCHullWhiteCapFloorEngine](#) & [withBrownianBridge](#) (bool b=true)
- [MakeMCHullWhiteCapFloorEngine](#) & [withSamples](#) (Size samples)
- [MakeMCHullWhiteCapFloorEngine](#) & [withTolerance](#) (Real tolerance)
- [MakeMCHullWhiteCapFloorEngine](#) & [withMaxSamples](#) (Size samples)
- [MakeMCHullWhiteCapFloorEngine](#) & [withSeed](#) (BigNatural seed)
- [MakeMCHullWhiteCapFloorEngine](#) & [withAntitheticVariate](#) (bool b=true)
- [operator boost::shared_ptr \(\)](#) const

9.602 MakeMCVarianceSwapEngine Class Template Reference

```
#include <ql/pricingengines/forward/mcvarianceswapengine.hpp>
```

9.602.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class  
QuantLib::MakeMCVarianceSwapEngine< RNG, S >
```

Monte Carlo variance-swap engine factory.

Public Member Functions

- [MakeMCVarianceSwapEngine](#) & **withSteps** (Size steps)
- [MakeMCVarianceSwapEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCVarianceSwapEngine](#) & **withBrownianBridge** (bool b=true)
- [MakeMCVarianceSwapEngine](#) & **withSamples** (Size samples)
- [MakeMCVarianceSwapEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCVarianceSwapEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCVarianceSwapEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCVarianceSwapEngine](#) & **withAntitheticVariate** (bool b=true)
- **operator boost::shared_ptr () const**

9.603 MakeSchedule Class Reference

```
#include <ql/time/schedule.hpp>
```

9.603.1 Detailed Description

helper class

This class provides a more comfortable interface to the argument list of Schedule's constructor.

Public Member Functions

- **MakeSchedule** (const [Date](#) &effectiveDate, const [Date](#) &terminationDate, const [Period](#) &tenor, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention)
- **MakeSchedule** & **terminationDateConvention** ([BusinessDayConvention](#) conv)
- **MakeSchedule** & **backwards** (bool flag=true)
- **MakeSchedule** & **forwards** (bool flag=true)
- **MakeSchedule** & **endOfMonth** (bool flag=true)
- **MakeSchedule** & **withFirstDate** (const [Date](#) &d)
- **MakeSchedule** & **withNextToLastDate** (const [Date](#) &d)
- **operator Schedule** () const

9.604 MakeVanillaSwap Class Reference

```
#include <ql/instruments/makevanillaswap.hpp>
```

9.604.1 Detailed Description

helper class

This class provides a more comfortable way to instantiate standard market swap.

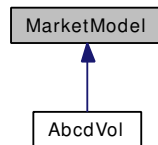
Public Member Functions

- **MakeVanillaSwap** (const [Period](#) &swapTenor, const boost::shared_ptr< [IborIndex](#) > &index, Rate fixedRate=[Null](#)< Rate >(), const [Period](#) &forwardStart=0 *Days)
- **operator VanillaSwap** () const
- **operator boost::shared_ptr** () const
- [MakeVanillaSwap](#) & **receiveFixed** (bool flag=true)
- [MakeVanillaSwap](#) & **withType** ([VanillaSwap::Type](#) type)
- [MakeVanillaSwap](#) & **withNominal** (Real n)
- [MakeVanillaSwap](#) & **withEffectiveDate** (const [Date](#) &)
- [MakeVanillaSwap](#) & **withDiscountingTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &discountingTermStructure)
- [MakeVanillaSwap](#) & **withFixedLegTenor** (const [Period](#) &t)
- [MakeVanillaSwap](#) & **withFixedLegCalendar** (const [Calendar](#) &cal)
- [MakeVanillaSwap](#) & **withFixedLegConvention** ([BusinessDayConvention](#) bdc)
- [MakeVanillaSwap](#) & **withFixedLegTerminationDateConvention** ([BusinessDayConvention](#) bdc)
- [MakeVanillaSwap](#) & **withFixedLegForward** (bool flag=true)
- [MakeVanillaSwap](#) & **withFixedLegEndOfMonth** (bool flag=true)
- [MakeVanillaSwap](#) & **withFixedLegFirstDate** (const [Date](#) &d)
- [MakeVanillaSwap](#) & **withFixedLegNextToLastDate** (const [Date](#) &d)
- [MakeVanillaSwap](#) & **withFixedLegDayCount** (const [DayCounter](#) &dc)
- [MakeVanillaSwap](#) & **withFloatingLegTenor** (const [Period](#) &t)
- [MakeVanillaSwap](#) & **withFloatingLegCalendar** (const [Calendar](#) &cal)
- [MakeVanillaSwap](#) & **withFloatingLegConvention** ([BusinessDayConvention](#) bdc)
- [MakeVanillaSwap](#) & **withFloatingLegTerminationDateConvention** ([BusinessDayConvention](#) bdc)
- [MakeVanillaSwap](#) & **withFloatingLegForward** (bool flag=true)
- [MakeVanillaSwap](#) & **withFloatingLegEndOfMonth** (bool flag=true)
- [MakeVanillaSwap](#) & **withFloatingLegFirstDate** (const [Date](#) &d)
- [MakeVanillaSwap](#) & **withFloatingLegNextToLastDate** (const [Date](#) &d)
- [MakeVanillaSwap](#) & **withFloatingLegDayCount** (const [DayCounter](#) &dc)
- [MakeVanillaSwap](#) & **withFloatingLegSpread** (Spread sp)

9.605 MarketModel Class Reference

```
#include <ql/models/marketmodels/marketmodel.hpp>
```

Inheritance diagram for MarketModel:



9.605.1 Detailed Description

base class for market models

For each time step, generates the pseudo-square root of the covariance matrix for that time step.

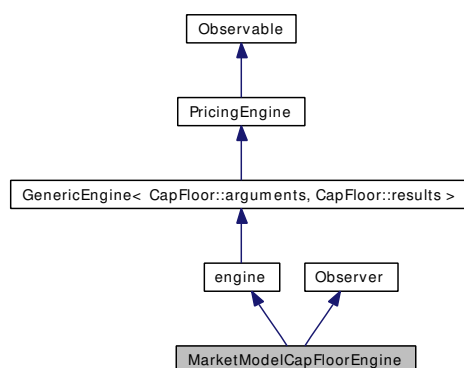
Public Member Functions

- virtual const std::vector< Rate > & **initialRates** () const=0
- virtual const std::vector< Spread > & **displacements** () const=0
- virtual const [EvolutionDescription](#) & **evolution** () const=0
- virtual Size **numberOfRates** () const=0
- virtual Size **numberOfFactors** () const=0
- virtual Size **numberOfSteps** () const=0
- virtual const [Matrix](#) & **pseudoRoot** (Size i) const=0
- virtual const [Matrix](#) & **covariance** (Size i) const
- virtual const [Matrix](#) & **totalCovariance** (Size endIndex) const

9.606 MarketModelCapFloorEngine Class Reference

#include <ql/pricingengines/capfloor/marketmodelcapfloorengine.hpp>

Inheritance diagram for MarketModelCapFloorEngine:



9.606.1 Detailed Description

Market-model cap/floor engine.

Bug

This engine is not yet working correctly (results are off the expected ones.)

Public Member Functions

- **MarketModelCapFloorEngine** (const boost::shared_ptr< [MarketModelFactory](#) > &)
- void **calculate** () const
- void **update** ()

9.606.2 Member Function Documentation

9.606.2.1 void update () [virtual]

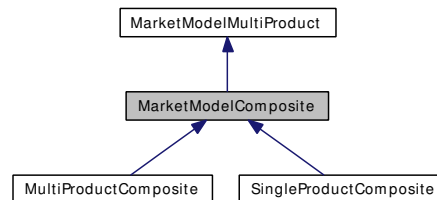
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.607 MarketModelComposite Class Reference

```
#include <ql/models/marketmodels/products/compositeproduct.hpp>
```

Inheritance diagram for MarketModelComposite:



9.607.1 Detailed Description

Composition of two or more market-model products.

Instances of this class build a market-model product by composing one or more subproducts.

Precondition:

All subproducts must have the same rate times.

Public Member Functions

MarketModelMultiProduct interface

- const [EvolutionDescription](#) & **evolution** () const
- std::vector< Size > **suggestedNumeraires** () const
- std::vector< Time > **possibleCashFlowTimes** () const
- void **reset** ()

during simulation put product at start of path

Composite facilities

- void **add** (const [Clone](#)< [MarketModelMultiProduct](#) > &, Real multiplier=1.0)
- void **subtract** (const [Clone](#)< [MarketModelMultiProduct](#) > &, Real multiplier=1.0)
- void **finalize** ()
- Size **size** () const
- const [MarketModelMultiProduct](#) & **item** (Size i) const
- [MarketModelMultiProduct](#) & **item** (Size i)
- Real **multiplier** (Size i) const

Protected Types

- typedef std::vector< SubProduct >::iterator **iterator**
- typedef std::vector< SubProduct >::const_iterator **const_iterator**

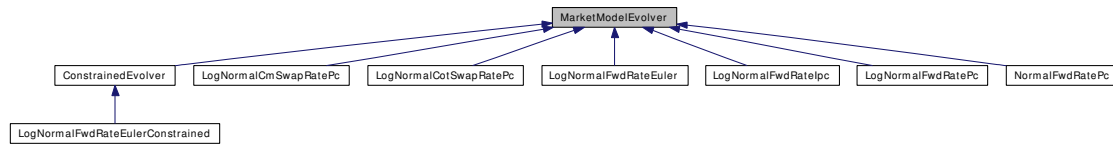
Protected Attributes

- `std::vector< SubProduct > components_`
- `std::vector< Time > rateTimes_`
- `std::vector< Time > evolutionTimes_`
- [EvolutionDescription](#) `evolution_`
- `bool finalized_`
- `Size currentIndex_`
- `std::vector< Time > cashflowTimes_`
- `std::vector< std::vector< Time > > allEvolutionTimes_`
- `std::vector< std::vector< bool > > isInSubset_`

9.608 MarketModelEvolver Class Reference

```
#include <ql/models/marketmodels/evolver.hpp>
```

Inheritance diagram for MarketModelEvolver:



9.608.1 Detailed Description

Market-model evolver.

Abstract base class. The evolver does the actual gritty work of evolving the forward rates from one time to the next.

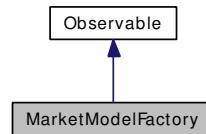
Public Member Functions

- virtual const std::vector< Size > & **numeraires** () const=0
- virtual Real **startNewPath** ()=0
- virtual Real **advanceStep** ()=0
- virtual Size **currentStep** () const=0
- virtual const [CurveState](#) & **currentState** () const=0
- virtual void **setInitialState** (const [CurveState](#) &)=0

9.609 MarketModelFactory Class Reference

```
#include <ql/models/marketmodels/marketmodel.hpp>
```

Inheritance diagram for MarketModelFactory:



9.609.1 Detailed Description

base class for market-model factories

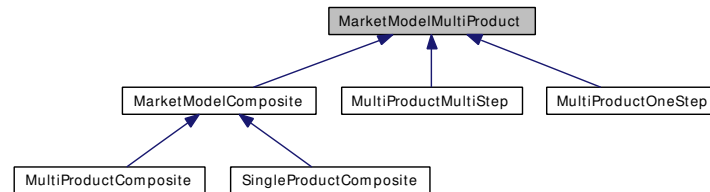
Public Member Functions

- virtual `boost::shared_ptr< MarketModel > create` (const [EvolutionDescription](#) &, Size numberOfFactors) const =0

9.610 MarketModelMultiProduct Class Reference

```
#include <ql/models/marketmodels/multiproduct.hpp>
```

Inheritance diagram for MarketModelMultiProduct:



9.610.1 Detailed Description

market-model product

This is the abstract base class that encapsulates the notion of a product: it contains the information that would be in the termsheet of the product.

It's useful to have it be able to do several products simultaneously. The products would have to have the same underlying rate times of course. The class is therefore really encapsulating the notion of a multi-product.

For each time evolved to, it generates the cash flows associated to that time for the state of the yield curve. If one was doing a callable product then this would encompass the product and its exercise strategy.

Public Member Functions

- virtual std::vector< Size > **suggestedNumeraires** () const=0
- virtual const [EvolutionDescription](#) & **evolution** () const=0
- virtual std::vector< Time > **possibleCashFlowTimes** () const=0
- virtual Size **numberOfProducts** () const=0
- virtual Size **maxNumberOfCashFlowsPerProductPerStep** () const=0
- virtual void **reset** ()=0
during simulation put product at start of path
- virtual bool **nextTimeStep** (const [CurveState](#) ¤tState, std::vector< Size > &number-CashFlowsThisStep, std::vector< std::vector< CashFlow > > &cashFlowsGenerated)=0
return value indicates whether path is finished, TRUE means done
- virtual std::auto_ptr< [MarketModelMultiProduct](#) > **clone** () const=0
returns a newly-allocated copy of itself

9.611 Matrix Class Reference

```
#include <ql/math/matrix.hpp>
```

9.611.1 Detailed Description

Matrix used in linear algebra.

This class implements the concept of [Matrix](#) as used in linear algebra. As such, it is **not** meant to be used as a container.

Public Types

- typedef Real * **iterator**
- typedef const Real * **const_iterator**
- typedef boost::reverse_iterator< iterator > **reverse_iterator**
- typedef boost::reverse_iterator< const_iterator > **const_reverse_iterator**
- typedef Real * **row_iterator**
- typedef const Real * **const_row_iterator**
- typedef boost::reverse_iterator< row_iterator > **reverse_row_iterator**
- typedef boost::reverse_iterator< const_row_iterator > **const_reverse_row_iterator**
- typedef [step_iterator](#)< iterator > **column_iterator**
- typedef [step_iterator](#)< const_iterator > **const_column_iterator**
- typedef boost::reverse_iterator< [column_iterator](#) > **reverse_column_iterator**
- typedef boost::reverse_iterator< [const_column_iterator](#) > **const_reverse_column_iterator**

Public Member Functions

Constructors, destructor, and assignment

- [Matrix](#) ()
creates a null matrix
- [Matrix](#) (Size rows, Size columns)
creates a matrix with the given dimensions
- [Matrix](#) (Size rows, Size columns, Real value)
creates the matrix and fills it with value
- [Matrix](#) (const [Matrix](#) &)
- [Matrix](#) (const [Disposable](#)< [Matrix](#) > &)
- [Matrix](#) & **operator=** (const [Matrix](#) &)
- [Matrix](#) & **operator=** (const [Disposable](#)< [Matrix](#) > &)

Algebraic operators

- const [Matrix](#) & **operator+=** (const [Matrix](#) &)
- const [Matrix](#) & **operator-=** (const [Matrix](#) &)
- const [Matrix](#) & **operator *=** (Real)
- const [Matrix](#) & **operator/=** (Real)

Iterator access

- `const_iterator` **begin** () const
- `iterator` **begin** ()
- `const_iterator` **end** () const
- `iterator` **end** ()
- `const_reverse_iterator` **rbegin** () const
- `reverse_iterator` **rbegin** ()
- `const_reverse_iterator` **rend** () const
- `reverse_iterator` **rend** ()
- `const_row_iterator` **row_begin** (Size i) const
- `row_iterator` **row_begin** (Size i)
- `const_row_iterator` **row_end** (Size i) const
- `row_iterator` **row_end** (Size i)
- `const_reverse_row_iterator` **row_rbegin** (Size i) const
- `reverse_row_iterator` **row_rbegin** (Size i)
- `const_reverse_row_iterator` **row_rend** (Size i) const
- `reverse_row_iterator` **row_rend** (Size i)
- `const_column_iterator` **column_begin** (Size i) const
- `column_iterator` **column_begin** (Size i)
- `const_column_iterator` **column_end** (Size i) const
- `column_iterator` **column_end** (Size i)
- `const_reverse_column_iterator` **column_rbegin** (Size i) const
- `reverse_column_iterator` **column_rbegin** (Size i)
- `const_reverse_column_iterator` **column_rend** (Size i) const
- `reverse_column_iterator` **column_rend** (Size i)

Element access

- `const_row_iterator` **operator[]** (Size) const
- `const_row_iterator` **at** (Size) const
- `row_iterator` **operator[]** (Size)
- `row_iterator` **at** (Size)
- `Disposable< Array >` **diagonal** (void) const

Inspectors

- Size **rows** () const
- Size **columns** () const
- bool **empty** () const

Utilities

- void **swap** (`Matrix` &)

Related Functions

(Note that these are not member functions.)

- `const Disposable< Matrix >` **operator+** (const `Matrix` &, const `Matrix` &)
- `const Disposable< Matrix >` **operator-** (const `Matrix` &, const `Matrix` &)
- `const Disposable< Matrix >` **operator *** (const `Matrix` &, Real)
- `const Disposable< Matrix >` **operator *** (Real, const `Matrix` &)
- `const Disposable< Matrix >` **operator/** (const `Matrix` &, Real)
- `const Disposable< Array >` **operator *** (const `Array` &, const `Matrix` &)

- `const Disposable< Array > operator * (const Matrix &, const Array &)`
- `const Disposable< Matrix > operator * (const Matrix &, const Matrix &)`
- `const Disposable< Matrix > transpose (const Matrix &)`
- `const Disposable< Matrix > outerProduct (const Array &v1, const Array &v2)`
- `template<class Iterator1, class Iterator2>
const Disposable< Matrix > outerProduct (Iterator1 v1begin, Iterator1 v1end, Iterator2
v2begin, Iterator2 v2end)`
- `void swap (Matrix &, Matrix &)`
- `std::ostream & operator<< (std::ostream &, const Matrix &)`
- `Disposable< Matrix > inverse (const Matrix &m)`
- `const Disposable< Matrix > CholeskyDecomposition (const Matrix &m, bool flexible=false)`
- `const Disposable< Matrix > pseudoSqrt (const Matrix &, SalvagingAlgo-
rithm::Type=SalvagingAlgorithm::None)`

Returns the pseudo square root of a real symmetric matrix.

- `const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, Real compo-
nentRetainedPercentage, SalvagingAlgorithm::Type)`

Returns the rank-reduced pseudo square root of a real symmetric matrix.

9.611.2 Member Function Documentation

9.611.2.1 `const Matrix & operator+= (const Matrix &)`

Precondition:

all matrices involved in an algebraic expression must have the same size.

9.611.3 Friends And Related Function Documentation

9.611.3.1 `const Disposable< Matrix > operator+ (const Matrix &, const Matrix &)` [related]

9.611.3.2 `const Disposable< Matrix > operator- (const Matrix &, const Matrix &)` [related]

9.611.3.3 `const Disposable< Matrix > operator * (const Matrix &, Real)` [related]

9.611.3.4 `const Disposable< Matrix > operator * (Real, const Matrix &)` [related]

9.611.3.5 `const Disposable< Matrix > operator/ (const Matrix &, Real)` [related]

9.611.3.6 `const Disposable< Array > operator * (const Array &, const Matrix &)` [related]

9.611.3.7 `const Disposable< Array > operator * (const Matrix &, const Array &)` [related]

9.611.3.8 `const Disposable< Matrix > operator * (const Matrix &, const Matrix &)` [related]

9.611.3.9 `const Disposable< Matrix > transpose (const Matrix &)` [related]

9.611.3.10 `const Disposable< Matrix > outerProduct (const Array & v1, const Array & v2)` [related]

9.611.3.11 `const Disposable< Matrix > outerProduct (Iterator1 v1begin, Iterator1 v1end,
Iterator2 v2begin, Iterator2 v2end)` [related]

9.611.3.12 `void swap (Matrix &, Matrix &)` [related]

9.611.3.13 `std::ostream & operator<< (std::ostream &, const Matrix &)` [related]

9.611.3.14 `Disposable< Matrix > inverse (const Matrix & m)` [related]

9.611.3.15 `const Disposable< Matrix > CholeskyDecomposition (const Matrix & m, bool
flexible = false)` [related]

9.611.3.16 `const Disposable< Matrix > pseudoSqrt (const Matrix &,
SalvagingAlgorithm::Type = SalvagingAlgorithm::None)` [related]

Returns the pseudo square root of a real symmetric matrix.

Given a matrix M , the result S is defined as the matrix such that $SS^T = M$. If the matrix is not positive semi definite, it can return an approximation of the pseudo square root using a (user selected) salvaging algorithm.

For more information see: "The most general methodology to create a valid correlation matrix for risk management and option pricing purposes", by R. Rebonato and P. Jäckel. The Journal of Risk, 2(2), Winter 1999/2000 <http://www.rebonato.com/correlationmatrix.pdf>

Revised and extended in "Monte Carlo Methods in Finance", by Peter Jäckel, Chapter 6.

Precondition:

the given matrix must be symmetric.

Warning

Higham algorithm only works for correlation matrices.

Tests

- the correctness of the results is tested by reproducing known good data.
- the correctness of the results is tested by checking returned values against numerical calculations.

9.611.3.17 `const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, Real componentRetainedPercentage, SalvagingAlgorithm::Type)` [related]

Returns the rank-reduced pseudo square root of a real symmetric matrix.

The result matrix has rank \leq maxRank. If maxRank \geq size, then the specified percentage of eigenvalues out of the eigenvalues' sum is retained.

If the input matrix is not positive semi definite, it can return an approximation of the pseudo square root using a (user selected) salvaging algorithm.

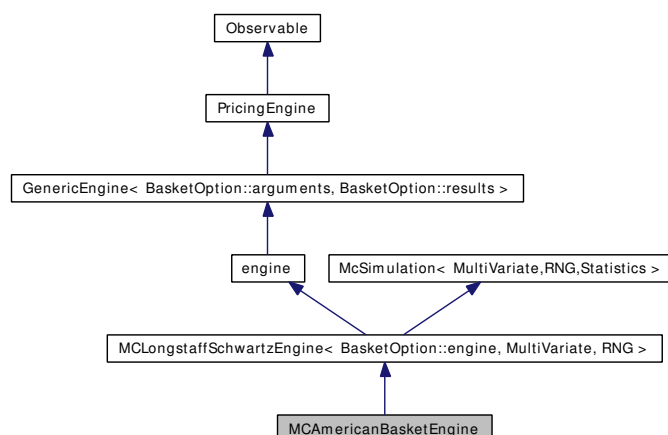
Precondition:

the given matrix must be symmetric.

9.612 MCAmericanBasketEngine Class Template Reference

```
#include <ql/pricingengines/basket/mcamericanbasketengine.hpp>
```

Inheritance diagram for MCAmericanBasketEngine:



9.612.1 Detailed Description

```
template<class RNG = PseudoRandom> class QuantLib::MCAmericanBasketEngine< RNG >
```

least-square Monte Carlo engine

Warning

This method is intrinsically weak for out-of-the-money options.

Public Member Functions

- **MCAmericanBasketEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed, Size nCalibrationSamples=[Null](#)< Size >())
- **MCAmericanBasketEngine** (Size requiredSamples, Size timeSteps, BigNatural seed=0, bool antitheticSampling=false)

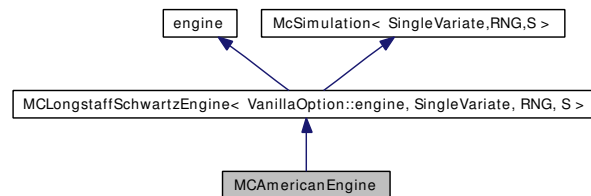
Protected Member Functions

- boost::shared_ptr< [LongstaffSchwartzPathPricer](#)< [MultiPath](#) > > **lsmPathPricer** () const

9.613 MCAmericanEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/mcamericanengine.hpp>
```

Inheritance diagram for MCAmericanEngine:



9.613.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MCAmericanEngine< RNG, S >
```

American Monte Carlo engine.

References:

Tests

the correctness of the returned value is tested by reproducing results available in web/literature

Public Member Functions

- **MCAmericanEngine** (Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed, Size polynomOrder, LsmBasisSystem::PolynomType polynomType, Size nCalibrationSamples=[Null](#)< Size >())

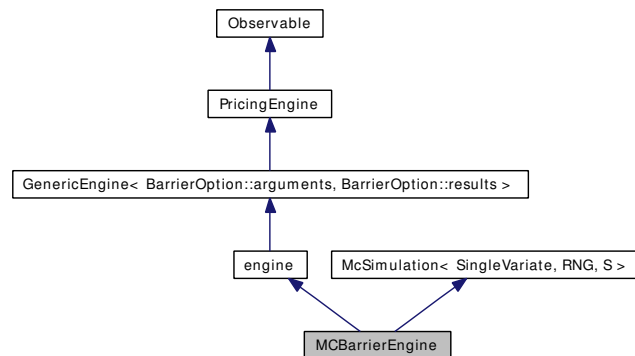
Protected Member Functions

- boost::shared_ptr< [LongstaffSchwartzPathPricer](#)< [Path](#) > > **lsmPathPricer** () const
- Real **controlVariateValue** () const
- boost::shared_ptr< [PricingEngine](#) > **controlPricingEngine** () const
- boost::shared_ptr< [PathPricer](#)< [Path](#) > > **controlPathPricer** () const

9.614 MBarrierEngine Class Template Reference

```
#include <ql/pricingengines/barrier/mcbarrierengine.hpp>
```

Inheritance diagram for MBarrierEngine:



9.614.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MBarrierEngine< RNG, S >
```

Pricing engine for barrier options using Monte Carlo simulation.

Uses the Brownian-bridge correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic Option Valuation* - D.R. Beaglehole, P.H. Dybvig and G. Zhou *Financial Analysts Journal*; Jan/Feb 1997; 53, 1. pg. 62-68 and *Simulating path-dependent options: A new approach* - M. El Babsiri and G. Noel *Journal of Derivatives*; Winter 1998; 6, 2; pg. 65-83

Tests

the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef `McSimulation< SingleVariate, RNG, S >::path_generator_type` **path_generator_type**
- typedef `McSimulation< SingleVariate, RNG, S >::path_pricer_type` **path_pricer_type**
- typedef `McSimulation< SingleVariate, RNG, S >::stats_type` **stats_type**

Public Member Functions

- **MBarrierEngine** (Size maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, bool isBiased, BigNatural seed)
- void **calculate** () const

Protected Member Functions

- [TimeGrid](#) **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const
- boost::shared_ptr< path_pricer_type > **pathPricer** () const

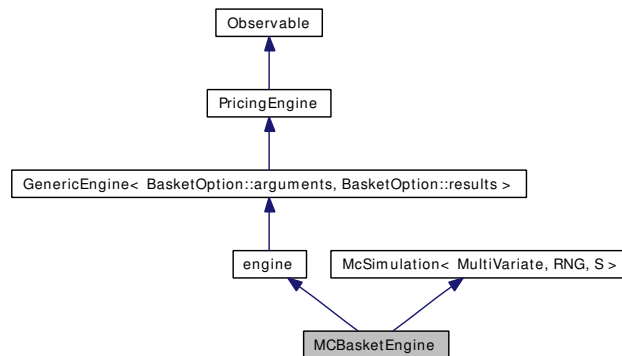
Protected Attributes

- Size **maxTimeStepsPerYear_**
- Size **requiredSamples_**
- Size **maxSamples_**
- Real **requiredTolerance_**
- bool **isBiased_**
- bool **brownianBridge_**
- BigNatural **seed_**

9.615 MCBasketEngine Class Template Reference

```
#include <ql/pricingengines/basket/mcbasketengine.hpp>
```

Inheritance diagram for MCBasketEngine:



9.615.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MCBasketEngine< RNG, S >
```

Pricing engine for basket options using Monte Carlo simulation.

Tests

the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef `McSimulation< MultiVariate, RNG, S >::path_generator_type` **path_generator_type**
- typedef `McSimulation< MultiVariate, RNG, S >::path_pricer_type` **path_pricer_type**
- typedef `McSimulation< MultiVariate, RNG, S >::stats_type` **stats_type**

Public Member Functions

- **MCBasketEngine** (Size maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const

Protected Member Functions

- `TimeGrid` **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const
- boost::shared_ptr< path_pricer_type > **pathPricer** () const

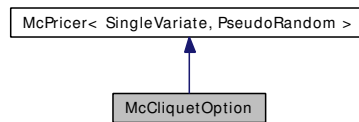
Protected Attributes

- Size `maxTimeStepsPerYear_`
- Size `requiredSamples_`
- Size `maxSamples_`
- Real `requiredTolerance_`
- bool `brownianBridge_`
- BigNatural `seed_`

9.616 McCliquetOption Class Reference

```
#include <ql/legacy/pricers/mccliquetoption.hpp>
```

Inheritance diagram for McCliquetOption:



9.616.1 Detailed Description

simple example of Monte Carlo pricer

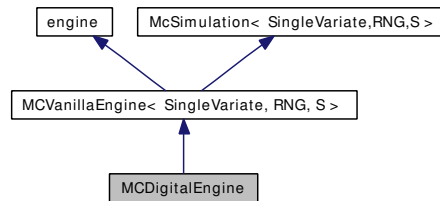
Public Member Functions

- **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)

9.617 MCDigitalEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/mcdigitalengine.hpp>
```

Inheritance diagram for MCDigitalEngine:



9.617.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MCDigitalEngine< RNG, S >
```

Pricing engine for digital options using Monte Carlo simulation.

Uses the Brownian Bridge correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic Option Valuation* - D.R. Beaglehole, P.H. Dybvig and G. Zhou *Financial Analysts Journal*; Jan/Feb 1997; 53, 1. pg. 62-68 and *Simulating path-dependent options: A new approach* - M. El Babsiri and G. Noel *Journal of Derivatives*; Winter 1998; 6, 2; pg. 65-83

Tests

the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing known good results.

Public Types

- typedef `MCVanillaEngine< SingleVariate, RNG, S >::path_generator_type` `path_generator_type`
- typedef `MCVanillaEngine< SingleVariate, RNG, S >::path_pricer_type` `path_pricer_type`
- typedef `MCVanillaEngine< SingleVariate, RNG, S >::stats_type` `stats_type`

Public Member Functions

- `MCDigitalEngine` (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

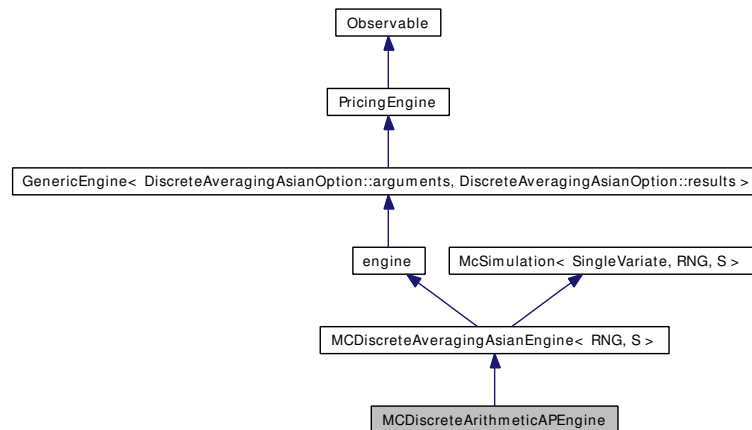
Protected Member Functions

- `boost::shared_ptr< path_pricer_type > pathPricer () const`

9.618 MCDiscreteArithmeticAPEngine Class Template Reference

```
#include <ql/pricingengines/asian/mc_discr_arith_av_price.hpp>
```

Inheritance diagram for MCDiscreteArithmeticAPEngine:



9.618.1 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)

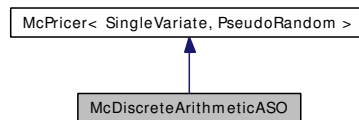
Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const
- boost::shared_ptr< path_pricer_type > **controlPathPricer** () const
- boost::shared_ptr< [PricingEngine](#) > **controlPricingEngine** () const

9.619 McDiscreteArithmeticASO Class Reference

```
#include <ql/legacy/pricers/mcdiscretearithmeticaso.hpp>
```

Inheritance diagram for McDiscreteArithmeticASO:



9.619.1 Detailed Description

Discrete arithmetic average-strike Asian option.

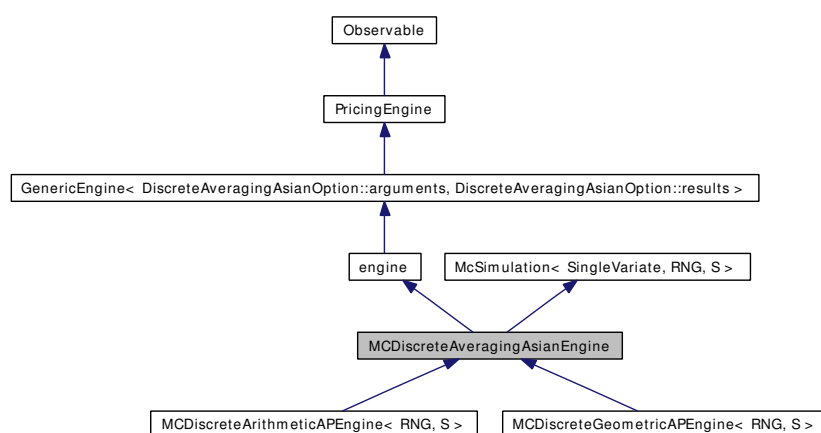
Public Member Functions

- **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)

9.620 MCDiscreteAveragingAsianEngine Class Template Reference

```
#include <ql/pricingengines/asian/mcdiscreteasianengine.hpp>
```

Inheritance diagram for MCDiscreteAveragingAsianEngine:



9.620.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MCDiscreteAveragingAsianEngine< RNG, S >
```

Pricing engine for discrete average Asians using Monte Carlo simulation.

Warning

control-variate calculation is disabled under VC++6.

Public Types

- typedef [McSimulation](#)< [SingleVariate](#), RNG, S >::path_generator_type **path_generator_type**
- typedef [McSimulation](#)< [SingleVariate](#), RNG, S >::path_pricer_type **path_pricer_type**
- typedef [McSimulation](#)< [SingleVariate](#), RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCDiscreteAveragingAsianEngine** (Size maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const

Protected Member Functions

- [TimeGrid](#) **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const
- Real **controlVariateValue** () const

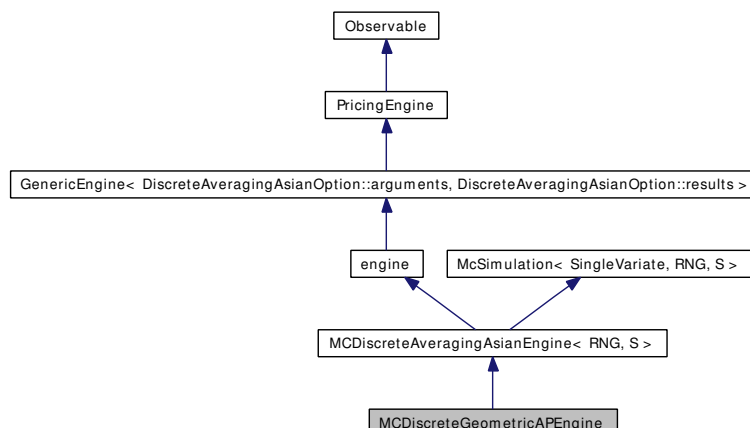
Protected Attributes

- Size **maxTimeStepsPerYear_**
- Size **requiredSamples_**
- Size **maxSamples_**
- Real **requiredTolerance_**
- bool **brownianBridge_**
- BigNatural **seed_**

9.621 MCDiscreteGeometricAPEngine Class Template Reference

```
#include <ql/pricingengines/asian/mc_discr_geom_av_price.hpp>
```

Inheritance diagram for MCDiscreteGeometricAPEngine:



9.621.1 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)

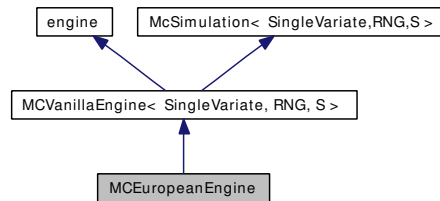
Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

9.622 MCEuropeanEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/mceuropeanengine.hpp>
```

Inheritance diagram for MCEuropeanEngine:



9.622.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MCEuropeanEngine< RNG, S >
```

European option pricing engine using Monte Carlo simulation.

Tests

the correctness of the returned value is tested by checking it against analytic results.

Public Types

- typedef `MCVanillaEngine< SingleVariate, RNG, S >::path_generator_type` **path_generator_type**
- typedef `MCVanillaEngine< SingleVariate, RNG, S >::path_pricer_type` **path_pricer_type**
- typedef `MCVanillaEngine< SingleVariate, RNG, S >::stats_type` **stats_type**

Public Member Functions

- **MCEuropeanEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

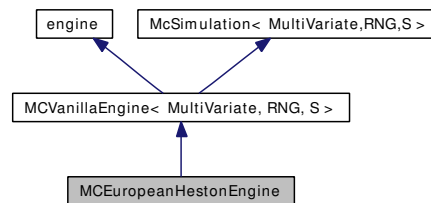
Protected Member Functions

- `boost::shared_ptr< path_pricer_type > pathPricer () const`

9.623 MCEuropeanHestonEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/mceuropeanhestonengine.hpp>
```

Inheritance diagram for MCEuropeanHestonEngine:



9.623.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MCEuropeanHestonEngine< RNG, S >
```

Monte Carlo Heston-model engine for European options.

Tests

the correctness of the returned value is tested by reproducing results available in web/literature

Public Types

- typedef [MCVanillaEngine< MultiVariate, RNG, S >::path_pricer_type](#) **path_pricer_type**

Public Member Functions

- **MCEuropeanHestonEngine** (Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

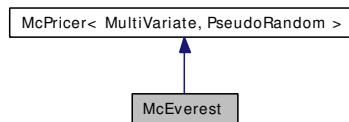
Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

9.624 McEverest Class Reference

```
#include <ql/legacy/pricers/mceverest.hpp>
```

Inheritance diagram for McEverest:



9.624.1 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

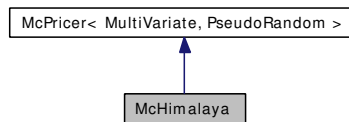
Public Member Functions

- **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)

9.625 McHimalaya Class Reference

```
#include <ql/legacy/pricers/mchimalaya.hpp>
```

Inheritance diagram for McHimalaya:



9.625.1 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.

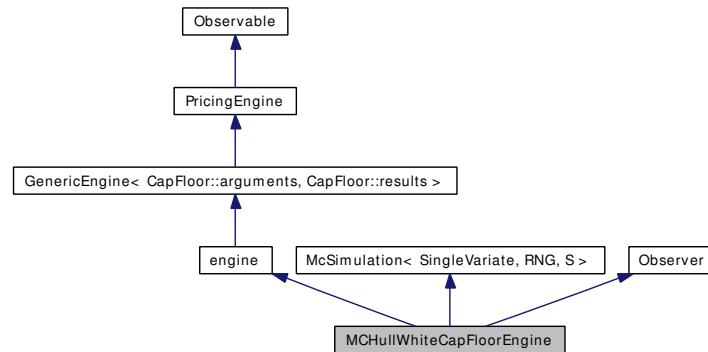
Public Member Functions

- **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)

9.626 MCHullWhiteCapFloorEngine Class Template Reference

```
#include <ql/pricingengines/capfloor/mchullwhiteengine.hpp>
```

Inheritance diagram for MCHullWhiteCapFloorEngine:



9.626.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MCHullWhiteCapFloorEngine< RNG, S >
```

Monte Carlo Hull-White engine for cap/floors.

Public Types

- typedef simulation::path_generator_type **path_generator_type**
- typedef simulation::path_pricer_type **path_pricer_type**
- typedef simulation::stats_type **stats_type**

Public Member Functions

- **MCHullWhiteCapFloorEngine** (const boost::shared_ptr< [HullWhite](#) > &model, bool brownianBridge, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const
- void [update](#) ()

Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const
- [TimeGrid](#) **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const

9.626.2 Member Function Documentation

9.626.2.1 void update () [virtual]

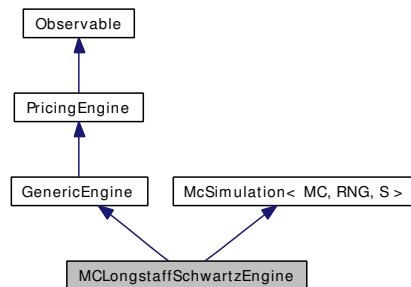
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

9.627 MCLongstaffSchwartzEngine Class Template Reference

```
#include <ql/pricingengines/mclongstaffschwartzengine.hpp>
```

Inheritance diagram for MCLongstaffSchwartzEngine:



9.627.1 Detailed Description

```
template<class GenericEngine, template< class > class MC, class RNG, class S = Statistics>
class QuantLib::MCLongstaffSchwartzEngine< GenericEngine, MC, RNG, S >
```

Longstaff-Schwarz Monte Carlo engine for early exercise options.

References:

Francis Longstaff, Eduardo Schwartz, 2001. Valuing American Options by Simulation: A Simple Least-Squares Approach, The Review of Financial Studies, Volume 14, No. 1, 113-147

Tests

the correctness of the returned value is tested by reproducing results available in web/literature

Public Types

- typedef MC< RNG >::path_type **path_type**
- typedef McSimulation< MC, RNG, S >::stats_type **stats_type**
- typedef McSimulation< MC, RNG, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< MC, RNG, S >::path_generator_type **path_generator_type**

Public Member Functions

- **MCLongstaffSchwartzEngine** (Size timeSteps, Size timeStepsPerYear, bool brownian-Bridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real required-Tolerance, Size maxSamples, BigNatural seed, Size nCalibrationSamples=Null< Size >())
- void **calculate** () const

Protected Member Functions

- virtual boost::shared_ptr< [LongstaffSchwartzPathPricer](#)< path_type > > **lsmPathPricer** () const=0
- [TimeGrid](#) **timeGrid** () const
- boost::shared_ptr< path_pricer_type > **pathPricer** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const

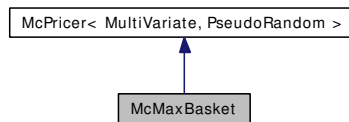
Protected Attributes

- const Size **timeSteps_**
- const Size **timeStepsPerYear_**
- const bool **brownianBridge_**
- const Size **requiredSamples_**
- const Real **requiredTolerance_**
- const Size **maxSamples_**
- const Size **seed_**
- const Size **nCalibrationSamples_**
- boost::shared_ptr< [LongstaffSchwartzPathPricer](#)< path_type > > **pathPricer_**

9.628 McMaxBasket Class Reference

```
#include <ql/legacy/pricers/mcmaxbasket.hpp>
```

Inheritance diagram for McMaxBasket:



9.628.1 Detailed Description

Max-basket Monte Carlo pricer.

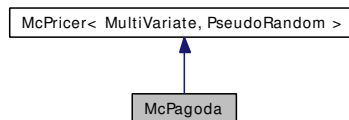
Public Member Functions

- **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)

9.629 McPagoda Class Reference

```
#include <ql/legacy/pricers/mcpagoda.hpp>
```

Inheritance diagram for McPagoda:



9.629.1 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.

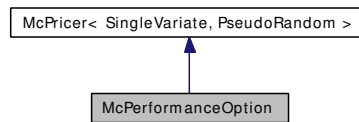
Public Member Functions

- **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)

9.630 McPerformanceOption Class Reference

```
#include <ql/legacy/pricers/mcperformanceoption.hpp>
```

Inheritance diagram for McPerformanceOption:



9.630.1 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)

9.631 McPricer Class Template Reference

```
#include <ql/legacy/pricers/mcpricer.hpp>
```

9.631.1 Detailed Description

```
template<template< class > class MC, class RNG, class S = Statistics> class  
QuantLib::McPricer< MC, RNG, S >
```

base class for Monte Carlo pricers

Eventually this class might be linked to the general tree of pricers, in order to have tools like `impliedVolatility` available. Also, it could, eventually, offer greeks methods. Deriving a class from [McPricer](#) gives an easy way to write a Monte Carlo Pricer. See [McEuropean](#) as example of one factor pricer, [Basket](#) as example of multi factor pricer.

Public Member Functions

- Real [value](#) (Real tolerance, Size maxSamples=QL_MAX_INTEGER, Size minSamples=1023) const
add samples until the required tolerance is reached
- Real [valueWithSamples](#) (Size samples, Size minSamples=1023) const
simulate a fixed number of samples
- Real [errorEstimate](#) () const
estimated error of the samples simulated so far
- const S & [sampleAccumulator](#) (void) const
access to the sample accumulator for more statistics

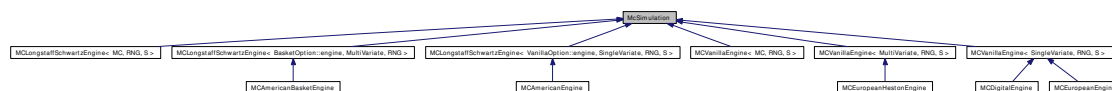
Protected Attributes

- boost::shared_ptr< [MonteCarloModel](#)< MC, RNG, S > > [mcModel_](#)

9.632 McSimulation Class Template Reference

```
#include <ql/pricingengines/mcsimulation.hpp>
```

Inheritance diagram for McSimulation:



9.632.1 Detailed Description

```
template<template< class > class MC, class RNG, class S = Statistics> class
QuantLib::McSimulation< MC, RNG, S >
```

base class for Monte Carlo engines

Eventually this class might offer greeks methods. Deriving a class from [McSimulation](#) gives an easy way to write a Monte Carlo engine.

See McVanillaEngine as an example.

Public Types

- typedef [MonteCarloModel](#)< MC, RNG, S >::path_generator_type **path_generator_type**
- typedef [MonteCarloModel](#)< MC, RNG, S >::path_pricer_type **path_pricer_type**
- typedef [MonteCarloModel](#)< MC, RNG, S >::stats_type **stats_type**
- typedef [MonteCarloModel](#)< MC, RNG, S >::result_type **result_type**

Public Member Functions

- result_type **value** (Real tolerance, Size maxSamples=QL_MAX_INTEGER, Size minSamples=1023) const
add samples until the required absolute tolerance is reached
- result_type **valueWithSamples** (Size samples) const
simulate a fixed number of samples
- result_type **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 result_type **controlVariateValue** () const

Static Protected Member Functions

- template<class Sequence>
static Real **maxError** (const Sequence &sequence)
- static Real **maxError** (Real error)

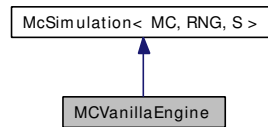
Protected Attributes

- boost::shared_ptr< [MonteCarloModel](#)< MC, RNG, S > > **mcModel_**
- bool **antitheticVariate_**
- bool **controlVariate_**

9.633 MCVanillaEngine Class Template Reference

```
#include <ql/pricingengines/vanilla/mcvanillaengine.hpp>
```

Inheritance diagram for MCVanillaEngine:



9.633.1 Detailed Description

```
template<template< class > class MC, class RNG, class S = Statistics, class Inst =
VanillaOption> class QuantLib::MCVanillaEngine< MC, RNG, S, Inst >
```

Pricing engine for vanilla options using Monte Carlo simulation.

Public Member Functions

- void **calculate** () const

Protected Types

- typedef [McSimulation](#)< MC, RNG, S >::path_generator_type **path_generator_type**
- typedef [McSimulation](#)< MC, RNG, S >::path_pricer_type **path_pricer_type**
- typedef [McSimulation](#)< MC, RNG, S >::stats_type **stats_type**
- typedef [McSimulation](#)< MC, RNG, S >::result_type **result_type**

Protected Member Functions

- **MCVanillaEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- [TimeGrid](#) **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const
- result_type **controlVariateValue** () const

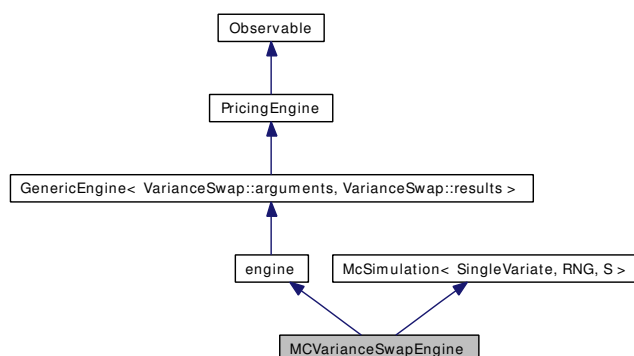
Protected Attributes

- Size **timeSteps_**
- Size **timeStepsPerYear_**
- Size **requiredSamples_**
- Size **maxSamples_**
- Real **requiredTolerance_**
- bool **brownianBridge_**
- BigNatural **seed_**

9.634 MCVarianceSwapEngine Class Template Reference

```
#include <ql/pricingengines/forward/mcvarianceswapengine.hpp>
```

Inheritance diagram for MCVarianceSwapEngine:



9.634.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class
QuantLib::MCVarianceSwapEngine< RNG, S >
```

Variance-swap pricing engine using Monte Carlo simulation,.

as described in Demeterfi, Derman, Kamal & Zou, "A Guide to Volatility and Variance Swaps", 1999

Todo

define tolerance of numerical integral and incorporate it in errorEstimate

Tests

returned fair variances checked for consistency with implied volatility curve.

Public Types

- typedef [McSimulation](#)< [SingleVariate](#), RNG, S >::path_generator_type **path_generator_type**
- typedef [McSimulation](#)< [SingleVariate](#), RNG, S >::path_pricer_type **path_pricer_type**
- typedef [McSimulation](#)< [SingleVariate](#), RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCVarianceSwapEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const

Protected Member Functions

- `boost::shared_ptr< path_pricer_type > pathPricer () const`
- `TimeGrid timeGrid () const`
- `boost::shared_ptr< path_generator_type > pathGenerator () const`

Protected Attributes

- `Size timeSteps_`
- `Size timeStepsPerYear_`
- `Size requiredSamples_`
- `Size maxSamples_`
- `Real requiredTolerance_`
- `bool brownianBridge_`
- `BigNatural seed_`

9.635 MersenneTwisterUniformRng Class Reference

```
#include <ql/math/randomnumbers/mt19937uniformrng.hpp>
```

9.635.1 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

9.635.2 Constructor & Destructor Documentation

9.635.2.1 MersenneTwisterUniformRng (unsigned long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

9.635.3 Member Function Documentation

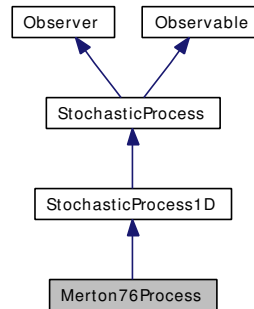
9.635.3.1 sample_type next () const

returns a sample with weight 1.0 containing a random number on (0.0, 1.0)-real-interval

9.636 Merton76Process Class Reference

```
#include <ql/processes/merton76process.hpp>
```

Inheritance diagram for Merton76Process:



9.636.1 Detailed Description

Merton-76 jump-diffusion process.

Public Member Functions

- **Merton76Process** (const [Handle](#)< [Quote](#) > &stateVariable, const [Handle](#)< [YieldTermStructure](#) > ÷ndTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &blackVolTS, const [Handle](#)< [Quote](#) > &jumpInt, const [Handle](#)< [Quote](#) > &logJMean, const [Handle](#)< [Quote](#) > &logJVol, const boost::shared_ptr< discretization > &d=boost::shared_ptr< discretization >(new [EulerDiscretization](#)))
- Time [time](#) (const [Date](#) &) const

StochasticProcess1D interface

- Real [x0](#) () const
returns the initial value of the state variable
- Real [drift](#) (Time, Real) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- Real [diffusion](#) (Time, Real) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- Real [apply](#) (Real, Real) const

Inspectors

- const [Handle](#)< [Quote](#) > & **stateVariable** () const
- const [Handle](#)< [YieldTermStructure](#) > & **dividendYield** () const
- const [Handle](#)< [YieldTermStructure](#) > & **riskFreeRate** () const
- const [Handle](#)< [BlackVolTermStructure](#) > & **blackVolatility** () const
- const [Handle](#)< [Quote](#) > & **jumpIntensity** () const
- const [Handle](#)< [Quote](#) > & **logMeanJump** () const
- const [Handle](#)< [Quote](#) > & **logJumpVolatility** () const

9.636.2 Member Function Documentation

9.636.2.1 Real apply (Real x_0 , Real dx) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented from [StochasticProcess1D](#).

9.636.2.2 Time time (const Date &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:

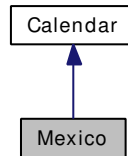
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

9.637 Mexico Class Reference

```
#include <ql/time/calendars/mexico.hpp>
```

Inheritance diagram for Mexico:



9.637.1 Detailed Description

Mexican calendars

Holidays for the Mexican stock exchange (data from <http://www.bmv.com.mx/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Constitution Day, February 5th
- Birthday of Benito Juarez, March 21st
- Holy Thursday
- Good Friday
- Labour Day, May 1st
- National Day, September 16th
- Our Lady of Guadalupe, December 12th
- Christmas, December 25th

Public Types

- enum [Market](#) { [BMV](#) }

Public Member Functions

- [Mexico](#) ([Market](#) m=BMV)

9.637.2 Member Enumeration Documentation

9.637.2.1 enum Market

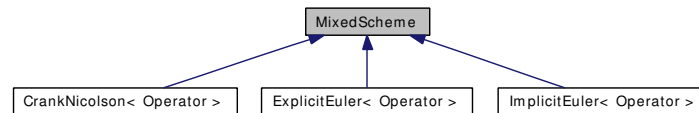
Enumerator:

BMV Mexican stock exchange.

9.638 MixedScheme Class Template Reference

```
#include <ql/methods/finitedifferences/mixedscheme.hpp>
```

Inheritance diagram for MixedScheme:



9.638.1 Detailed Description

```
template<class Operator> class QuantLib::MixedScheme< Operator >
```

Mixed (explicit/implicit) scheme for finite difference methods.

In this implementation, the passed operator must be derived from either `TimeConstantOperator` or `TimeDependentOperator`. Also, it must implement at least the following interface:

```

typedef ... array_type;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
array_type applyTo(const array_type&);
array_type solveFor(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, const Operator&);
Operator operator+(const Operator&, const Operator&);
Operator operator+(const Operator&, const Operator&);

```

Warning

The differential operator must be linear for this evolver to work.

Todo

- derive variable theta schemes
- introduce multi time-level schemes.

Public Types

- `typedef OperatorTraits< Operator > traits`
- `typedef traits::operator_type operator_type`

- typedef traits::array_type **array_type**
- typedef traits::bc_set **bc_set**
- typedef [traits::condition_type](#) **condition_type**

Public Member Functions

- **MixedScheme** (const operator_type &L, Real theta, const bc_set &bcs)
- void **step** (array_type &a, Time t)
- void **setStep** (Time dt)

Protected Attributes

- operator_type **L_**
- operator_type **I_**
- operator_type **explicitPart_**
- operator_type **implicitPart_**
- Time **dt_**
- Real **theta_**
- bc_set **bcs_**

9.639 Money Class Reference

```
#include <ql/money.hpp>
```

9.639.1 Detailed Description

amount of cash

Tests

money arithmetic is tested with and without currency conversions.

Conversion settings

These parameters are used for combining money amounts in different currencies

- enum [ConversionType](#) { [NoConversion](#), [BaseCurrencyConversion](#), [AutomatedConversion](#) }
- static [ConversionType](#) [conversionType](#)
- static [Currency](#) [baseCurrency](#)

Public Member Functions

Constructors

- [Money](#) (const [Currency](#) ¤cy, Decimal value)
- [Money](#) (Decimal value, const [Currency](#) ¤cy)

Inspectors

- const [Currency](#) & [currency](#) () const
- Decimal [value](#) () const
- [Money](#) [rounded](#) () const

Money arithmetics

See below for non-member functions and for settings which determine the behavior of the operators.

- [Money](#) [operator+](#) () const
- [Money](#) [operator-](#) () const
- [Money](#) & [operator+=](#) (const [Money](#) &)
- [Money](#) & [operator-=](#) (const [Money](#) &)
- [Money](#) & [operator*=](#) (Decimal)
- [Money](#) & [operator/=](#) (Decimal)

Related Functions

(Note that these are not member functions.)

- [Money](#) [operator+](#) (const [Money](#) &, const [Money](#) &)
- [Money](#) [operator-](#) (const [Money](#) &, const [Money](#) &)

- `Money operator *` (const `Money` &, Decimal)
- `Money operator *` (Decimal, const `Money` &)
- `Money operator/` (const `Money` &, Decimal)
- `Decimal operator/` (const `Money` &, const `Money` &)
- `bool operator==` (const `Money` &, const `Money` &)
- `bool operator!=` (const `Money` &, const `Money` &)
- `bool operator<` (const `Money` &, const `Money` &)
- `bool operator<=` (const `Money` &, const `Money` &)
- `bool operator>` (const `Money` &, const `Money` &)
- `bool operator>=` (const `Money` &, const `Money` &)
- `bool close` (const `Money` &, const `Money` &, Size n=42)
- `bool close_enough` (const `Money` &, const `Money` &, Size n=42)
- `Money operator *` (Decimal, const `Currency` &)
- `Money operator *` (const `Currency` &, Decimal)
- `std::ostream & operator<<` (std::ostream &, const `Money` &)

9.639.2 Member Enumeration Documentation

9.639.2.1 enum ConversionType

Enumerator:

NoConversion do not perform conversions

BaseCurrencyConversion convert both operands to the base currency before converting

AutomatedConversion return the result in the currency of the first operand

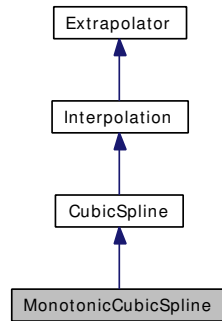
9.639.3 Friends And Related Function Documentation

- 9.639.3.1 Money operator+ (const Money &, const Money &) [related]
- 9.639.3.2 Money operator- (const Money &, const Money &) [related]
- 9.639.3.3 Money operator * (const Money &, Decimal) [related]
- 9.639.3.4 Money operator * (Decimal, const Money &) [related]
- 9.639.3.5 Money operator/ (const Money &, Decimal) [related]
- 9.639.3.6 Decimal operator/ (const Money &, const Money &) [related]
- 9.639.3.7 bool operator== (const Money &, const Money &) [related]
- 9.639.3.8 bool operator!= (const Money &, const Money &) [related]
- 9.639.3.9 bool operator< (const Money &, const Money &) [related]
- 9.639.3.10 bool operator<= (const Money &, const Money &) [related]
- 9.639.3.11 bool operator> (const Money &, const Money &) [related]
- 9.639.3.12 bool operator>= (const Money &, const Money &) [related]
- 9.639.3.13 bool close (const Money &, const Money &, Size $n = 42$) [related]
- 9.639.3.14 bool close_enough (const Money &, const Money &, Size $n = 42$) [related]
- 9.639.3.15 Money operator * (Decimal, const Currency &) [related]
- 9.639.3.16 Money operator * (const Currency &, Decimal) [related]
- 9.639.3.17 std::ostream & operator<< (std::ostream &, const Money &) [related]

9.640 MonotonicCubicSpline Class Reference

```
#include <ql/math/interpolations/cubicspline.hpp>
```

Inheritance diagram for MonotonicCubicSpline:



9.640.1 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`)

9.640.2 Constructor & Destructor Documentation

- 9.640.2.1 `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.

9.641 MonteCarloModel Class Template Reference

```
#include <ql/methods/montecarlo/montecarlomodel.hpp>
```

9.641.1 Detailed Description

```
template<template< class > class MC, class RNG, class S = Statistics> class
QuantLib::MonteCarloModel< MC, RNG, S >
```

General-purpose Monte Carlo model for path samples.

The template arguments of this class correspond to available policies for the particular model to be instantiated—i.e., whether it is single- or multi-asset, or whether it should use pseudo-random or low-discrepancy numbers for path generation. Such decisions are grouped in trait classes so as to be orthogonal—see [mctraits.hpp](#) for examples.

The constructor accepts two safe references, i.e. two smart pointers, one to a path generator and the other to a path pricer. In case of control variate technique the user should provide the additional control option, namely the option path pricer and the option value.

Examples:

[DiscreteHedging.cpp](#).

Public Types

- typedef MC< RNG > **mc_traits**
- typedef RNG **rng_traits**
- typedef MC< RNG >::path_generator_type **path_generator_type**
- typedef MC< RNG >::path_pricer_type **path_pricer_type**
- typedef path_generator_type::sample_type **sample_type**
- typedef path_pricer_type::result_type **result_type**
- typedef S **stats_type**

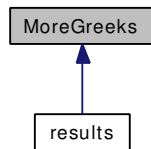
Public Member Functions

- **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

9.642 MoreGreeks Class Reference

```
#include <ql/option.hpp>
```

Inheritance diagram for MoreGreeks:



9.642.1 Detailed Description

more additional option results

Public Member Functions

- `void reset ()`

Public Attributes

- Real `itmCashProbability`
- Real `deltaForward`
- Real `elasticity`
- Real `thetaPerDay`
- Real `strikeSensitivity`

9.643 MoroInverseCumulativeNormal Class Reference

```
#include <ql/math/distributions/normaldistribution.hpp>
```

9.643.1 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

9.644 MTBrownianGenerator Class Reference

```
#include <ql/models/marketmodels/browniangenerators/mtbrowniangenerator.hpp>
```

9.644.1 Detailed Description

Mersenne-twister Brownian generator for market-model simulations.

Incremental Brownian generator using a Mersenne-twister uniform generator and inverse-cumulative Gaussian method.

Note:

At this time, generation of the underlying uniform sequence is eager, while its transformation into Gaussian variates is lazy. Further optimization might be possible by using the Mersenne twister directly instead of a [RandomSequenceGenerator](#); however, it is not clear how much of a difference this would make when compared to the inverse-cumulative Gaussian calculation.

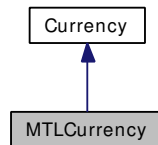
Public Member Functions

- **MTBrownianGenerator** (Size factors, Size steps, unsigned long seed=0)
- Real **nextStep** (std::vector< Real > &)
- Real **nextPath** ()
- Size **numberOfFactors** () const
- Size **numberOfSteps** () const

9.645 MTLCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for MTLCurrency:



9.645.1 Detailed Description

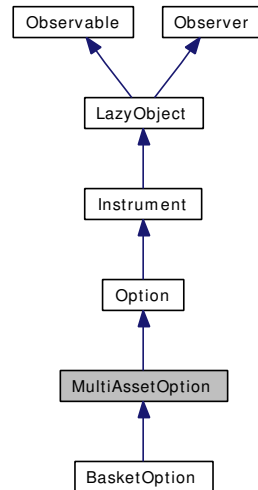
Maltese lira.

The ISO three-letter code is MTL; the numeric code is 470. It is divided in 100 cents.

9.646 MultiAssetOption Class Reference

```
#include <ql/instruments/multiassetoption.hpp>
```

Inheritance diagram for MultiAssetOption:



9.646.1 Detailed Description

Base class for options on multiple assets.

Public Member Functions

- **MultiAssetOption** (const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [Payoff](#) > &, const boost::shared_ptr< [Exercise](#) > &, const boost::shared_ptr< [PricingEngine](#) > & engine=boost::shared_ptr< [PricingEngine](#) >())
- void **setupArguments** ([PricingEngine::arguments](#) *) const
- void **fetchResults** (const [PricingEngine::results](#) *) const

Instrument interface

- bool **isExpired** () const
returns whether the instrument is still tradable.

greeks

- Real **delta** () const
- Real **gamma** () const
- Real **theta** () const
- Real **vega** () const
- Real **rho** () const
- Real **dividendRho** () const

Protected Member Functions

- void [setupExpired](#) () const

Protected Attributes

- Real [delta_](#)
- Real [gamma_](#)
- Real [theta_](#)
- Real [vega_](#)
- Real [rho_](#)
- Real [dividendRho_](#)
- boost::shared_ptr<[StochasticProcess](#)> [stochasticProcess_](#)

Classes

- class [arguments](#)
Arguments for multi-asset option calculation
- class [results](#)
Results from multi-asset option calculation

9.646.2 Member Function Documentation

9.646.2.1 void [fetchResults](#) (const PricingEngine::results *) const [virtual]

When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

9.646.2.2 void [setupExpired](#) () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from [Instrument](#).

9.647 MultiAssetOption::arguments Class Reference

```
#include <ql/instruments/multiassetoption.hpp>
```

9.647.1 Detailed Description

Arguments for multi-asset option calculation

Public Member Functions

- void **validate** () const

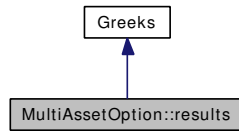
Public Attributes

- boost::shared_ptr< [StochasticProcess](#) > **stochasticProcess**

9.648 MultiAssetOption::results Class Reference

```
#include <ql/instruments/multiassetoption.hpp>
```

Inheritance diagram for MultiAssetOption::results:



9.648.1 Detailed Description

Results from multi-asset option calculation

Public Member Functions

- void **reset** ()

9.649 MultiCubicSpline Class Template Reference

```
#include <ql/math/interpolations/multicubicspline.hpp>
```

9.649.1 Detailed Description

template<Size i> class QuantLib::MultiCubicSpline< i >

N-dimensional cubic spline interpolation between discrete points.

Tests

interpolated values are checked against the original function.

Todo

- allow extrapolation as for the other interpolations
- investigate if and how to implement Hyman filters and different boundary conditions

Bug

cannot interpolate at the grid points on the boundary surface of the N-dimensional region

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

9.650 MultiPath Class Reference

```
#include <ql/methods/montecarlo/multipath.hpp>
```

9.650.1 Detailed Description

Correlated multiple asset paths.

[MultiPath](#) contains the list of paths for each asset, i.e., `multipath[j]` is the path followed by the j -th asset.

Public Member Functions

- **MultiPath** (Size nAsset, const [TimeGrid](#) &timeGrid)
- **MultiPath** (const std::vector< [Path](#) > &multiPath)

inspectors

- Size **assetNumber** () const
- Size **pathSize** () const

read/write access to components

- const [Path](#) & **operator**[] (Size j) const
- const [Path](#) & **at** (Size j) const
- [Path](#) & **operator**[] (Size j)
- [Path](#) & **at** (Size j)

9.651 MultiPathGenerator Class Template Reference

```
#include <ql/methods/montecarlo/multipathgenerator.hpp>
```

9.651.1 Detailed Description

template<class GSG> class QuantLib::MultiPathGenerator< GSG >

Generates a multipath from a random number generator.

RSG is a sample generator which returns a random sequence. It must have the minimal interface:

```
RSG {  
    Sample<Array> next();  
};
```

Tests

the generated paths are checked against cached results

Public Types

- typedef [Sample< MultiPath >](#) **sample_type**

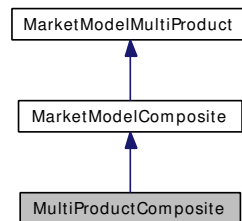
Public Member Functions

- **MultiPathGenerator** (const boost::shared_ptr< [StochasticProcess](#) > &, const [TimeGrid](#) &, GSG generator, bool brownianBridge=false)
- const [sample_type](#) & **next** () const
- const [sample_type](#) & **antithetic** () const

9.652 MultiProductComposite Class Reference

```
#include <ql/models/marketmodels/products/multiproductcomposite.hpp>
```

Inheritance diagram for MultiProductComposite:



9.652.1 Detailed Description

Composition of one or more market-model products.

Instances of this class build a multiple market-model product by composing two or more sub-products.

Precondition:

All subproducts must have the same rate times.

Public Member Functions

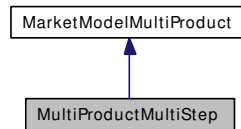
MarketModelMultiProduct interface

- Size **numberOfProducts** () const
- Size **maxNumberOfCashFlowsPerProductPerStep** () const
- bool **nextTimeStep** (const **CurveState** ¤tState, std::vector< Size > &numberCashFlowsThisStep, std::vector< std::vector< **CashFlow** > > &cashFlowsGenerated)
return value indicates whether path is finished, TRUE means done
- std::auto_ptr< **MarketModelMultiProduct** > **clone** () const
returns a newly-allocated copy of itself

9.653 MultiProductMultiStep Class Reference

```
#include <ql/models/marketmodels/products/multiproductmultistep.hpp>
```

Inheritance diagram for MultiProductMultiStep:



9.653.1 Detailed Description

Multiple-step market-model product.

This is the abstract base class that encapsulates the notion of a [MarketModelMultiProduct](#) which can be evaluated in a more than one step (aka Rebonato's long jump).

Public Member Functions

- **MultiProductMultiStep** (const std::vector< Time > &rateTimes)

MarketModelMultiProduct interface

- std::vector< Size > **suggestedNumeraires** () const
- const [EvolutionDescription](#) & **evolution** () const

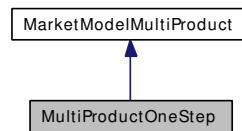
Protected Attributes

- std::vector< Time > **rateTimes_**
- [EvolutionDescription](#) **evolution_**

9.654 MultiProductOneStep Class Reference

```
#include <ql/models/marketmodels/products/multiproductonestep.hpp>
```

Inheritance diagram for MultiProductOneStep:



9.654.1 Detailed Description

Single-step market-model product.

This is the abstract base class that encapsulates the notion of a [MarketModelMultiProduct](#) which can be evaluated in one step (aka Rebonato's very long jump).

Public Member Functions

- **MultiProductOneStep** (const std::vector< Time > &rateTimes)

MarketModelMultiProduct interface

- const [EvolutionDescription](#) & **evolution** () const
- std::vector< Size > **suggestedNumeraires** () const

Protected Attributes

- std::vector< Time > **rateTimes_**
- [EvolutionDescription](#) **evolution_**

9.655 MultiVariate Struct Template Reference

```
#include <ql/methods/montecarlo/mctraits.hpp>
```

9.655.1 Detailed Description

template<class RNG = PseudoRandom> struct QuantLib::MultiVariate< RNG >

default Monte Carlo traits for multi-variate models

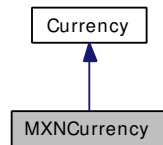
Public Types

- enum { **allowsErrorEstimate** = RNG::allowsErrorEstimate }
- typedef RNG **rng_traits**
- typedef [MultiPath](#) **path_type**
- typedef [PathPricer](#)< [path_type](#) > **path_pricer_type**
- typedef RNG::rsg_type **rsg_type**
- typedef [MultiPathGenerator](#)< rsg_type > **path_generator_type**

9.656 MXNCurrency Class Reference

```
#include <ql/currencies/america.hpp>
```

Inheritance diagram for MXNCurrency:



9.656.1 Detailed Description

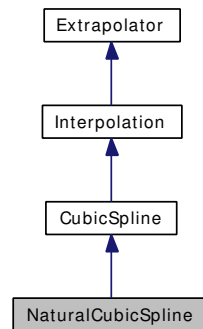
Mexican peso.

The ISO three-letter code is MXN; the numeric code is 484. It is divided in 100 centavos.

9.657 NaturalCubicSpline Class Reference

```
#include <ql/math/interpolations/cubicspline.hpp>
```

Inheritance diagram for NaturalCubicSpline:



9.657.1 Detailed Description

Cubic spline with null second derivative at end points

Public Member Functions

- `template<class I1, class I2>`
[NaturalCubicSpline](#) (const I1 &*xBegin*, const I1 &*xEnd*, const I2 &*yBegin*)

9.657.2 Constructor & Destructor Documentation

9.657.2.1 NaturalCubicSpline (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

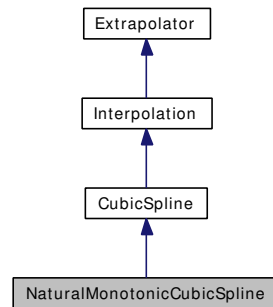
Precondition:

the *x* values must be sorted.

9.658 NaturalMonotonicCubicSpline Class Reference

```
#include <ql/math/interpolations/cubicspline.hpp>
```

Inheritance diagram for NaturalMonotonicCubicSpline:



9.658.1 Detailed Description

Natural cubic spline with monotonicity constraint.

Public Member Functions

- `template<class I1, class I2>`
`NaturalMonotonicCubicSpline` (const I1 &*xBegin*, const I1 &*xEnd*, const I2 &*yBegin*)

9.658.2 Constructor & Destructor Documentation

9.658.2.1 `NaturalMonotonicCubicSpline` (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

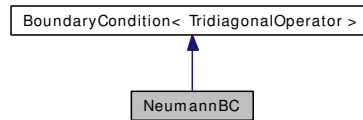
Precondition:

the *x* values must be sorted.

9.659 NeumannBC Class Reference

```
#include <ql/methods/finitedifferences/boundarycondition.hpp>
```

Inheritance diagram for NeumannBC:



9.659.1 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)

9.659.2 Member Function Documentation

9.659.2.1 void [applyBeforeApplying](#) ([TridiagonalOperator](#) &) const [virtual]

This method modifies an operator L before it is applied to an array u so that $v = Lu$ will satisfy the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

9.659.2.2 void [applyAfterApplying](#) ([Array](#) &) const [virtual]

This method modifies an array u so that it satisfies the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

9.659.2.3 void applyBeforeSolving (TridiagonalOperator &, Array & rhs) const [virtual]

This method modifies an operator L before the linear system $Lu' = u$ is solved so that u' will satisfy the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

9.659.2.4 void applyAfterSolving (Array &) const [virtual]

This method modifies an array u so that it satisfies the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

9.659.2.5 void setTime (Time t) [virtual]

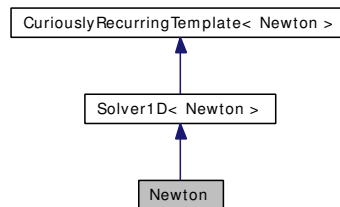
This method sets the current time for time-dependent boundary conditions.

Implements [BoundaryCondition< TridiagonalOperator >](#).

9.660 Newton Class Reference

```
#include <ql/math/solvers1d/newton.hpp>
```

Inheritance diagram for Newton:



9.660.1 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.

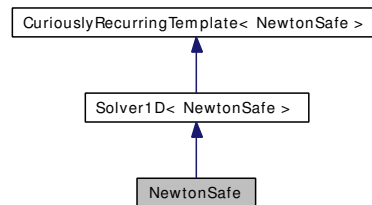
Public Member Functions

- `template<class F>`
`Real solveImpl (const F &f, Real xAccuracy) const`

9.661 NewtonSafe Class Reference

```
#include <ql/math/solvers1d/newtonsafe.hpp>
```

Inheritance diagram for NewtonSafe:



9.661.1 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.

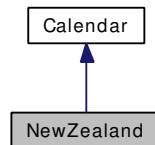
Public Member Functions

- `template<class F>`
`Real solveImpl (const F &f, Real xAccuracy) const`

9.662 NewZealand Class Reference

```
#include <ql/time/calendars/newzealand.hpp>
```

Inheritance diagram for NewZealand:



9.662.1 Detailed Description

New Zealand calendar.

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday or Tuesday)
- Day after New Year's Day, January 2st (possibly moved to Monday or Tuesday)
- Anniversary Day, Monday nearest January 22nd
- Waitangi Day. February 6th
- Good Friday
- Easter Monday
- ANZAC Day. April 25th
- Queen's Birthday, first Monday in June
- Labour Day, fourth Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

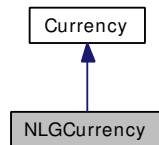
Note:

The holiday rules for New Zealand were documented by David Gilbert for IDB (<http://www.jrefinery.com/ibd/>)

9.663 NLGCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for NLGCurrency:



9.663.1 Detailed Description

Dutch guilder.

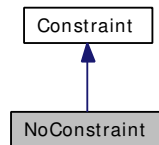
The ISO three-letter code was NLG; the numeric code was 528. It was divided in 100 cents.

Obsoleted by the Euro since 1999.

9.664 NoConstraint Class Reference

```
#include <ql/math/optimization/constraint.hpp>
```

Inheritance diagram for NoConstraint:



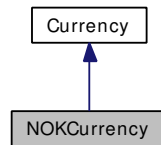
9.664.1 Detailed Description

No constraint.

9.665 NOKCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for NOKCurrency:



9.665.1 Detailed Description

Norwegian krone.

The ISO three-letter code is NOK; the numeric code is 578. It is divided in 100 øre.

9.666 NonLinearLeastSquare Class Reference

```
#include <ql/math/optimization/leastsquare.hpp>
```

9.666.1 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

9.667 NormalDistribution Class Reference

```
#include <ql/math/distributions/normaldistribution.hpp>
```

9.667.1 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.

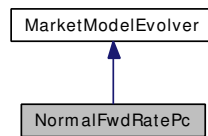
Public Member Functions

- **NormalDistribution** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real x) const
- Real **derivative** (Real x) const

9.668 NormalFwdRatePc Class Reference

```
#include <ql/models/marketmodels/evolvers/normalfwdratepc.hpp>
```

Inheritance diagram for NormalFwdRatePc:



9.668.1 Detailed Description

Predictor-Corrector.

Public Member Functions

- **NormalFwdRatePc** (const boost::shared_ptr< [MarketModel](#) > &, const BrownianGeneratorFactory &, const std::vector< Size > &numeraires, Size initialStep=0)

MarketModel interface

- const std::vector< Size > & **numeraires** () const
- Real **startNewPath** ()
- Real **advanceStep** ()
- Size **currentStep** () const
- const [CurveState](#) & **currentState** () const
- void **setInitialState** (const [CurveState](#) &)

9.669 Norway Class Reference

```
#include <ql/time/calendars/norway.hpp>
```

Inheritance diagram for Norway:



9.669.1 Detailed Description

Norwegian calendar.

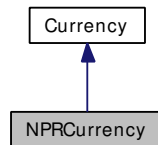
Holidays:

- Saturdays
- Sundays
- Holy Thursday
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- May Day, May 1st
- National Independence Day, May 17st
- Christmas, December 25th
- Boxing Day, December 26th

9.670 NPRCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for NPRCurrency:



9.670.1 Detailed Description

Nepal rupee.

The ISO three-letter code is NPR; the numeric code is 524. It is divided in 100 paise.

9.671 Null Class Template Reference

```
#include <ql/utilities/null.hpp>
```

9.671.1 Detailed Description

```
template<class Type> class QuantLib::Null< Type >
```

template class providing a null value for a given type.

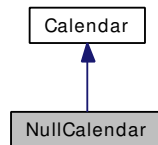
Public Member Functions

- `operator Type () const`

9.672 NullCalendar Class Reference

```
#include <ql/time/calendars/nullcalendar.hpp>
```

Inheritance diagram for NullCalendar:



9.672.1 Detailed Description

Calendar for reproducing theoretical calculations.

This calendar has no holidays. It ensures that dates at whole-month distances have the same day of month.

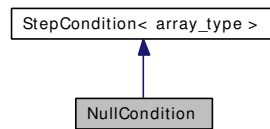
Examples:

[Replication.cpp](#), and [Repo.cpp](#).

9.673 NullCondition Class Template Reference

```
#include <ql/methods/finitedifferences/stepcondition.hpp>
```

Inheritance diagram for NullCondition:



9.673.1 Detailed Description

```
template<class array_type> class QuantLib::NullCondition< array_type >
```

null step condition

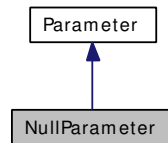
Public Member Functions

- void **applyTo** (array_type &, Time) const

9.674 NullParameter Class Reference

```
#include <ql/models/parameter.hpp>
```

Inheritance diagram for NullParameter:



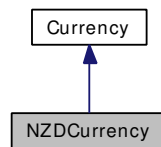
9.674.1 Detailed Description

Parameter which is always zero $a(t) = 0$

9.675 NZDCurrency Class Reference

```
#include <ql/currencies/oceania.hpp>
```

Inheritance diagram for NZDCurrency:



9.675.1 Detailed Description

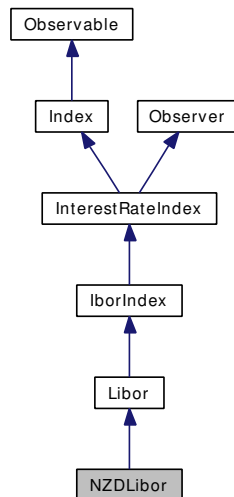
New Zealand dollar.

The ISO three-letter code is NZD; the numeric code is 554. It is divided in 100 cents.

9.676 NZDLibor Class Reference

```
#include <ql/indexes/ibor/nzdlibor.hpp>
```

Inheritance diagram for NZDLibor:



9.676.1 Detailed Description

NZD LIBOR rate

New Zealand Dollar LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

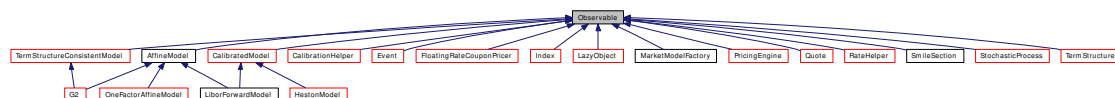
Public Member Functions

- **NZDLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), Natural settlementDays=2)

9.677 Observable Class Reference

```
#include <ql/patterns/observable.hpp>
```

Inheritance diagram for Observable:



9.677.1 Detailed Description

Object that notifies its changes to a set of observables.

Public Member Functions

- **Observable** (const [Observable](#) &)
- **Observable** & **operator=** (const [Observable](#) &)
- void **notifyObservers** ()

Friends

- class **Observer**

9.677.2 Member Function Documentation

9.677.2.1 **Observable** & **operator=** (const [Observable](#) & *o*)

Warning

notification is sent before the copy constructor has a chance of actually change the data members. Therefore, observers whose `update()` method tries to use their observables will not see the updated values. It is suggested that the `update()` method just raise a flag in order to trigger a later recalculation.

9.677.2.2 **void notifyObservers** ()

This method should be called at the end of non-const methods or when the programmer desires to notify any changes.

9.678 ObservableValue Class Template Reference

```
#include <ql/utilities/observablevalue.hpp>
```

9.678.1 Detailed Description

template<class T> class QuantLib::ObservableValue< T >

observable and assignable proxy to concrete value

Observers can be registered with instances of this class so that they are notified when a different value is assigned to such instances. Client code can copy the contained value or pass it to functions via implicit conversion.

Note:

it is not possible to call non-const method on the returned value. This is by design, as this possibility would necessarily bypass the notification code; client code should modify the value via re-assignment instead.

Public Member Functions

- **ObservableValue** (const T &)
- **ObservableValue** (const [ObservableValue](#)< T > &)
- **operator T** () const
implicit conversion
- **operator boost::shared_ptr** () const
- const T & **value** () const
explicit inspector

controlled assignment

- [ObservableValue](#)< T > & **operator=** (const T &)
- [ObservableValue](#)< T > & **operator=** (const [ObservableValue](#)< T > &)

9.679 Observer Class Reference

```
#include <ql/patterns/observable.hpp>
```

Inheritance diagram for Observer:



9.679.1 Detailed Description

Object that gets notified when a given observable changes.

Public Member Functions

- **Observer** (const [Observer](#) &)
- **Observer & operator=** (const [Observer](#) &)
- void **registerWith** (const boost::shared_ptr< [Observable](#) > &)
- void **unregisterWith** (const boost::shared_ptr< [Observable](#) > &)
- virtual void **update** ()=0

9.679.2 Member Function Documentation

9.679.2.1 virtual void update () [pure virtual]

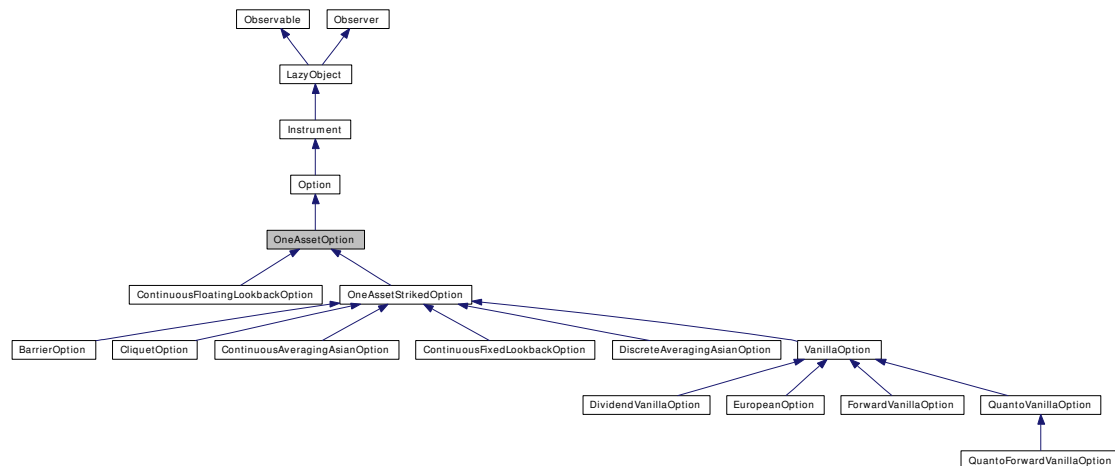
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implemented in [CappedFlooredCoupon](#), [FloatingRateCouponPricer](#), [FloatingRateCoupon](#), [InterestRateIndex](#), [CalibrationHelper](#), [CalibratedModel](#), [LazyObject](#), [BlackCapFloorEngine](#), [MarketModelCapFloorEngine](#), [MCHullWhiteCapFloorEngine](#), [GenericModelEngine](#), [LatticeShortRateModelEngine](#), [BlackSwaptionEngine](#), [GeneralizedBlackScholesProcess](#), [HestonProcess](#), [CompositeQuote](#), [DerivedQuote](#), [ForwardValueQuote](#), [FuturesConvAdjustmentQuote](#), [StochasticProcess](#), [TermStructure](#), [CapVolatilityVector](#), [DecInterpCapletVolStructure](#), [CmsMarket](#), [SwaptionVolatilityCube](#), [SwaptionVolatilityMatrix](#), [ExtendedDiscountCurve](#), [FlatForward](#), [RateHelper](#), [PiecewiseYieldCurve](#), [PiecewiseZeroSpreadedTermStructure](#), [RelativeDateRateHelper](#), [GenericModelEngine< ShortRateModel, Arguments, Results >](#), [GenericModelEngine< LiborForwardModel, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< ShortRateModel, VanillaSwap::arguments, VanillaSwap::results >](#), [GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< G2, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >](#), [GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results >](#), [GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< HestonModel, VanillaOption::arguments, VanillaOption::results >](#), [LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >](#), [LatticeShortRateModelEngine< VanillaSwap::arguments, VanillaSwap::results >](#), and [LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >](#).

9.680 OneAssetOption Class Reference

```
#include <ql/instruments/oneassetoption.hpp>
```

Inheritance diagram for OneAssetOption:



9.680.1 Detailed Description

Base class for options on a single asset.

Public Member Functions

- **OneAssetOption** (const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [Payoff](#) > &, const boost::shared_ptr< [Exercise](#) > &, const boost::shared_ptr< [PricingEngine](#) > & engine=boost::shared_ptr< [PricingEngine](#) >())
- Volatility [impliedVolatility](#) (Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility minVol=1.0e-7, Volatility maxVol=4.0) const
- void **setupArguments** ([PricingEngine::arguments](#) *) const
- void [fetchResults](#) (const [PricingEngine::results](#) *) const

Instrument interface

- bool [isExpired](#) () const
returns whether the instrument is still tradable.

greeks

- Real **delta** () const
- Real **deltaForward** () const
- Real **elasticity** () const
- Real **gamma** () const
- Real **theta** () const
- Real **thetaPerDay** () const
- Real **vega** () const
- Real **rho** () const
- Real **dividendRho** () const
- Real **itmCashProbability** () const

Protected Member Functions

- void [setupExpired](#) () const

Protected Attributes

- Real [delta](#)_
- Real [deltaForward](#)_
- Real [elasticity](#)_
- Real [gamma](#)_
- Real [theta](#)_
- Real [thetaPerDay](#)_
- Real [vega](#)_
- Real [rho](#)_
- Real [dividendRho](#)_
- Real [itmCashProbability](#)_
- boost::shared_ptr< [StochasticProcess](#) > [stochasticProcess](#)_

Classes

- class [arguments](#)
Arguments for single-asset option calculation
- class [results](#)
Results from single-asset option calculation

9.680.2 Member Function Documentation

- 9.680.2.1 **Volatility** [impliedVolatility](#) (Real *price*, Real *accuracy* = 1.0e-4, Size *maxEvaluations* = 100, Volatility *minVol* = 1.0e-7, Volatility *maxVol* = 4.0) const

Warning

currently, this method returns the Black-Scholes implied volatility. It will give inconsistent [results](#) if the pricing was performed with any other methods (such as jump-diffusion models.)

Warning

options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g binary options. In these cases the calculation can fail and the result (if any) is almost meaningless. Another possible source of failure is to have a target value that is not attainable with any volatility, e.g., a target value lower than the intrinsic value in the case of American options.

9.680.2.2 void fetchResults (const PricingEngine::results *) const [virtual]

When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

Reimplemented in [ForwardVanillaOption](#), [OneAssetStrikedOption](#), and [QuantoVanillaOption](#).

9.680.2.3 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 [OneAssetStrikedOption](#), and [QuantoVanillaOption](#).

9.681 OneAssetOption::arguments Class Reference

```
#include <ql/instruments/oneassetoption.hpp>
```

9.681.1 Detailed Description

Arguments for single-asset option calculation

Public Member Functions

- void **validate** () const

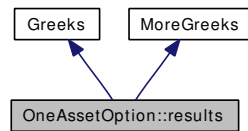
Public Attributes

- boost::shared_ptr< [StochasticProcess](#) > **stochasticProcess**

9.682 OneAssetOption::results Class Reference

```
#include <ql/instruments/oneassetoption.hpp>
```

Inheritance diagram for OneAssetOption::results:



9.682.1 Detailed Description

Results from single-asset option calculation

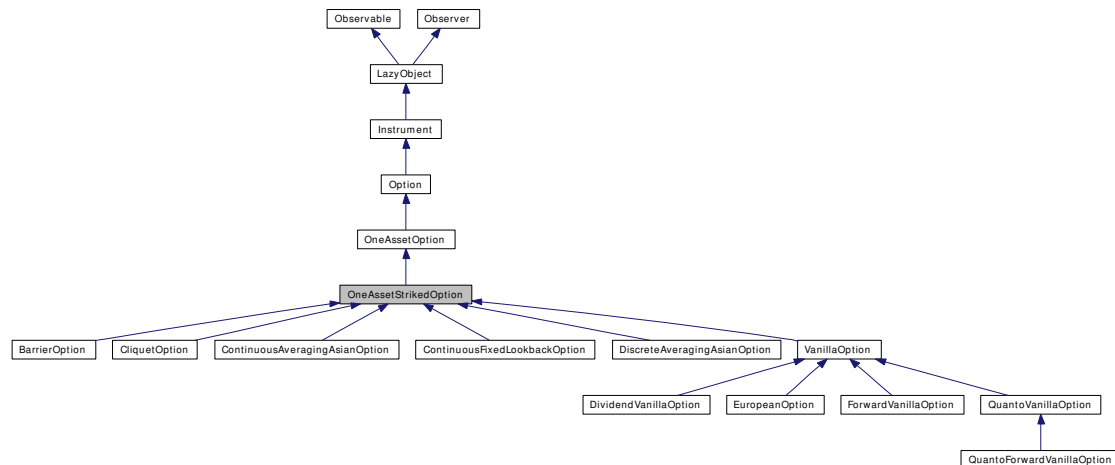
Public Member Functions

- void **reset** ()

9.683 OneAssetStrikedOption Class Reference

```
#include <ql/instruments/oneassetstrikedoption.hpp>
```

Inheritance diagram for OneAssetStrikedOption:



9.683.1 Detailed Description

Base class for options on a single asset with striked payoff.

Public Member Functions

- **OneAssetStrikedOption** (const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &, const boost::shared_ptr< [Exercise](#) > &, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void **setupArguments** ([PricingEngine::arguments](#) *) const
- void **fetchResults** (const [PricingEngine::results](#) *) const

greeks

- Real **strikeSensitivity** () const

Protected Member Functions

- void **setupExpired** () const

Protected Attributes

- Real **strikeSensitivity_**

9.683.2 Member Function Documentation

9.683.2.1 `void fetchResults (const PricingEngine::results *) const` [virtual]

When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetOption](#).

Reimplemented in [ForwardVanillaOption](#), and [QuantoVanillaOption](#).

9.683.2.2 `void setupExpired () const` [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

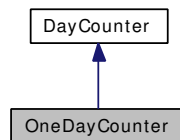
Reimplemented from [OneAssetOption](#).

Reimplemented in [QuantoVanillaOption](#).

9.684 OneDayCounter Class Reference

```
#include <ql/time/daycounters/one.hpp>
```

Inheritance diagram for OneDayCounter:



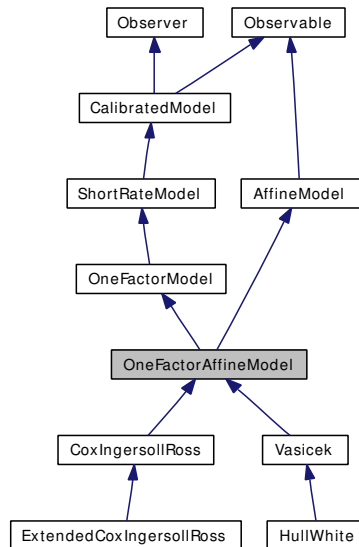
9.684.1 Detailed Description

1/1 day count convention

9.685 OneFactorAffineModel Class Reference

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

Inheritance diagram for OneFactorAffineModel:



9.685.1 Detailed Description

Single-factor affine base class.

Single-factor models with an analytical formula for discount bonds should inherit from this class. They must then implement the functions $A(t, T)$ and $B(t, T)$ such that

$$P(t, T, r_t) = A(t, T)e^{-B(t, T)r_t}.$$

Public Member Functions

- **OneFactorAffineModel** (Size nArguments)
- virtual Real **discountBond** (Time now, Time maturity, [Array](#) factors) const
- Real **discountBond** (Time now, Time maturity, Rate rate) const
- DiscountFactor [discount](#) (Time t) const

Implied discount curve.

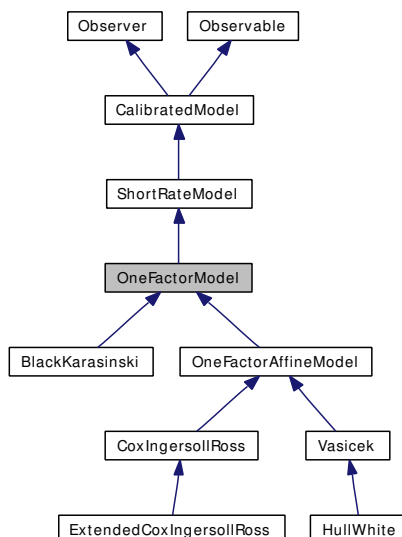
Protected Member Functions

- virtual Real **A** (Time t, Time T) const=0
- virtual Real **B** (Time t, Time T) const=0

9.686 OneFactorModel Class Reference

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

Inheritance diagram for OneFactorModel:



9.686.1 Detailed Description

Single-factor short-rate model abstract class.

Public Member Functions

- **OneFactorModel** (Size nArguments)
- virtual `boost::shared_ptr< ShortRateDynamics > dynamics () const=0`
returns the short-rate dynamics
- `boost::shared_ptr< Lattice > tree (const TimeGrid &grid) const`
Return by default a trinomial recombining tree.

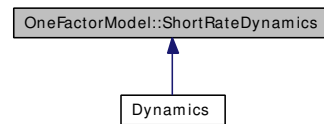
Classes

- class **ShortRateDynamics**
Base class describing the short-rate dynamics.
- class **ShortRateTree**
Recombining trinomial tree discretizing the state variable.

9.687 OneFactorModel::ShortRateDynamics Class Reference

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

Inheritance diagram for OneFactorModel::ShortRateDynamics:



9.687.1 Detailed Description

Base class describing the short-rate dynamics.

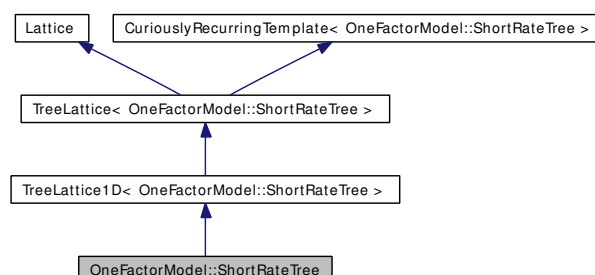
Public Member Functions

- **ShortRateDynamics** (const boost::shared_ptr< [StochasticProcess1D](#) > &process)
- virtual Real [variable](#) (Time t, Rate r) const=0
Compute state variable from short rate.
- virtual Rate [shortRate](#) (Time t, Real variable) const=0
Compute short rate from state variable.
- const boost::shared_ptr< [StochasticProcess1D](#) > & [process](#) ()
Returns the risk-neutral dynamics of the state variable.

9.688 OneFactorModel::ShortRateTree Class Reference

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

Inheritance diagram for OneFactorModel::ShortRateTree:



9.688.1 Detailed Description

Recombining trinomial tree discretizing the state variable.

Public Member Functions

- [ShortRateTree](#) (const boost::shared_ptr< [TrinomialTree](#) > &tree, const boost::shared_ptr< [ShortRateDynamics](#) > &dynamics, const [TimeGrid](#) &timeGrid)
Plain tree build-up from short-rate dynamics.
- [ShortRateTree](#) (const boost::shared_ptr< [TrinomialTree](#) > &tree, const boost::shared_ptr< [ShortRateDynamics](#) > &dynamics, const boost::shared_ptr< [TermStructureFittingParameter::NumericalImpl](#) > &phi, const [TimeGrid](#) &timeGrid)
Tree build-up + numerical fitting to term-structure.
- Size **size** (Size i) const
- DiscountFactor **discount** (Size i, Size index) const
- Real **underlying** (Size i, Size index) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

9.689 OperatorFactory Class Reference

```
#include <ql/methods/finitedifferences/operatorfactory.hpp>
```

9.689.1 Detailed Description

Black-Scholes-Merton differential operator.

Tests

coefficients are tested against constant BSM operator

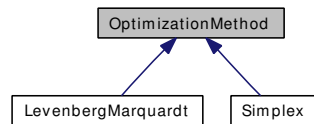
Static Public Member Functions

- static [TridiagonalOperator](#) **getOperator** (const boost::shared_ptr< [GeneralizedBlackScholesProcess](#) > &process, const [Array](#) &grid, Time residualTime, bool timeDependent)
- static [TridiagonalOperator](#) **getOperator** (const boost::shared_ptr< [OneFactorModel::ShortRateDynamics](#) > &process, const [Array](#) &grid)

9.690 OptimizationMethod Class Reference

```
#include <ql/math/optimization/method.hpp>
```

Inheritance diagram for OptimizationMethod:



9.690.1 Detailed Description

Abstract class for constrained optimization method.

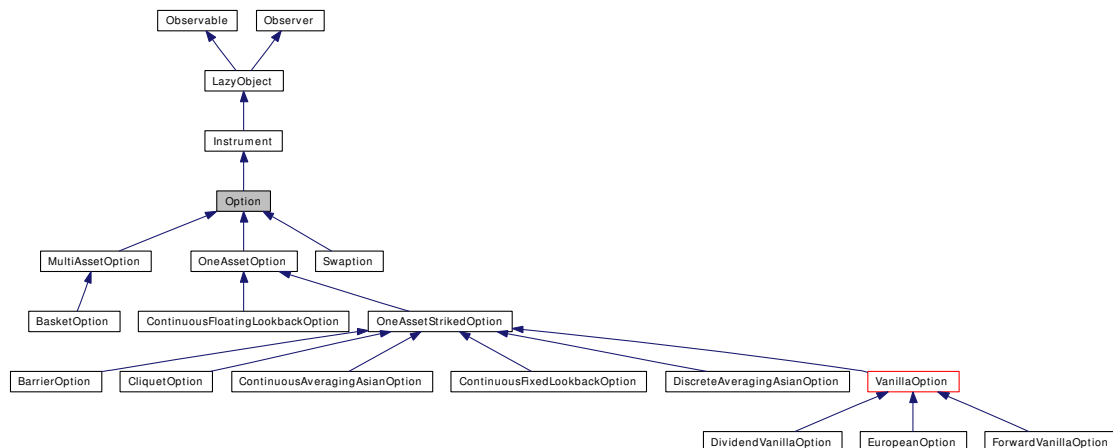
Public Member Functions

- virtual EndCriteria::Type [minimize](#) ([Problem](#) &P, const [EndCriteria](#) &endCriteria)=0
minimize the optimization problem P

9.691 Option Class Reference

```
#include <ql/option.hpp>
```

Inheritance diagram for Option:



9.691.1 Detailed Description

base option class

Public Types

- enum **Type** { Put = -1, Call = 1 }

Public Member Functions

- Option** (const boost::shared_ptr< [Payoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())

Protected Attributes

- boost::shared_ptr< [Payoff](#) > **payoff_**
- boost::shared_ptr< [Exercise](#) > **exercise_**

Related Functions

(Note that these are not member functions.)

- std::ostream & [operator<<](#) (std::ostream &, Option::Type)

Classes

- class [arguments](#)
basic option arguments

9.691.2 Friends And Related Function Documentation

9.691.2.1 `std::ostream & operator<< (std::ostream &, Option::Type)` [related]

9.692 Option::arguments Class Reference

```
#include <ql/option.hpp>
```

9.692.1 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

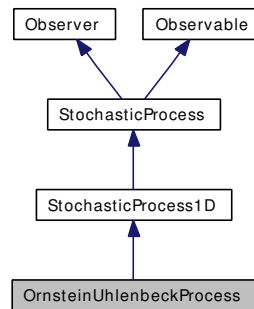
Public Attributes

- boost::shared_ptr< [Payoff](#) > **payoff**
- boost::shared_ptr< [Exercise](#) > **exercise**
- std::vector< Time > **stoppingTimes**

9.693 OrnsteinUhlenbeckProcess Class Reference

```
#include <ql/processes/ornsteinuhlenbeckprocess.hpp>
```

Inheritance diagram for OrnsteinUhlenbeckProcess:



9.693.1 Detailed Description

Ornstein-Uhlenbeck process class.

This class describes the Ornstein-Uhlenbeck process governed by

$$dx = a(r - x_t)dt + \sigma dW_t.$$

Public Member Functions

- **OrnsteinUhlenbeckProcess** (Real speed, Volatility vol, Real x0=0.0, Real level=0.0)

StochasticProcess interface

- Real **x0** () const
returns the initial value of the state variable
- Real **speed** () const
- Real **volatility** () const
- Real **level** () const
- Real **drift** (Time t, Real x) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- Real **diffusion** (Time t, Real x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- Real **expectation** (Time t0, Real x0, Time dt) const
- Real **stdDeviation** (Time t0, Real x0, Time dt) const
- Real **variance** (Time t0, Real x0, Time dt) const

9.693.2 Member Function Documentation

9.693.2.1 Real expectation (Time t0, Real x0, Time dt) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code

a particular discretization.

Reimplemented from [StochasticProcess1D](#).

9.693.2.2 Real stdDeviation (Time t_0 , Real x_0 , Time dt) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

9.693.2.3 Real variance (Time t_0 , Real x_0 , Time dt) const [virtual]

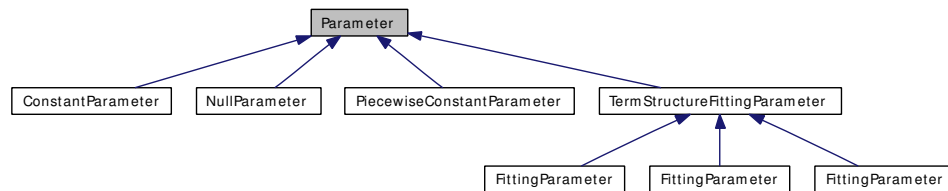
returns the variance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

9.694 Parameter Class Reference

```
#include <ql/models/parameter.hpp>
```

Inheritance diagram for Parameter:



9.694.1 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< [Impl](#) > & **implementation** () const

Protected Member Functions

- **Parameter** (Size size, const boost::shared_ptr< [Impl](#) > &impl, const [Constraint](#) &constraint)

Protected Attributes

- boost::shared_ptr< [Impl](#) > **impl_**
- [Array](#) **params_**
- [Constraint](#) **constraint_**

Classes

- class [Impl](#)
Base class for model parameter implementation.

9.695 Parameter::Impl Class Reference

```
#include <ql/models/parameter.hpp>
```

9.695.1 Detailed Description

Base class for model parameter implementation.

Public Member Functions

- virtual Real **value** (const [Array](#) ¶ms, Time t) const=0

9.696 Path Class Reference

```
#include <ql/methods/montecarlo/path.hpp>
```

9.696.1 Detailed Description

single-factor random walk

Note:

the path includes the initial asset value as its first point.

Examples:

[DiscreteHedging.cpp](#).

iterators

- typedef Array::const_iterator **iterator**
- typedef Array::const_reverse_iterator **reverse_iterator**
- iterator **begin** () const
- iterator **end** () const
- reverse_iterator **rbegin** () const
- reverse_iterator **rend** () const

Public Member Functions

- Path (const [TimeGrid](#) &timeGrid, const [Array](#) &values=[Array](#)())

inspectors

- bool **empty** () const
- Size **length** () const
- Real **operator[]** (Size i) const
asset value at the i-th point
- Real **at** (Size i) const
- Real & **operator[]** (Size i)
- Real & **at** (Size i)
- Real **value** (Size i) const
- Real & **value** (Size i)
- Time **time** (Size i) const
time at the i-th point
- Real **front** () const
initial asset value
- Real & **front** ()
- Real **back** () const
final asset value
- Real & **back** ()
- const [TimeGrid](#) & **timeGrid** () const
time grid

9.697 PathGenerator Class Template Reference

```
#include <ql/methods/montecarlo/pathgenerator.hpp>
```

9.697.1 Detailed Description

template<class GSG> class QuantLib::PathGenerator< GSG >

Generates random paths using a sequence generator.

Generates random paths with drift(S,t) and variance(S,t) using a gaussian sequence generator

Tests

the generated paths are checked against cached results

Public Types

- typedef [Sample< Path >](#) **sample_type**

Public Member Functions

- **PathGenerator** (const boost::shared_ptr< [StochasticProcess](#) > &, Time length, Size timeSteps, const GSG &generator, bool brownianBridge)
- **PathGenerator** (const boost::shared_ptr< [StochasticProcess](#) > &, const [TimeGrid](#) &timeGrid, const GSG &generator, bool brownianBridge)

inspectors

- const [sample_type](#) & **next** () const
- const [sample_type](#) & **antithetic** () const
- Size **size** () const
- const [TimeGrid](#) & **timeGrid** () const

9.698 PathPricer Class Template Reference

```
#include <ql/methods/montecarlo/pathpricer.hpp>
```

9.698.1 Detailed Description

```
template<class PathType, class ValueType = Real> class QuantLib::PathPricer< PathType,  
ValueType >
```

base class for path pricers

Returns the value of an option on a given path.

Examples:

[DiscreteHedging.cpp](#).

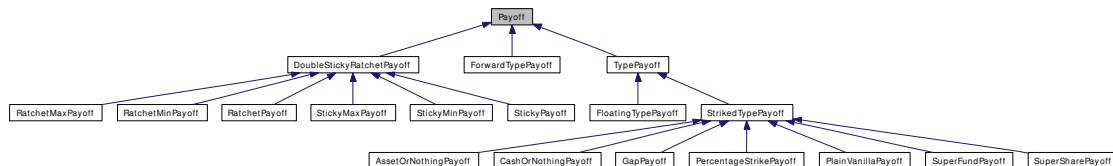
Public Member Functions

- virtual ValueType **operator()** (const PathType &path) const =0

9.699 Payoff Class Reference

```
#include <ql/payoff.hpp>
```

Inheritance diagram for Payoff:



9.699.1 Detailed Description

Abstract base class for option payoffs.

Public Member Functions

Payoff interface

- virtual std::string **name** () const=0
- virtual std::string **description** () const=0
- virtual Real **operator()** (Real price) const=0

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

9.699.2 Member Function Documentation

9.699.2.1 virtual std::string name () const [pure virtual]

Warning

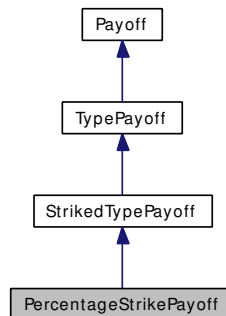
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implemented in [ForwardTypePayoff](#), [FloatingTypePayoff](#), [PlainVanillaPayoff](#), [PercentageStrikePayoff](#), [AssetOrNothingPayoff](#), [CashOrNothingPayoff](#), [GapPayoff](#), [SuperFundPayoff](#), [SuperSharePayoff](#), [DoubleStickyRatchetPayoff](#), [RatchetPayoff](#), [StickyPayoff](#), [RatchetMaxPayoff](#), [RatchetMinPayoff](#), [StickyMaxPayoff](#), and [StickyMinPayoff](#).

9.700 PercentageStrikePayoff Class Reference

```
#include <ql/instruments/payoffs.hpp>
```

Inheritance diagram for PercentageStrikePayoff:



9.700.1 Detailed Description

Payoff with strike expressed as percentage

Public Member Functions

- **PercentageStrikePayoff** (Option::Type type, Real moneyiness)

Payoff interface

- std::string **name** () const
- Real **operator()** (Real price) const
- virtual void **accept** (AcyclicVisitor &)

9.700.2 Member Function Documentation

9.700.2.1 std::string name () const [virtual]

Warning

This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implements [Payoff](#).

9.701 Period Class Reference

```
#include <ql/time/period.hpp>
```

9.701.1 Detailed Description

Time period described by a number of a given time unit.

Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

Public Member Functions

- **Period** (Integer n, [TimeUnit](#) units)
- **Period** ([Frequency](#) f)
- Integer **length** () const
- [TimeUnit](#) **units** () const
- [Frequency](#) **frequency** () const

Related Functions

(Note that these are not member functions.)

- template<typename T>
[Period operator *](#) (T n, [TimeUnit](#) units)
- template<typename T>
[Period operator *](#) ([TimeUnit](#) units, T n)
- [Period operator-](#) (const [Period](#) &)
- [Period operator *](#) (Integer n, const [Period](#) &)
- [Period operator *](#) (const [Period](#) &, Integer n)
- bool [operator<](#) (const [Period](#) &, const [Period](#) &)
- bool [operator==](#) (const [Period](#) &, const [Period](#) &)
- bool [operator!=](#) (const [Period](#) &, const [Period](#) &)
- bool [operator>](#) (const [Period](#) &, const [Period](#) &)
- bool [operator<=](#) (const [Period](#) &, const [Period](#) &)
- bool [operator>=](#) (const [Period](#) &, const [Period](#) &)
- std::ostream & [operator<<](#) (std::ostream &, const [Period](#) &)

9.701.2 Friends And Related Function Documentation

9.701.2.1 Period operator * (T *n*, TimeUnit *units*) [related]

9.701.2.2 Period operator * (TimeUnit *units*, T *n*) [related]

9.701.2.3 Period operator- (const Period &) [related]

9.701.2.4 Period operator * (Integer *n*, const Period &) [related]

9.701.2.5 Period operator * (const Period &, Integer *n*) [related]

9.701.2.6 bool operator< (const Period &, const Period &) [related]

9.701.2.7 bool operator== (const Period &, const Period &) [related]

9.701.2.8 bool operator!= (const Period &, const Period &) [related]

9.701.2.9 bool operator> (const Period &, const Period &) [related]

9.701.2.10 bool operator<= (const Period &, const Period &) [related]

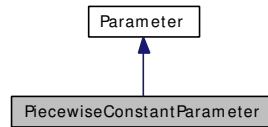
9.701.2.11 bool operator>= (const Period &, const Period &) [related]

9.701.2.12 std::ostream & operator<< (std::ostream &, const Period &) [related]

9.702 PiecewiseConstantParameter Class Reference

```
#include <ql/models/parameter.hpp>
```

Inheritance diagram for PiecewiseConstantParameter:



9.702.1 Detailed Description

Piecewise-constant parameter.

$a(t) = a_i$ if $t_{i-1} \leq t < t_i$. This kind of parameter is usually used to enhance the fitting of a model

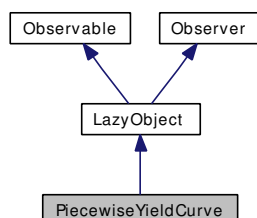
Public Member Functions

- **PiecewiseConstantParameter** (const std::vector< Time > ×)

9.703 PiecewiseYieldCurve Class Template Reference

```
#include <ql/termstructures/yieldcurves/piecewiseyieldcurve.hpp>
```

Inheritance diagram for PiecewiseYieldCurve:



9.703.1 Detailed Description

```
template<class Traits, class Interpolator> class QuantLib::PiecewiseYieldCurve< Traits, Interpolator >
```

Piecewise yield term structure.

This term structure is bootstrapped on a number of interest rate instruments which are passed as a vector of handles to [RateHelper](#) instances. Their maturities mark the boundaries of the interpolated segments.

Each segment is determined sequentially starting from the earliest period to the latest and is chosen so that the instrument whose maturity marks the end of such segment is correctly repriced on the curve.

Warning

The bootstrapping algorithm will raise an exception if any two instruments have the same maturity date.

Tests

- the correctness of the returned values is tested by checking them against the original inputs.
- the observability of the term structure is tested.

Examples:

[FRA.cpp](#), and [swapvaluation.cpp](#).

Public Member Functions

Constructors

- **PiecewiseYieldCurve** (const [Date](#) &referenceDate, const std::vector< boost::shared_ptr< [RateHelper](#) > > &instruments, const [DayCounter](#) &dayCounter, Real accuracy=1.0e-12, const Interpolator &i=Interpolator())

- **PiecewiseYieldCurve** (Natural settlementDays, const [Calendar](#) &calendar, const std::vector< boost::shared_ptr< [RateHelper](#) > > &instruments, const [DayCounter](#) &day-Counter, Real accuracy=1.0e-12, const Interpolator &i=Interpolator())

YieldTermStructure interface

- const std::vector< [Date](#) > & **dates** () const
- [Date](#) **maxDate** () const
- const std::vector< Time > & **times** () const

Inspectors

- std::vector< std::pair< [Date](#), Real > > **nodes** () const

Observer interface

- void [update](#) ()

Friends

- class **ObjectiveFunction**

9.703.2 Member Function Documentation

9.703.2.1 void update () [virtual]

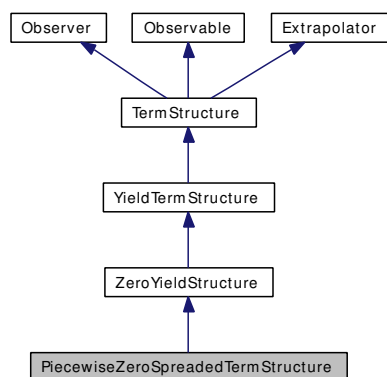
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [LazyObject](#).

9.704 PiecewiseZeroSpreadedTermStructure Class Reference

```
#include <ql/termstructures/yieldcurves/piecewisezerospreadedtermstructure.hpp>
```

Inheritance diagram for PiecewiseZeroSpreadedTermStructure:



9.704.1 Detailed Description

Term structure with an added vector of spreads on the zero-yield rate.

The zero-yield spread at any given date is linearly interpolated between the input data.

Note:

This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Public Member Functions

- **PiecewiseZeroSpreadedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const std::vector< [Handle](#)< [Quote](#) > &spreads, const std::vector< [Date](#) > &dates)

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- const [Date](#) & [referenceDate](#) () const
the date at which discount = 1.0 and/or variance = 0.0
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return values

Protected Member Functions

- Rate [zeroYieldImpl](#) (Time) const
returns the spreaded zero yield rate
- void [update](#) ()

9.704.2 Member Function Documentation

9.704.2.1 void [update](#) () [protected, virtual]

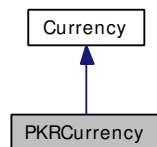
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

9.705 PKRCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for PKRCurrency:



9.705.1 Detailed Description

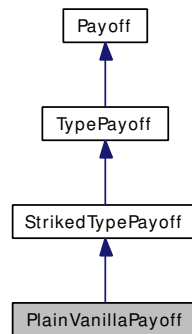
Pakistani rupee.

The ISO three-letter code is PKR; the numeric code is 586. It is divided in 100 paisa.

9.706 PlainVanillaPayoff Class Reference

```
#include <ql/instruments/payoffs.hpp>
```

Inheritance diagram for PlainVanillaPayoff:



9.706.1 Detailed Description

Plain-vanilla payoff.

Examples:

[DiscreteHedging.cpp](#), [EquityOption.cpp](#), and [Replication.cpp](#).

Public Member Functions

- **PlainVanillaPayoff** (Option::Type type, Real strike)

Payoff interface

- std::string **name** () const
- Real **operator()** (Real price) const
- virtual void **accept** ([AcyclicVisitor](#) &)

9.706.2 Member Function Documentation

9.706.2.1 std::string name () const [virtual]

Warning

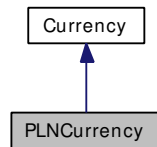
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implements [Payoff](#).

9.707 PLNCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for PLNCurrency:



9.707.1 Detailed Description

Polish zloty.

The ISO three-letter code is PLN; the numeric code is 985. It is divided in 100 groszy.

9.708 PoissonDistribution Class Reference

```
#include <ql/math/distributions/poissondistribution.hpp>
```

9.708.1 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.

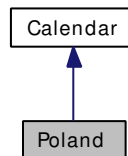
Public Member Functions

- **PoissonDistribution** (Real mu)
- Real **operator()** (BigNatural k) const

9.709 Poland Class Reference

```
#include <ql/time/calendars/poland.hpp>
```

Inheritance diagram for Poland:



9.709.1 Detailed Description

Polish calendar.

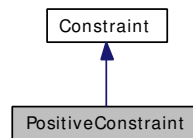
Holidays:

- Saturdays
- Sundays
- Easter Monday
- Corpus Christi
- New Year's Day, January 1st
- May Day, May 1st
- Constitution Day, May 3rd
- Assumption of the Blessed Virgin Mary, August 15th
- All Saints Day, November 1st
- Independence Day, November 11th
- Christmas, December 25th
- 2nd Day of Christmas, December 26th

9.710 PositiveConstraint Class Reference

```
#include <ql/math/optimization/constraint.hpp>
```

Inheritance diagram for PositiveConstraint:



9.710.1 Detailed Description

Constraint imposing positivity to all arguments

9.711 PricingEngine Class Reference

```
#include <ql/pricingengine.hpp>
```

Inheritance diagram for PricingEngine:



9.711.1 Detailed Description

interface for pricing engines

Public Member Functions

- virtual arguments * **getArguments** () const=0
- virtual const results * **getResults** () const=0
- virtual void **reset** ()=0
- virtual void **calculate** () const=0

9.712 PrimeNumbers Class Reference

```
#include <ql/math/primenumbers.hpp>
```

9.712.1 Detailed Description

Prime numbers calculator.

Taken from "Monte Carlo Methods in Finance", by Peter Jäckel

Static Public Member Functions

- static `BigNatural` [get](#) (Size `absoluteIndex`)
Get and store one after another.

9.713 Problem Class Reference

```
#include <ql/math/optimization/problem.hpp>
```

9.713.1 Detailed Description

Constrained optimization problem.

Public Member Functions

- **Problem** (**CostFunction** &costFunction, **Constraint** &constraint, const **Array** &initial-Value=**Array**())
default constructor
- void **reset** ()
- Real **value** (const **Array** &x)
call cost function computation and increment evaluation counter
- **Disposable**< **Array** > **values** (const **Array** &x)
call cost values computation and increment evaluation counter
- void **gradient** (**Array** &grad_f, const **Array** &x)
call cost function gradient computation and increment
- Real **valueAndGradient** (**Array** &grad_f, const **Array** &x)
call cost function computation and it gradient
- **Constraint** & **constraint** () const
Constraint.
- **CostFunction** & **costFunction** () const
Cost function.
- void **setCurrentValue** (const **Array** ¤tValue)
- const **Array** & **currentValue** () const
current value of the local minimum
- void **setFunctionValue** (Real functionValue)
- Real **functionValue** () const
value of cost function
- void **setGradientNormValue** (Real squaredNorm)
- Real **gradientNormValue** () const
value of cost function gradient norm
- Integer **functionEvaluation** () const
number of evaluation of cost function
- Integer **gradientEvaluation** () const
number of evaluation of cost function gradient

Protected Attributes

- [CostFunction](#) & [costFunction_](#)
Unconstrained cost function.
- [Constraint](#) & [constraint_](#)
Constraint.
- [Array](#) [currentValue_](#)
current value of the local minimum
- [Real](#) [functionValue_](#)
function and gradient norm values at the [currentValue_](#) (i.e. the last step)
- [Real](#) [squaredNorm_](#)
- [Integer](#) [functionEvaluation_](#)
number of evaluation of cost function and its gradient
- [Integer](#) [gradientEvaluation_](#)

9.713.2 Member Function Documentation

9.713.2.1 void reset ()

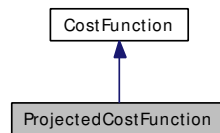
[Warning](#)

it does not reset the current minimum to any initial value

9.714 ProjectedCostFunction Class Reference

```
#include <ql/math/optimization/projectedcostfunction.hpp>
```

Inheritance diagram for ProjectedCostFunction:



9.714.1 Detailed Description

Parameterized cost function.

This class creates a proxy cost function which can depend on any arbitrary subset of parameters (the other being fixed)

Public Member Functions

- **ProjectedCostFunction** (const [CostFunction](#) &costFunction, const [Array](#) ¶metersValues, const std::vector< bool > ¶metersFreedom)
- virtual [Disposable](#)< [Array](#) > [project](#) (const [Array](#) ¶meters) const
returns the subset of free parameters corresponding
- virtual [Disposable](#)< [Array](#) > [include](#) (const [Array](#) &projectedParameters) const
returns whole set of parameters corresponding to the set

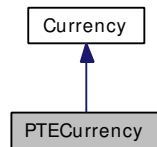
CostFunction interface

- virtual Real [value](#) (const [Array](#) &freeParameters) const
method to overload to compute the cost function value in x
- virtual [Disposable](#)< [Array](#) > [values](#) (const [Array](#) &freeParameters) const
method to overload to compute the cost function values in x

9.715 PTECurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for PTECurrency:



9.715.1 Detailed Description

Portuguese escudo.

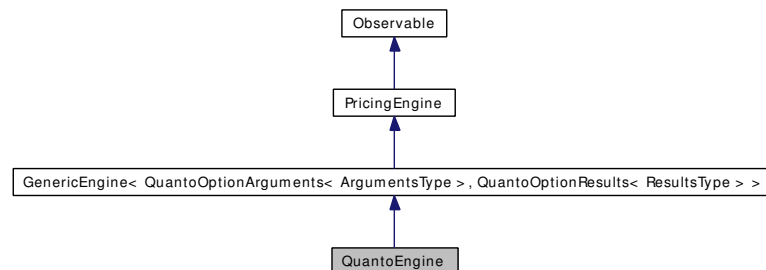
The ISO three-letter code was PTE; the numeric code was 620. It was divided in 100 centavos.

Obsoleted by the Euro since 1999.

9.716 QuantoEngine Class Template Reference

```
#include <ql/pricingengines/quanto/quantoengine.hpp>
```

Inheritance diagram for QuantoEngine:



9.716.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::QuantoEngine< ArgumentsType, ResultsType >
```

Quanto engine base class.

Warning

for the time being, this engine will only work with simple Black-Scholes processes (i.e., no Merton.)

Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **QuantoEngine** (const boost::shared_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > &)
- void **calculate** () const
- ArgumentsType * **underlyingArgs** () const

Protected Attributes

- boost::shared_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > > **originalEngine_**
- ArgumentsType * **originalArguments_**
- const ResultsType * **originalResults_**

9.716.2 Member Function Documentation

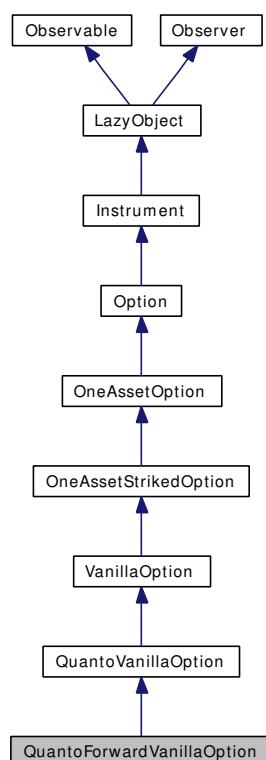
9.716.2.1 `ArgumentsType* underlyingArgs () const`

Access to the arguments of the underlying engine is needed as this engine is not able to set them completely. When necessary, it must be done by the instrument: see [QuantoForwardVanillaOption](#) for an example.

9.717 QuantoForwardVanillaOption Class Reference

```
#include <ql/instruments/quantoforwardvanillaoption.hpp>
```

Inheritance diagram for QuantoForwardVanillaOption:



9.717.1 Detailed Description

Quanto version of a forward vanilla option.

Public Types

- typedef [QuantoOptionArguments](#)< [ForwardVanillaOption::arguments](#) > **arguments**
- typedef [QuantoOptionResults](#)< [ForwardVanillaOption::results](#) > **results**
- typedef [QuantoEngine](#)< [ForwardVanillaOption::arguments](#), [ForwardVanillaOption::results](#) > **engine**

Public Member Functions

- **QuantoForwardVanillaOption** (const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &exchRateVolTS, const [Handle](#)< [Quote](#) > &correlation, Real moneyness, [Date](#) resetDate, const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &, const boost::shared_ptr< [Exercise](#) > &, const boost::shared_ptr< [PricingEngine](#) > &engine)
- void **setupArguments** ([PricingEngine::arguments](#) *) const

9.718 QuantoOptionArguments Class Template Reference

```
#include <ql/pricingengines/quanto/quantoengine.hpp>
```

9.718.1 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`

9.719 QuantoOptionResults Class Template Reference

```
#include <ql/pricingengines/quanto/quantoengine.hpp>
```

9.719.1 Detailed Description

```
template<class ResultsType> class QuantLib::QuantoOptionResults< ResultsType >
```

Results from quanto option calculation

Public Member Functions

- void reset ()

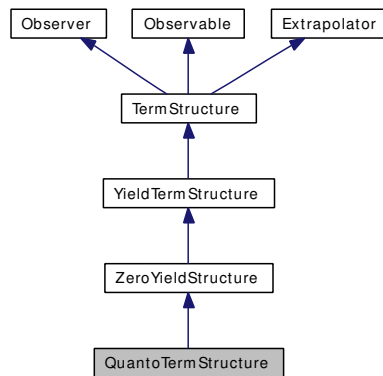
Public Attributes

- Real qvega
- Real qrho
- Real qlambda

9.720 QuantoTermStructure Class Reference

```
#include <ql/termstructures/yieldcurves/quantotermstructure.hpp>
```

Inheritance diagram for QuantoTermStructure:



9.720.1 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 exchRateATMlevel, Real underlyingExchRateCorrelation)

YieldTermStructure interface

- [DayCounter](#) **dayCounter** () const
the day counter used for date/time conversion
- [Calendar](#) **calendar** () const
the calendar used for reference date calculation
- const [Date](#) & **referenceDate** () const
the date at which discount = 1.0 and/or variance = 0.0
- [Date](#) **maxDate** () const
the latest date for which the curve can return values

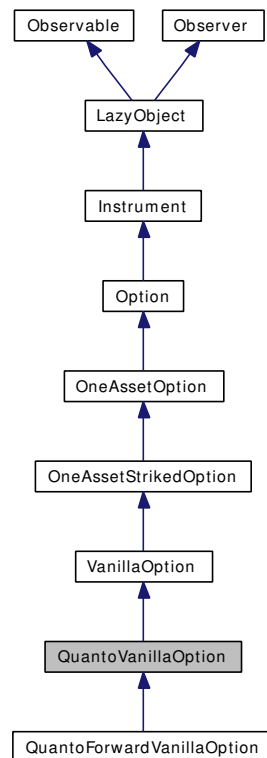
Protected Member Functions

- Rate [zeroYieldImpl](#) (Time) const
returns the zero yield as seen from the evaluation date

9.721 QuantoVanillaOption Class Reference

```
#include <ql/instruments/quantovanillaoption.hpp>
```

Inheritance diagram for QuantoVanillaOption:



9.721.1 Detailed Description

quanto version of a vanilla option

Public Types

- typedef [QuantoOptionArguments](#)< [VanillaOption::arguments](#) > **arguments**
- typedef [QuantoOptionResults](#)< [VanillaOption::results](#) > **results**
- typedef [QuantoEngine](#)< [VanillaOption::arguments](#), [VanillaOption::results](#) > **engine**

Public Member Functions

- **QuantoVanillaOption** (const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &exchRateVolTS, const [Handle](#)< [Quote](#) > &correlation, const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &, const boost::shared_ptr< [Exercise](#) > &, const boost::shared_ptr< [PricingEngine](#) > &)
- void **setupArguments** ([PricingEngine::arguments](#) *) const
- void **fetchResults** (const [PricingEngine::results](#) *) const

greeks

- Real **qvega** () const
- Real **qrho** () const
- Real **qlambda** () const

Protected Member Functions

- void **setupExpired** () const

Protected Attributes

- [Handle< YieldTermStructure >](#) **foreignRiskFreeTS_**
- [Handle< BlackVolTermStructure >](#) **exchRateVolTS_**
- [Handle< Quote >](#) **correlation_**
- Real **qvega_**
- Real **qrho_**
- Real **qlambda_**

9.721.2 Member Function Documentation

9.721.2.1 void fetchResults (const PricingEngine::results *) const [virtual]

When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetStrikedOption](#).

9.721.2.2 void setupExpired () const [protected, virtual]

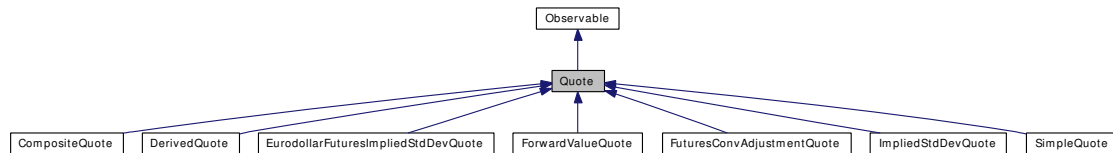
This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from [OneAssetStrikedOption](#).

9.722 Quote Class Reference

```
#include <ql/quote.hpp>
```

Inheritance diagram for Quote:



9.722.1 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

9.723 RandomizedLDS Class Template Reference

```
#include <ql/math/randomnumbers/randomizedlds.hpp>
```

9.723.1 Detailed Description

```
template<class LDS, class PRS = RandomSequenceGenerator<MersenneTwisterUniformRng>>
class QuantLib::RandomizedLDS< LDS, PRS >
```

Randomized (random shift) low-discrepancy sequence.

Random-shifts a uniform low-discrepancy sequence of dimension N by adding (modulo 1 for each coordinate) a pseudo-random uniform deviate in $(0, 1)^N$. It is used for implementing Randomized Quasi Monte Carlo.

The uniform low discrepancy sequence is supplied by LDS; the uniform pseudo-random sequence is supplied by PRS.

Both class LDS and PRS must implement the following interface:

```
LDS::sample_type LDS::nextSequence() const;
Size LDS::dimension() const;
```

Precondition:

LDS and PRS must have the same dimension N

Warning

Inverting LDS and PRS is possible, but it doesn't make sense.

Todo

implement the other randomization algorithms

Tests

correct initialization is tested.

Public Types

- typedef [Sample](#)< std::vector< Real > > **sample_type**

Public Member Functions

- **RandomizedLDS** (const LDS &lds, const PRS &prsg)
- **RandomizedLDS** (const LDS &lds)
- **RandomizedLDS** (Size dimensionality, BigNatural ldsSeed=0, BigNatural prsSeed=0)
- const [sample_type](#) & **nextSequence** () const
returns next sample using a given randomizing vector
- const [sample_type](#) & **lastSequence** () const
- void **nextRandomizer** ()
- Size **dimension** () const

9.723.2 Member Function Documentation

9.723.2.1 void nextRandomizer ()

update the randomizing vector and re-initialize the low discrepancy generator

9.724 RandomSequenceGenerator Class Template Reference

```
#include <ql/math/randomnumbers/randomsequencegenerator.hpp>
```

9.724.1 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](#)< std::vector< Real > > **sample_type**

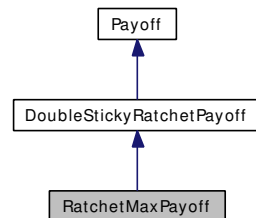
Public Member Functions

- **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

9.725 RatchetMaxPayoff Class Reference

```
#include <ql/instruments/sticky_ratchet.hpp>
```

Inheritance diagram for RatchetMaxPayoff:



9.725.1 Detailed Description

RatchetMax payoff (double option).

Public Member Functions

- **RatchetMaxPayoff** (Real gearing1, Real gearing2, Real gearing3, Real spread1, Real spread2, Real spread3, Real initialValue1, Real initialValue2, Real accrualFactor)

Payoff interface

- std::string [name](#) () const

9.725.2 Member Function Documentation

9.725.2.1 std::string name () const [virtual]

Warning

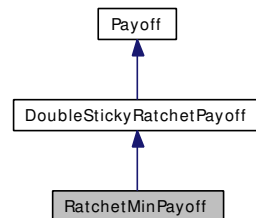
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Reimplemented from [DoubleStickyRatchetPayoff](#).

9.726 RatchetMinPayoff Class Reference

```
#include <ql/instruments/sticky_ratchet.hpp>
```

Inheritance diagram for RatchetMinPayoff:



9.726.1 Detailed Description

RatchetMin payoff (double option).

Public Member Functions

- **RatchetMinPayoff** (Real gearing1, Real gearing2, Real gearing3, Real spread1, Real spread2, Real spread3, Real initialValue1, Real initialValue2, Real accrualFactor)

Payoff interface

- `std::string name () const`

9.726.2 Member Function Documentation

9.726.2.1 `std::string name () const` [virtual]

Warning

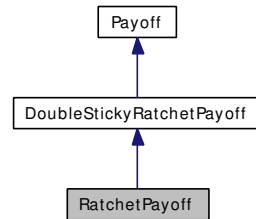
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Reimplemented from [DoubleStickyRatchetPayoff](#).

9.727 RatchetPayoff Class Reference

```
#include <ql/instruments/stickyratchet.hpp>
```

Inheritance diagram for RatchetPayoff:



9.727.1 Detailed Description

Ratchet payoff (single option).

Public Member Functions

- **RatchetPayoff** (Real gearing1, Real gearing2, Real spread1, Real spread2, Real initialValue, Real accrualFactor)

Payoff interface

- `std::string name () const`

9.727.2 Member Function Documentation

9.727.2.1 `std::string name () const` [virtual]

Warning

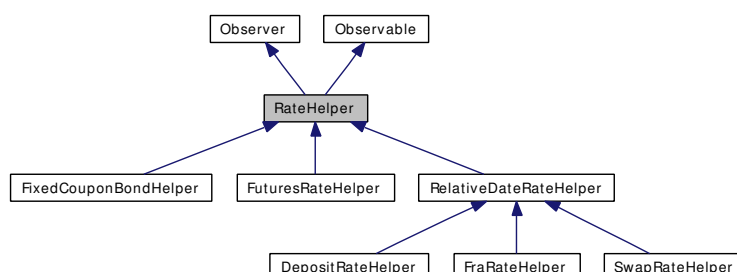
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Reimplemented from [DoubleStickyRatchetPayoff](#).

9.728 RateHelper Class Reference

```
#include <ql/termstructures/yieldcurves/piecewiseyieldcurve.hpp>
```

Inheritance diagram for RateHelper:



9.728.1 Detailed Description

Base helper class for yield-curve bootstrapping.

This class provides an abstraction for the instruments used to bootstrap a term structure. It is advised that a rate helper for an instrument contains an instance of the actual instrument class to ensure consistency between the algorithms used during bootstrapping and later instrument pricing. This is not yet fully enforced in the available rate helpers, though - only [SwapRateHelper](#) and [FixedCouponBondHelper](#) contain their corresponding instrument for the time being.

Public Member Functions

- **RateHelper** (const [Handle](#)< [Quote](#) > "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](#) **earliestDate** () const
earliest relevant date
- virtual [Date](#) **latestDate** () const
latest relevant date

Observer interface

- virtual void **update** ()

Protected Attributes

- [Handle< Quote >](#) quote_
- [YieldTermStructure](#) * termStructure_
- [Date](#) earliestDate_
- [Date](#) latestDate_

9.728.2 Member Function Documentation

9.728.2.1 virtual void setTermStructure (YieldTermStructure *) [virtual]

sets the term structure to be used for pricing

Warning

Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented in [FixedCouponBondHelper](#), [DepositRateHelper](#), [FraRateHelper](#), and [SwapRateHelper](#).

9.728.2.2 virtual Date earliestDate () const [virtual]

earliest relevant date

The earliest date at which discounts are needed by the helper in order to provide a quote.

9.728.2.3 virtual Date latestDate () const [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

9.728.2.4 virtual void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

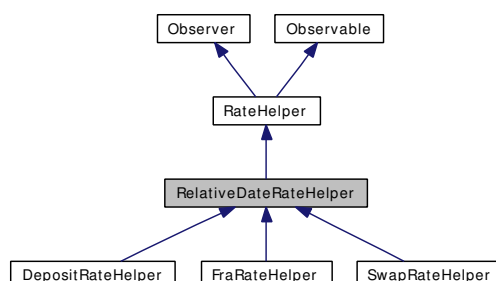
Implements [Observer](#).

Reimplemented in [RelativeDateRateHelper](#).

9.729 RelativeDateRateHelper Class Reference

```
#include <ql/termstructures/yieldcurves/ratehelpers.hpp>
```

Inheritance diagram for RelativeDateRateHelper:



9.729.1 Detailed Description

Rate helper with date schedule relative to the global evaluation date.

This class takes care of rebuilding the date schedule when the global evaluation date changes

Public Member Functions

- **RelativeDateRateHelper** (const [Handle](#)< [Quote](#) > "e)
- **RelativeDateRateHelper** (Real quote)
- void [update](#) ()

Protected Member Functions

- virtual void **initializeDates** ()=0

Protected Attributes

- [Date](#) **evaluationDate_**

9.729.2 Member Function Documentation

9.729.2.1 void update () [virtual]

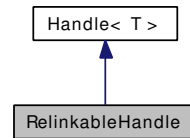
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [RateHelper](#).

9.730 RelinkableHandle Class Template Reference

```
#include <ql/handle.hpp>
```

Inheritance diagram for RelinkableHandle:



9.730.1 Detailed Description

```
template<class T> class QuantLib::RelinkableHandle< T >
```

Relinkable handle to an observable.

An instance of this class can be relinked so that it points to another observable. The change will be propagated to all handles that were created as copies of such instance.

Precondition:

Class T must inherit from [Observable](#)

Examples:

[FRA.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

Public Member Functions

- [RelinkableHandle](#) (const boost::shared_ptr< T > &h=boost::shared_ptr< T >(), bool registerAsObserver=true)
- void [linkTo](#) (const boost::shared_ptr< T > &, bool registerAsObserver=true)

9.730.2 Constructor & Destructor Documentation

9.730.2.1 [RelinkableHandle](#) (const boost::shared_ptr< T > &h = boost::shared_ptr< T >(), bool *registerAsObserver* = true) [explicit]

Warning

see the [Handle](#) documentation for issues relatives to registerAsObserver.

9.730.3 Member Function Documentation

9.730.3.1 void [linkTo](#) (const boost::shared_ptr< T > &, bool *registerAsObserver* = true)

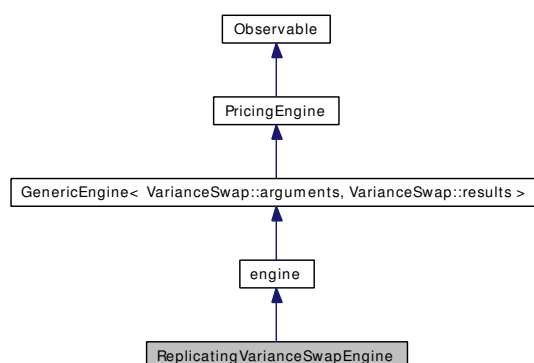
Warning

see the [Handle](#) documentation for issues relatives to registerAsObserver.

9.731 ReplicatingVarianceSwapEngine Class Reference

#include <ql/pricingengines/forward/replicatingvarianceswapengine.hpp>

Inheritance diagram for ReplicatingVarianceSwapEngine:



9.731.1 Detailed Description

Variance-swap pricing engine using replicating cost.

as described in Demeterfi, Derman, Kamal & Zou, "A Guide to Volatility and Variance Swaps", 1999

Tests

returned fair variances verified against results from literature

Public Types

- typedef std::vector< Real > **StrikesType**

Public Member Functions

- **ReplicatingVarianceSwapEngine** (const Real dk=5.0, const StrikesType &callStrikes=StrikesType(), const StrikesType &putStrikes=StrikesType(), const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void **calculate** () const

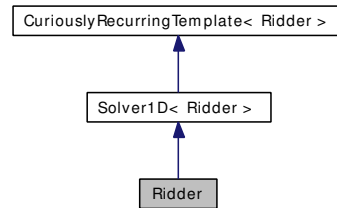
Protected Member Functions

- void **computeOptionWeights** (const StrikesType &, const Option::Type) const
- Real **computeLogPayoff** (const Real, const Real) const
- Real **computeReplicatingPortfolio** () const
- Rate **riskFreeRate** () const
- DiscountFactor **riskFreeDiscount** () const
- Real **underlying** () const
- Time **residualTime** () const

9.732 Ridder Class Reference

```
#include <ql/math/solvers1d/ridder.hpp>
```

Inheritance diagram for Ridder:



9.732.1 Detailed Description

Ridder 1-D solver

Tests

the correctness of the returned values is tested by checking them against known good results.

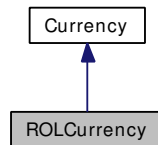
Public Member Functions

- `template<class F>`
`Real solveImpl (const F &f, Real xAcc) const`

9.733 ROLCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for ROLCurrency:



9.733.1 Detailed Description

Romanian leu.

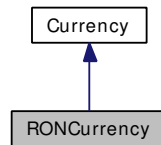
The ISO three-letter code was ROL; the numeric code was 642. It was divided in 100 bani.

Obsoleted by the new leu since July 2005.

9.734 RONCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for RONCurrency:



9.734.1 Detailed Description

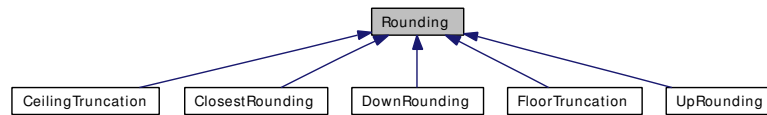
Romanian new leu.

The ISO three-letter code is RON; the numeric code is 946. It is divided in 100 bani.

9.735 Rounding Class Reference

```
#include <ql/math/rounding.hpp>
```

Inheritance diagram for Rounding:



9.735.1 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

9.735.2 Member Enumeration Documentation

9.735.2.1 enum Type

rounding methods

The rounding methods follow the OMG specification available at <ftp://ftp.omg.org/pub/docs/formal/00-06-29.pdf>

Warning

the names of the [Floor](#) and Ceiling methods might be misleading. Check the provided reference.

Enumerator:

None do not round: return the number unmodified

Up the first decimal place past the precision will be rounded up. This differs from the OMG rule which rounds up only if the decimal to be rounded is greater than or equal to the rounding digit

Down all decimal places past the precision will be truncated

Closest the first decimal place past the precision will be rounded up if greater than or equal to the rounding digit; this corresponds to the OMG round-up rule. When the rounding digit is 5, the result will be the one closest to the original number, hence the name.

Floor positive numbers will be rounded up and negative numbers will be rounded down using the OMG round up and round down rules

Ceiling positive numbers will be rounded down and negative numbers will be rounded up using the OMG round up and round down rules

9.735.3 Constructor & Destructor Documentation

9.735.3.1 Rounding ()

default constructor

Instances built through this constructor don't perform any rounding.

9.736 SABR Class Reference

```
#include <ql/math/interpolations/sabrinterpolation.hpp>
```

9.736.1 Detailed Description

SABR interpolation factory

Public Types

- enum { **global** = 1 }

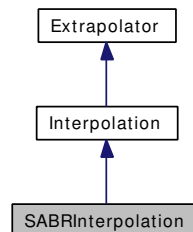
Public Member Functions

- **SABR** (Time t, Real forward, Real alpha, Real beta, Real nu, Real rho, bool alphaIsFixed, bool betaIsFixed, bool nuIsFixed, bool rhoIsFixed, bool vegaWeighted=false, const boost::shared_ptr< [EndCriteria](#) > endCriteria=boost::shared_ptr< [EndCriteria](#) >(), const boost::shared_ptr< [OptimizationMethod](#) > method=boost::shared_ptr< [OptimizationMethod](#) >())
- template<class I1, class I2>
[Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

9.737 SABRInterpolation Class Reference

```
#include <ql/math/interpolations/sabrinterpolation.hpp>
```

Inheritance diagram for SABRInterpolation:



9.737.1 Detailed Description

SABR smile interpolation between discrete volatility points.

Public Member Functions

- `template<class I1, class I2>`
SABRInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, Time t, const Real &forward, Real alpha, Real beta, Real nu, Real rho, bool alphaIsFixed, bool betaIsFixed, bool nuIsFixed, bool rhoIsFixed, bool vegaWeighted=false, const boost::shared_ptr< [EndCriteria](#) > &endCriteria=boost::shared_ptr< [EndCriteria](#) >(), const boost::shared_ptr< [OptimizationMethod](#) > &method=boost::shared_ptr< [OptimizationMethod](#) >())
- Real **expiry** () const
- Real **forward** () const
- Real **alpha** () const
- Real **beta** () const
- Real **nu** () const
- Real **rho** () const
- Real **interpolationError** () const
- Real **interpolationMaxError** () const
- EndCriteria::Type **endCriteria** ()

9.738 SalvagingAlgorithm Struct Reference

```
#include <ql/math/matrixutilities/pseudosqrt.hpp>
```

9.738.1 Detailed Description

algorithm used for matricial pseudo square root

Public Types

- enum Type {
 None, Spectral, Hypersphere, LowerDiagonal,
 Higham }

9.739 Sample Struct Template Reference

```
#include <ql/methods/montecarlo/sample.hpp>
```

9.739.1 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**

9.740 SampledCurve Class Reference

```
#include <ql/math/sampledcurve.hpp>
```

9.740.1 Detailed Description

This class contains a sampled curve.

Initially the class will contain one indexed curve

Public Member Functions

- **SampledCurve** (Size gridSize=0)
- **SampledCurve** (const [Array](#) &grid)
- **SampledCurve** & **operator=** (const [SampledCurve](#) &)

inspectors

- const [Array](#) & **grid** () const
- [Array](#) & **grid** ()
- const [Array](#) & **values** () const
- [Array](#) & **values** ()
- Real **gridValue** (Size i) const
- Real & **gridValue** (Size i)
- Real **value** (Size i) const
- Real & **value** (Size i)
- Size **size** () const
- bool **empty** () const

modifiers

- void **setGrid** (const [Array](#) &)
- void **setValues** (const [Array](#) &)
- template<class F>
void **sample** (const F &f)

calculations

- Real **valueAtCenter** () const
- Real **firstDerivativeAtCenter** () const
- Real **secondDerivativeAtCenter** () const

utilities

- void **swap** ([SampledCurve](#) &)
- void **setLogGrid** (Real min, Real max)
- void **regridLogGrid** (Real min, Real max)
- void **shiftGrid** (Real s)
- void **scaleGrid** (Real s)
- void **regrid** (const [Array](#) &new_grid)
- template<class T>
void **regrid** (const [Array](#) &new_grid, T func)
- template<class T>
const [SampledCurve](#) & **transform** (T x)
- template<class T>
const [SampledCurve](#) & **transformGrid** (T x)

9.740.2 Member Function Documentation

9.740.2.1 Real valueAtCenter () const

Todo

replace or complement with a more general function valueAt(spot)

9.740.2.2 Real firstDerivativeAtCenter () const

Todo

replace or complement with a more general function firstDerivativeAt(spot)

9.740.2.3 Real secondDerivativeAtCenter () const

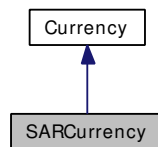
Todo

replace or complement with a more general function secondDerivativeAt(spot)

9.741 SARCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for SARCurrency:



9.741.1 Detailed Description

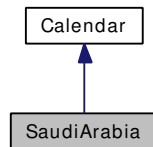
Saudi riyal.

The ISO three-letter code is SAR; the numeric code is 682. It is divided in 100 halalat.

9.742 SaudiArabia Class Reference

```
#include <ql/time/calendars/saudiarabia.hpp>
```

Inheritance diagram for SaudiArabia:



9.742.1 Detailed Description

Saudi Arabian calendar.

Holidays for the Tadawul financial market (data from <http://www.tadawul.com.sa>):

- Thursdays
- Fridays
- National Day of Saudi Arabia, September 23rd

Other holidays for which no rule is given (data available for 2004-2005 only:)

- Eid Al-Adha
- Eid Al-Fitr

Public Types

- enum `Market` { `Tadawul` }

Public Member Functions

- `SaudiArabia` (`Market` m=`Tadawul`)

9.742.2 Member Enumeration Documentation

9.742.2.1 enum `Market`

Enumerator:

Tadawul Tadawul financial market.

9.743 Schedule Class Reference

```
#include <ql/time/schedule.hpp>
```

9.743.1 Detailed Description

Payment schedule.

Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

Iterators

- typedef std::vector< [Date](#) >::const_iterator **const_iterator**
- const_iterator **begin** () const
- const_iterator **end** () const

Public Member Functions

- **Schedule** (const std::vector< [Date](#) > &, const [Calendar](#) &calendar=[NullCalendar](#)(), [BusinessDayConvention](#) convention=[Unadjusted](#))
- **Schedule** (const [Date](#) &effectiveDate, const [Date](#) &terminationDate, const [Period](#) &tenor, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, [BusinessDayConvention](#) terminationDateConvention, bool backward, bool endOfMonth, const [Date](#) &firstDate=[Date](#)(), const [Date](#) &nextToLastDate=[Date](#)())

Date access

- Size **size** () const
- const [Date](#) & **operator[]** (Size i) const
- const [Date](#) & **at** (Size i) const
- const [Date](#) & **date** (Size i) const
- const std::vector< [Date](#) > & **dates** () const
- bool **isRegular** (Size i) const

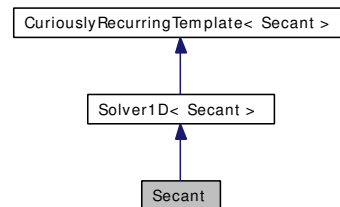
Other inspectors

- bool **empty** () const
- const [Calendar](#) & **calendar** () const
- const [Date](#) & **startDate** () const
- const [Date](#) & **endDate** () const
- const [Period](#) & **tenor** () const
- [BusinessDayConvention](#) **businessDayConvention** () const

9.744 Secant Class Reference

```
#include <ql/math/solvers1d/secant.hpp>
```

Inheritance diagram for Secant:



9.744.1 Detailed Description

Secant 1-D solver

Tests

the correctness of the returned values is tested by checking them against known good results.

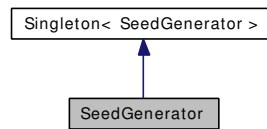
Public Member Functions

- `template<class F>`
`Real solveImpl (const F &f, Real xAccuracy) const`

9.745 SeedGenerator Class Reference

```
#include <ql/math/randomnumbers/seedgenerator.hpp>
```

Inheritance diagram for SeedGenerator:



9.745.1 Detailed Description

Random seed generator.

Random number generator used for automatic generation of initialization seeds.

Tests

correct initialization of the single instance is tested.

Public Member Functions

- unsigned long `get ()`

Friends

- class `Singleton< SeedGenerator >`

9.746 SegmentIntegral Class Reference

```
#include <ql/math/integrals/segmentintegral.hpp>
```

9.746.1 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)

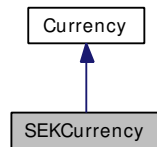
Protected Member Functions

- virtual Real **integrate** (const boost::function< Real(Real)> &f, Real a, Real b) const

9.747 SEKCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for SEKCurrency:



9.747.1 Detailed Description

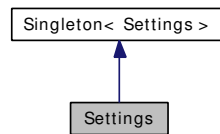
Swedish krona.

The ISO three-letter code is SEK; the numeric code is 752. It is divided in 100 öre.

9.748 Settings Class Reference

```
#include <ql/settings.hpp>
```

Inheritance diagram for Settings:



9.748.1 Detailed Description

global repository for run-time library settings

Public Member Functions

- DateProxy & [evaluationDate](#) ()
the date at which pricing is to be performed.
- const DateProxy & [evaluationDate](#) () const
- bool [enforceTodaysHistoricFixings](#) () const
- void [setEnforceTodaysHistoricFixings](#) (bool b=true)

Friends

- class [Singleton< Settings >](#)
- std::ostream & [operator<<](#) (std::ostream &, const DateProxy &)

9.748.2 Member Function Documentation

9.748.2.1 Settings::DateProxy & evaluationDate ()

the date at which pricing is to be performed.

Client code can inspect the evaluation date, as in:

```
Date d = Settings::instance().evaluationDate();
```

where today's date is returned if the evaluation date is set to the null date (its default value;) can set it to a new value, as in:

```
Settings::instance().evaluationDate() = d;
```

and can register with it, as in:

```
registerWith(Settings::instance().evaluationDate());
```

to be notified when it is set to a new value.

Warning

a notification is not sent when the evaluation date changes for natural causes—i.e., a date was not explicitly set (which results in today's date being used for pricing) and the current date changes as the clock strikes midnight.

9.749 Settlement Struct Reference

```
#include <ql/instruments/swaption.hpp>
```

9.749.1 Detailed Description

settlement information

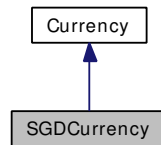
Public Types

- enum Type { Physical, Cash }

9.750 SGDCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for SGDCurrency:



9.750.1 Detailed Description

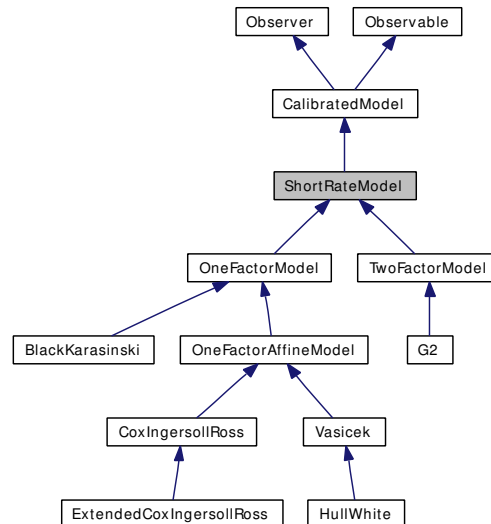
Singapore dollar

The ISO three-letter code is SGD; the numeric code is 702. It is divided in 100 cents.

9.751 ShortRateModel Class Reference

```
#include <ql/models/model.hpp>
```

Inheritance diagram for ShortRateModel:



9.751.1 Detailed Description

Abstract short-rate model class.

Public Member Functions

- **ShortRateModel** (Size nArguments)
- virtual boost::shared_ptr< [Lattice](#) > **tree** (const [TimeGrid](#) &) const=0

9.752 ShoutCondition Class Reference

```
#include <ql/methods/finitedifferences/shoutcondition.hpp>
```

9.752.1 Detailed Description

Shout option condition.

A shout option is an option where the holder has the right to lock in a minimum value for the payoff at one (shout) time during the option's life. The minimum value is the option's intrinsic value at the shout time.

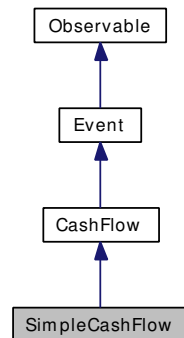
Public Member Functions

- **ShoutCondition** (Option::Type type, Real strike, Time resTime, Rate rate)
- **ShoutCondition** (const [Array](#) &intrinsicValues, Time resTime, Rate rate)
- void **applyTo** ([Array](#) &a, Time t) const

9.753 SimpleCashFlow Class Reference

```
#include <ql/cashflows/simplecashflow.hpp>
```

Inheritance diagram for SimpleCashFlow:



9.753.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

Public Member Functions

- **SimpleCashFlow** (Real amount, const [Date](#) &date)

CashFlow interface

- Real [amount](#) () const
returns the amount of the cash flow
- [Date](#) [date](#) () const
Note:
This is inherited from the event class

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

9.753.2 Member Function Documentation

9.753.2.1 Real amount () const [virtual]

returns the amount of the cash flow

Note:

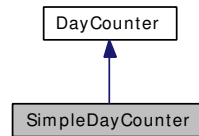
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

9.754 SimpleDayCounter Class Reference

```
#include <ql/time/daycounters/simpliedaycounter.hpp>
```

Inheritance diagram for SimpleDayCounter:



9.754.1 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.

9.755 SimpleLocalEstimator Class Reference

```
#include <ql/models/volatility/simplelocalestimator.hpp>
```

9.755.1 Detailed Description

Local-estimator volatility model.

Volatilities are assumed to be expressed on an annual basis.

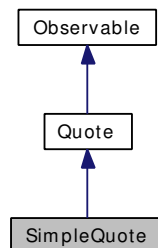
Public Member Functions

- **SimpleLocalEstimator** (Real y)
- [TimeSeries](#)< Volatility > **calculate** (const [TimeSeries](#)< Real > "eSeries)

9.756 SimpleQuote Class Reference

```
#include <ql/quotes/simplequote.hpp>
```

Inheritance diagram for SimpleQuote:



9.756.1 Detailed Description

market element returning a stored value

Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), [FRA.cpp](#), [Replication.cpp](#), and [swapvaluation.cpp](#).

Public Member Functions

- **SimpleQuote** (Real value=[Null](#)< Real >())

Quote interface

- Real [value](#) () const
returns the current value

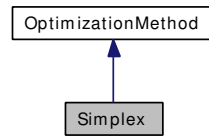
Modifiers

- Real [setValue](#) (Real value)
returns the difference between the new value and the old value

9.757 Simplex Class Reference

```
#include <ql/math/optimization/simplex.hpp>
```

Inheritance diagram for Simplex:



9.757.1 Detailed Description

Multi-dimensional simplex class.

Public Member Functions

- [Simplex](#) (Real lambda)
- virtual EndCriteria::Type [minimize](#) ([Problem](#) &P, const [EndCriteria](#) &endCriteria)
minimize the optimization problem P

9.757.2 Constructor & Destructor Documentation

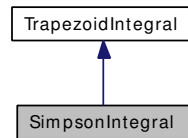
9.757.2.1 Simplex (Real *lambda*)

Constructor taking as input the characteristic length

9.758 SimpsonIntegral Class Reference

```
#include <ql/math/integrals/simpsonintegral.hpp>
```

Inheritance diagram for SimpsonIntegral:



9.758.1 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 >())

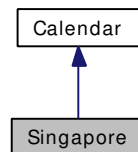
Protected Member Functions

- Real **integrate** (const boost::function< Real(Real)> &f, Real a, Real b) const

9.759 Singapore Class Reference

```
#include <ql/time/calendars/singapore.hpp>
```

Inheritance diagram for Singapore:



9.759.1 Detailed Description

Singapore calendars

Holidays for the [Singapore](http://www.ses.com.sg) exchange (data from <http://www.ses.com.sg>):

- Saturdays
- Sundays
- New Year's day, January 1st
- Good Friday
- Labour Day, May 1st
- National Day, August 9th
- Christmas, December 25th

Other holidays for which no rule is given (data available for 2004-2007 only:)

- Chinese New Year
- Hari Raya Haji
- Vesak Poya Day
- Deepavali
- Diwali
- Hari Raya Puasa

Public Types

- enum [Market](#) { [SGX](#) }

Public Member Functions

- [Singapore](#) ([Market](#) m=SGX)

9.759.2 Member Enumeration Documentation

9.759.2.1 enum Market

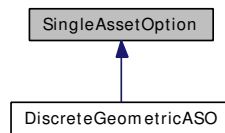
Enumerator:

SGX [Singapore](#) exchange.

9.760 SingleAssetOption Class Reference

```
#include <ql/legacy/pricers/singleassetoption.hpp>
```

Inheritance diagram for SingleAssetOption:



9.760.1 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=1.0e-7, Volatility maxVol=4.0) const
- Spread **impliedDivYield** (Real targetValue, Real accuracy=1e-4, Size maxEvaluations=100, Spread minYield=1.0e-7, Spread maxYield=4.0) const
- virtual boost::shared_ptr< [SingleAssetOption](#) > **clone** () const=0

Protected Attributes

- Real **underlying_**
- [PlainVanillaPayoff](#) **payoff_**
- Spread **dividendYield_**
- Rate **riskFreeRate_**
- Time **residualTime_**
- Volatility **volatility_**
- bool **hasBeenCalculated_**
- Real **rho_**
- Real **dividendRho_**
- Real **vega_**
- Real **theta_**

- bool `rhoComputed_`
- bool `dividendRhoComputed_`
- bool `vegaComputed_`
- bool `thetaComputed_`

Static Protected Attributes

- static const Real `dVolMultiplier_`
- static const Real `dRMultiplier_`

Friends

- class `VolatilityFunction`
- class `DivYieldFunction`

9.760.2 Member Function Documentation

9.760.2.1 **Volatility** `impliedVolatility (Real targetValue, Real accuracy = 1e-4, Size maxEvaluations = 100, Volatility minVol = 1.0e-7, Volatility maxVol = 4.0) const`

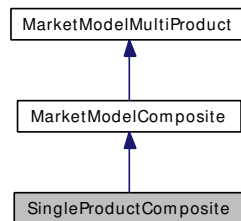
Warning

Options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g. binary options. In these cases `impliedVolatility` can fail and in any case is meaningless. Another possible source of failure is to have a `targetValue` that is not attainable with any volatility, e.g. a `targetValue` lower than the intrinsic value in the case of American options.

9.761 SingleProductComposite Class Reference

```
#include <ql/models/marketmodels/products/singleproductcomposite.hpp>
```

Inheritance diagram for SingleProductComposite:



9.761.1 Detailed Description

Composition of one or more market-model products.

Instances of this class build a single market-model product by composing two or more subproducts.

Precondition:

All subproducts must have the same rate times.

Public Member Functions

MarketModelMultiProduct interface

- Size **numberOfProducts** () const
- Size **maxNumberOfCashFlowsPerProductPerStep** () const
- bool **nextTimeStep** (const [CurveState](#) ¤tState, std::vector< Size > &numberCashFlowsThisStep, std::vector< std::vector< [CashFlow](#) > > &cashFlowsGenerated)
return value indicates whether path is finished, TRUE means done
- std::auto_ptr< [MarketModelMultiProduct](#) > **clone** () const
returns a newly-allocated copy of itself

9.762 Singleton Class Template Reference

```
#include <ql/patterns/singleton.hpp>
```

9.762.1 Detailed Description

```
template<class T> class QuantLib::Singleton< T >
```

Basic support for the singleton pattern.

The typical use of this class is:

```
class Foo : public Singleton<Foo> {  
    friend class Singleton<Foo>;  
private:  
    Foo() {}  
public:  
    ...  
};
```

which, albeit sub-optimal, frees one from the concerns of creating and managing the unique instance and can serve later as a single implementation point should synchronization features be added.

Static Public Member Functions

- static T & [instance](#) ()
access to the unique instance

9.763 SingleVariate Struct Template Reference

```
#include <ql/methods/montecarlo/mctraits.hpp>
```

9.763.1 Detailed Description

`template<class RNG = PseudoRandom> struct QuantLib::SingleVariate< RNG >`

default Monte Carlo traits for single-variate models

Examples:

[DiscreteHedging.cpp](#).

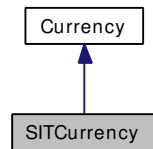
Public Types

- enum { **allowsErrorEstimate** = RNG::allowsErrorEstimate }
- typedef RNG **rng_traits**
- typedef [Path](#) **path_type**
- typedef [PathPricer](#)< [path_type](#) > **path_pricer_type**
- typedef RNG::rsg_type **rsg_type**
- typedef [PathGenerator](#)< rsg_type > **path_generator_type**

9.764 SITCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for SITCurrency:



9.764.1 Detailed Description

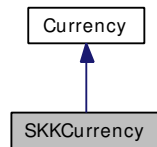
Slovenian tolar.

The ISO three-letter code is SIT; the numeric code is 705. It is divided in 100 stotinov.

9.765 SKKCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for SKKCurrency:



9.765.1 Detailed Description

Slovak koruna.

The ISO three-letter code is SKK; the numeric code is 703. It is divided in 100 halierov.

9.766 Slovakia Class Reference

```
#include <ql/time/calendars/slovakia.hpp>
```

Inheritance diagram for Slovakia:



9.766.1 Detailed Description

Slovak calendars.

Holidays for the Bratislava stock exchange (data from <http://www.bsse.sk/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Good Friday
- Easter Monday
- May Day, May 1st
- Liberation of the Republic, May 8th
- SS. Cyril and Methodius, July 5th
- Slovak National Uprising, August 29th
- Constitution of the Slovak Republic, September 1st
- Our Lady of the Seven Sorrows, September 15th
- All Saints Day, November 1st
- Freedom and Democracy of the Slovak Republic, November 17th
- Christmas Eve, December 24th
- Christmas, December 25th
- St. Stephen, December 26th

Public Types

- enum [Market](#) { [BSSE](#) }

Public Member Functions

- Slovakia ([Market](#) m=BSSE)

9.766.2 Member Enumeration Documentation

9.766.2.1 enum Market

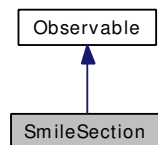
Enumerator:

BSSE Bratislava stock exchange.

9.767 SmileSection Class Reference

```
#include <ql/termstructures/volatilities/smilesection.hpp>
```

Inheritance diagram for SmileSection:



9.767.1 Detailed Description

interest rate volatility smile section

This abstract class provides volatility smile section interface

Public Member Functions

- **SmileSection** (const [Date](#) &d, const [DayCounter](#) &dc=[Actual365Fixed](#)(), const [Date](#) &referenceDate=[Date](#)())
- **SmileSection** (Time exerciseTime, const [DayCounter](#) &dc=[Actual365Fixed](#)())
- virtual Real **minStrike** () const=0
- virtual Real **maxStrike** () const=0
- virtual Real **variance** (Rate strike) const =0
- virtual Volatility **volatility** (Rate strike) const =0
- const [Date](#) & **exerciseDate** () const
- Time **exerciseTime** () const
- const [DayCounter](#) & **dayCounter** () const

Protected Attributes

- [Date](#) **exerciseDate_**
- [DayCounter](#) **dc_**
- Time **exerciseTime_**

9.768 SMMDriftCalculator Class Reference

```
#include <ql/models/marketmodels/driftcomputation/smmdriftcalculator.hpp>
```

9.768.1 Detailed Description

Drift computation for coterminal swap market models.

Returns the drift $\mu\Delta t$. See Mark Joshi, Lorenzo Liesch, *Effective Implementation Of Generic Market Models*.

Public Member Functions

- **SMMDriftCalculator** (const [Matrix](#) &pseudo, const std::vector< Spread > &displacements, const std::vector< Time > &taus, Size numeraire, Size alive)
- void [compute](#) (const [CoterminalSwapCurveState](#) &cs, std::vector< Real > &drifts) const
Computes the drifts.

9.769 SobolBrownianGenerator Class Reference

```
#include <ql/models/marketmodels/browniangenerators/sobolbrowniangenerator.hpp>
```

9.769.1 Detailed Description

Sobol Brownian generator for market-model simulations.

Incremental Brownian generator using a Sobol generator, inverse-cumulative Gaussian method, and Brownian bridging.

Public Types

- enum [Ordering](#) { [Factors](#), [Steps](#), [Diagonal](#) }

Public Member Functions

- **SobolBrownianGenerator** (Size factors, Size steps, [Ordering](#) ordering, unsigned long seed=0, SobolRsg::DirectionIntegers directionIntegers=SobolRsg::Jaeckel)
- Real **nextPath** ()
- Real **nextStep** (std::vector< Real > &)
- Size **numberOfFactors** () const
- Size **numberOfSteps** () const

9.769.2 Member Enumeration Documentation

9.769.2.1 enum Ordering

Enumerator:

Factors The variates with the best quality will be used for the evolution of the first factor.

Steps The variates with the best quality will be used for the largest steps of all factors.

Diagonal A diagonal schema will be used to assign the variates with the best quality to the most important factors and the largest steps.

9.770 SobolRsg Class Reference

```
#include <ql/math/randomnumbers/sobolrsg.hpp>
```

9.770.1 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 in QuantLib. Jäckel has calculated 8 129 334 polynomials: if you need that many dimensions you can replace the `primitivepolynomials.c` file included in QuantLib with the one provided in the CD of the "Monte Carlo Methods in Finance" book.

The choice of initialization numbers (also know as free direction integers) is crucial for the homogeneity properties of the sequence. Sobol defines two homogeneity properties: Property A and Property A'.

The unit initialization numbers suggested in "Numerical Recipes in C", 2nd edition, by Press, Teukolsky, Vetterling, and Flannery (section 7.7) fail the test for Property A even for low dimensions.

Bratley and Fox published coefficients of the free direction integers up to dimension 40, crediting unpublished work of Sobol' and Levitan. See Bratley, P., Fox, B.L. (1988) "Algorithm 659: Implementing Sobol's quasirandom sequence generator," ACM Transactions on Mathematical Software 14:88-100. These values satisfy Property A for $d \leq 20$ and $d = 23, 31, 33, 34, 37$; Property A' holds for $d \leq 6$.

Jäckel provides in his book (section 8.3) initialization numbers up to dimension 32. Coefficients for $d \leq 8$ are the same as in Bradley-Fox, so Property A' holds for $d \leq 6$ but Property A holds for $d \leq 32$.

The implementation of Lemieux, Cieslak, and Luttmmer includes coefficients of the free direction integers up to dimension 360. Coefficients for $d \leq 40$ are the same as in Bradley-Fox. For dimension $40 < d \leq 360$ the coefficients have been calculated as optimal values based on the "resolution" criterion. See "RandQMC user's guide - A package for randomized quasi-Monte Carlo methods in C," by C. Lemieux, M. Cieslak, and K. Luttmmer, version January 13 2004, and references cited there (<http://www.math.ualgary.ca/~lemieux/randqmc.html>). The values up to $d \leq 360$ has been provided to the QuantLib team by Christiane Lemieux, private communication, September 2004.

For more info on Sobol' sequences see also "Monte Carlo Methods in Financial Engineering," by P. Glasserman, 2004, Springer, section 5.2.3

Tests

- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

Public Types

- enum `DirectionIntegers` { `Unit`, `Jaekel`, `SobolLevitan`, `SobolLevitanLemieux` }

- typedef [Sample](#)< std::vector< Real > > **sample_type**

Public Member Functions

- [SobolRsg](#) (Size dimensionality, unsigned long seed=0, DirectionIntegers directionIntegers=Jaeckel)
- void [skipTo](#) (unsigned long n)
- const std::vector< unsigned long > & [nextInt32Sequence](#) () const
- const [SobolRsg::sample_type](#) & [nextSequence](#) () const
- const [sample_type](#) & [lastSequence](#) () const
- Size [dimension](#) () const

9.770.2 Constructor & Destructor Documentation

9.770.2.1 SobolRsg (Size *dimensionality*, unsigned long *seed* = 0, DirectionIntegers *directionIntegers* = Jaeckel)

Precondition:

dimensionality must be <= PPMT_MAX_DIM

9.770.3 Member Function Documentation

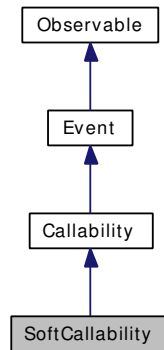
9.770.3.1 void skipTo (unsigned long *n*)

skip to the n-th sample in the low-discrepancy sequence

9.771 SoftCallability Class Reference

```
#include <ql/instruments/convertiblebond.hpp>
```

Inheritance diagram for SoftCallability:



9.771.1 Detailed Description

callability leaving to the holder the possibility to convert

Examples:

[ConvertibleBonds.cpp](#).

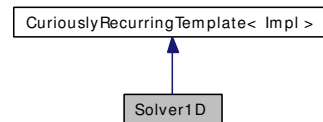
Public Member Functions

- **SoftCallability** (const Price &price, const [Date](#) &date, Real trigger)
- Real **trigger** () const

9.772 Solver1D Class Template Reference

```
#include <ql/math/solver1d.hpp>
```

Inheritance diagram for Solver1D:



9.772.1 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_`

9.772.2 Member Function Documentation

9.772.2.1 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 ϵ ; depending on the particular solver, this might mean that the returned x is such that $|f(x)| < \epsilon$, or that $|x - \xi| < \epsilon$ where ξ is the real zero.

This method contains a bracketing routine to which an initial guess must be supplied as well as a step used to scan the range of the possible bracketing values.

9.772.2.2 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 ϵ ; depending on the particular solver, this might mean that the returned x is such that $|f(x)| < \epsilon$, or that $|x - \xi| < \epsilon$ where ξ is the real zero.

An initial guess must be supplied, as well as two values x_{\min} and x_{\max} which must bracket the zero (i.e., either $f(x_{\min}) \leq 0 \leq f(x_{\max})$, or $f(x_{\max}) \leq 0 \leq f(x_{\min})$ must be true).

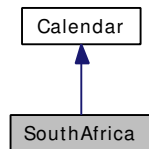
9.772.2.3 void setMaxEvaluations (Size evaluations)

This method sets the maximum number of function evaluations for the bracketing routine. An error is thrown if a bracket is not found after this number of evaluations.

9.773 SouthAfrica Class Reference

```
#include <ql/time/calendars/southafrica.hpp>
```

Inheritance diagram for SouthAfrica:



9.773.1 Detailed Description

South-African calendar.

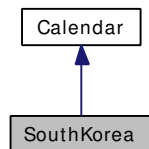
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Family Day, Easter Monday
- Human Rights Day, March 21st (possibly moved to Monday)
- Freedom Day, April 27th (possibly moved to Monday)
- Workers Day, May 1st (possibly moved to Monday)
- Youth Day, June 16th (possibly moved to Monday)
- National Women's Day, August 9th (possibly moved to Monday)
- Heritage Day, September 24th (possibly moved to Monday)
- Day of Reconciliation, December 16th (possibly moved to Monday)
- Christmas December 25th
- Day of Goodwill December 26th (possibly moved to Monday)

9.774 SouthKorea Class Reference

```
#include <ql/time/calendars/southkorea.hpp>
```

Inheritance diagram for SouthKorea:



9.774.1 Detailed Description

South Korean calendars.

Holidays for the Korea exchange (data from <http://www.krx.co.kr>):

- 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 2004-2007 only:)

- Lunar New Year
- Election Day 2004
- Buddha's birthday
- Harvest Moon Day

Public Types

- enum [Market](#) { [KRX](#) }

Public Member Functions

- SouthKorea ([Market](#) m=KRX)

9.774.2 Member Enumeration Documentation

9.774.2.1 enum Market

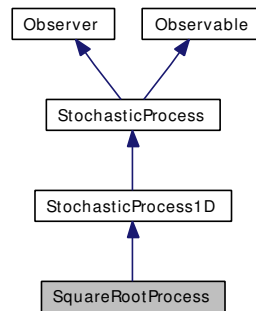
Enumerator:

KRX Korea exchange.

9.775 SquareRootProcess Class Reference

```
#include <ql/processes/squarerootprocess.hpp>
```

Inheritance diagram for SquareRootProcess:



9.775.1 Detailed Description

Square-root process class.

This class describes a square-root process governed by

$$dx = a(b - x_t)dt + \sigma \sqrt{x_t}dW_t.$$

Public Member Functions

- **SquareRootProcess** (Real b, Real a, Volatility sigma, Real x0=0.0, const boost::shared_ptr< discretization > &d=boost::shared_ptr< discretization >(new [EulerDiscretization](#)))

StochasticProcess interface

- Real [x0](#) () const
returns the initial value of the state variable
- Real [drift](#) (Time t, Real x) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- Real [diffusion](#) (Time t, Real x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

9.776 StatsHolder Class Reference

```
#include <ql/math/statistics/gaussianstatistics.hpp>
```

9.776.1 Detailed Description

Helper class for precomputed distributions.

Public Types

- typedef Real **value_type**

Public Member Functions

- **StatsHolder** (Real mean, Real standardDeviation)
- Real **mean** () const
- Real **standardDeviation** () const

9.777 SteepestDescent Class Reference

```
#include <ql/math/optimization/steepestdescent.hpp>
```

9.777.1 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** (const boost::shared_ptr< [LineSearch](#) > &lineSearch=boost::shared_ptr< [LineSearch](#) >())
- virtual EndCriteria::Type [minimize](#) ([Problem](#) &P, const [EndCriteria](#) &endCriteria)
minimize the optimization problem P

9.778 `step_iterator` Class Template Reference

```
#include <ql/utilities/steppingiterator.hpp>
```

9.778.1 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

9.778.2 Friends And Related Function Documentation

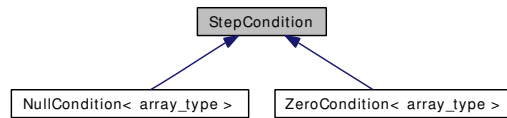
9.778.2.1 `step_iterator`< Iterator > [make_step_iterator](#) (Iterator it, Size step) [related]

helper function to create step iterators

9.779 StepCondition Class Template Reference

```
#include <ql/methods/finitedifferences/stepcondition.hpp>
```

Inheritance diagram for StepCondition:



9.779.1 Detailed Description

```
template<class array_type> class QuantLib::StepCondition< array_type >
```

condition to be applied at every time step

Public Member Functions

- virtual void **applyTo** (array_type &a, Time t) const=0

9.780 StepConditionSet Class Template Reference

```
#include <ql/methods/finitedifferences/parallelevolver.hpp>
```

9.780.1 Detailed Description

```
template<typename array_type> class QuantLib::StepConditionSet< array_type >
```

Parallel evolver for multiple arrays.

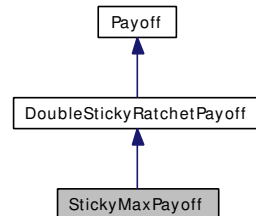
Public Member Functions

- void **applyTo** (std::vector< array_type > &a, Time t) const
- void **push_back** (const itemType &a)

9.781 StickyMaxPayoff Class Reference

```
#include <ql/instruments/stickyratchet.hpp>
```

Inheritance diagram for StickyMaxPayoff:



9.781.1 Detailed Description

StickyMax payoff (double option).

Public Member Functions

- **StickyMaxPayoff** (Real gearing1, Real gearing2, Real gearing3, Real spread1, Real spread2, Real spread3, Real initialValue1, Real initialValue2, Real accrualFactor)

Payoff interface

- `std::string name () const`

9.781.2 Member Function Documentation

9.781.2.1 `std::string name () const` [virtual]

Warning

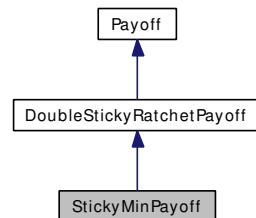
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Reimplemented from [DoubleStickyRatchetPayoff](#).

9.782 StickyMinPayoff Class Reference

```
#include <ql/instruments/stickyratchet.hpp>
```

Inheritance diagram for StickyMinPayoff:



9.782.1 Detailed Description

StickyMin payoff (double option).

Public Member Functions

- **StickyMinPayoff** (Real gearing1, Real gearing2, Real gearing3, Real spread1, Real spread2, Real spread3, Real initialValue1, Real initialValue2, Real accrualFactor)

Payoff interface

- `std::string name () const`

9.782.2 Member Function Documentation

9.782.2.1 `std::string name () const` [virtual]

Warning

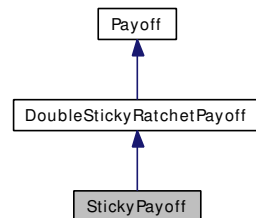
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Reimplemented from [DoubleStickyRatchetPayoff](#).

9.783 StickyPayoff Class Reference

```
#include <ql/instruments/stickyrate.hpp>
```

Inheritance diagram for StickyPayoff:



9.783.1 Detailed Description

Sticky payoff (single option).

Public Member Functions

- **StickyPayoff** (Real gearing1, Real gearing2, Real spread1, Real spread2, Real initialValue, Real accrualFactor)

Payoff interface

- `std::string name () const`

9.783.2 Member Function Documentation

9.783.2.1 `std::string name () const` [virtual]

Warning

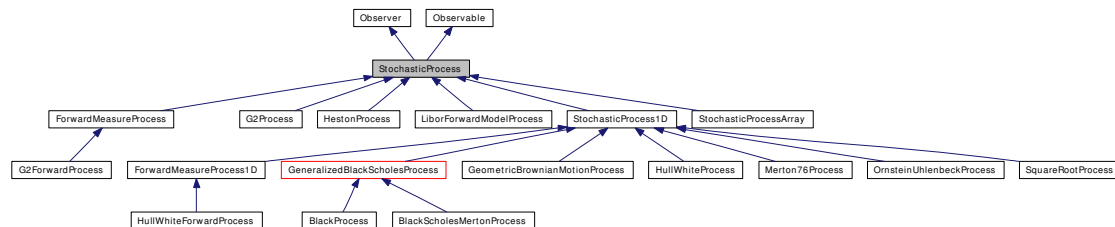
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Reimplemented from [DoubleStickyRatchetPayoff](#).

9.784 StochasticProcess Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for StochasticProcess:



9.784.1 Detailed Description

multi-dimensional stochastic process class.

This class describes a stochastic process governed by

$$dx_t = \mu(t, x_t)dt + \sigma(t, x_t) \cdot dW_t.$$

Public Member Functions

Stochastic process interface

- virtual `Size` `size` () const=0
returns the number of dimensions of the stochastic process
- virtual `Size` `factors` () const
returns the number of independent factors of the process
- virtual `Disposable`< `Array` > `initialValues` () const=0
returns the initial values of the state variables
- virtual `Disposable`< `Array` > `drift` (Time t, const `Array` &x) const=0
returns the drift part of the equation, i.e., $\mu(t, x_t)$
- virtual `Disposable`< `Matrix` > `diffusion` (Time t, const `Array` &x) const=0
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- virtual `Disposable`< `Array` > `expectation` (Time t0, const `Array` &x0, Time dt) const
- virtual `Disposable`< `Matrix` > `stdDeviation` (Time t0, const `Array` &x0, Time dt) const
- virtual `Disposable`< `Matrix` > `covariance` (Time t0, const `Array` &x0, Time dt) const
- virtual `Disposable`< `Array` > `evolve` (Time t0, const `Array` &x0, Time dt, const `Array` &dw) const
- virtual `Disposable`< `Array` > `apply` (const `Array` &x0, const `Array` &dx) const

utilities

- virtual Time `time` (const `Date` &) const

Observer interface

- void `update` ()

Protected Member Functions

- `StochasticProcess` (const boost::shared_ptr< [discretization](#) > &)

Protected Attributes

- boost::shared_ptr< [discretization](#) > `discretization_`

Classes

- class [discretization](#)
[discretization](#) of a stochastic process over a given time interval

9.784.2 Member Function Documentation

9.784.2.1 virtual Disposable<Array> expectation (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given [discretization](#). This method can be overridden in derived classes which want to hard-code a particular [discretization](#).

Reimplemented in [G2Process](#), [G2ForwardProcess](#), and [StochasticProcessArray](#).

9.784.2.2 virtual Disposable<Matrix> stdDeviation (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given [discretization](#). This method can be overridden in derived classes which want to hard-code a particular [discretization](#).

Reimplemented in [G2Process](#), [G2ForwardProcess](#), and [StochasticProcessArray](#).

9.784.2.3 virtual Disposable<Matrix> covariance (Time *t0*, const Array & *x0*, Time *dt*) const [virtual]

returns the covariance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given [discretization](#). This method can be overridden in derived classes which want to hard-code a particular [discretization](#).

Reimplemented in [G2Process](#), [G2ForwardProcess](#), [LiborForwardModelProcess](#), and [StochasticProcessArray](#).

9.784.2.4 virtual Disposable<Array> evolve (Time *t0*, const Array & *x0*, Time *dt*, const Array & *dw*) const [virtual]

returns the asset value after a time interval Δt according to the given [discretization](#). By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where E is the expectation and S the standard deviation.

Reimplemented in [HestonProcess](#), [LiborForwardModelProcess](#), and [StochasticProcessArray](#).

9.784.2.5 virtual Disposable<Array> apply (const Array & x_0 , const Array & dx) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented in [HestonProcess](#), [LiborForwardModelProcess](#), and [StochasticProcessArray](#).

9.784.2.6 virtual Time time (const Date &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:

As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented in [GeneralizedBlackScholesProcess](#), [HestonProcess](#), [Merton76Process](#), and [StochasticProcessArray](#).

9.784.2.7 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

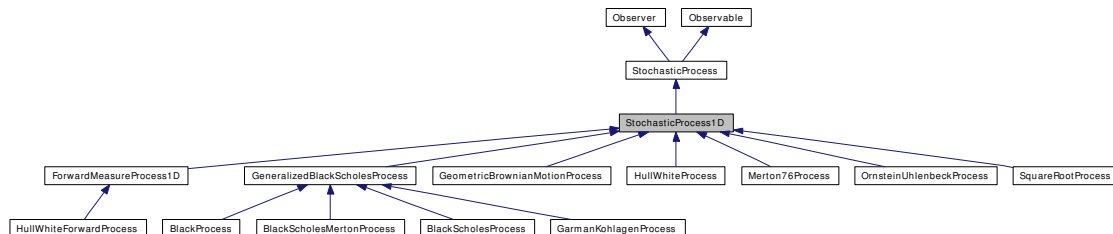
Implements [Observer](#).

Reimplemented in [GeneralizedBlackScholesProcess](#), and [HestonProcess](#).

9.785 StochasticProcess1D Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for StochasticProcess1D:



9.785.1 Detailed Description

1-dimensional stochastic process

This class describes a stochastic process governed by

$$dx_t = \mu(t, x_t)dt + \sigma(t, x_t)dW_t.$$

Public Member Functions

1-D stochastic process interface

- virtual Real **x0** () const=0
returns the initial value of the state variable
- virtual Real **drift** (Time t, Real x) const=0
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- virtual Real **diffusion** (Time t, Real x) const=0
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- virtual Real **expectation** (Time t0, Real x0, Time dt) const
- virtual Real **stdDeviation** (Time t0, Real x0, Time dt) const
- virtual Real **variance** (Time t0, Real x0, Time dt) const
- virtual Real **evolve** (Time t0, Real x0, Time dt, Real dw) const
- virtual Real **apply** (Real x0, Real dx) const

Protected Member Functions

- StochasticProcess1D (const boost::shared_ptr< **discretization** > &)

Protected Attributes

- boost::shared_ptr< **discretization** > **discretization_**

Classes

- class [discretization](#)
discretization of a 1-D stochastic process

9.785.2 Member Function Documentation

9.785.2.1 virtual Real expectation (Time t_0 , Real x_0 , Time dt) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given [discretization](#). This method can be overridden in derived classes which want to hard-code a particular [discretization](#).

Reimplemented in [HullWhiteProcess](#), [HullWhiteForwardProcess](#), and [OrnsteinUhlenbeckProcess](#).

9.785.2.2 virtual Real stdDeviation (Time t_0 , Real x_0 , Time dt) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given [discretization](#). This method can be overridden in derived classes which want to hard-code a particular [discretization](#).

Reimplemented in [HullWhiteProcess](#), [HullWhiteForwardProcess](#), and [OrnsteinUhlenbeckProcess](#).

9.785.2.3 virtual Real variance (Time t_0 , Real x_0 , Time dt) const [virtual]

returns the variance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given [discretization](#). This method can be overridden in derived classes which want to hard-code a particular [discretization](#).

Reimplemented in [HullWhiteProcess](#), [HullWhiteForwardProcess](#), and [OrnsteinUhlenbeckProcess](#).

9.785.2.4 virtual Real evolve (Time t_0 , Real x_0 , Time dt , Real dw) const [virtual]

returns the asset value after a time interval Δt according to the given [discretization](#). By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where E is the expectation and S the standard deviation.

9.785.2.5 virtual Real apply (Real x_0 , Real dx) const [virtual]

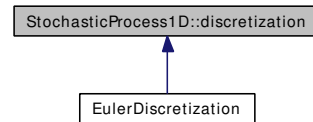
applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented in [GeneralizedBlackScholesProcess](#), and [Merton76Process](#).

9.786 StochasticProcess1D::discretization Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for StochasticProcess1D::discretization:



9.786.1 Detailed Description

[discretization](#) of a 1-D stochastic process

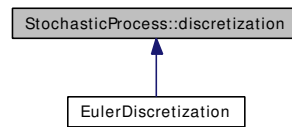
Public Member Functions

- virtual Real **drift** (const [StochasticProcess1D](#) &, Time t0, Real x0, Time dt) const=0
- virtual Real **diffusion** (const [StochasticProcess1D](#) &, Time t0, Real x0, Time dt) const=0
- virtual Real **variance** (const [StochasticProcess1D](#) &, Time t0, Real x0, Time dt) const=0

9.787 StochasticProcess::discretization Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for StochasticProcess::discretization:



9.787.1 Detailed Description

[discretization](#) of a stochastic process over a given time interval

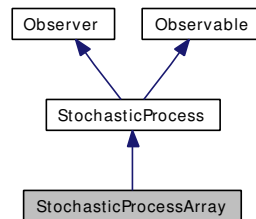
Public Member Functions

- virtual [Disposable](#)< [Array](#) > **drift** (const [StochasticProcess](#) &, Time t0, const [Array](#) &x0, Time dt) const=0
- virtual [Disposable](#)< [Matrix](#) > **diffusion** (const [StochasticProcess](#) &, Time t0, const [Array](#) &x0, Time dt) const=0
- virtual [Disposable](#)< [Matrix](#) > **covariance** (const [StochasticProcess](#) &, Time t0, const [Array](#) &x0, Time dt) const=0

9.788 StochasticProcessArray Class Reference

```
#include <ql/processes/stochasticprocessarray.hpp>
```

Inheritance diagram for StochasticProcessArray:



9.788.1 Detailed Description

Array of correlated 1-D stochastic processes

Public Member Functions

- **StochasticProcessArray** (const std::vector< boost::shared_ptr< [StochasticProcess1D](#) > > &, const [Matrix](#) &correlation)
- Size [size](#) () const
returns the number of dimensions of the stochastic process
- [Disposable](#)< [Array](#) > [initialValues](#) () const
returns the initial values of the state variables
- [Disposable](#)< [Array](#) > [drift](#) (Time t, const [Array](#) &x) const
returns the drift part of the equation, i.e., $\mu(t, x_t)$
- [Disposable](#)< [Array](#) > [expectation](#) (Time t0, const [Array](#) &x0, Time dt) const
- [Disposable](#)< [Matrix](#) > [diffusion](#) (Time t, const [Array](#) &x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- [Disposable](#)< [Matrix](#) > [covariance](#) (Time t0, const [Array](#) &x0, Time dt) const
- [Disposable](#)< [Matrix](#) > [stdDeviation](#) (Time t0, const [Array](#) &x0, Time dt) const
- [Disposable](#)< [Array](#) > [apply](#) (const [Array](#) &x0, const [Array](#) &dx) const
- [Disposable](#)< [Array](#) > [evolve](#) (Time t0, const [Array](#) &x0, Time dt, const [Array](#) &dw) const
- Time [time](#) (const [Date](#) &) const
- const boost::shared_ptr< [StochasticProcess1D](#) > & [process](#) (Size i) const
- [Disposable](#)< [Matrix](#) > [correlation](#) () const

Protected Attributes

- std::vector< boost::shared_ptr< [StochasticProcess1D](#) > > [processes_](#)
- [Matrix](#) [sqrtCorrelation_](#)

9.788.2 Member Function Documentation

9.788.2.1 Disposable<Array> expectation (Time t_0 , const Array & x_0 , Time dt) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval 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](#).

9.788.2.2 Disposable<Matrix> covariance (Time t_0 , const Array & x_0 , Time dt) const [virtual]

returns the covariance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval 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](#).

9.788.2.3 Disposable<Matrix> stdDeviation (Time t_0 , const Array & x_0 , Time dt) const [virtual]

returns the standard deviation $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval 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](#).

9.788.2.4 Disposable<Array> apply (const Array & x_0 , const Array & dx) const [virtual]

applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented from [StochasticProcess](#).

9.788.2.5 Disposable<Array> evolve (Time t_0 , const Array & x_0 , Time dt , const Array & dw) const [virtual]

returns the asset value after a time interval Δt according to the given discretization. By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where E is the expectation and S the standard deviation.

Reimplemented from [StochasticProcess](#).

9.788.2.6 Time time (const Date &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:

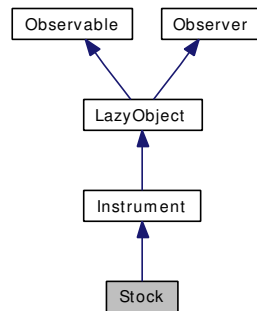
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

9.789 Stock Class Reference

```
#include <ql/instruments/stock.hpp>
```

Inheritance diagram for Stock:



9.789.1 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

9.789.2 Member Function Documentation

9.789.2.1 void performCalculations () const [protected, virtual]

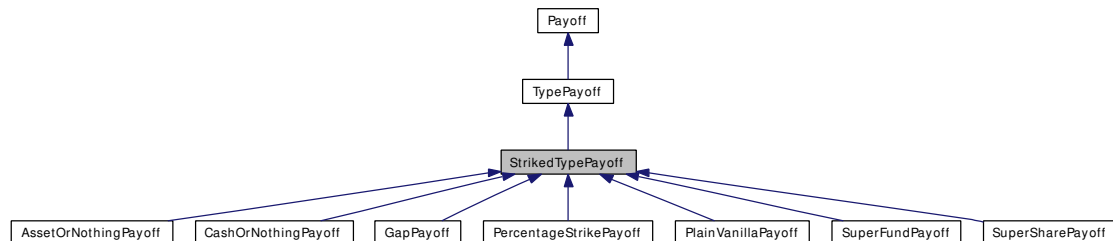
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

9.790 StrikedTypePayoff Class Reference

```
#include <ql/instruments/payoffs.hpp>
```

Inheritance diagram for StrikedTypePayoff:



9.790.1 Detailed Description

Intermediate class for payoffs based on a fixed strike.

Public Member Functions

- **StrikedTypePayoff** (Option::Type type, Real strike)
- Real **strike** () const

Payoff interface

- std::string **description** () const

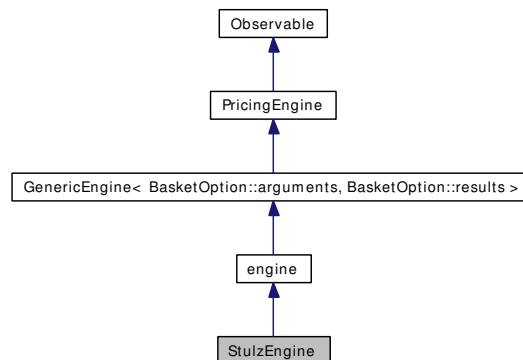
Protected Attributes

- Real **strike_**

9.791 StulzEngine Class Reference

```
#include <ql/pricingengines/basket/stulzengine.hpp>
```

Inheritance diagram for StulzEngine:



9.791.1 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.

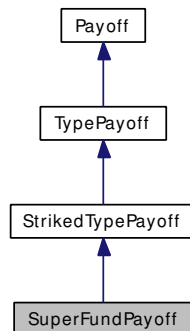
Public Member Functions

- void **calculate** () const

9.792 SuperFundPayoff Class Reference

```
#include <ql/instruments/payoffs.hpp>
```

Inheritance diagram for SuperFundPayoff:



9.792.1 Detailed Description

Binary superfund payoff.

This payoff is equivalent to being (1/lowerstrike) a) long (short) an AssetOrNothing Call (Put) at the lower strike and b) short (long) an AssetOrNothing Call (Put) at the higher strike

Public Member Functions

- **SuperFundPayoff** (Real strike, Real secondStrike)
- Real **secondStrike** () const

Payoff interface

- std::string **name** () const
- Real **operator()** (Real price) const
- virtual void **accept** ([AcyclicVisitor](#) &)

Protected Attributes

- Real **secondStrike_**

9.792.2 Member Function Documentation

9.792.2.1 std::string name () const [virtual]

Warning

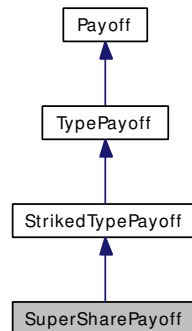
This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implements [Payoff](#).

9.793 SuperSharePayoff Class Reference

```
#include <ql/instruments/payoffs.hpp>
```

Inheritance diagram for SuperSharePayoff:



9.793.1 Detailed Description

Binary supershare payoff.

Public Member Functions

- **SuperSharePayoff** (Real strike, Real secondStrike, Real cashPayoff)
- Real **strike** () const
- Real **secondStrike** () const
- Real **cashPayoff** () const

Payoff interface

- std::string **name** () const
- std::string **description** () const
- Real **operator()** (Real price) const
- virtual void **accept** ([AcyclicVisitor](#) &)

Protected Attributes

- Real **strike_**
- Real **secondStrike_**
- Real **cashPayoff_**

9.793.2 Member Function Documentation

9.793.2.1 std::string name () const [virtual]

Warning

This method is used for output and comparison between payoffs. It is **not** meant to be used for writing switch-on-type code.

Implements [Payoff](#).

9.794 Surface Class Reference

```
#include <ql/math/surface.hpp>
```

9.794.1 Detailed Description

Surface abstract class

Public Member Functions

- virtual Real **operator()** (Real x, Real y) const=0
- virtual boost::shared_ptr< [Domain](#) > **domain** () const=0

9.795 SVD Class Reference

```
#include <ql/math/matrixutilities/svd.hpp>
```

9.795.1 Detailed Description

Singular value decomposition.

Refer to Golub and Van Loan: [Matrix](#) computation, The Johns Hopkins University Press

Tests

the correctness of the returned values is tested by checking their properties.

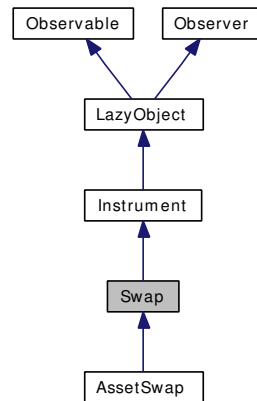
Public Member Functions

- **SVD** (const [Matrix](#) &)
- const [Matrix](#) & **U** () const
- const [Matrix](#) & **V** () const
- const [Array](#) & **singularValues** () const
- [Disposable](#)< [Matrix](#) > **S** () const
- Real **norm2** ()
- Real **cond** ()
- Integer **rank** ()
- [Disposable](#)< [Array](#) > **solveFor** (const [Array](#) &) const

9.796 Swap Class Reference

```
#include <ql/instruments/swap.hpp>
```

Inheritance diagram for Swap:



9.796.1 Detailed Description

Interest rate swap.

The cash flows belonging to the first leg are paid; the ones belonging to the second leg are received.

Public Member Functions

- **Swap** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, const Leg &firstLeg, const Leg &secondLeg)
- **Swap** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, const std::vector< Leg > &legs, const std::vector< bool > &payer)

Instrument interface

- bool **isExpired** () const
returns whether the instrument is still tradable.

Additional interface

- [Date](#) **startDate** () const
- [Date](#) **maturityDate** () const
- Real **legBPS** (Size j) const
- Real **legNPV** (Size j) const
- const Leg & **leg** (Size j) const
- const [Handle](#)< [YieldTermStructure](#) > & **termStructure** () const

Protected Member Functions

- void **setupExpired** () const
- void **performCalculations** () const

Protected Attributes

- [Handle< YieldTermStructure >](#) **termStructure_**
- `std::vector< Leg >` **legs_**
- `std::vector< Real >` **payer_**
- `std::vector< Real >` **legNPV_**
- `std::vector< Real >` **legBPS_**

9.796.2 Constructor & Destructor Documentation

9.796.2.1 Swap (const [Handle< YieldTermStructure >](#) & *termStructure*, const Leg & *firstLeg*, const Leg & *secondLeg*)

The cash flows belonging to the first leg are paid; the ones belonging to the second leg are received.

9.796.2.2 Swap (const [Handle< YieldTermStructure >](#) & *termStructure*, const `std::vector< Leg >` & *legs*, const `std::vector< bool >` & *payer*)

Multi leg constructor.

9.796.3 Member Function Documentation

9.796.3.1 void **setupExpired ()** const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met. Reimplemented from [Instrument](#).

9.796.3.2 void **performCalculations ()** const [protected, virtual]

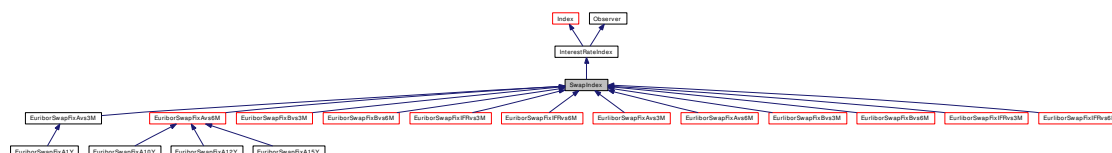
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

9.797 SwapIndex Class Reference

```
#include <ql/indexes/swapindex.hpp>
```

Inheritance diagram for SwapIndex:



9.797.1 Detailed Description

base class for swap-rate indexes

Public Member Functions

- **SwapIndex** (const std::string &familyName, const [Period](#) &tenor, Natural settlement-Days, [Currency](#) currency, const [Calendar](#) &calendar, const [Period](#) &fixedLegTenor, [BusinessDayConvention](#) fixedLegConvention, const [DayCounter](#) &fixedLegDayCounter, const boost::shared_ptr< [IborIndex](#) > &iborIndex)

InterestRateIndex interface

- [Handle](#)< [YieldTermStructure](#) > **termStructure** () const
- Rate **forecastFixing** (const [Date](#) &fixingDate) const
- [Date](#) **maturityDate** (const [Date](#) &valueDate) const

Inspectors

- [Period](#) **fixedLegTenor** () const
- [BusinessDayConvention](#) **fixedLegConvention** () const
- boost::shared_ptr< [IborIndex](#) > **iborIndex** () const
- [Schedule](#) **fixedRateSchedule** (const [Date](#) &fixingDate) const
- boost::shared_ptr< [VanillaSwap](#) > **underlyingSwap** (const [Date](#) &fixingDate) const

Protected Attributes

- [Period](#) **tenor_**
- boost::shared_ptr< [IborIndex](#) > **iborIndex_**
- [Period](#) **fixedLegTenor_**
- [BusinessDayConvention](#) **fixedLegConvention_**

9.797.2 Member Function Documentation

9.797.2.1 boost::shared_ptr<VanillaSwap> underlyingSwap (const [Date](#) & *fixingDate*) const

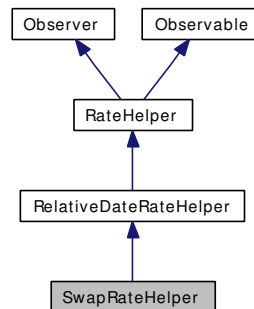
Warning

Relinking the term structure underlying the index will not have effect on the returned swap.

9.798 SwapRateHelper Class Reference

```
#include <ql/termstructures/yieldcurves/ratehelpers.hpp>
```

Inheritance diagram for SwapRateHelper:



9.798.1 Detailed Description

Rate helper for bootstrapping over swap rates.

Examples:

[swapvaluation.cpp](#).

Public Member Functions

- **SwapRateHelper** (const [Handle](#)< [Quote](#) > &rate, const [Period](#) &tenor, Natural settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, const boost::shared_ptr< [IborIndex](#) > &index)
- **SwapRateHelper** (Rate rate, const [Period](#) &tenor, Natural settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, const boost::shared_ptr< [IborIndex](#) > &index)
- Real **impliedQuote** () const
- void **setTermStructure** ([YieldTermStructure](#) *)
sets the term structure to be used for pricing

Protected Member Functions

- void **initializeDates** ()

Protected Attributes

- [Period](#) tenor_
- Natural settlementDays_
- [Calendar](#) calendar_
- [BusinessDayConvention](#) fixedConvention_

- [Frequency](#) `fixedFrequency_`
- [DayCounter](#) `fixedDayCount_`
- `boost::shared_ptr< IborIndex > index_`
- `boost::shared_ptr< VanillaSwap > swap_`
- [RelinkableHandle](#)< [YieldTermStructure](#) > `termStructureHandle_`

9.798.2 Member Function Documentation

9.798.2.1 `void setTermStructure (YieldTermStructure *)` [virtual]

sets the term structure to be used for pricing

[Warning](#)

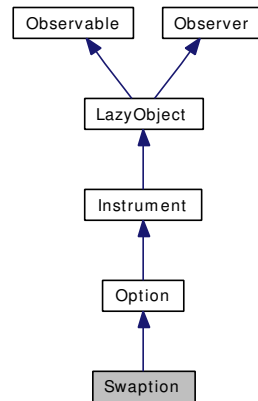
Being a pointer and not a `shared_ptr`, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

9.799 Swaption Class Reference

```
#include <ql/instruments/swaption.hpp>
```

Inheritance diagram for Swaption:



9.799.1 Detailed Description

Swaption class

Tests

- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption decreases (resp. increases) with the strike.
- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption increases (resp. decreases) with the spread.
- the correctness of the returned value is tested by checking it against that of a swaption on a swap with no spread and a correspondingly adjusted fixed rate.
- the correctness of the returned value is tested by checking it against a known good value.
- the correctness of the returned value of cash settled swaptions is tested by checking the modified annuity against a value calculated without using the [Swaption](#) class.

Todo

add greeks and explicit exercise lag

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

- **Swaption** (const boost::shared_ptr< VanillaSwap > &swap, const boost::shared_ptr< [Exercise](#) > &exercise, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared_ptr< [PricingEngine](#) > &engine, Settlement::Type delivery=Settlement::Physical)
- void **setupArguments** ([PricingEngine::arguments](#) *) const

- Volatility [impliedVolatility](#) (Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility minVol=1.0e-7, Volatility maxVol=4.0) const

implied volatility

- Rate **atmRate** () const

Instrument interface

- bool [isExpired](#) () const

returns whether the instrument is still tradable.

Inspectors

- Settlement::Type **settlementType** () const
- VanillaSwap::Type **type** () const
- const boost::shared_ptr< VanillaSwap > & **underlyingSwap** () const

Classes

- class [arguments](#)
Arguments for swaption calculation
- class [engine](#)
base class for swaption engines

9.800 Swaption::arguments Class Reference

```
#include <ql/instruments/swaption.hpp>
```

9.800.1 Detailed Description

Arguments for swaption calculation

Public Member Functions

- void **validate** () const

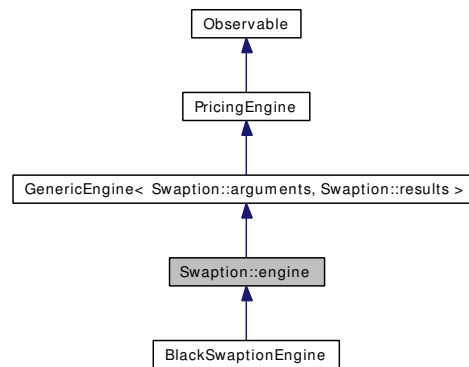
Public Attributes

- Rate **fairRate**
- Rate **fixedRate**
- Real **fixedBPS**
- Real **fixedCashBPS**
- Settlement::Type **settlementType**

9.801 Swaption::engine Class Reference

```
#include <ql/instruments/swaption.hpp>
```

Inheritance diagram for Swaption::engine:



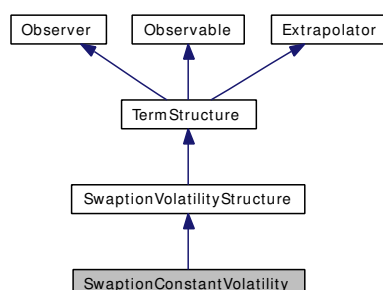
9.801.1 Detailed Description

base class for swaption engines

9.802 SwaptionConstantVolatility Class Reference

```
#include <ql/termstructures/volatilities/swaptionconstantvol.hpp>
```

Inheritance diagram for SwaptionConstantVolatility:



9.802.1 Detailed Description

Constant swaption volatility, no time-strike dependence.

SwaptionConstantVolatility interface

- const [Period](#) & [maxSwapTenor](#) () const
the largest length for which the term structure can return vols
- Time [maxSwapLength](#) () const
the largest swapLength 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
- Volatility [volatilityImpl](#) (Time, Time, Rate) const
implements the actual volatility calculation in derived classes
- boost::shared_ptr< [SmileSection](#) > [smileSectionImpl](#) (Time optionTime, Time swapLength) const
return smile section
- Volatility [volatilityImpl](#) (const [Date](#) &, const [Period](#) &, Rate) const

Public Member Functions

- [SwaptionConstantVolatility](#) (const [Date](#) &referenceDate, Volatility volatility, const [DayCounter](#) &dayCounter)
- [SwaptionConstantVolatility](#) (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)

- **SwaptionConstantVolatility** (Natural settlementDays, const [Calendar](#) &, Volatility volatility, const [DayCounter](#) &dayCounter)
- **SwaptionConstantVolatility** (Natural settlementDays, const [Calendar](#) &, const [Handle<Quote>](#) &volatility, const [DayCounter](#) &dayCounter)

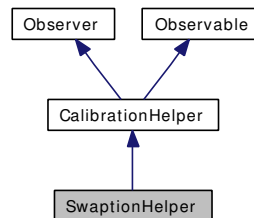
TermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return values

9.803 SwaptionHelper Class Reference

```
#include <ql/models/shortrate/calibrationhelpers/swaptionhelper.hpp>
```

Inheritance diagram for SwaptionHelper:



9.803.1 Detailed Description

calibration helper for ATM swaption

Examples:

[BermudanSwaption.cpp](#).

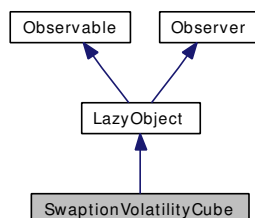
Public Member Functions

- **SwaptionHelper** (const [Period](#) &maturity, const [Period](#) &length, const [Handle](#)< [Quote](#) > &volatility, const boost::shared_ptr< [IborIndex](#) > &index, const [Period](#) &fixedLegTenor, const [DayCounter](#) &fixedLegDayCounter, const [DayCounter](#) &floatingLegDayCounter, const [Handle](#)< [YieldTermStructure](#) > &termStructure, bool calibrateVolatility=false)
- virtual void **addTimesTo** (std::list< Time > ×) const
- virtual Real **modelValue** () const
returns the price of the instrument according to the model
- virtual Real **blackPrice** (Volatility volatility) const
Black price given a volatility.

9.804 SwaptionVolatilityCube Class Reference

```
#include <ql/termstructures/volatilities/swaptionvolcube.hpp>
```

Inheritance diagram for SwaptionVolatilityCube:



9.804.1 Detailed Description

swaption-volatility cube

Warning

this class is not finalized and its interface might change in subsequent releases.

Public Member Functions

- **SwaptionVolatilityCube** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &atmVolStructure, const std::vector< [Period](#) > &optionTenors, const std::vector< [Period](#) > &swapTenors, const std::vector< [Spread](#) > &strikeSpreads, const std::vector< std::vector< [Handle](#)< [Quote](#) > > &volSpreads, const boost::shared_ptr< [SwapIndex](#) > &swapIndexBase, bool vegaWeightedSmileFit)

TermStructure interface

- [DayCounter](#) **dayCounter** () const
the day counter used for date/time conversion
- [Date](#) **maxDate** () const
the latest date for which the curve can return values
- [Time](#) **maxTime** () const
the latest time for which the curve can return values
- const [Date](#) & **referenceDate** () const
the date at which discount = 1.0 and/or variance = 0.0
- [Calendar](#) **calendar** () const
the calendar used for reference date calculation

LazyObject interface

- void **update** ()

SwaptionVolatilityStructure interface

- const [Period](#) & [maxSwapTenor](#) () const
the largest length for which the term structure can return vols
- Time [maxSwapLength](#) () const
the largest swapLength for which the term structure can return vols
- Rate [minStrike](#) () const
the minimum strike for which the term structure can return vols
- Rate [maxStrike](#) () const
the maximum strike for which the term structure can return vols

Other inspectors

- Rate [atmStrike](#) (const [Date](#) &optionDate, const [Period](#) &swapTenor) const
- Rate [atmStrike](#) (const [Period](#) &optionTenor, const [Period](#) &swapTenor) const

Protected Member Functions

SwaptionVolatilityStructure interface

- std::pair< Time, Time > [convertDates](#) (const [Date](#) &optionDate, const [Period](#) &swapTenor) const
implements the conversion between dates and times
- void [registerWithVolatilitySpread](#) ()
- Volatility [volatilityImpl](#) (Time optionTime, Time swapLength, Rate strike) const
implements the actual volatility calculation in derived classes
- Volatility [volatilityImpl](#) (const [Date](#) &optionDate, const [Period](#) &swapTenor, Rate strike) const
- Volatility [volatilityImpl](#) (const [Period](#) &optionTenor, const [Period](#) &swapTenor, Rate strike) const

Protected Attributes

- [Handle](#)< [SwaptionVolatilityStructure](#) > atmVol_
- Size nStrikes_
- std::vector< Spread > strikeSpreads_
- std::vector< Rate > localStrikes_
- std::vector< Volatility > localSmile_
- std::vector< std::vector< [Handle](#)< [Quote](#) > > > volSpreads_
- boost::shared_ptr< [SwapIndex](#) > swapIndexBase_
- bool vegaWeightedSmileFit_

9.804.2 Member Function Documentation

9.804.2.1 void update () [virtual]

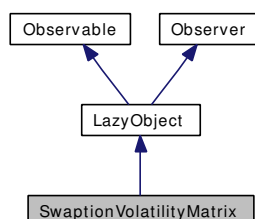
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [LazyObject](#).

9.805 SwaptionVolatilityMatrix Class Reference

```
#include <ql/termstructures/volatilities/swaptionvolmatrix.hpp>
```

Inheritance diagram for SwaptionVolatilityMatrix:



9.805.1 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 option date and swapLength.

The volatility matrix M must be defined so that:

- the number of rows equals the number of option dates;
- the number of columns equals the number of swap tenors;
- $M[i][j]$ contains the volatility corresponding to the i -th option and j -th tenor.

Public Member Functions

- **SwaptionVolatilityMatrix** (const [Calendar](#) &calendar, const std::vector< [Period](#) > &optionTenors, const std::vector< [Period](#) > &swapTenors, const std::vector< std::vector< [Handle](#)< [Quote](#) > > > &vols, const [DayCounter](#) &dayCounter=[Actual365Fixed](#)(), [BusinessDayConvention](#) bdc=[Following](#))
floating reference date, floating market data
- **SwaptionVolatilityMatrix** (const [Date](#) &referenceDate, const [Calendar](#) &calendar, const std::vector< [Period](#) > &optionTenors, const std::vector< [Period](#) > &swapTenors, const std::vector< std::vector< [Handle](#)< [Quote](#) > > > &vols, const [DayCounter](#) &dayCounter=[Actual365Fixed](#)(), [BusinessDayConvention](#) bdc=[Following](#))
fixed reference date, floating market data
- **SwaptionVolatilityMatrix** (const [Calendar](#) &calendar, const std::vector< [Period](#) > &optionTenors, const std::vector< [Period](#) > &swapTenors, const [Matrix](#) &volatilities, const [DayCounter](#) &dayCounter=[Actual365Fixed](#)(), [BusinessDayConvention](#) bdc=[Following](#))
floating reference date, fixed market data
- **SwaptionVolatilityMatrix** (const [Date](#) &referenceDate, const [Calendar](#) &calendar, const std::vector< [Period](#) > &optionTenors, const std::vector< [Period](#) > &swapTenors, const [Matrix](#) &volatilities, const [DayCounter](#) &dayCounter=[Actual365Fixed](#)(), [BusinessDayConvention](#) bdc=[Following](#))

fixed reference date, fixed market data

- **SwaptionVolatilityMatrix** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &optionDates, const std::vector< [Period](#) > &swapTenors, const [Matrix](#) &volatilities, const [DayCounter](#) &dayCounter)

TermStructure interface

- [Date](#) **maxDate** () const
the latest date for which the curve can return values

LazyObject interface

- void **update** ()
- void **performCalculations** () const

SwaptionVolatilityStructure interface

- const [Period](#) & **maxSwapTenor** () const
the largest length for which the term structure can return vols
- Time **maxSwapLength** () const
the largest swapLength for which the term structure can return vols
- Rate **minStrike** () const
the minimum strike for which the term structure can return vols
- Rate **maxStrike** () const
the maximum strike for which the term structure can return vols
- boost::shared_ptr< [SmileSection](#) > **smileSectionImpl** (Time optionTime, Time swapLength) const
return trivial smile section

Other inspectors

- std::pair< Size, Size > **locate** (const [Date](#) &optionDates, const [Period](#) &swapTenor) const
returns the lower indexes of surrounding volatility matrix corners
- std::pair< Size, Size > **locate** (Time optionTime, Time swapLength) const
returns the lower indexes of surrounding volatility matrix corners

9.805.2 Constructor & Destructor Documentation

- 9.805.2.1 **SwaptionVolatilityMatrix** (const [Date](#) & *referenceDate*, const std::vector< [Date](#) > & *optionDates*, const std::vector< [Period](#) > & *swapTenors*, const [Matrix](#) & *volatilities*, const [DayCounter](#) & *dayCounter*)

Deprecated

alternative constructors instead

9.805.3 Member Function Documentation

9.805.3.1 `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](#).

9.805.3.2 `void performCalculations () const` [virtual]

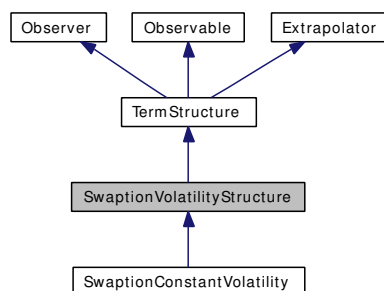
This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implements [LazyObject](#).

9.806 SwaptionVolatilityStructure Class Reference

```
#include <ql/swaptionvolstructure.hpp>
```

Inheritance diagram for SwaptionVolatilityStructure:



9.806.1 Detailed Description

Swaption-volatility structure

This class is purely abstract and defines the interface of concrete swaption volatility structures which will be derived from this one.

Public Member Functions

- virtual `std::pair< Time, Time > convertDates (const Date &optionDate, const Period &swapTenor) const`
implements the conversion between dates and times
- virtual `BusinessDayConvention businessDayConvention () const`
the business day convention used for option date calculation
- `Date optionDateFromTenor (const Period &optionTenor) const`
implements the conversion between optionTenors and optionDates

Constructors

See the `TermStructure` documentation for issues regarding constructors.

- `SwaptionVolatilityStructure (const DayCounter &dc=Actual365Fixed(), BusinessDayConvention bdc=Following)`
default constructor
- `SwaptionVolatilityStructure (const Date &referenceDate, const Calendar &calendar=Calendar(), const DayCounter &dc=Actual365Fixed(), BusinessDayConvention bdc=Following)`
initialize with a fixed reference date
- `SwaptionVolatilityStructure (Natural settlementDays, const Calendar &, const DayCounter &dc=Actual365Fixed(), BusinessDayConvention bdc=Following)`

calculate the reference date based on the global evaluation date

Volatility, variance and smile

- Volatility [volatility](#) (Time optionTime, Time swapLength, Rate strike, bool extrapolate=false) const
returns the volatility for a given option time and swapLength
- Real [blackVariance](#) (Time optionTime, Time swapLength, Rate strike, bool extrapolate=false) const
returns the Black variance for a given option time and swapLength
- Volatility [volatility](#) (const [Date](#) &optionDate, const [Period](#) &swapTenor, Rate strike, bool extrapolate=false) const
returns the volatility for a given option date and swap tenor
- Real [blackVariance](#) (const [Date](#) &optionDate, const [Period](#) &swapTenor, Rate strike, bool extrapolate=false) const
returns the Black variance for a given option date and swap tenor
- virtual boost::shared_ptr< [SmileSection](#) > [smileSection](#) (const [Date](#) &optionDate, const [Period](#) &swapTenor) const
- Volatility [volatility](#) (const [Period](#) &optionTenor, const [Period](#) &swapTenor, Rate strike, bool extrapolate=false) const
returns the volatility for a given option tenor and swap tenor
- Real [blackVariance](#) (const [Period](#) &optionTenor, const [Period](#) &swapTenor, Rate strike, bool extrapolate=false) const
returns the Black variance for a given option tenor and swap tenor
- boost::shared_ptr< [SmileSection](#) > [smileSection](#) (const [Period](#) &optionTenor, const [Period](#) &swapTenor) const

Limits

- virtual const [Period](#) & [maxSwapTenor](#) () const=0
the largest length for which the term structure can return vols
- virtual Time [maxSwapLength](#) () const
the largest swapLength for which the term structure can return vols
- virtual Rate [minStrike](#) () const=0
the minimum strike for which the term structure can return vols
- virtual Rate [maxStrike](#) () const=0
the maximum strike for which the term structure can return vols

Protected Member Functions

- virtual boost::shared_ptr< [SmileSection](#) > [smileSectionImpl](#) (Time optionTime, Time swapLength) const=0

return smile section

- virtual Volatility [volatilityImpl](#) (Time optionTime, Time swapLength, Rate strike) const=0
implements the actual volatility calculation in derived classes
- virtual Volatility **volatilityImpl** (const [Date](#) &optionDate, const [Period](#) &swapTenor, Rate strike) const
- void **checkRange** (Time, Time, Rate strike, bool extrapolate) const
- void **checkRange** (const [Date](#) &optionDate, const [Period](#) &swapTenor, Rate strike, bool extrapolate) const

9.806.2 Constructor & Destructor Documentation

9.806.2.1 SwaptionVolatilityStructure (const DayCounter & dc = Actual365Fixed(), BusinessDayConvention bdc = Following)

default constructor

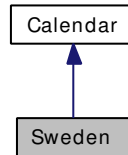
[Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

9.807 Sweden Class Reference

```
#include <ql/time/calendars/sweden.hpp>
```

Inheritance diagram for Sweden:



9.807.1 Detailed Description

Swedish calendar.

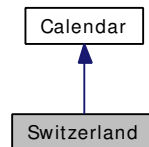
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- May Day, May 1st
- National Day, June 6th
- Midsummer Eve (Friday between June 18-24)
- Christmas Eve, December 24th
- Christmas Day, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31th

9.808 Switzerland Class Reference

```
#include <ql/time/calendars/switzerland.hpp>
```

Inheritance diagram for Switzerland:



9.808.1 Detailed Description

Swiss calendar.

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Berchtoldstag, January 2nd
- Good Friday
- Easter Monday
- Ascension Day
- Whit Monday
- Labour Day, May 1st
- National Day, August 1st
- Christmas, December 25th
- St. Stephen's Day, December 26th

9.809 SymmetricSchurDecomposition Class Reference

```
#include <ql/math/matrixutilities/symmetricschurdecomposition.hpp>
```

9.809.1 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

9.809.2 Constructor & Destructor Documentation

9.809.2.1 SymmetricSchurDecomposition (const Matrix & s)

Precondition:

s must be symmetric

9.810 TabulatedGaussLegendre Class Reference

```
#include <ql/math/integrals/gaussianquadratures.hpp>
```

9.810.1 Detailed Description

tabulated Gauss-Legendre quadratures

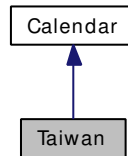
Public Member Functions

- **TabulatedGaussLegendre** (Size n=20)
- `template<class F>`
Real **operator()** (const F &f) const
- void **order** (Size)
- Size **order** () const

9.811 Taiwan Class Reference

```
#include <ql/time/calendars/taiwan.hpp>
```

Inheritance diagram for Taiwan:



9.811.1 Detailed Description

Taiwanese calendars.

Holidays for the [Taiwan](#) stock exchange (data from http://www.tse.com.tw/en/trading/trading_days.php):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Peace Memorial Day, February 28
- Labor Day, May 1st
- Double Tenth National Day, October 10th

Other holidays for which no rule is given (data available for 2002-2007 only:)

- Chinese Lunar New Year
- Tomb Sweeping Day
- Dragon Boat Festival
- Moon Festival

Public Types

- enum [Market](#) { [TSEC](#) }

Public Member Functions

- [Taiwan](#) ([Market](#) m=TSEC)

9.811.2 Member Enumeration Documentation

9.811.2.1 enum Market

Enumerator:

TSEC [Taiwan](#) stock exchange.

9.812 TARGET Class Reference

```
#include <ql/time/calendars/target.hpp>
```

Inheritance diagram for TARGET:



9.812.1 Detailed Description

TARGET calendar

Holidays (see <http://www.ecb.int>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday (since 2000)
- Easter Monday (since 2000)
- Labour Day, May 1st (since 2000)
- Christmas, December 25th
- Day of Goodwill, December 26th (since 2000)
- December 31st (1998, 1999, and 2001)

Tests

the correctness of the returned results is tested against a list of known holidays.

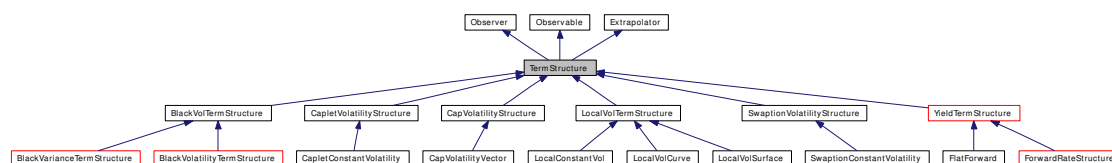
Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), and [swapvaluation.cpp](#).

9.813 TermStructure Class Reference

```
#include <ql/termstructure.hpp>
```

Inheritance diagram for TermStructure:



9.813.1 Detailed Description

Basic term-structure functionality.

Public Member Functions

Constructors

There are three ways in which a term structure can keep track of its reference date. The first is that such date is fixed; the second is that it is determined by advancing the current date of a given number of business days; and the third is that it is based on the reference date of some other structure.

In the first case, the constructor taking a date is to be used; the default implementation of `referenceDate()` will then return such date. In the second case, the constructor taking a number of days and a calendar is to be used; `referenceDate()` will return a date calculated based on the current evaluation date, and the term structure and its observers will be notified when the evaluation date changes. In the last case, the `referenceDate()` method must be overridden in derived classes so that it fetches and return the appropriate date.

- **TermStructure** (const **DayCounter** &dc=**Actual365Fixed**())
default constructor
- **TermStructure** (const **Date** &referenceDate, const **Calendar** &calendar=**Calendar**(), const **DayCounter** &dc=**Actual365Fixed**())
initialize with a fixed reference date
- **TermStructure** (Natural settlementDays, const **Calendar** &, const **DayCounter** &dc=**Actual365Fixed**())
calculate the reference date based on the global evaluation date

Dates

- virtual **DayCounter** **dayCounter** () const
the day counter used for date/time conversion
- virtual **Date** **maxDate** () const=0
the latest date for which the curve can return values
- virtual **Time** **maxTime** () const

the latest time for which the curve can return values

- virtual const [Date](#) & [referenceDate](#) () const
the date at which discount = 1.0 and/or variance = 0.0
- virtual [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation

Observer interface

- void [update](#) ()

Protected Member Functions

- Time [timeFromReference](#) (const [Date](#) &date) const
date/time conversion
- void [checkRange](#) (const [Date](#) &, bool extrapolate) const
date-range check
- void [checkRange](#) (Time, bool extrapolate) const
time-range check

Protected Attributes

- bool [moving_](#)

9.813.2 Constructor & Destructor Documentation

9.813.2.1 TermStructure (const DayCounter & dc = Actual365Fixed())

default constructor

Warning

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

9.813.3 Member Function Documentation

9.813.3.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

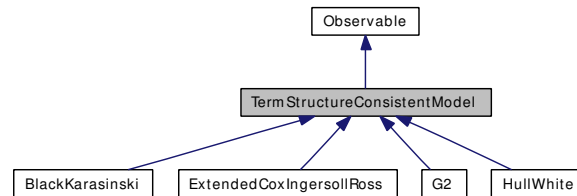
Implements [Observer](#).

Reimplemented in [CapVolatilityVector](#), [DecInterpCapletVolStructure](#), [SwaptionVolatilityCube](#), [SwaptionVolatilityMatrix](#), [ExtendedDiscountCurve](#), [FlatForward](#), and [PiecewiseZeroSpreadedTermStructure](#).

9.814 TermStructureConsistentModel Class Reference

```
#include <ql/models/model.hpp>
```

Inheritance diagram for TermStructureConsistentModel:



9.814.1 Detailed Description

Term-structure consistent model class.

This is a base class for models that can reprice exactly any discount bond.

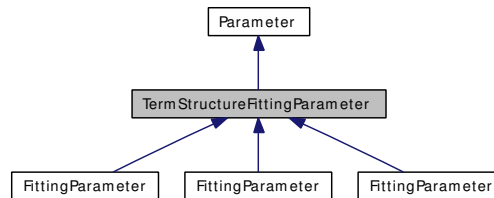
Public Member Functions

- **TermStructureConsistentModel** (const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- const [Handle](#)< [YieldTermStructure](#) > & **termStructure** () const

9.815 TermStructureFittingParameter Class Reference

```
#include <ql/models/parameter.hpp>
```

Inheritance diagram for TermStructureFittingParameter:



9.815.1 Detailed Description

Deterministic time-dependent parameter used for yield-curve fitting.

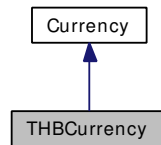
Public Member Functions

- **TermStructureFittingParameter** (const boost::shared_ptr< [Parameter::Impl](#) > &impl)
- **TermStructureFittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &term)

9.816 THBCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for THBCurrency:



9.816.1 Detailed Description

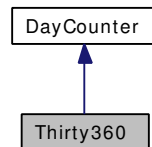
Thai baht.

The ISO three-letter code is THB; the numeric code is 764. It is divided in 100 stang.

9.817 Thirty360 Class Reference

```
#include <ql/time/daycounters/thirty360.hpp>
```

Inheritance diagram for Thirty360:



9.817.1 Detailed Description

30/360 day count convention

The 30/360 day count can be calculated according to US, European, or Italian conventions.

US (NASD) convention: if the starting date is the 31st of a month, it becomes equal to the 30th of the same month. If the ending date is the 31st of a month and the starting date is earlier than the 30th of a month, the ending date becomes equal to the 1st of the next month, otherwise the ending date becomes equal to the 30th of the same month. Also known as "30/360", "360/360", or "Bond Basis"

European convention: starting dates or ending dates that occur on the 31st of a month become equal to the 30th of the same month. Also known as "30E/360", or "Eurobond Basis"

Italian convention: starting dates or ending dates that occur on February and are greater than 27 become equal to 30 for computational sake.

Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

Public Types

- enum **Convention** {
 USA, **BondBasis**, **European**, **EurobondBasis**,
 Italian }

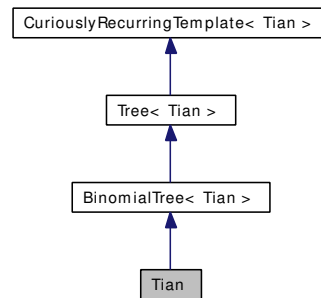
Public Member Functions

- Thirty360** (Convention c=Thirty360::BondBasis)

9.818 Tian Class Reference

```
#include <ql/methods/lattices/binomialtree.hpp>
```

Inheritance diagram for Tian:



9.818.1 Detailed Description

Tian tree: third moment matching, multiplicative approach

Public Member Functions

- **Tian** (const boost::shared_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size branch) const

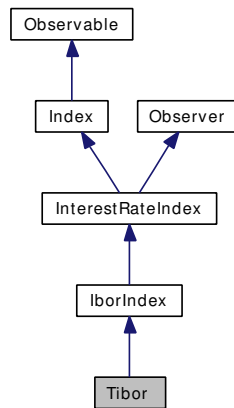
Protected Attributes

- Real **up_**
- Real **down_**
- Real **pu_**
- Real **pd_**

9.819 Tibor Class Reference

```
#include <ql/indexes/ibor/tibor.hpp>
```

Inheritance diagram for Tibor:



9.819.1 Detailed Description

JPY TIBOR index

Tokyo Interbank Offered Rate.

Warning

This is the rate fixed in Tokio by JBA. Use [JPYLibor](#) if you're interested in the London fixing by BBA.

Todo

check settlement days and end-of-month adjustment.

Public Member Functions

- **Tibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.820 TimeBasket Class Reference

```
#include <ql/cashflows/timebasket.hpp>
```

9.820.1 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

9.821 TimeGrid Class Reference

```
#include <ql/timegrid.hpp>
```

9.821.1 Detailed Description

time grid class

Todo

what was the rationale for limiting the grid to positive times? Investigate and see whether we can use it for negative ones as well.

Examples:

[BermudanSwaption.cpp](#).

sequence interface

- typedef std::vector< Time >::const_iterator **const_iterator**
- typedef std::vector< Time >::const_reverse_iterator **const_reverse_iterator**
- Time **operator[]** (Size i) const
- Time **at** (Size i) const
- Size **size** () const
- bool **empty** () const
- const_iterator **begin** () const
- const_iterator **end** () const
- const_reverse_iterator **rbegin** () const
- const_reverse_iterator **rend** () const
- Time **front** () const
- Time **back** () const

Public Member Functions

Constructors

- [TimeGrid](#) (Time end, Size steps)
Regularly spaced time-grid.
- template<class Iterator>
[TimeGrid](#) (Iterator begin, Iterator end)
Time grid with mandatory time points.
- template<class Iterator>
[TimeGrid](#) (Iterator begin, Iterator end, Size steps)
Time grid with mandatory time points.

Time grid interface

- Size [index](#) (Time t) const

returns the index i such that $grid[i] = t$

- Size `closestIndex` (Time t) const
returns the index i such that $grid[i]$ is closest to t
- Time `closestTime` (Time t) const
returns the time on the grid closest to the given t
- const std::vector< Time > & `mandatoryTimes` () const
- Time `dt` (Size i) const

9.821.2 Constructor & Destructor Documentation

9.821.2.1 TimeGrid (Iterator *begin*, Iterator *end*)

Time grid with mandatory time points.

Mandatory points are guaranteed to belong to the grid. No additional points are added.

9.821.2.2 TimeGrid (Iterator *begin*, Iterator *end*, Size *steps*)

Time grid with mandatory time points.

Mandatory points are guaranteed to belong to the grid. Additional points are then added with regular spacing between pairs of mandatory times in order to reach the desired number of steps.

9.822 TimeSeries Class Template Reference

```
#include <ql/timeseries.hpp>
```

9.822.1 Detailed Description

template<class T, class Container = std::map<Date, T>> class QuantLib::TimeSeries< T, Container >

Container for historical data.

This class acts as a generic repository for a set of historical data. Any single datum can be accessed through its date, while sets of consecutive data can be accessed through iterators.

Precondition:

The Container type must satisfy the requirements set by the C++ standard for associative containers.

Iterators

- typedef Container::const_iterator **const_iterator**
- typedef Container::const_reverse_iterator **const_reverse_iterator**
- const_iterator **begin** () const
- const_iterator **end** () const
- const_reverse_iterator **rbegin** () const
- const_reverse_iterator **rend** () const

Public Types

- typedef [Date](#) **key_type**
- typedef T **value_type**

Public Member Functions

- [TimeSeries](#) ()
- template<class DateIterator, class ValueIterator>
[TimeSeries](#) (DateIterator dBegin, DateIterator dEnd, ValueIterator vBegin)
- template<class ValueIterator>
[TimeSeries](#) (const [Date](#) &firstDate, ValueIterator begin, ValueIterator end)

Inspectors

- [Date](#) **firstDate** () const
returns the first date for which a historical datum exists
- [Date](#) **lastDate** () const
returns the last date for which a historical datum exists
- Size [size](#) () const

returns the number of historical data including null ones

- `bool empty () const`
returns whether the series contains any data

Historical data access

- `T operator[] (const Date &d) const`
returns the (possibly null) datum corresponding to the given date
- `T & operator[] (const Date &d)`

Utilities

- `const_iterator find (const Date &)`
- `std::vector< Date > dates () const`
- `std::vector< T > values () const`

9.822.2 Constructor & Destructor Documentation

9.822.2.1 TimeSeries ()

Default constructor

9.822.2.2 TimeSeries (DateIterator *dBegin*, DateIterator *dEnd*, ValueIterator *vBegin*)

This constructor initializes the history with a set of values passed as two sequences, the first containing dates and the second containing corresponding values.

9.822.2.3 TimeSeries (const Date &*firstDate*, ValueIterator *begin*, ValueIterator *end*)

This constructor initializes the history with a set of values. Such values are assigned to a corresponding number of consecutive dates starting from *firstDate* included.

9.823 TqrEigenDecomposition Class Reference

```
#include <ql/math/matrixutilities/tqreigendecomposition.hpp>
```

9.823.1 Detailed Description

tridiag. QR eigen decomposition with explicite shift aka Wilkinson

References:

Wilkinson, J.H. and Reinsch, C. 1971, [Linear](#) Algebra, vol. II of Handbook for Automatic Computation (New York: Springer-Verlag)

"Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

Tests

the correctness of the result is tested by checking it against known good values.

Public Types

- enum `EigenVectorCalculation` { `WithEigenVector`, `WithoutEigenVector`, `OnlyFirstRowEigenVector` }
- enum `ShiftStrategy` { `NoShift`, `Overrelaxation`, `CloseEigenValue` }

Public Member Functions

- `TqrEigenDecomposition` (const [Array](#) &diag, const [Array](#) &sub, EigenVectorCalculation calc=WithEigenVector, ShiftStrategy strategy=CloseEigenValue)
- const [Array](#) & `eigenvalues` () const
- const [Matrix](#) & `eigenvectors` () const
- Size `iterations` () const

9.824 TransformedGrid Class Reference

```
#include <ql/math/transformedgrid.hpp>
```

9.824.1 Detailed Description

transformed grid

This package encapsulates an array of grid points. It is used primarily in PDE calculations.

Public Member Functions

- **TransformedGrid** (const [Array](#) &grid)
- template<class T>
 TransformedGrid (const [Array](#) &grid, T func)
- const [Array](#) & **gridArray** () const
- const [Array](#) & **transformedGridArray** () const
- const [Array](#) & **dxmArray** () const
- const [Array](#) & **dxpArray** () const
- const [Array](#) & **dxArray** () const
- Real **grid** (Size i) const
- Real **transformedGrid** (Size i) const
- Real **dxm** (Size i) const
- Real **dxp** (Size i) const
- Real **dx** (Size i) const
- Size **size** () const

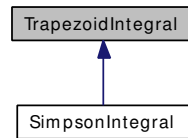
Protected Attributes

- [Array](#) **grid_**
- [Array](#) **transformedGrid_**
- [Array](#) **dxm_**
- [Array](#) **dxp_**
- [Array](#) **dx_**

9.825 TrapezoidIntegral Class Reference

```
#include <ql/math/integrals/trapezoidintegral.hpp>
```

Inheritance diagram for TrapezoidIntegral:



9.825.1 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 >())

Protected Member Functions

- Method **method** () const
- Method & **method** ()
- Real **integrate** (const boost::function< Real(Real)> &f, Real a, Real b) const
- Real **defaultIteration** (const boost::function< Real(Real)> &f, Real a, Real b, Real I, Size N) const
- Real **midPointIteration** (const boost::function< Real(Real)> &f, Real a, Real b, Real I, Size N) const

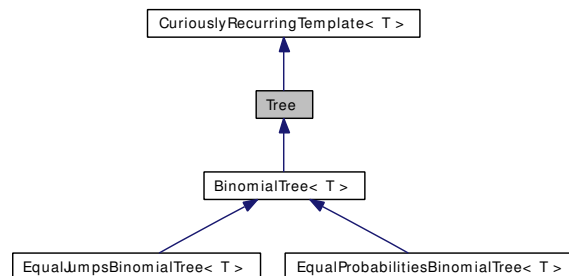
Protected Attributes

- Method **method_**

9.826 Tree Class Template Reference

```
#include <ql/methods/lattices/tree.hpp>
```

Inheritance diagram for Tree:



9.826.1 Detailed Description

```
template<class T> class QuantLib::Tree< T >
```

Tree approximating a single-factor diffusion

Derived classes must implement the following interface:

```

public:
    Real underlying(Size i, Size index) const;
    Size size(Size i) const;
    Size descendant(Size i, Size index, Size branch) const;
    Real probability(Size i, Size index, Size branch) const;
  
```

and provide a public enumeration

```
enum { branches = N };
```

where N is a suitable constant (2 for binomial, 3 for trinomial...)

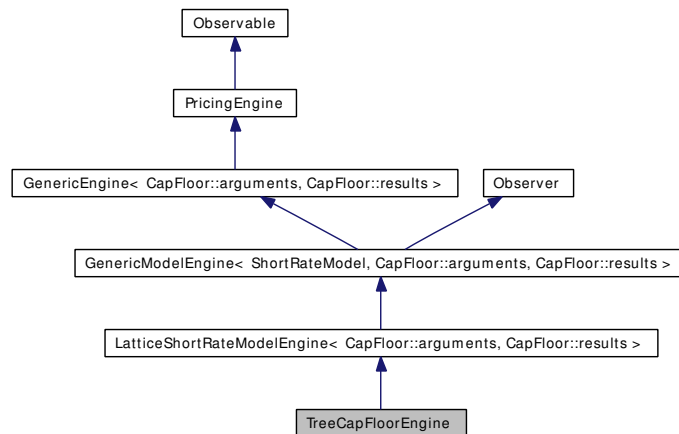
Public Member Functions

- **Tree** (Size columns)
- Size **columns** () const

9.827 TreeCapFloorEngine Class Reference

#include <ql/pricingengines/capfloor/treecapfloorengine.hpp>

Inheritance diagram for TreeCapFloorEngine:



9.827.1 Detailed Description

Numerical lattice engine for cap/floors.

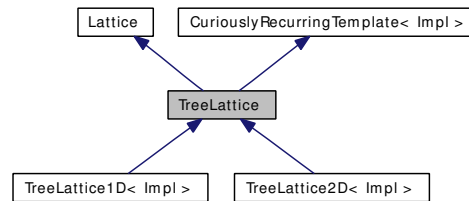
Public Member Functions

- **TreeCapFloorEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, Size timeSteps)
- **TreeCapFloorEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

9.828 TreeLattice Class Template Reference

```
#include <ql/methods/lattices/lattice.hpp>
```

Inheritance diagram for TreeLattice:



9.828.1 Detailed Description

template<class Impl> class QuantLib::TreeLattice< Impl >

Tree-based lattice-method base class.

This class defines a lattice method that is able to rollback (with discount) a discretized asset object. It will be based on one or more trees.

Derived classes must implement the following interface:

```
public:
    DiscountFactor discount(Size i, Size index) const;
    Size descendant(Size i, Size index, Size branch) const;
    Real probability(Size i, Size index, Size branch) const;
```

and may implement the following:

```
public:
    void stepback(Size i,
                  const Array& values,
                  Array& newValues) const;
```

Public Member Functions

- **TreeLattice** (const [TimeGrid](#) &timeGrid, Size n)
- const [Array](#) & **statePrices** (Size i) const
- void **stepback** (Size i, const [Array](#) &values, [Array](#) &newValues) const

Lattice interface

- void **initialize** ([DiscretizedAsset](#) &, Time t) const
initialize an asset at the given time.
- void **rollback** ([DiscretizedAsset](#) &, Time to) const
- void **partialRollback** ([DiscretizedAsset](#) &, Time to) const
- Real **presentValue** ([DiscretizedAsset](#) &) const
Computes the present value of an asset using Arrow-Debreu prices.

Protected Member Functions

- void **computeStatePrices** (Size until) const

Protected Attributes

- std::vector< [Array](#) > **statePrices_**

9.828.2 Member Function Documentation

9.828.2.1 void rollback (DiscretizedAsset &, Time to) const [virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implements [Lattice](#).

Reimplemented in [TsiveriotisFernandesLattice](#).

9.828.2.2 void partialRollback (DiscretizedAsset &, Time to) const [virtual]

Roll back an asset until the given time, but do not perform the final adjustment.

Warning

In version 0.3.7 and earlier, this method was called `rollAlmostBack` method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

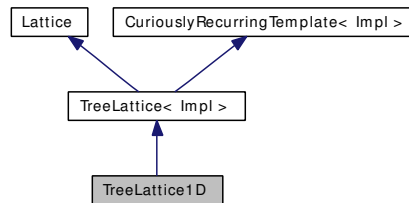
Implements [Lattice](#).

Reimplemented in [TsiveriotisFernandesLattice](#).

9.829 TreeLattice1D Class Template Reference

```
#include <ql/methods/lattices/lattice1d.hpp>
```

Inheritance diagram for TreeLattice1D:



9.829.1 Detailed Description

template<class Impl> class QuantLib::TreeLattice1D< Impl >

One-dimensional tree-based lattice.

Derived classes must implement the following interface:

```
Real underlying(Size i, Size index) const;
```

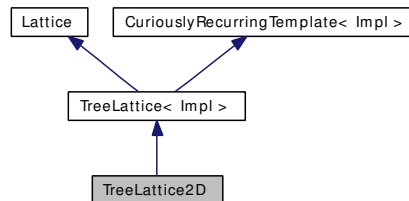
Public Member Functions

- **TreeLattice1D** (const [TimeGrid](#) &timeGrid, Size n)
- [Disposable](#)< [Array](#) > **grid** (Time t) const
- Real **underlying** (Size i, Size index) const

9.830 TreeLattice2D Class Template Reference

```
#include <ql/methods/lattices/lattice2d.hpp>
```

Inheritance diagram for TreeLattice2D:



9.830.1 Detailed Description

```
template<class Impl, class T = TrinomialTree> class QuantLib::TreeLattice2D< Impl, T >
```

Two-dimensional tree-based lattice.

This lattice is based on two trinomial trees and primarily used for the [G2](#) short-rate model.

Public Member Functions

- **TreeLattice2D** (const boost::shared_ptr< T > &tree1, const boost::shared_ptr< T > &tree2, Real correlation)
- Size **size** (Size i) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

Protected Member Functions

- [Disposable](#)< [Array](#) > **grid** (Time) const

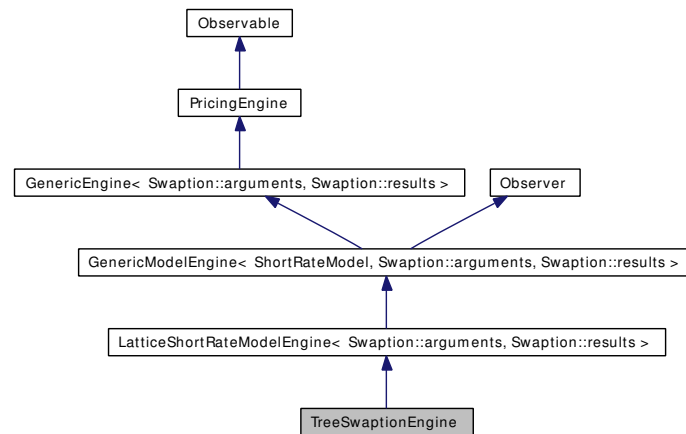
Protected Attributes

- boost::shared_ptr< T > **tree1_**
- boost::shared_ptr< T > **tree2_**

9.831 TreeSwaptionEngine Class Reference

```
#include <ql/pricingengines/swaption/treeswaptionengine.hpp>
```

Inheritance diagram for TreeSwaptionEngine:



9.831.1 Detailed Description

Numerical lattice engine for swaptions.

Warning

This engine is not guaranteed to work if the underlying swap has a start date in the past, i.e., before today's date. When using this engine, prune the initial part of the swap so that it starts at $t \geq 0$.

Tests

calculations are checked against cached results

Examples:

[BermudanSwaption.cpp](#).

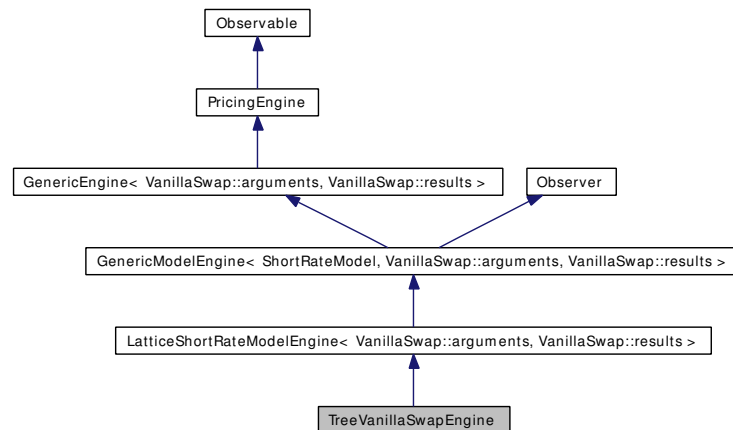
Public Member Functions

- **TreeSwaptionEngine** (const boost::shared_ptr< [ShortRateModel](#) > &, Size timeSteps)
- **TreeSwaptionEngine** (const boost::shared_ptr< [ShortRateModel](#) > &, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

9.832 TreeVanillaSwapEngine Class Reference

#include <ql/pricingengines/swaption/treeswaptionengine.hpp>

Inheritance diagram for TreeVanillaSwapEngine:



9.832.1 Detailed Description

Numerical lattice engine for simple swaps.

Tests

calculations are checked against known good results

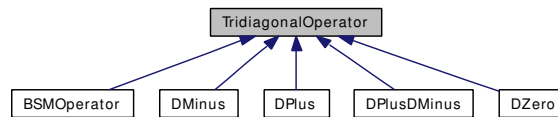
Public Member Functions

- **TreeVanillaSwapEngine** (const boost::shared_ptr< [ShortRateModel](#) > &, Size timeSteps)
- **TreeVanillaSwapEngine** (const boost::shared_ptr< [ShortRateModel](#) > &, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

9.833 TridiagonalOperator Class Reference

```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

Inheritance diagram for TridiagonalOperator:



9.833.1 Detailed Description

Base implementation for tridiagonal operator.

Warning

to use real time-dependant algebra, you must overload the corresponding operators in the inheriting time-dependent class.

Operator interface

- `Disposable< Array > applyTo (const Array &v) const`
apply operator to a given array
- `Disposable< Array > solveFor (const Array &rhs) const`
solve linear system for a given right-hand side
- `Disposable< Array > SOR (const Array &rhs, Real tol) const`
solve linear system with SOR approach
- static `Disposable< TridiagonalOperator > identity (Size size)`
identity instance

Public Types

- typedef `Array array_type`

Public Member Functions

- `TridiagonalOperator (Size size=0)`
- `TridiagonalOperator (const Array &low, const Array &mid, const Array &high)`
- `TridiagonalOperator (const Disposable< TridiagonalOperator > &)`
- `TridiagonalOperator & operator= (const Disposable< TridiagonalOperator > &)`

Inspectors

- `Size size () const`

- bool **isTimeDependent** ()
- const [Array](#) & **lowerDiagonal** () const
- const [Array](#) & **diagonal** () const
- const [Array](#) & **upperDiagonal** () const

Modifiers

- void **setFirstRow** (Real, Real)
- void **setMidRow** (Size, Real, Real, Real)
- void **setMidRows** (Real, Real, Real)
- void **setLastRow** (Real, Real)
- void **setTime** (Time t)

Utilities

- void **swap** ([TridiagonalOperator](#) &)

Protected Attributes

- [Array](#) **diagonal_**
- [Array](#) **lowerDiagonal_**
- [Array](#) **upperDiagonal_**
- boost::shared_ptr< [TimeSetter](#) > **timeSetter_**

Friends

- [Disposable](#)< [TridiagonalOperator](#) > **operator+** (const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator-** (const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator+** (const [TridiagonalOperator](#) &, const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator-** (const [TridiagonalOperator](#) &, const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator *** (Real, const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator *** (const [TridiagonalOperator](#) &, Real)
- [Disposable](#)< [TridiagonalOperator](#) > **operator/** (const [TridiagonalOperator](#) &, Real)

Classes

- class [TimeSetter](#)
encapsulation of time-setting logic

9.834 TridiagonalOperator::TimeSetter Class Reference

```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

9.834.1 Detailed Description

encapsulation of time-setting logic

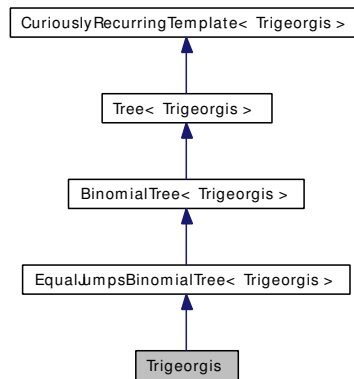
Public Member Functions

- virtual void **setTime** (Time t, [TridiagonalOperator](#) &L) const=0

9.835 Trigeorgis Class Reference

```
#include <ql/methods/lattices/binomialtree.hpp>
```

Inheritance diagram for Trigeorgis:



9.835.1 Detailed Description

Trigeorgis (additive equal jumps) binomial tree

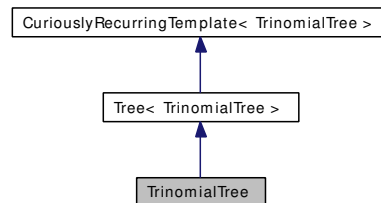
Public Member Functions

- **Trigeorgis** (const boost::shared_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)

9.836 TrinomialTree Class Reference

```
#include <ql/methods/lattices/trinomialtree.hpp>
```

Inheritance diagram for TrinomialTree:



9.836.1 Detailed Description

Recombining trinomial tree class.

This class defines a recombining trinomial tree approximating a 1-D stochastic process.

Warning

The diffusion term of the SDE must be independent of the underlying process.

Public Types

- enum **Branches** { **branches** = 3 }

Public Member Functions

- **TrinomialTree** (const boost::shared_ptr< [StochasticProcess1D](#) > &process, const [TimeGrid](#) &timeGrid, bool isPositive=false)
- Real **dx** (Size i) const
- const [TimeGrid](#) & **timeGrid** () const
- Size **size** (Size i) const
- Real **underlying** (Size i, Size index) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

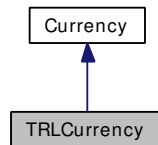
Protected Attributes

- std::vector< Branching > **branchings_**
- Real **x0_**
- std::vector< Real > **dx_**
- [TimeGrid](#) **timeGrid_**

9.837 TRLCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for TRLCurrency:



9.837.1 Detailed Description

Turkish lira.

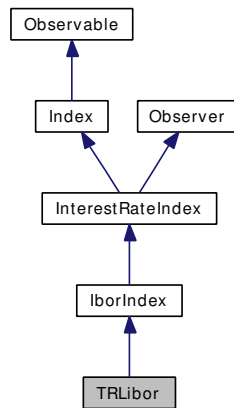
The ISO three-letter code was TRL; the numeric code was 792. It was divided in 100 kurus.

Obsoleted by the new Turkish lira since 2005.

9.838 TRLibor Class Reference

```
#include <ql/indexes/ibor/trlibor.hpp>
```

Inheritance diagram for TRLibor:



9.838.1 Detailed Description

TRY LIBOR rate

TRY LIBOR fixed by TBA.

See <<http://www.trlibor.org/trlibor/english/default.asp>>

Todo

check end-of-month adjustment.

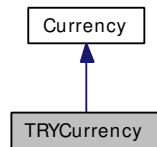
Public Member Functions

- **TRLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

9.839 TRYCurrency Class Reference

```
#include <ql/currencies/europe.hpp>
```

Inheritance diagram for TRYCurrency:



9.839.1 Detailed Description

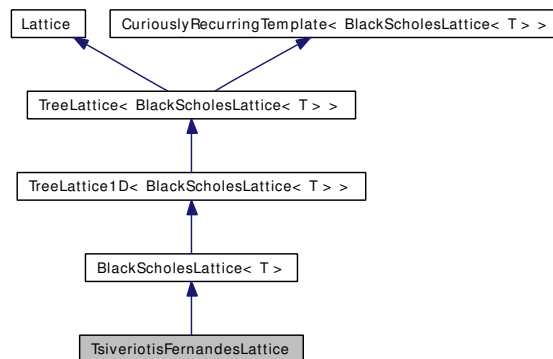
New Turkish lira.

The ISO three-letter code is TRY; the numeric code is 949. It is divided in 100 new kurus.

9.840 TsiveriotisFernandesLattice Class Template Reference

```
#include <ql/methods/lattices/tflattice.hpp>
```

Inheritance diagram for TsiveriotisFernandesLattice:



9.840.1 Detailed Description

```
template<class T> class QuantLib::TsiveriotisFernandesLattice< T >
```

Binomial lattice approximating the Tsiveriotis-Fernandes model.

Public Member Functions

- **TsiveriotisFernandesLattice** (const boost::shared_ptr< T > &tree, Rate riskFreeRate, Time end, Size steps, Real creditSpread, Volatility volatility, Spread divYield)
- Rate **riskFreeRate** () const
- Real **creditSpread** () const
- Real **dt** () const

Protected Member Functions

- void **stepback** (Size i, const [Array](#) &values, const [Array](#) &conversionProbability, const [Array](#) &spreadAdjustedRate, [Array](#) &newValues, [Array](#) &newConversionProbability, [Array](#) &newSpreadAdjustedRate) const
- void **rollback** ([DiscretizedAsset](#) &, Time to) const
- void **partialRollback** ([DiscretizedAsset](#) &, Time to) const

9.840.2 Member Function Documentation

9.840.2.1 void rollback ([DiscretizedAsset](#) &, Time to) const [protected, virtual]

Roll back an asset until the given time, performing any needed adjustment.

Reimplemented from [TreeLattice< BlackScholesLattice< T > >](#).

9.840.2.2 void partialRollback (DiscretizedAsset &, Time *to*) const [protected, virtual]

Roll back an asset until the given time, but do not perform the final adjustment.

Warning

In version 0.3.7 and earlier, this method was called `rollAlmostBack` method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

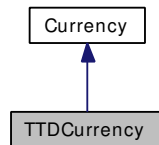
```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

Reimplemented from [TreeLattice< BlackScholesLattice< T > >](#).

9.841 TTDCurrency Class Reference

```
#include <ql/currencies/america.hpp>
```

Inheritance diagram for TTDCurrency:



9.841.1 Detailed Description

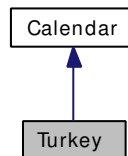
Trinidad & Tobago dollar.

The ISO three-letter code is TTD; the numeric code is 780. It is divided in 100 cents.

9.842 Turkey Class Reference

```
#include <ql/time/calendars/turkey.hpp>
```

Inheritance diagram for Turkey:



9.842.1 Detailed Description

Turkish calendar.

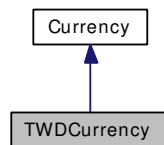
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- National Holidays (April 23rd, May 19th, August 30th, October 29th)
- Local Holidays (Kurban, Ramadan; 2004 to 2009 only)

9.843 TWDCurrency Class Reference

```
#include <ql/currencies/asia.hpp>
```

Inheritance diagram for TWDCurrency:



9.843.1 Detailed Description

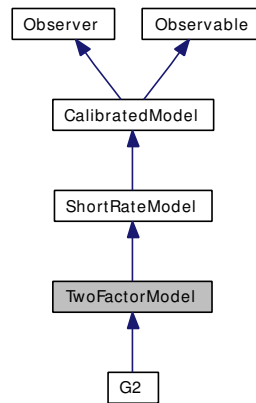
Taiwan dollar

The ISO three-letter code is TWD; the numeric code is 901. It is divided in 100 cents.

9.844 TwoFactorModel Class Reference

```
#include <ql/models/shortrate/twofactormodel.hpp>
```

Inheritance diagram for TwoFactorModel:



9.844.1 Detailed Description

Abstract base-class for two-factor models.

Public Member Functions

- **TwoFactorModel** (Size nParams)
- virtual boost::shared_ptr< [ShortRateDynamics](#) > [dynamics](#) () const=0
Returns the short-rate dynamics.
- boost::shared_ptr< [Lattice](#) > [tree](#) (const [TimeGrid](#) &grid) const
Returns a two-dimensional trinomial tree.

Classes

- class [ShortRateDynamics](#)
Class describing the dynamics of the two state variables.
- class [ShortRateTree](#)
Recombining two-dimensional tree discretizing the state variable.

9.845 TwoFactorModel::ShortRateDynamics Class Reference

```
#include <ql/models/shortrate/twofactormodel.hpp>
```

9.845.1 Detailed Description

Class describing the dynamics of the two state variables.

We assume here that the short-rate is a function of two state variables x and y .

$$r_t = f(t, x_t, y_t)$$

of two state variables x_t and y_t . These stochastic processes satisfy

$$x_t = \mu_x(t, x_t)dt + \sigma_x(t, x_t)dW_t^x$$

and

$$y_t = \mu_y(t, y_t)dt + \sigma_y(t, y_t)dW_t^y$$

where W^x and W^y are two brownian motions satisfying

$$dW_t^x dW_t^y = \rho dt$$

.

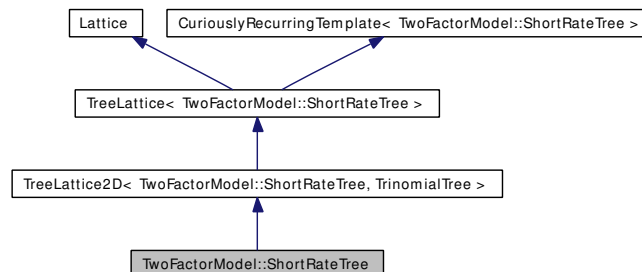
Public Member Functions

- **ShortRateDynamics** (const boost::shared_ptr< [StochasticProcess1D](#) > &xProcess, const boost::shared_ptr< [StochasticProcess1D](#) > &yProcess, Real correlation)
- virtual Rate **shortRate** (Time t, Real x, Real y) const=0
- const boost::shared_ptr< [StochasticProcess1D](#) > & **xProcess** () const
Risk-neutral dynamics of the first state variable x.
- const boost::shared_ptr< [StochasticProcess1D](#) > & **yProcess** () const
Risk-neutral dynamics of the second state variable y.
- Real **correlation** () const
Correlation ρ between the two brownian motions.
- boost::shared_ptr< [StochasticProcess](#) > **process** () const
Joint process of the two variables.

9.846 TwoFactorModel::ShortRateTree Class Reference

```
#include <ql/models/shortrate/twofactormodel.hpp>
```

Inheritance diagram for TwoFactorModel::ShortRateTree:



9.846.1 Detailed Description

Recombining two-dimensional tree discretizing the state variable.

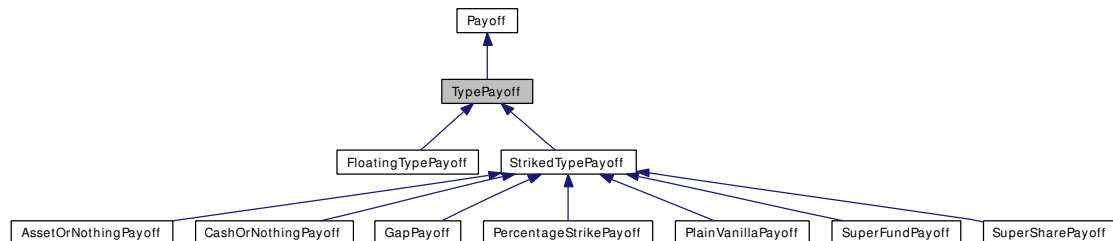
Public Member Functions

- [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

9.847 TypePayoff Class Reference

```
#include <ql/instruments/payoffs.hpp>
```

Inheritance diagram for TypePayoff:



9.847.1 Detailed Description

Intermediate class for put/call payoffs.

Public Member Functions

- `TypePayoff` (`Option::Type` type)
- `Option::Type` `optionType` () const

Payoff interface

- `std::string` `description` () const

Protected Attributes

- `Option::Type` `type_`

9.848 Ukraine Class Reference

```
#include <ql/time/calendars/ukraine.hpp>
```

Inheritance diagram for Ukraine:



9.848.1 Detailed Description

Ukrainian calendars.

Holidays for the Ukrainian stock exchange (data from <http://www.ukrse.kiev.ua/eng/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Orthodox Christmas, January 7th
- International Women's Day, March 8th
- Easter Monday
- Holy Trinity Day, 50 days after Easter
- International Workers' Solidarity Days, May 1st and 2nd
- Victory Day, May 9th
- Constitution Day, June 28th
- Independence Day, August 24th

Holidays falling on a Saturday or Sunday are moved to the following Monday.

Public Types

- enum [Market](#) { [USE](#) }

Public Member Functions

- [Ukraine](#) ([Market](#) m=USE)

9.848.2 Member Enumeration Documentation

9.848.2.1 enum Market

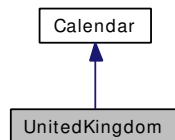
Enumerator:

USE Ukrainian stock exchange.

9.849 UnitedKingdom Class Reference

```
#include <ql/time/calendars/unitedkingdom.hpp>
```

Inheritance diagram for UnitedKingdom:



9.849.1 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](#))

9.849.2 Member Enumeration Documentation

9.849.2.1 enum Market

UK calendars.

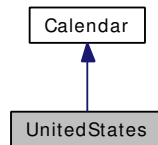
Enumerator:

Settlement generic settlement calendar
Exchange London stock-exchange calendar.

9.850 UnitedStates Class Reference

```
#include <ql/time/calendars/unitedstates.hpp>
```

Inheritance diagram for UnitedStates:



9.850.1 Detailed Description

United States calendars.

Public holidays (see: <http://www.opm.gov/fedhol/>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday, or to Friday if on Saturday)
- Martin Luther King's birthday, third Monday in January
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Columbus Day, second Monday in October
- Veterans' Day, November 11th (moved to Monday if Sunday or Friday if Saturday)
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)

Holidays for the stock exchange (data from <http://www.nyse.com>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Martin Luther King's birthday, third Monday in January (since 1998)
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Good Friday
- Memorial Day, last Monday in May

- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Thanksgiving Day, fourth Thursday in November
- Presidential election day, first Tuesday in November of election years (until 1980)
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)
- Special historic closings (see <http://www.nyse.com/about/1022221392381.html>)

Holidays for the government bond market (data from <http://www.bondmarkets.com>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Martin Luther King's birthday, third Monday in January
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Good Friday
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Columbus Day, second Monday in October
- Veterans' Day, November 11th (moved to Monday if Sunday or Friday if Saturday)
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)

Holidays for the North American Energy Reliability Council (data from <http://www.nerc.com/~oc/offpeaks.html>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday)
- Labor Day, first Monday in September
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday)

Tests

the correctness of the returned results is tested against a list of known holidays.

Public Types

- enum [Market](#) { [Settlement](#), [NYSE](#), [GovernmentBond](#), [NERC](#) }
US calendars.

Public Member Functions

- [UnitedStates](#) ([Market](#) market=[Settlement](#))

9.850.2 Member Enumeration Documentation

9.850.2.1 enum Market

US calendars.

Enumerator:

Settlement generic settlement calendar

NYSE New York stock exchange calendar.

GovernmentBond government-bond calendar

NERC off-peak days for NERC

9.851 UpperBoundEngine Class Reference

```
#include <ql/models/marketmodels/callability/upperboundengine.hpp>
```

9.851.1 Detailed Description

Market-model engine for upper-bound estimation.

Precondition:

product and hedge must have the same rate times and exercise times

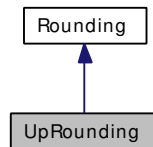
Public Member Functions

- **UpperBoundEngine** (const boost::shared_ptr< [MarketModelEvolver](#) > &evolver, const std::vector< boost::shared_ptr< [MarketModelEvolver](#) > > &innerEvolvers, const [MarketModelMultiProduct](#) &underlying, const MarketModelExerciseValue &rebate, const [MarketModelMultiProduct](#) &hedge, const MarketModelExerciseValue &hedgeRebate, const ExerciseStrategy< [CurveState](#) > &hedgeStrategy, Real initialNumeraireValue)
- void **multiplePathValues** ([Statistics](#) &stats, Size outerPaths, Size innerPaths)
- std::pair< Real, Real > **singlePathValue** (Size innerPaths)

9.852 UpRounding Class Reference

```
#include <ql/math/rounding.hpp>
```

Inheritance diagram for UpRounding:



9.852.1 Detailed Description

Up-rounding.

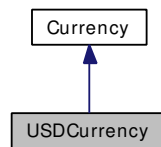
Public Member Functions

- **UpRounding** (Integer precision, Integer digit=5)

9.853 USDCurrency Class Reference

```
#include <ql/currencies/america.hpp>
```

Inheritance diagram for USDCurrency:



9.853.1 Detailed Description

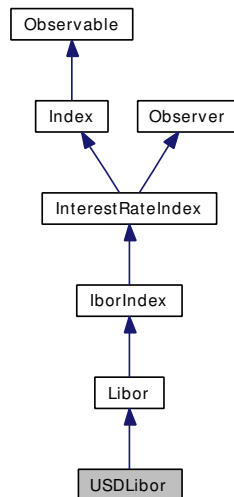
U.S. dollar.

The ISO three-letter code is USD; the numeric code is 840. It is divided in 100 cents.

9.854 USDLibor Class Reference

```
#include <ql/indexes/ibor/usdlibor.hpp>
```

Inheritance diagram for USDLibor:



9.854.1 Detailed Description

USD LIBOR rate

US Dollar LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

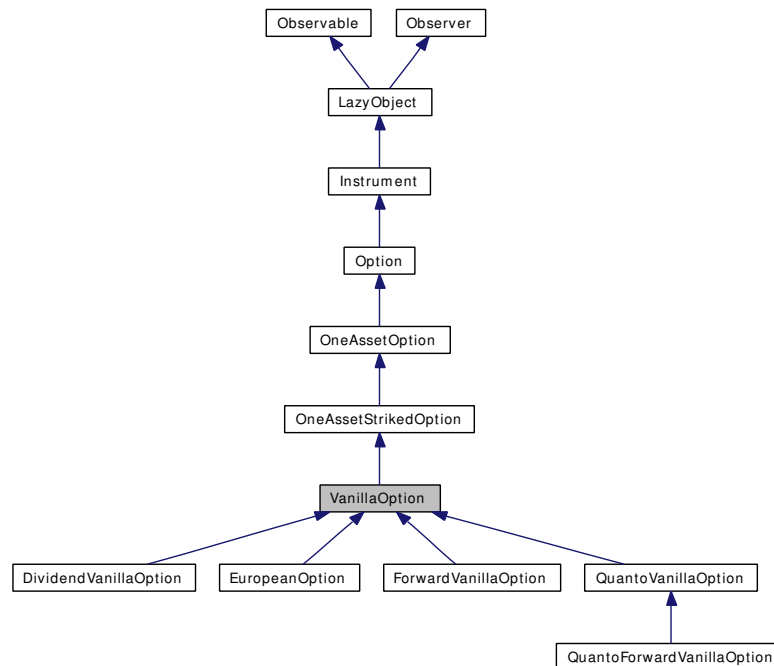
Public Member Functions

- **USDLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), Natural settlementDays=2)

9.855 VanillaOption Class Reference

```
#include <ql/instruments/vanillaoption.hpp>
```

Inheritance diagram for VanillaOption:



9.855.1 Detailed Description

Vanilla option (no discrete dividends, no barriers) on a single asset.

Examples:

[EquityOption.cpp](#).

Public Member Functions

- **VanillaOption** (const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &, const boost::shared_ptr< [Exercise](#) > &, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())

Classes

- class [engine](#)
Vanilla option engine base class.

```
#include <ql/instruments/vanillaoption.hpp>
```

```

graph TD
    MCQEngine[MCQEngine]
    AnalyticEngine[AnalyticEngine]
    SemiAdaptiveApproximationEngine[SemiAdaptiveApproximationEngine]
    SimplifiedStandardApproximationEngine[SimplifiedStandardApproximationEngine]
    FDRestimatorEngine[FDRestimatorEngine]
    Jmp2DistributionsEngine[Jmp2DistributionsEngine]
    JmpQuadratureApproximationEngine[JmpQuadratureApproximationEngine]
    MCMCParallelSemiAdaptiveEngine[MCMCParallelSemiAdaptiveEngine]
    VariationalEngine[VariationalEngine]
    SingleVariable_Engine[SingleVariable_Engine]
    MCParallelEngine_MultiVariable_Engine[MCParallelEngine_MultiVariable_Engine]
    MCParallelEngine_SingleVariable_Engine[MCParallelEngine_SingleVariable_Engine]
    MCMCParallelEngine[MCMCParallelEngine]
    MCQuadratureEngine1[MCQuadratureEngine]
    MCMCParallelEngine2[MCMCParallelEngine]
    MCQuadratureEngine2[MCQuadratureEngine]

    MCQEngine --> AnalyticEngine
    MCQEngine --> SemiAdaptiveApproximationEngine
    MCQEngine --> SimplifiedStandardApproximationEngine
    MCQEngine --> FDRestimatorEngine
    MCQEngine --> Jmp2DistributionsEngine
    MCQEngine --> JmpQuadratureApproximationEngine
    MCQEngine --> MCMCParallelSemiAdaptiveEngine
    MCQEngine --> VariationalEngine
    MCQEngine --> SingleVariable_Engine
    MCQEngine --> MCParallelEngine_MultiVariable_Engine
    MCQEngine --> MCParallelEngine_SingleVariable_Engine
    MCQEngine --> MCMCParallelEngine
    MCQEngine --> MCQuadratureEngine1
    MCQEngine --> MCMCParallelEngine2
    MCQEngine --> MCQuadratureEngine2

    MCMCParallelSemiAdaptiveEngine --> MCMCParallelEngine
    VariationalEngine --> MCMCParallelEngine
    SingleVariable_Engine --> MCMCParallelEngine
    MCParallelEngine_MultiVariable_Engine --> MCMCParallelEngine
    MCParallelEngine_SingleVariable_Engine --> MCMCParallelEngine
    MCMCParallelEngine --> MCMCParallelEngine
    MCQuadratureEngine1 --> MCMCParallelEngine
    MCMCParallelEngine2 --> MCMCParallelEngine
    MCQuadratureEngine2 --> MCMCParallelEngine
  
```

Vanilla option engine base class.

Generated on Fri Jun 1 18:19:44 2007 for QuantLib by Doxygen

9.857 VanillaSwap::arguments Class Reference

```
#include <ql/instruments/vanillaswap.hpp>
```

9.857.1 Detailed Description

Arguments for simple swap calculation

Public Member Functions

- void **validate** () const

Public Attributes

- Type **type**
- Real **nominal**
- std::vector< Time > **fixedResetTimes**
- std::vector< Time > **fixedPayTimes**
- std::vector< Real > **fixedCoupons**
- std::vector< Time > **floatingAccrualTimes**
- std::vector< Time > **floatingResetTimes**
- std::vector< Time > **floatingFixingTimes**
- std::vector< Time > **floatingPayTimes**
- std::vector< Spread > **floatingSpreads**
- Real **currentFloatingCoupon**

9.858 VanillaSwap::results Class Reference

```
#include <ql/instruments/vanillaswap.hpp>
```

9.858.1 Detailed Description

Results from simple swap calculation

Public Member Functions

- void **reset** ()

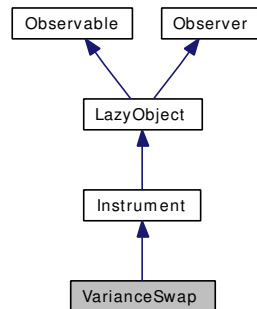
Public Attributes

- Real **fixedLegBPS**
- Real **floatingLegBPS**
- Rate **fairRate**
- Spread **fairSpread**

9.859 VarianceSwap Class Reference

```
#include <ql/instruments/varianceswap.hpp>
```

Inheritance diagram for VarianceSwap:



9.859.1 Detailed Description

Variance swap.

Warning

This class does not manage seasoned variance swaps.

Public Types

- typedef std::vector< std::pair< boost::shared_ptr< [StrikedTypePayoff](#) >, Real > > **WeightsType**

Public Member Functions

- **VarianceSwap** (Position::Type position, Real strike, Real notional, const boost::shared_ptr< [StochasticProcess](#) > &process, const [Date](#) &maturityDate, const boost::shared_ptr< [PricingEngine](#) > &engine)
- void **setupArguments** ([PricingEngine::arguments](#) *args) const
- void **fetchResults** (const PricingEngine::results *) const

Instrument interface

- bool **isExpired** () const
returns whether the instrument is still tradable.

Additional interface

- Real **strike** () const
- Position::Type **position** () const
- [Date](#) **maturityDate** () const
- [Date](#) **settlementDate** () const
- Real **notional** () const
- Real **fairVariance** () const
- std::vector< std::pair< Real, Real > > **optionWeights** (Option::Type) const

Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

Protected Attributes

- boost::shared_ptr< [GeneralizedBlackScholesProcess](#) > **process_**
- Position::Type **position_**
- Real **strike_**
- Real **notional_**
- [Date](#) **maturityDate_**
- WeightsType **optionWeights_**
- Real **fairVariance_**

Classes

- class [arguments](#)
Arguments for forward fair-variance calculation
- class [engine](#)
base class for variance-swap engines
- class [results](#)
Results from variance-swap calculation

9.859.2 Member Function Documentation

9.859.2.1 void fetchResults (const PricingEngine::results *) const [virtual]

When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing [engine](#) is used.

Reimplemented from [Instrument](#).

9.859.2.2 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from [Instrument](#).

9.859.2.3 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](#).

9.860 VarianceSwap::arguments Class Reference

```
#include <ql/instruments/varianceswap.hpp>
```

9.860.1 Detailed Description

Arguments for forward fair-variance calculation

Public Member Functions

- void **validate** () const

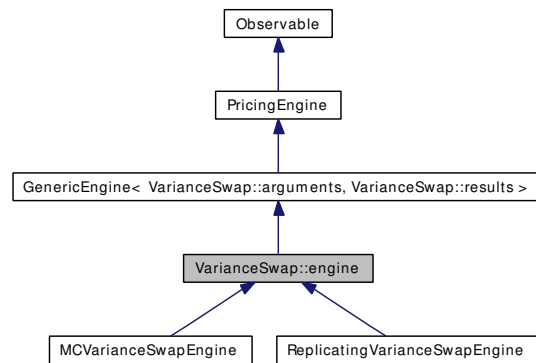
Public Attributes

- boost::shared_ptr< [GeneralizedBlackScholesProcess](#) > **stochasticProcess**
- Position::Type **position**
- Real **strike**
- Real **notional**
- [Date](#) **maturityDate**

9.861 VarianceSwap::engine Class Reference

```
#include <ql/instruments/varianceswap.hpp>
```

Inheritance diagram for VarianceSwap::engine:



9.861.1 Detailed Description

base class for variance-swap engines

9.862 VarianceSwap::results Class Reference

```
#include <ql/instruments/varianceswap.hpp>
```

9.862.1 Detailed Description

Results from variance-swap calculation

Public Member Functions

- void **reset** ()

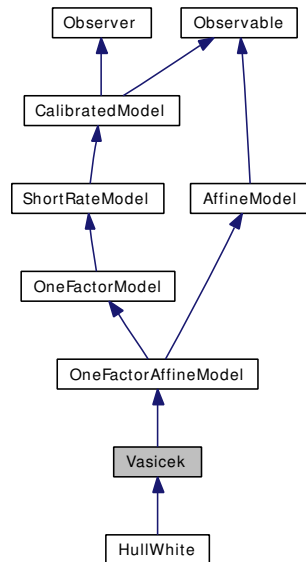
Public Attributes

- Real **fairVariance**
- WeightsType **optionWeights**
- WeightsType::const_iterator **iterator**

9.863 Vasicek Class Reference

```
#include <ql/models/shortrate/onefactormodels/vasicek.hpp>
```

Inheritance diagram for Vasicek:



9.863.1 Detailed Description

Vasicek model class

This class implements the [Vasicek](#) model defined by

$$dr_t = a(b - r_t)dt + \sigma dW_t,$$

where a , b and σ are constants; a risk premium λ can also be specified.

Public Member Functions

- **Vasicek** (Rate $r_0=0.05$, Real $a=0.1$, Real $b=0.05$, Real $\sigma=0.01$, Real $\lambda=0.0$)
- virtual Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const
- virtual boost::shared_ptr< ShortRateDynamics > **dynamics** () const
returns the short-rate dynamics

Protected Member Functions

- virtual Real **A** (Time t , Time T) const
- virtual Real **B** (Time t , Time T) const
- Real **a** () const
- Real **b** () const
- Real **lambda** () const
- Real **sigma** () const

Protected Attributes

- Real `r0_`
- [Parameter](#) & `a_`
- [Parameter](#) & `b_`
- [Parameter](#) & `sigma_`
- [Parameter](#) & `lambda_`

Classes

- class [Dynamics](#)
Short-rate dynamics in the Vasicek model.

9.864 Vasicek::Dynamics Class Reference

```
#include <ql/models/shortrate/onefactormodels/vasicek.hpp>
```

9.864.1 Detailed Description

Short-rate dynamics in the Vasicek model.

The short-rate follows an Ornstein-Uhlenbeck process with mean b .

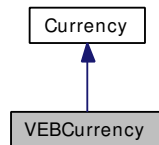
Public Member Functions

- **Dynamics** (Real a , Real b , Real σ , Real r_0)
- virtual Real **variable** (Time, Rate r) const
- virtual Real **shortRate** (Time, Real x) const

9.865 VEBCurrency Class Reference

```
#include <ql/currencies/america.hpp>
```

Inheritance diagram for VEBCurrency:



9.865.1 Detailed Description

Venezuelan bolivar.

The ISO three-letter code is VEB; the numeric code is 862. It is divided in 100 centimos.

9.866 Visitor Class Template Reference

```
#include <ql/patterns/visitor.hpp>
```

9.866.1 Detailed Description

```
template<class T> class QuantLib::Visitor< T >
```

Visitor for a specific class

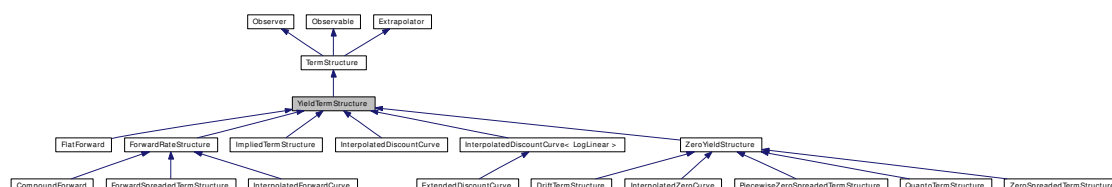
Public Member Functions

- virtual void **visit** (T &)=0

9.867 YieldTermStructure Class Reference

```
#include <ql/yieldtermstructure.hpp>
```

Inheritance diagram for YieldTermStructure:



9.867.1 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

Constructors

See the [TermStructure](#) documentation for issues regarding constructors.

- [YieldTermStructure](#) (const [DayCounter](#) &dc=[Actual365Fixed](#)())
default constructor
- [YieldTermStructure](#) (const [Date](#) &referenceDate, const [Calendar](#) &cal=[Calendar](#)(), const [DayCounter](#) &dc=[Actual365Fixed](#)())
initialize with a fixed reference date
- [YieldTermStructure](#) (Natural settlementDays, const [Calendar](#) &, const [DayCounter](#) &dc=[Actual365Fixed](#)())
calculate the reference date based on the global evaluation date

zero-yield rates

These methods return the implied zero-yield rate for a given date or time. In the former case, the time is calculated as a fraction of year from the reference date.

- [InterestRate](#) zeroRate (const [Date](#) &d, const [DayCounter](#) &resultDayCounter, Compounding comp, [Frequency](#) freq=[Annual](#), bool extrapolate=false) const

- [InterestRate](#) [zeroRate](#) (Time t, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const

discount factors

These methods return the discount factor for a given date or time. In the former case, the time is calculated as a fraction of year from the reference date.

- DiscountFactor **discount** (const [Date](#) &, bool extrapolate=false) const
- DiscountFactor [discount](#) (Time, bool extrapolate=false) const

forward rates

These methods returns the implied forward interest rate between two dates or times. In the former case, times are calculated as fractions of year from the reference date.

- [InterestRate](#) [forwardRate](#) (const [Date](#) &d1, const [Date](#) &d2, const [DayCounter](#) &resultDayCounter, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const
- [InterestRate](#) [forwardRate](#) (const [Date](#) &d, const [Period](#) &p, const [DayCounter](#) &resultDayCounter, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const
- [InterestRate](#) [forwardRate](#) (Time t1, Time t2, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const

par rates

These methods returns the implied par rate for a given sequence of payments at the given dates or times. In the former case, times are calculated as fractions of year from the reference date.

Warning

though somewhat related to a swap rate, this method is not to be used for the fair rate of a real swap, since it does not take into account all the market conventions' details. The correct way to evaluate such rate is to instantiate a [SimpleSwap](#) with the correct conventions, pass it the term structure and call the swap's [fairRate\(\)](#) method.

- Rate **parRate** (Integer tenor, const [Date](#) &startDate, [Frequency](#) freq=Annual, bool extrapolate=false) const
- Rate [parRate](#) (const std::vector< [Date](#) > &dates, [Frequency](#) freq=Annual, bool extrapolate=false) const
- Rate [parRate](#) (const std::vector< Time > ×, [Frequency](#) freq=Annual, bool extrapolate=false) const

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

9.867.2 Constructor & Destructor Documentation

9.867.2.1 YieldTermStructure (const DayCounter & *dc* = Actual365Fixed())

default constructor

Warning

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

9.867.3 Member Function Documentation

9.867.3.1 InterestRate zeroRate (const Date & *d*, const DayCounter & *resultDayCounter*, Compounding *comp*, Frequency *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the required daycounting rule.

9.867.3.2 InterestRate zeroRate (Time *t*, Compounding *comp*, Frequency *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the same day-counting rule used by the term structure. The same rule should be used for calculating the passed time *t*.

9.867.3.3 DiscountFactor discount (Time, bool *extrapolate* = false) const

The same day-counting rule used by the term structure should be used for calculating the passed time *t*.

9.867.3.4 InterestRate forwardRate (const Date & *d1*, const Date & *d2*, const DayCounter & *resultDayCounter*, Compounding *comp*, Frequency *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the required day-counting rule.

9.867.3.5 InterestRate forwardRate (const Date & *d*, const Period & *p*, const DayCounter & *resultDayCounter*, Compounding *comp*, Frequency *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the required day-counting rule.

Warning

dates are not adjusted for holidays

9.867.3.6 InterestRate forwardRate (Time *t1*, Time *t2*, Compounding *comp*, Frequency *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the same day-counting rule used by the term structure. The same rule should be used for the calculating the passed times *t1* and *t2*.

9.867.3.7 `Rate parRate (const std::vector< Date > & dates, Frequency freq = Annual, bool extrapolate = false) const`

the first date in the vector must equal the start date; the following dates must equal the payment dates.

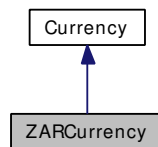
9.867.3.8 `Rate parRate (const std::vector< Time > & times, Frequency freq = Annual, bool extrapolate = false) const`

the first time in the vector must equal the start time; the following times must equal the payment times.

9.868 ZARCurrency Class Reference

```
#include <ql/currencies/africa.hpp>
```

Inheritance diagram for ZARCurrency:



9.868.1 Detailed Description

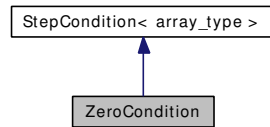
South-African rand.

The ISO three-letter code is ZAR; the numeric code is 710. It is divided into 100 cents.

9.869 ZeroCondition Class Template Reference

```
#include <ql/methods/finitedifferences/zerocondition.hpp>
```

Inheritance diagram for ZeroCondition:



9.869.1 Detailed Description

```
template<class array_type> class QuantLib::ZeroCondition< array_type >
```

Zero exercise condition.

Used in CEV models

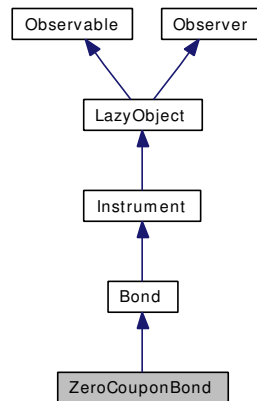
Public Member Functions

- void **applyTo** (array_type &a, Time) const

9.870 ZeroCouponBond Class Reference

```
#include <ql/instruments/zerocouponbond.hpp>
```

Inheritance diagram for ZeroCouponBond:



9.870.1 Detailed Description

zero-coupon bond

Tests

calculations are tested by checking results against cached values.

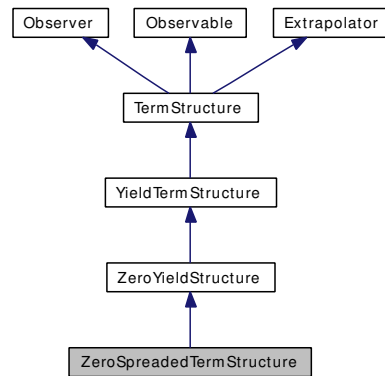
Public Member Functions

- **ZeroCouponBond** (Natural settlementDays, Real faceAmount, const [Calendar](#) &calendar, const [Date](#) &maturityDate, const [DayCounter](#) &dayCounter, [BusinessDayConvention](#) paymentConvention=Following, Real redemption=100.0, const [Date](#) &issueDate=[Date](#)(), const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())

9.871 ZeroSpreadedTermStructure Class Reference

```
#include <ql/termstructures/yieldcurves/zerospreadedtermstructure.hpp>
```

Inheritance diagram for ZeroSpreadedTermStructure:



9.871.1 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, Compounding comp=Continuous, [Frequency](#) freq=NoFrequency, const [DayCounter](#) &dc=[DayCounter](#)())

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- const [Date](#) & [referenceDate](#) () const
the date at which discount = 1.0 and/or variance = 0.0

- [Date maxDate](#) () const
the latest date for which the curve can return values
- [Time maxTime](#) () const
the latest time for which the curve can return values

Protected Member Functions

- [Rate zeroYieldImpl](#) (Time) const
returns the spreaded zero yield rate
- [Rate forwardImpl](#) (Time) const
returns the spreaded forward rate

9.872 ZeroYield Struct Reference

```
#include <ql/termstructures/yieldcurves/bootstraptraits.hpp>
```

9.872.1 Detailed Description

Zero-curve traits.

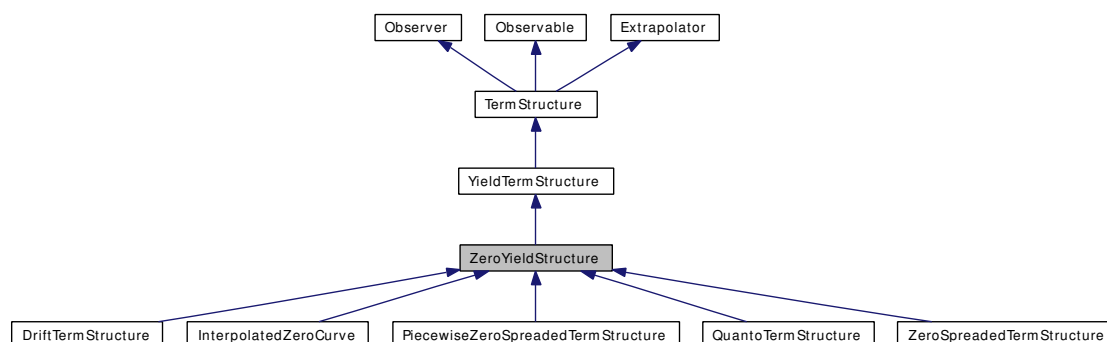
Static Public Member Functions

- static Rate **initialValue** ()
- static Rate **initialGuess** ()
- static Rate **guess** (const [YieldTermStructure](#) *c, const [Date](#) &d)
- static Rate **minValueAfter** (Size, const std::vector< Real > &)
- static Rate **maxValueAfter** (Size, const std::vector< Real > &)
- static void **updateGuess** (std::vector< Rate > &data, Rate rate, Size i)

9.873 ZeroYieldStructure Class Reference

```
#include <ql/termstructures/yieldcurves/zeroyieldstructure.hpp>
```

Inheritance diagram for ZeroYieldStructure:



9.873.1 Detailed Description

Zero-yield term structure.

This abstract class acts as an adapter to [YieldTermStructure](#) allowing the programmer to implement only the `zeroYieldImpl(Time, bool)` method in derived classes. [Discount](#) and forward are calculated from zero yields.

Rates are assumed to be annual continuous compounding.

Public Member Functions

Constructors

See the [TermStructure](#) documentation for issues regarding constructors.

- **ZeroYieldStructure** (const [DayCounter](#) &dc=[Actual365Fixed](#)())
- **ZeroYieldStructure** (const [Date](#) &referenceDate, const [Calendar](#) &calendar=[Calendar](#)(), const [DayCounter](#) &dc=[Actual365Fixed](#)())
- **ZeroYieldStructure** (Natural settlementDays, const [Calendar](#) &, const [DayCounter](#) &dc=[Actual365Fixed](#)())

Protected Member Functions

YieldTermStructure implementation

- DiscountFactor [discountImpl](#) (Time) const
- virtual Rate [zeroYieldImpl](#) (Time) const=0
zero-yield calculation

9.873.2 Member Function Documentation

9.873.2.1 DiscountFactor discountImpl (Time) const [protected, virtual]

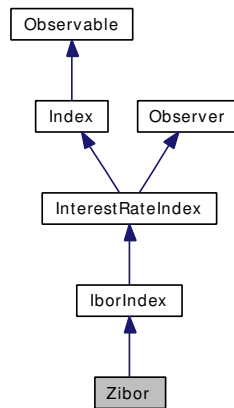
Returns the discount factor for the given date calculating it from the zero yield.

Implements [YieldTermStructure](#).

9.874 Zibor Class Reference

```
#include <ql/indexes/ibor/zibor.hpp>
```

Inheritance diagram for Zibor:



9.874.1 Detailed Description

CHF ZIBOR rate

Zurich Interbank Offered Rate.

Warning

This is the rate fixed in Zurich by BBA. Use [CHFLibor](#) if you're interested in the London fixing by BBA.

Todo

check settlement days, end-of-month adjustment, and day-count convention.

Public Member Functions

- **Zibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

Chapter 10

QuantLib File Documentation

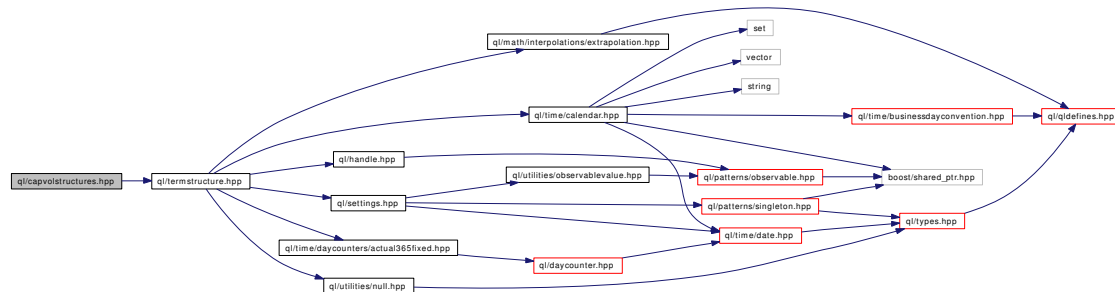
10.1 ql/capvolstructures.hpp File Reference

10.1.1 Detailed Description

cap/floor volatility structures

#include <ql/termstructure.hpp>

Include dependency graph for capvolstructures.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CapVolatilityStructure**
Cap/floor term-volatility structure.
- class **CapletVolatilityStructure**
Caplet/floorlet forward-volatility structure.

10.2 ql/cashflow.hpp File Reference

10.2.1 Detailed Description

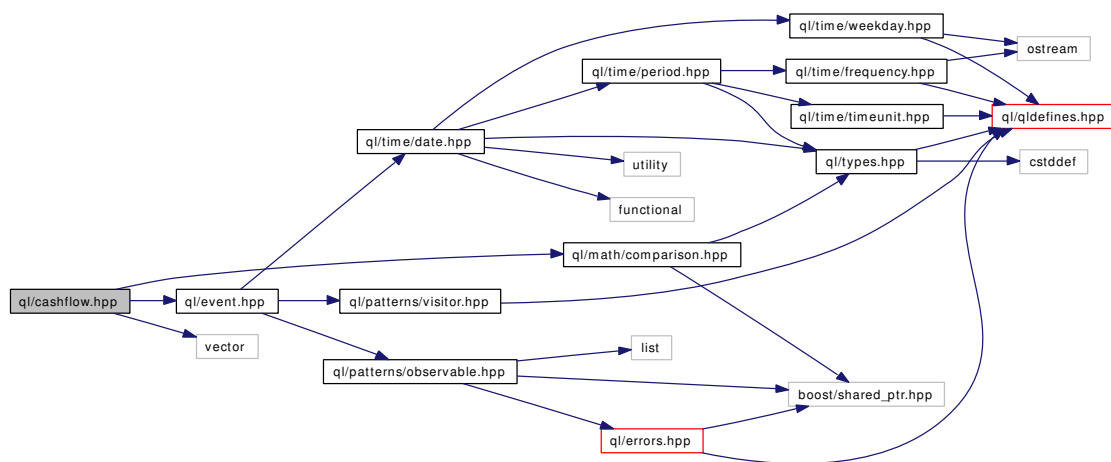
Base class for cash flows.

```
#include <ql/event.hpp>
```

```
#include <ql/math/comparison.hpp>
```

```
#include <vector>
```

Include dependency graph for cashflow.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CashFlow](#)
Base class for cash flows.

Typedefs

- typedef `std::vector< boost::shared_ptr< CashFlow > >` **Leg**

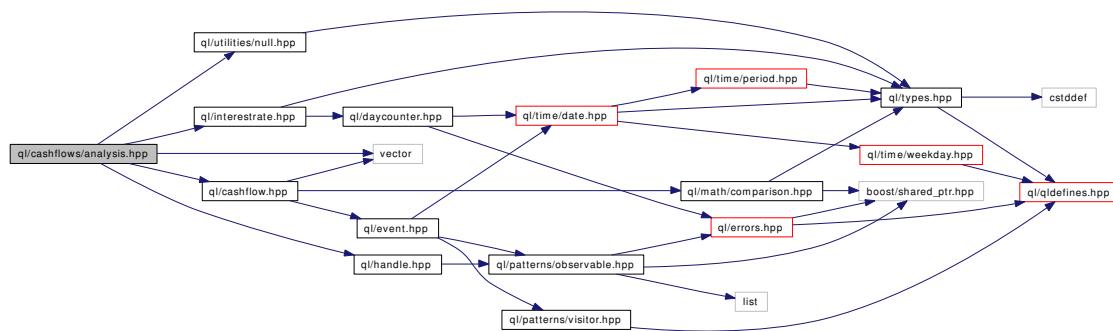
10.3 ql/cashflows/analysis.hpp File Reference

10.3.1 Detailed Description

Cash-flow analysis functions.

```
#include <ql/cashflow.hpp>
#include <ql/interestrate.hpp>
#include <ql/handle.hpp>
#include <ql/utilities/null.hpp>
#include <vector>
```

Include dependency graph for analysis.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct [Duration](#)
duration type
- class [CashFlows](#)
cashflow-analysis functions

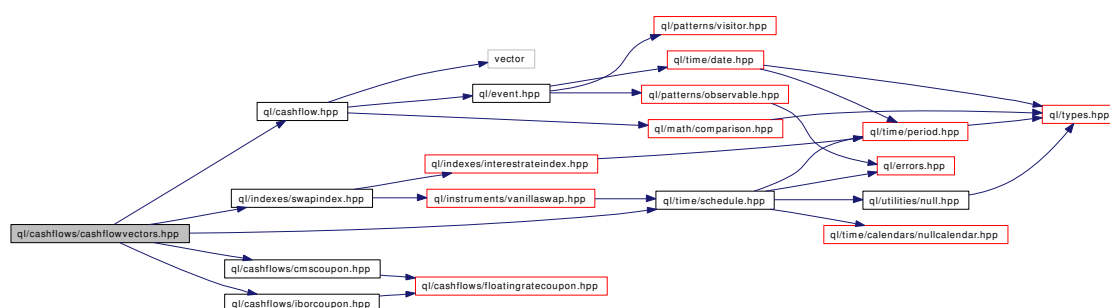
10.5 ql/cashflows/cashflowvectors.hpp File Reference

10.5.1 Detailed Description

Cash flow vector builders.

```
#include <ql/cashflow.hpp>
#include <ql/time/schedule.hpp>
#include <ql/cashflows/iborcoupon.hpp>
#include <ql/cashflows/cmscoupon.hpp>
#include <ql/indexes/swapindex.hpp>
```

Include dependency graph for cashflowvectors.hpp:



Namespaces

- namespace **QuantLib**

Functions

- Leg **FixedRateLeg** (const std::vector< Real > &nominals, const Schedule &schedule, const std::vector< Rate > &couponRates, const DayCounter &paymentDayCounter, BusinessDayConvention paymentAdjustment=Following, const DayCounter &firstPeriodDayCounter=DayCounter())
helper function building a sequence of fixed rate coupons
- Leg **IborLeg** (const std::vector< Real > &nominals, const Schedule &schedule, const boost::shared_ptr< IborIndex > &index, const DayCounter &paymentDayCounter=DayCounter(), const BusinessDayConvention paymentConvention=Following, Natural fixingDays=Null< Size >(), const std::vector< Real > &gearings=std::vector< Real >(), const std::vector< Spread > &spreads=std::vector< Spread >(), const std::vector< Rate > &caps=std::vector< Rate >(), const std::vector< Rate > &floors=std::vector< Rate >(), bool isInArrears=false)
helper function building a sequence of capped/floored ibor rate coupons
- Leg **CmsLeg** (const std::vector< Real > &nominals, const Schedule &schedule, const boost::shared_ptr< SwapIndex > &index, const DayCounter &paymentDayCounter=DayCounter(), BusinessDayConvention paymentConvention=Following, Natural fixingDays=Null< Size >(), const std::vector< Real > &gearings=std::vector< Real >(),

```
const std::vector< Spread > &spreads=std::vector< Spread >(), const std::vector< Rate >
&caps=std::vector< Rate >(), const std::vector< Rate > &floors=std::vector< Rate >(), bool
isInArrears=false)
```

helper function building a sequence of capped/floored cms rate coupons

- Leg [CmsZeroLeg](#) (const std::vector< Real > &nominals, const Schedule &sched-
ule, const boost::shared_ptr< SwapIndex > &index, const DayCounter &paymentDay-
Counter=DayCounter(), BusinessDayConvention paymentConvention=Following, Natu-
ral fixingDays=Null< Natural >(), const std::vector< Real > &gearings=std::vector< Real
>(), const std::vector< Spread > &spreads=std::vector< Spread >(), const std::vector< Rate
> &caps=std::vector< Rate >(), const std::vector< Rate > &floors=std::vector< Rate >())

helper function building a sequence of capped/floored cms zero rate coupons

- Leg [RangeAccrualLeg](#) (const std::vector< Real > &nominals, const Schedule &sched-
ule, const boost::shared_ptr< IborIndex > &index, const DayCounter &paymentDay-
Counter=DayCounter(), BusinessDayConvention paymentConvention=Following, Natu-
ral fixingDays=Null< Natural >(), const std::vector< Real > &gearings=std::vector< Real
>(), const std::vector< Spread > &spreads=std::vector< Spread >(), const
std::vector< Rate > &lowerTriggers=std::vector< Rate >(), const std::vector< Rate >
&upperTriggers=std::vector< Rate >(), const Period &observationTenor=1 *Days, Business-
DayConvention observationConvention=ModifiedFollowing)

helper function building a sequence of range accrual floaters coupons

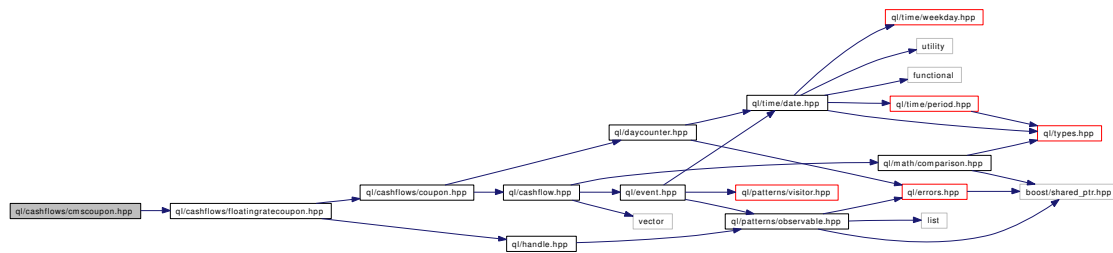
10.6 ql/cashflows/cmscoupon.hpp File Reference

10.6.1 Detailed Description

CMS coupon.

```
#include <ql/cashflows/floatingratecoupon.hpp>
```

Include dependency graph for cmscoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CmsCoupon**
CMS coupon class.

10.7 ql/cashflows/conundrumpricer.hpp File Reference

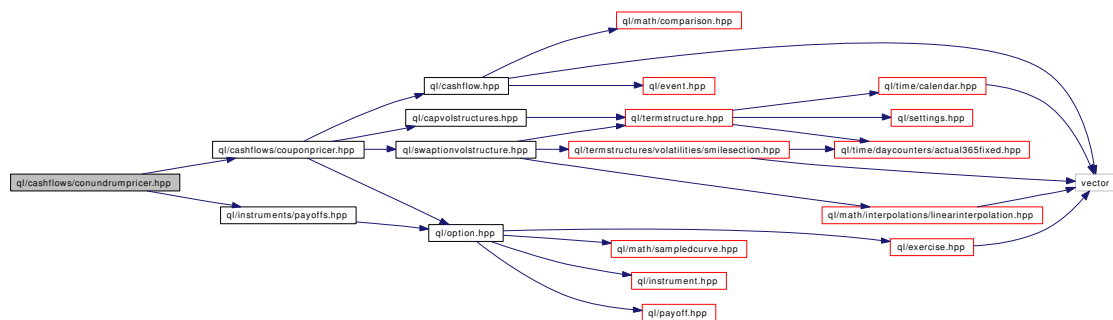
10.7.1 Detailed Description

CMS-coupon pricer.

```
#include <ql/cashflows/couponpricer.hpp>
```

```
#include <ql/instruments/payoffs.hpp>
```

Include dependency graph for conundrumpricer.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **ConundrumPricer**
CMS-coupon pricer.
- class **ConundrumPricerByNumericalIntegration**
CMS-coupon pricer.
- class **ConundrumPricerByBlack**
CMS-coupon pricer.

10.8 ql/cashflows/coupon.hpp File Reference

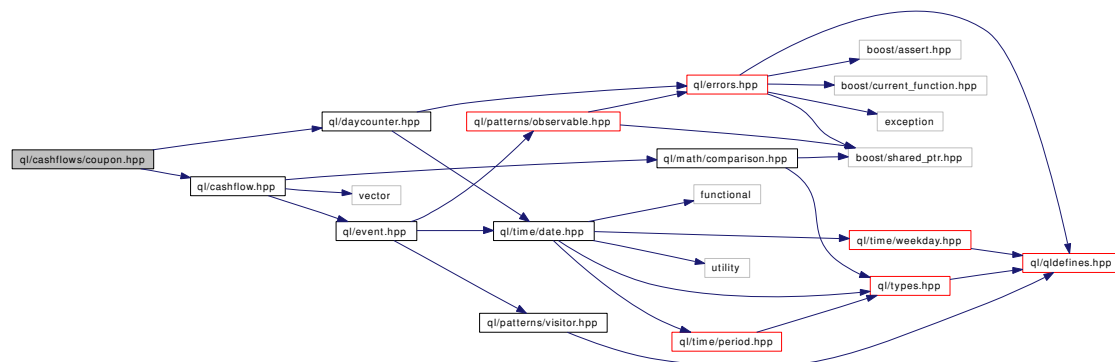
10.8.1 Detailed Description

Coupon accruing over a fixed period.

```
#include <ql/cashflow.hpp>
```

```
#include <ql/daycounter.hpp>
```

Include dependency graph for coupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Coupon**
coupon accruing over a fixed period

10.9 ql/cashflows/couponpricer.hpp File Reference

10.9.1 Detailed Description

Coupon pricers.

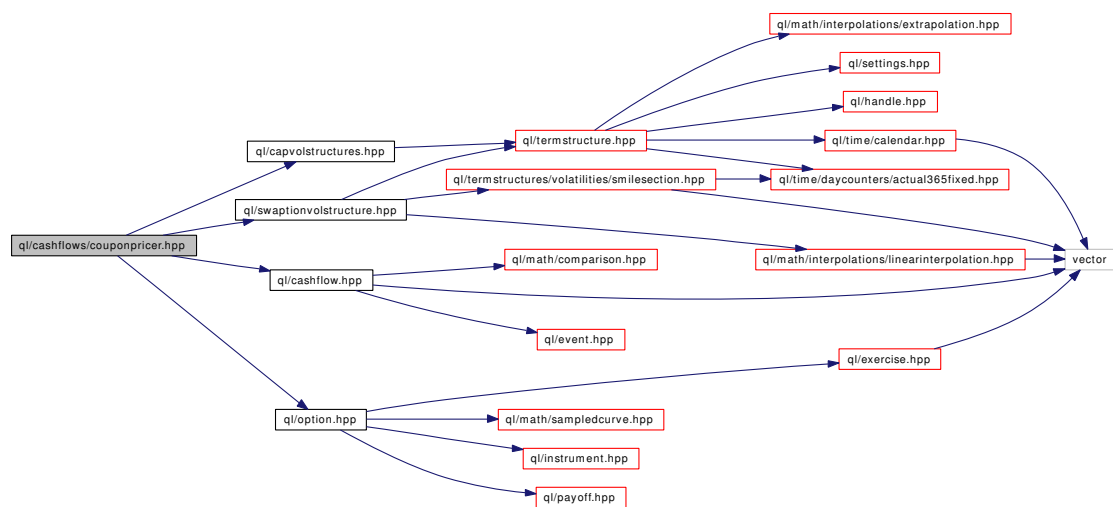
```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/swaptionvolstructure.hpp>
```

```
#include <ql/cashflow.hpp>
```

```
#include <ql/option.hpp>
```

Include dependency graph for couponpricer.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FloatingRateCouponPricer**
generic pricer for floating-rate coupons
- class **IborCouponPricer**
base pricer for capped/floored Ibor coupons
- class **BlackIborCouponPricer**
Black-formula pricer for capped/floored Ibor coupons.
- class **CmsCouponPricer**
base pricer for vanilla CMS coupons

Functions

- void **setCouponPricer** (const Leg &leg, const boost::shared_ptr< FloatingRateCouponPricer > &)
- void **setCouponPricers** (const Leg &leg, const std::vector< boost::shared_ptr< FloatingRateCouponPricer > > &)

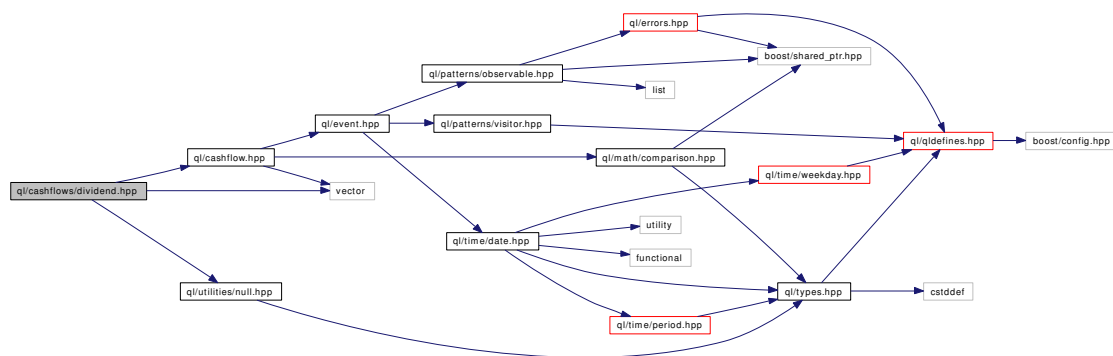
10.10 ql/cashflows/dividend.hpp File Reference

10.10.1 Detailed Description

A stock dividend.

```
#include <ql/cashflow.hpp>
#include <ql/utilities/null.hpp>
#include <vector>
```

Include dependency graph for dividend.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Dividend**
Predetermined cash flow.
- class **FixedDividend**
Predetermined cash flow.
- class **FractionalDividend**
Predetermined cash flow.

Functions

- std::vector< boost::shared_ptr< Dividend > > **DividendVector** (const std::vector< Date > ÷ndDates, const std::vector< Real > ÷nds)
helper function building a sequence of fixed dividends

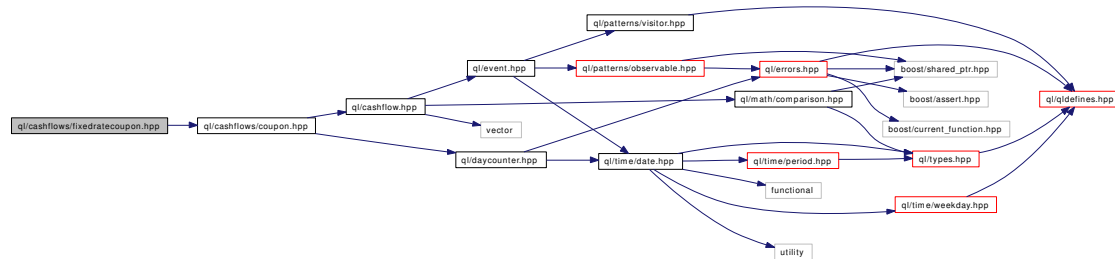
10.11 ql/cashflows/fixedratecoupon.hpp File Reference

10.11.1 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

10.12 ql/cashflows/floatingratecoupon.hpp File Reference

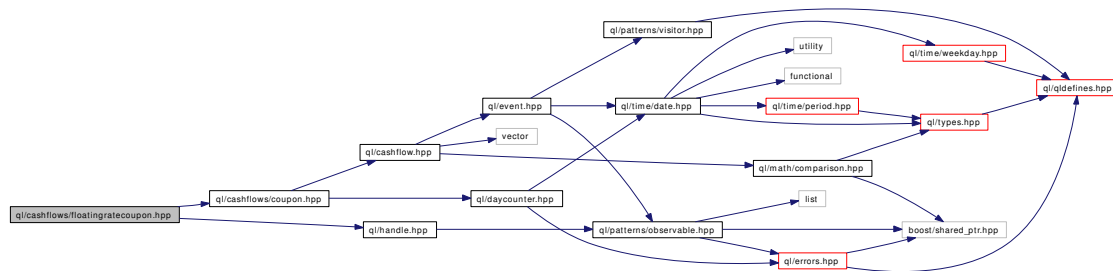
10.12.1 Detailed Description

Coupon paying a variable index-based rate.

```
#include <ql/cashflows/coupon.hpp>
```

```
#include <ql/handle.hpp>
```

Include dependency graph for floatingratecoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FloatingRateCoupon**
base floating-rate coupon class

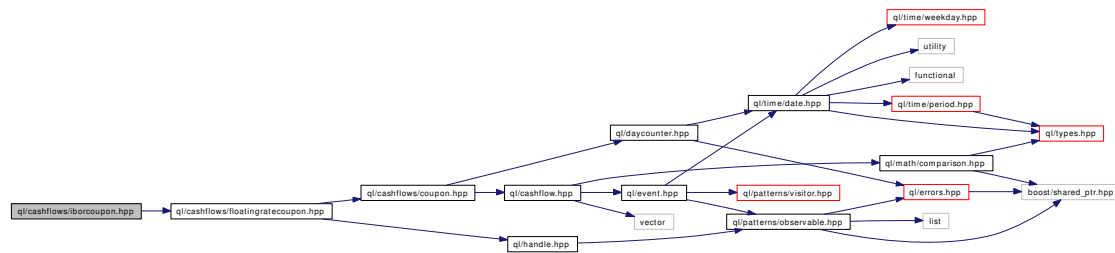
10.13 ql/cashflows/iborcoupon.hpp File Reference

10.13.1 Detailed Description

Coupon paying a Libor-type index.

```
#include <ql/cashflows/floatingratecoupon.hpp>
```

Include dependency graph for iborcoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **IborCoupon**
Coupon paying a Libor-type index

10.14 ql/cashflows/rangeaccrual.hpp File Reference

10.14.1 Detailed Description

range-accrual coupon

```
#include <ql/termstructures/volatilities/smilesection.hpp>
```

```
#include <ql/indexes/iborindex.hpp>
```

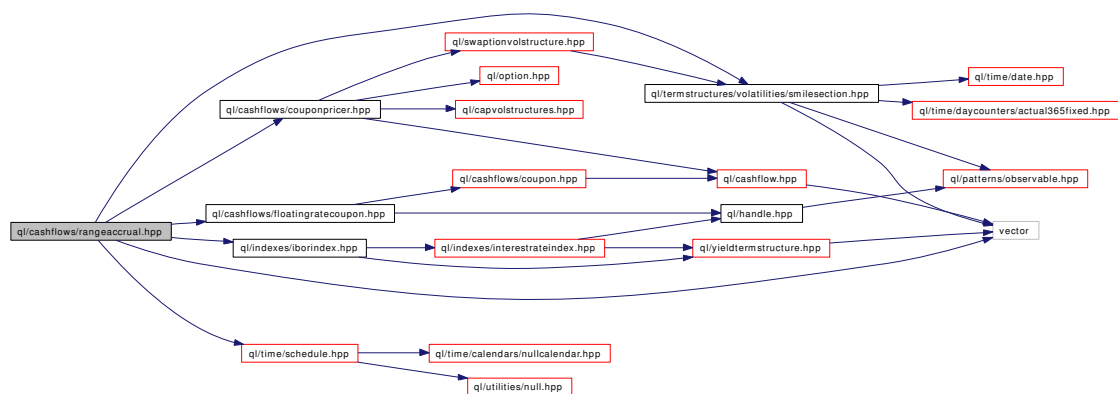
```
#include <ql/cashflows/couponpricer.hpp>
```

```
#include <ql/cashflows/floatingratecoupon.hpp>
```

```
#include <ql/time/schedule.hpp>
```

```
#include <vector>
```

Include dependency graph for rangeaccrual.hpp:



Namespaces

- namespace **QuantLib**

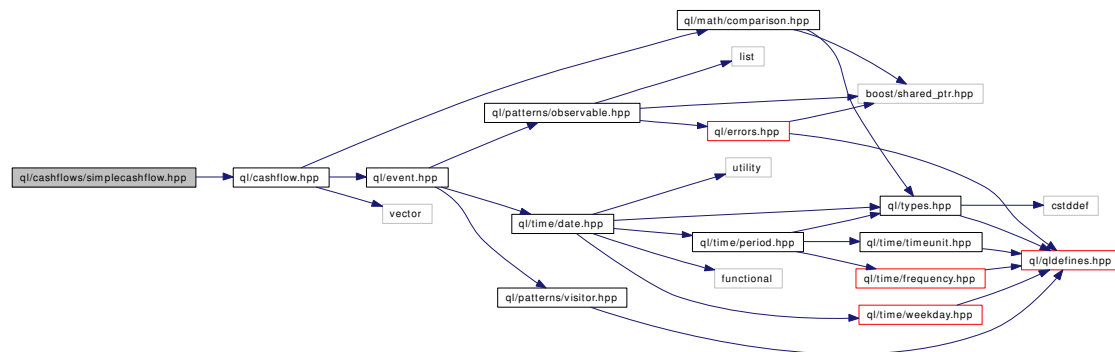
10.15 ql/cashflows/simplecashflow.hpp File Reference

10.15.1 Detailed Description

Predetermined cash flow.

```
#include <ql/cashflow.hpp>
```

Include dependency graph for simplecashflow.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **SimpleCashFlow**
Predetermined cash flow.

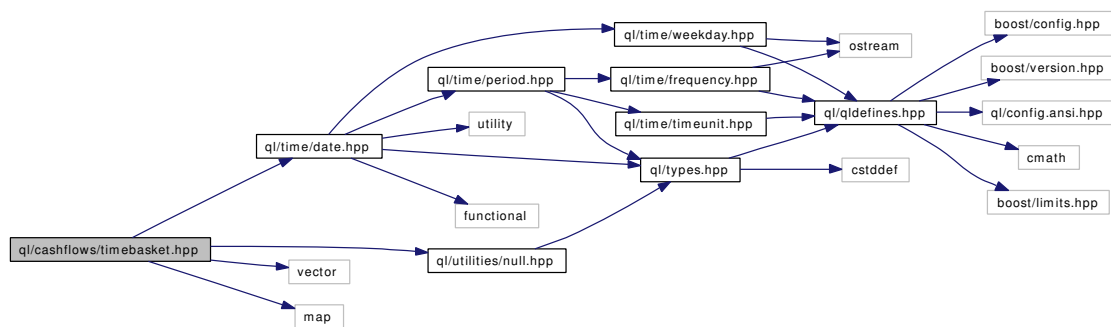
10.16 ql/cashflows/timebasket.hpp File Reference

10.16.1 Detailed Description

distribution over a number of date ranges

```
#include <ql/time/date.hpp>
#include <ql/utilities/null.hpp>
#include <vector>
#include <map>
```

Include dependency graph for timebasket.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TimeBasket](#)
Distribution over a number of dates.

10.17 ql/currencies/africa.hpp File Reference

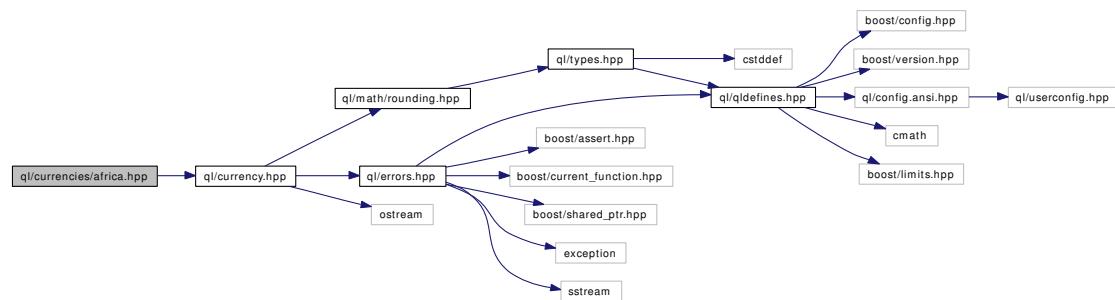
10.17.1 Detailed Description

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.

10.18 ql/currencies/america.hpp File Reference

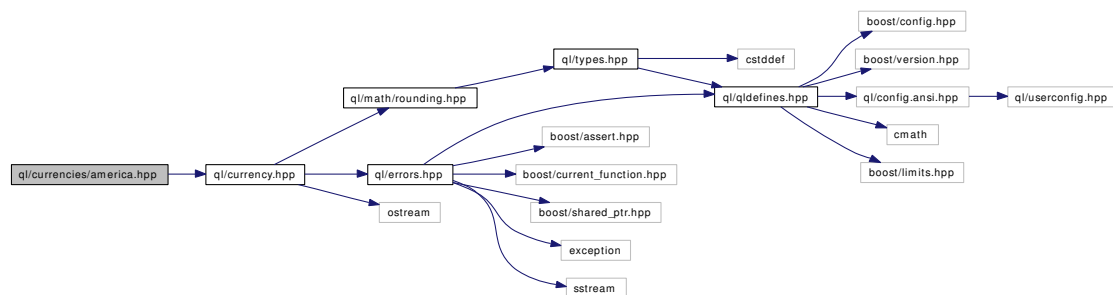
10.18.1 Detailed Description

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.

10.19 ql/currencies/asia.hpp File Reference

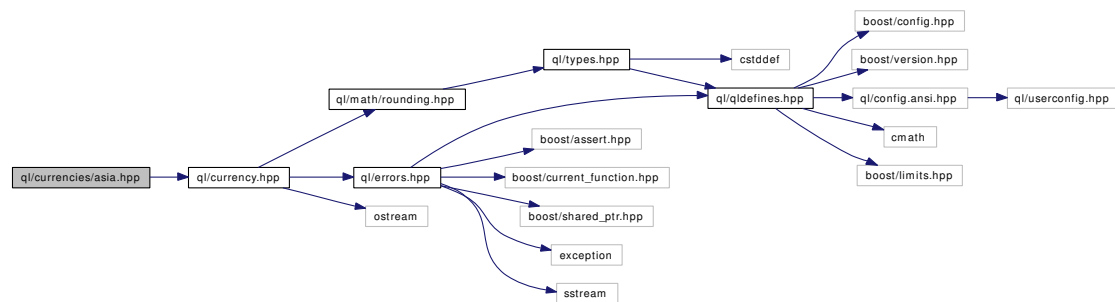
10.19.1 Detailed Description

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

10.20 ql/currencies/europe.hpp File Reference

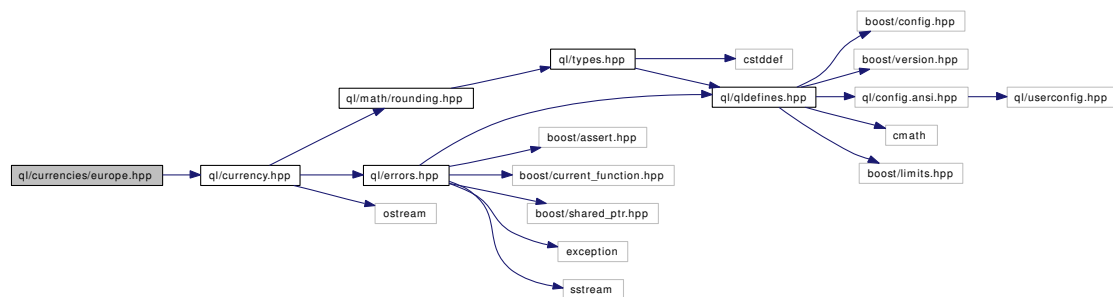
10.20.1 Detailed Description

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](#)
Icelandic krona.
- class [LTLCurrency](#)
Lithuanian litas.
- class [LVLCurrency](#)
Latvian lat.
- class [MTCurrency](#)
Maltese lira.
- class [NOKCurrency](#)
Norwegian krone.
- class [PLNCurrency](#)
Polish zloty.
- class [ROLCurrency](#)
Romanian leu.
- class [RONCurrency](#)
Romanian new leu.
- class [SEKCurrency](#)
Swedish krona.
- class [SITCurrency](#)
Slovenian tolar.
- class [SKKCurrency](#)
Slovak koruna.
- class [TRLCurrency](#)
Turkish lira.
- class [TRYCurrency](#)
New Turkish lira.
- class [ATSCurrency](#)
Austrian shilling.

- class [BEFCurrency](#)
Belgian franc.
- class [DEMCurrency](#)
Deutsche mark.
- class [ESPCurrency](#)
Spanish peseta.
- class [FIMCurrency](#)
Finnish markka.
- class [FRFCurrency](#)
French franc.
- class [GRDCurrency](#)
Greek drachma.
- class [IEPCurrency](#)
Irish punt.
- class [ITLCurrency](#)
Italian lira.
- class [LUFCurrency](#)
Luxembourg franc.
- class [NLGCurrency](#)
Dutch guilder.
- class [PTECurrency](#)
Portuguese escudo.

10.21 ql/currencies/exchangeratemanager.hpp File Reference

10.21.1 Detailed Description

exchange-rate repository

```
#include <ql/exchangerate.hpp>
```

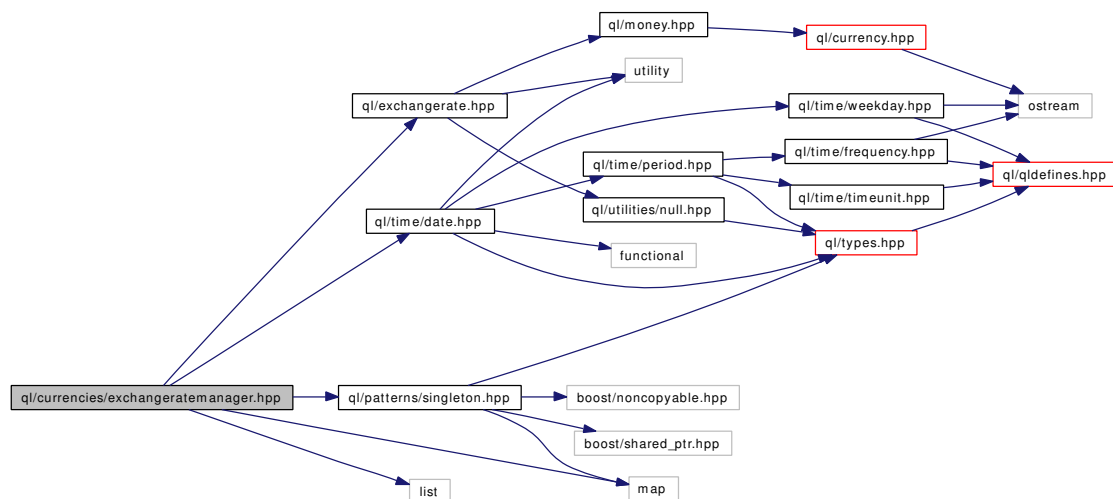
```
#include <ql/time/date.hpp>
```

```
#include <ql/patterns/singleton.hpp>
```

```
#include <list>
```

```
#include <map>
```

Include dependency graph for `exchangeratemanager.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class **ExchangeRateManager**
exchange-rate repository

10.22 ql/currencies/oceania.hpp File Reference

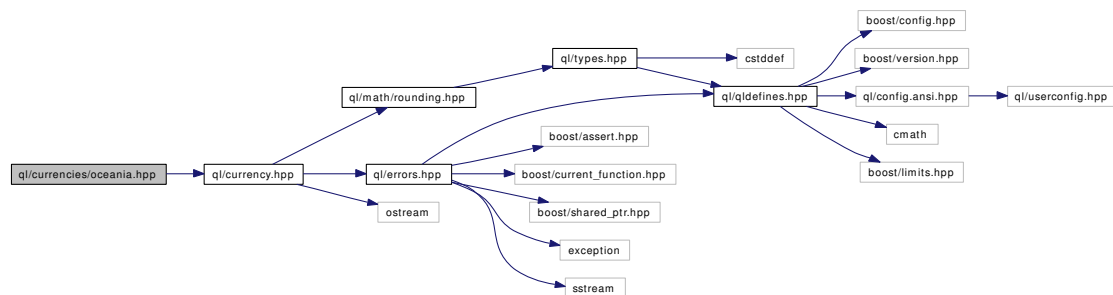
10.22.1 Detailed Description

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.

10.23 ql/currency.hpp File Reference

10.23.1 Detailed Description

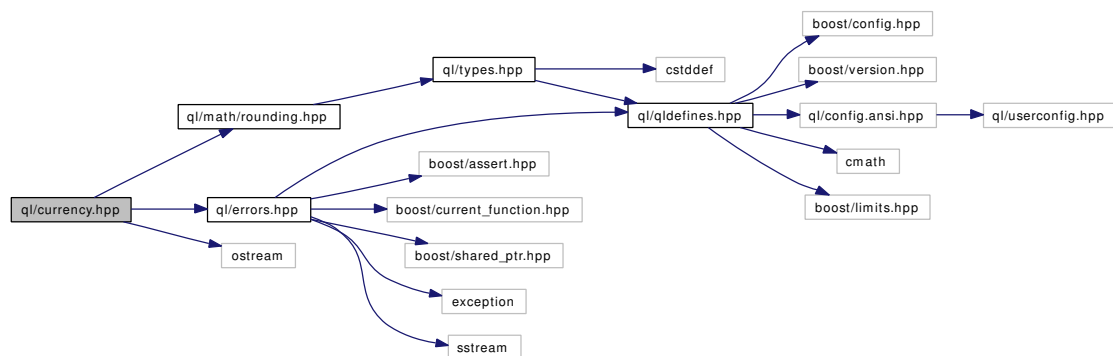
Currency specification.

```
#include <ql/math/rounding.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <ostream>
```

Include dependency graph for currency.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Currency](#)
Currency specification

10.24 ql/daycounter.hpp File Reference

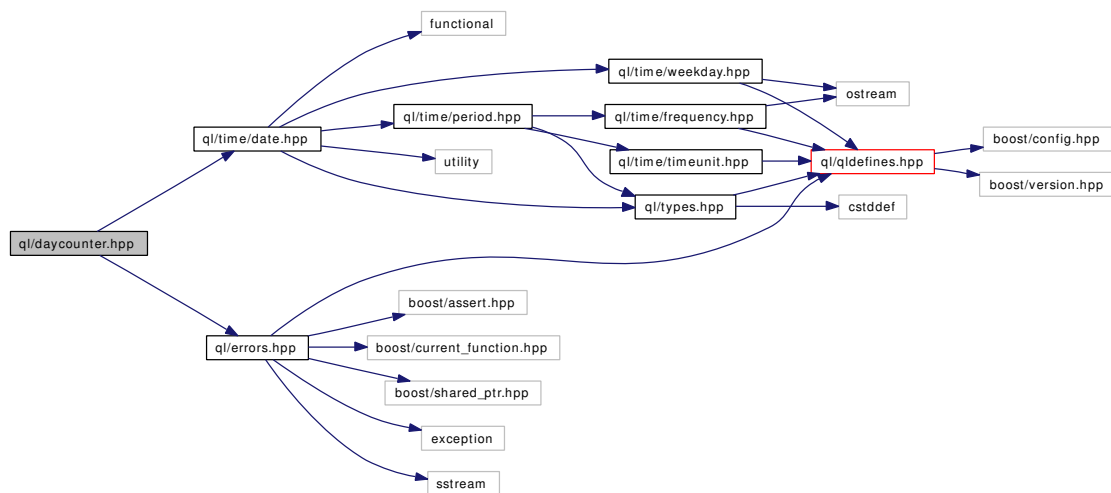
10.24.1 Detailed Description

day counter class

```
#include <ql/time/date.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for daycounter.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **DayCounter**
day counter class
- class **DayCounter::Impl**
abstract base class for day counter implementations

10.25 ql/discretizedasset.hpp File Reference

10.25.1 Detailed Description

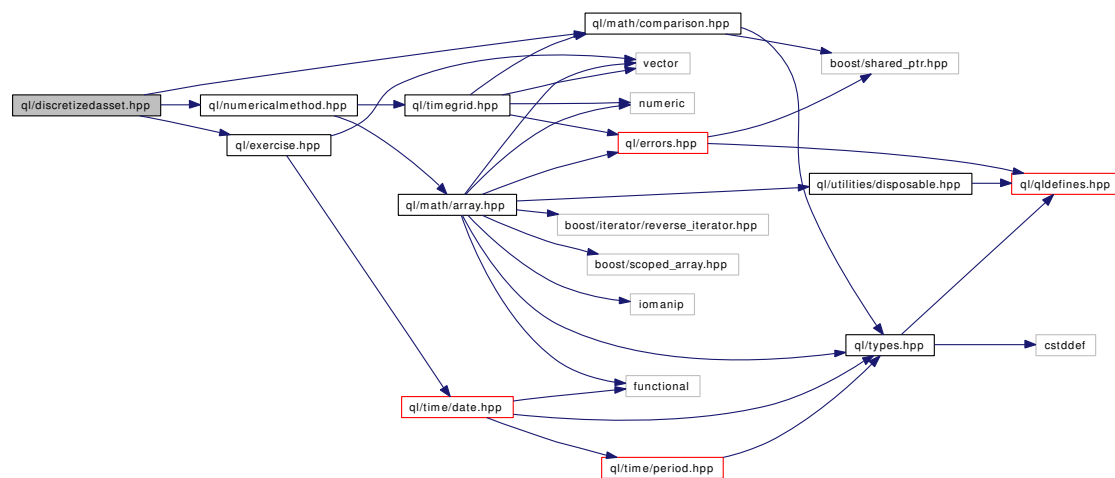
Discretized asset classes.

```
#include <ql/numericalmethod.hpp>
```

```
#include <ql/math/comparison.hpp>
```

```
#include <ql/exercise.hpp>
```

Include dependency graph for discretizedasset.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DiscretizedAsset](#)
Discretized asset class used by numerical methods.
- class [DiscretizedDiscountBond](#)
Useful discretized discount bond asset.
- class [DiscretizedOption](#)
Discretized option on a given asset.

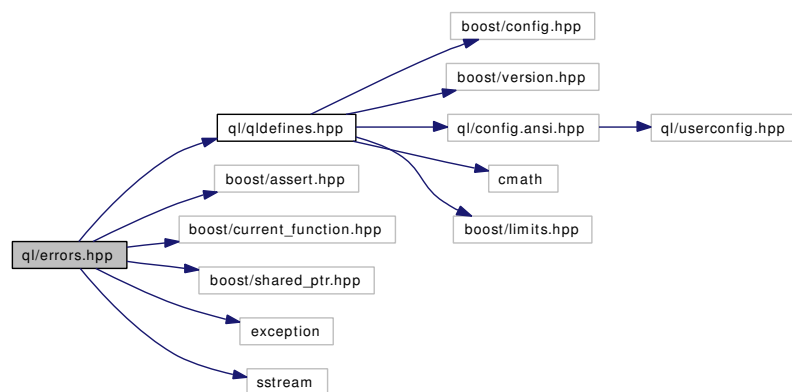
10.26 ql/errors.hpp File Reference

10.26.1 Detailed Description

Classes and functions for error handling.

```
#include <ql/qldefines.hpp>
#include <boost/assert.hpp>
#include <boost/current_function.hpp>
#include <boost/shared_ptr.hpp>
#include <exception>
#include <sstream>
```

Include dependency graph for errors.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Error**
Base error class.

Defines

- #define **QL_FAIL**(message)
throw an error (possibly with file and line information)
- #define **QL_ASSERT**(condition, message)
throw an error if the given condition is not verified
- #define **QL_REQUIRE**(condition, message)

throw an error if the given pre-condition is not verified

- `#define QL_ENSURE(condition, message)`
throw an error if the given post-condition is not verified

10.26.2 Define Documentation

10.26.2.1 `#define QL_FAIL(message)`

Value:

```
do { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__, __LINE__, \
                          BOOST_CURRENT_FUNCTION, _ql_msg_stream.str()); \
} while (false)
```

throw an error (possibly with file and line information)

10.26.2.2 `#define QL_ASSERT(condition, message)`

Value:

```
if (!(condition)) { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__, __LINE__, \
                          BOOST_CURRENT_FUNCTION, _ql_msg_stream.str()); \
} else
```

throw an error if the given condition is not verified

10.26.2.3 `#define QL_REQUIRE(condition, message)`

Value:

```
if (!(condition)) { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__, __LINE__, \
                          BOOST_CURRENT_FUNCTION, _ql_msg_stream.str()); \
} else
```

throw an error if the given pre-condition is not verified

Examples:

[DiscreteHedging.cpp](#), and [swapvaluation.cpp](#).

10.26.2.4 #define QL_ENSURE(condition, message)

Value:

```
if (!(condition)) { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__, __LINE__, \
                          BOOST_CURRENT_FUNCTION, _ql_msg_stream.str()); \
} else
```

throw an error if the given post-condition is not verified

10.27 ql/event.hpp File Reference

10.27.1 Detailed Description

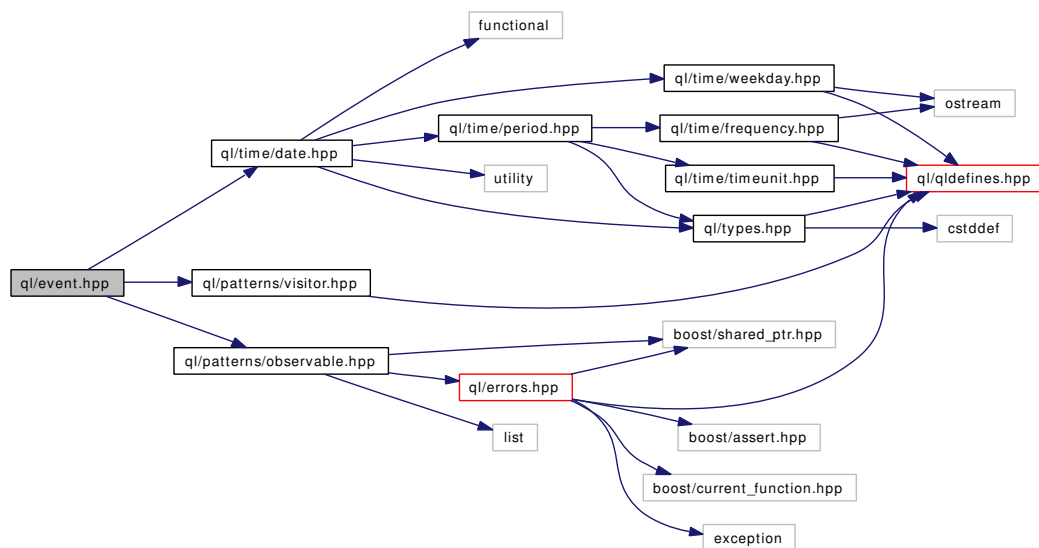
Base class for events associated with a given date.

```
#include <ql/time/date.hpp>
```

```
#include <ql/patterns/observable.hpp>
```

```
#include <ql/patterns/visitor.hpp>
```

Include dependency graph for event.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Event](#)
Base class for event.

10.28 ql/exchangerate.hpp File Reference

10.28.1 Detailed Description

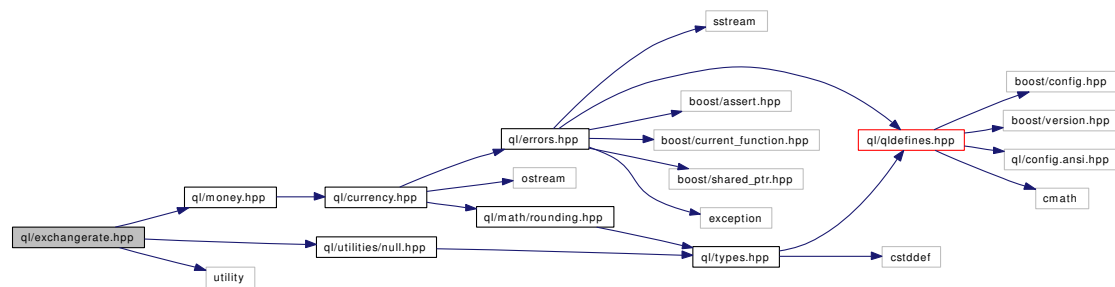
exchange rate between two currencies

```
#include <ql/money.hpp>
```

```
#include <ql/utilities/null.hpp>
```

```
#include <utility>
```

Include dependency graph for exchangerate.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ExchangeRate](#)
exchange rate between two currencies

10.29 ql/exercise.hpp File Reference

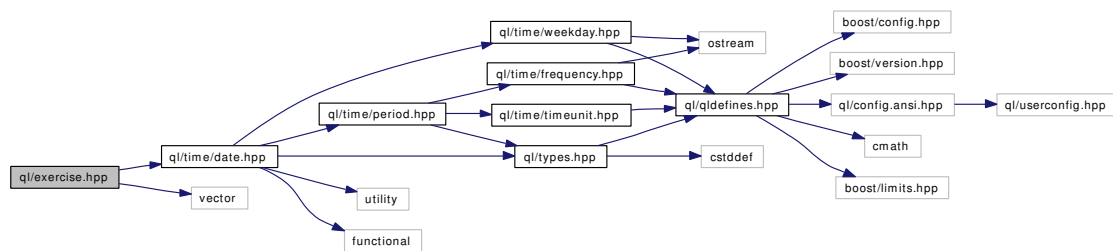
10.29.1 Detailed Description

Option exercise classes and payoff function.

```
#include <ql/time/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.

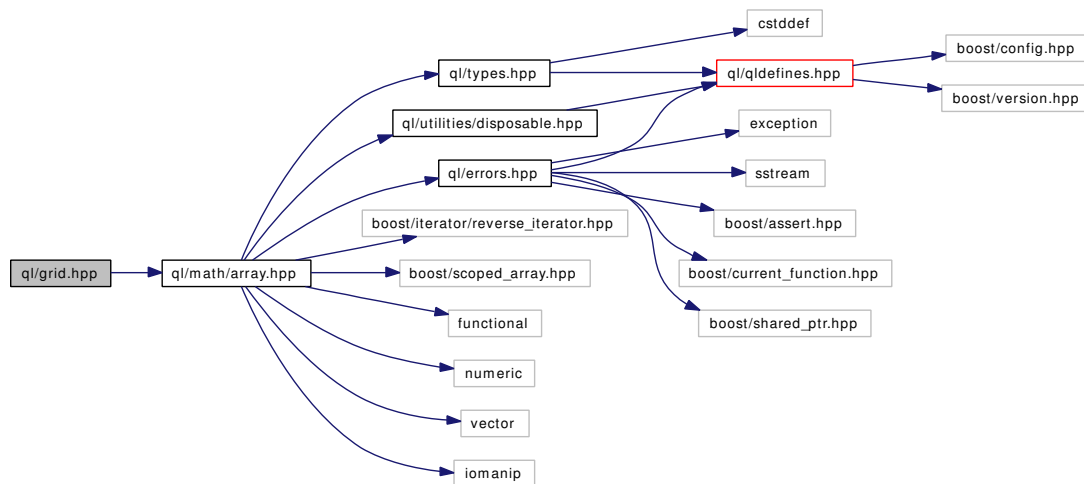
10.30 ql/grid.hpp File Reference

10.30.1 Detailed Description

Grid constructors.

```
#include <ql/math/array.hpp>
```

Include dependency graph for grid.hpp:



Namespaces

- namespace **QuantLib**

Functions

- Disposable< Array > **CenteredGrid** (Real center, Real dx, Size steps)
- Disposable< Array > **BoundedGrid** (Real xMin, Real xMax, Size steps)
- Disposable< Array > **BoundedLogGrid** (Real xMin, Real xMax, Size steps)

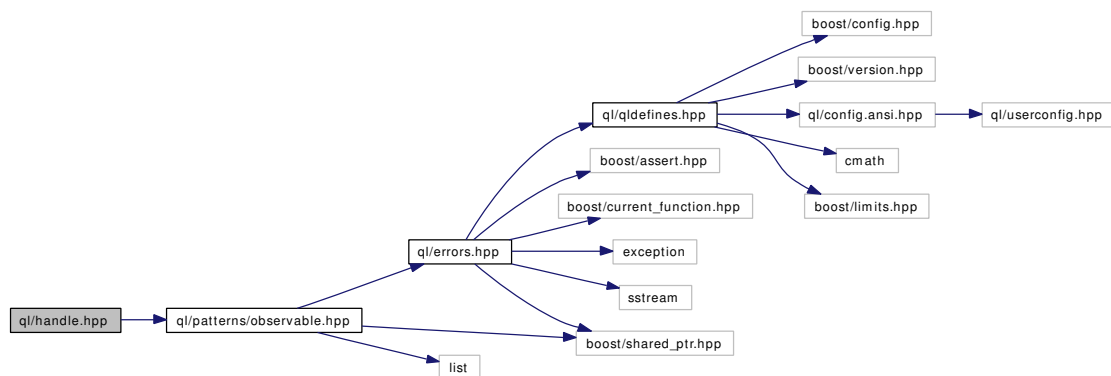
10.31 ql/handle.hpp File Reference

10.31.1 Detailed Description

Globally accessible relinkable pointer.

```
#include <ql/patterns/observable.hpp>
```

Include dependency graph for handle.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Handle](#)
Shared handle to an observable.
- class [RelinkableHandle](#)
Relinkable handle to an observable.

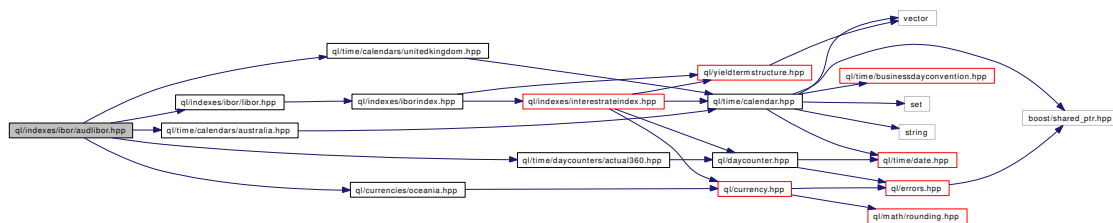
10.33 ql/indexes/ibor/audlibor.hpp File Reference

10.33.1 Detailed Description

AUD LIBOR rate

```
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/unitedkingdom.hpp>
#include <ql/time/calendars/australia.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/oceania.hpp>
```

Include dependency graph for audlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **AUDLibor**
AUD LIBOR rate

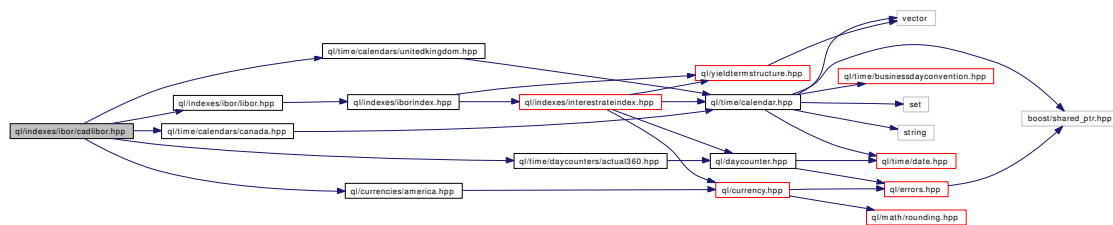
10.34 ql/indexes/ibor/cadlibor.hpp File Reference

10.34.1 Detailed Description

CAD LIBOR rate

```
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/unitedkingdom.hpp>
#include <ql/time/calendars/canada.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/america.hpp>
```

Include dependency graph for cadlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CADLibor**
CAD LIBOR rate

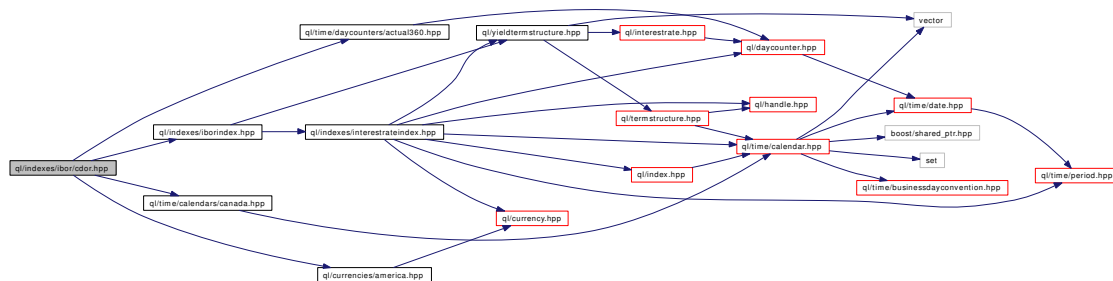
10.35 ql/indexes/ibor/cdor.hpp File Reference

10.35.1 Detailed Description

CDOR rate

```
#include <ql/indexes/iborindex.hpp>
#include <ql/time/calendars/canada.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/america.hpp>
```

Include dependency graph for cdor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Cdor**
CDOR rate

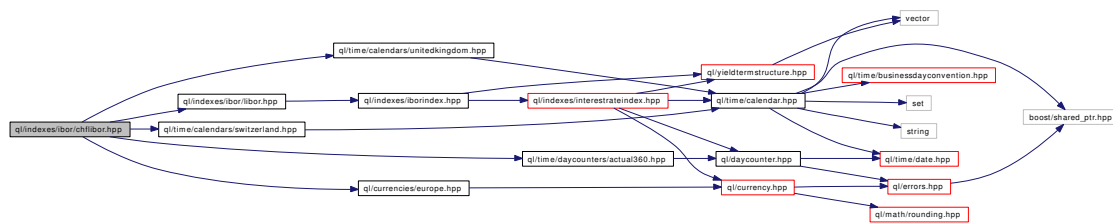
10.36 ql/indexes/ibor/chflibor.hpp File Reference

10.36.1 Detailed Description

CHF LIBOR rate

```
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/unitedkingdom.hpp>
#include <ql/time/calendars/switzerland.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for chflibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CHFLibor**
CHF LIBOR rate

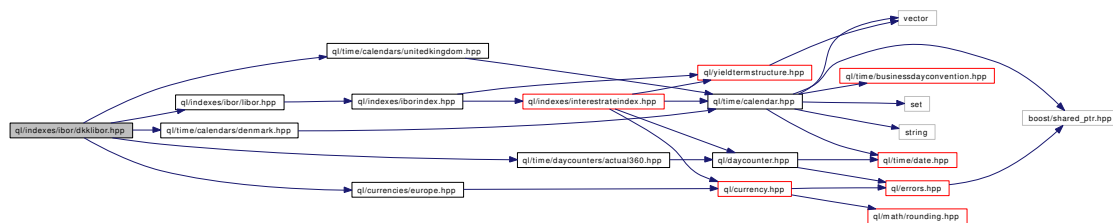
10.37 ql/indexes/ibor/dkklibor.hpp File Reference

10.37.1 Detailed Description

DKK LIBOR rate

```
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/unitedkingdom.hpp>
#include <ql/time/calendars/denmark.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for dkklibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **DKKLibor**
DKK LIBOR rate

- class [Euribor4M](#)
4-months Euribor index
- class [Euribor5M](#)
5-months Euribor index
- class [Euribor6M](#)
6-months Euribor index
- class [Euribor7M](#)
7-months Euribor index
- class [Euribor8M](#)
8-months Euribor index
- class [Euribor9M](#)
9-months Euribor index
- class [Euribor10M](#)
10-months Euribor index
- class [Euribor11M](#)
11-months Euribor index
- class [Euribor1Y](#)
1-year Euribor index
- class [Euribor365_SW](#)
1-week Euribor365 index
- class [Euribor365_2W](#)
2-weeks Euribor365 index
- class [Euribor365_3W](#)
3-weeks Euribor365 index
- class [Euribor365_1M](#)
1-month Euribor365 index
- class [Euribor365_2M](#)
2-months Euribor365 index
- class [Euribor365_3M](#)
3-months Euribor365 index
- class [Euribor365_4M](#)
4-months Euribor365 index
- class [Euribor365_5M](#)

5-months Euribor365 index

- class [Euribor365_6M](#)

6-months Euribor365 index

- class [Euribor365_7M](#)

7-months Euribor365 index

- class [Euribor365_8M](#)

8-months Euribor365 index

- class [Euribor365_9M](#)

9-months Euribor365 index

- class [Euribor365_10M](#)

10-months Euribor365 index

- class [Euribor365_11M](#)

11-months Euribor365 index

- class [Euribor365_1Y](#)

1-year Euribor365 index

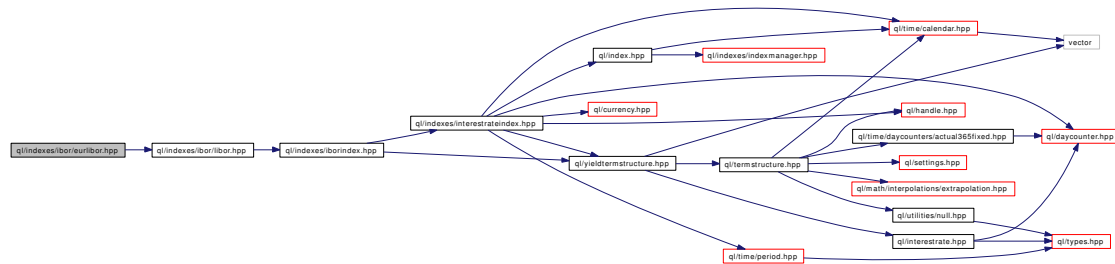
10.39 ql/indexes/ibor/eurlibor.hpp File Reference

10.39.1 Detailed Description

EUR LIBOR rate

```
#include <ql/indexes/ibor/libor.hpp>
```

Include dependency graph for eurlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **EURLibor**
EUR LIBOR rate
- class **EURLiborSW**
1-week EURLibor index
- class **EURLibor2W**
2-weeks Euribor index
- class **EURLibor1M**
1-month EURLibor index
- class **EURLibor2M**
2-months EURLibor index
- class **EURLibor3M**
3-months EURLibor index
- class **EURLibor4M**
4-months EURLibor index
- class **EURLibor5M**
5-months EURLibor index

- class [EURLibor6M](#)
6-months EURLibor index
- class [EURLibor7M](#)
7-months EURLibor index
- class [EURLibor8M](#)
8-months EURLibor index
- class [EURLibor9M](#)
9-months EURLibor index
- class [EURLibor10M](#)
10-months EURLibor index
- class [EURLibor11M](#)
11-months EURLibor index
- class [EURLibor1Y](#)
1-year EURLibor index

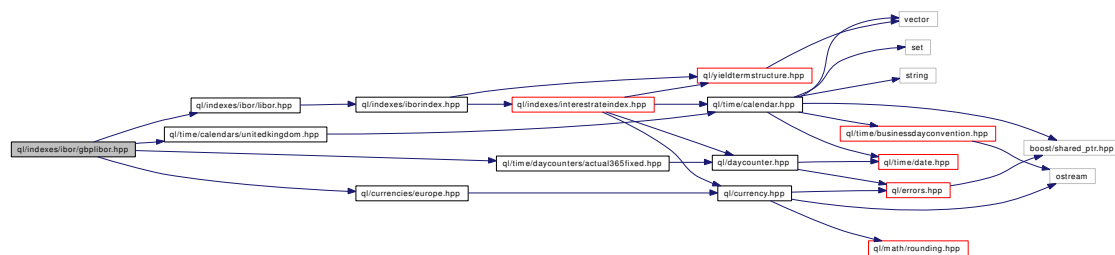
10.40 ql/indexes/ibor/gbplibor.hpp File Reference

10.40.1 Detailed Description

GBP LIBOR rate

```
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/unitedkingdom.hpp>
#include <ql/time/daycounters/actual365fixed.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for gbplibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **GBPLibor**
GBP LIBOR rate

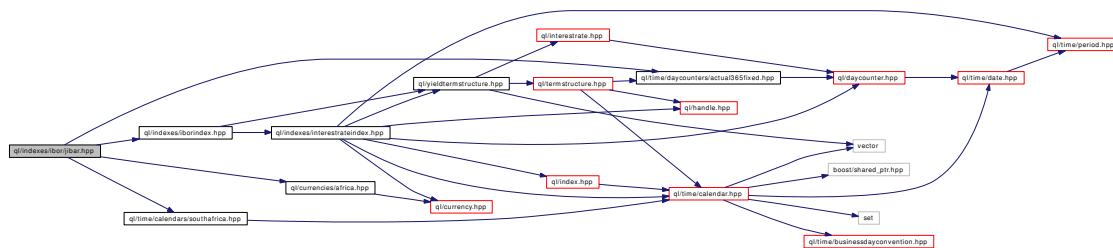
10.41 ql/indexes/ibor/jibar.hpp File Reference

10.41.1 Detailed Description

JIBAR rate

```
#include <ql/indexes/iborindex.hpp>
#include <ql/time/calendars/southafrica.hpp>
#include <ql/time/daycounters/actual365fixed.hpp>
#include <ql/currencies/africa.hpp>
```

Include dependency graph for jibar.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Jibar**
JIBAR rate

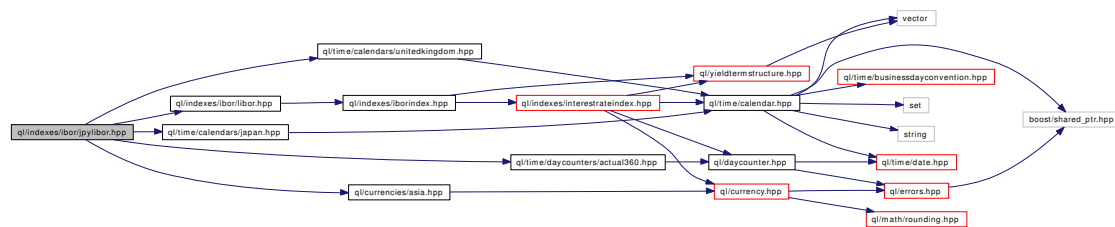
10.42 ql/indexes/ibor/jpylibor.hpp File Reference

10.42.1 Detailed Description

JPY LIBOR rate

```
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/unitedkingdom.hpp>
#include <ql/time/calendars/japan.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/asia.hpp>
```

Include dependency graph for jpylibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **JPYLibor**
JPY LIBOR rate

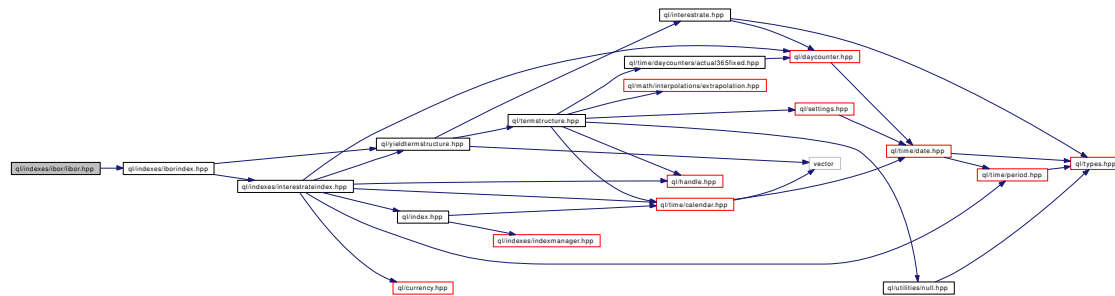
10.43 ql/indexes/ibor/libor.hpp File Reference

10.43.1 Detailed Description

base class for BBA LIBOR indexes

```
#include <ql/indexes/iborindex.hpp>
```

Include dependency graph for libor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Libor**
base class for all BBA LIBOR indexes but the EUR ones

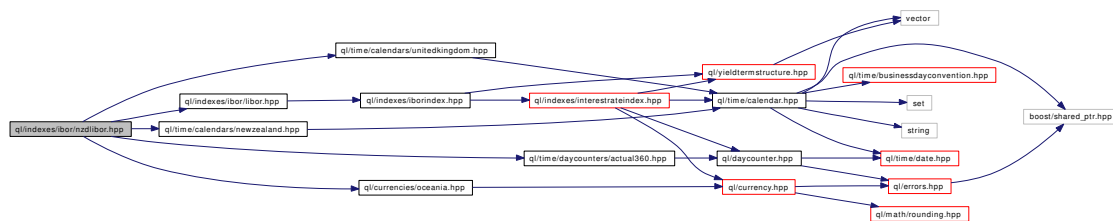
10.44 ql/indexes/ibor/nzdlibor.hpp File Reference

10.44.1 Detailed Description

NZD LIBOR rate

```
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/unitedkingdom.hpp>
#include <ql/time/calendars/newzealand.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/oceania.hpp>
```

Include dependency graph for nzdlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **NZDLibor**
NZD LIBOR rate

10.45 ql/indexes/ibor/tibor.hpp File Reference

10.45.1 Detailed Description

JPY TIBOR rate

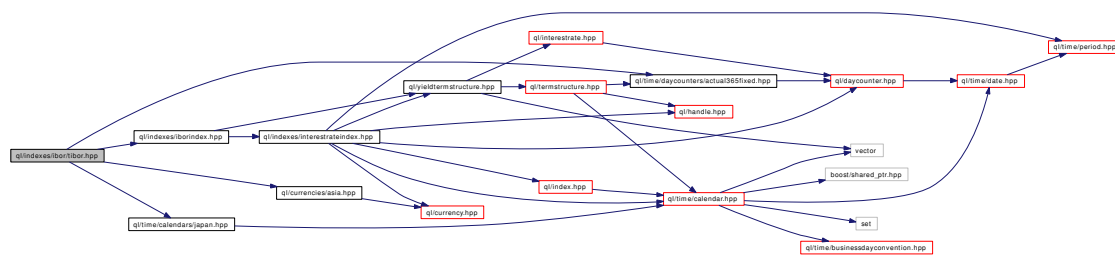
```
#include <ql/indexes/iborindex.hpp>
```

```
#include <ql/time/calendars/japan.hpp>
```

```
#include <ql/time/daycounters/actual365fixed.hpp>
```

```
#include <ql/currencies/asia.hpp>
```

Include dependency graph for tibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Tibor**
JPY TIBOR index

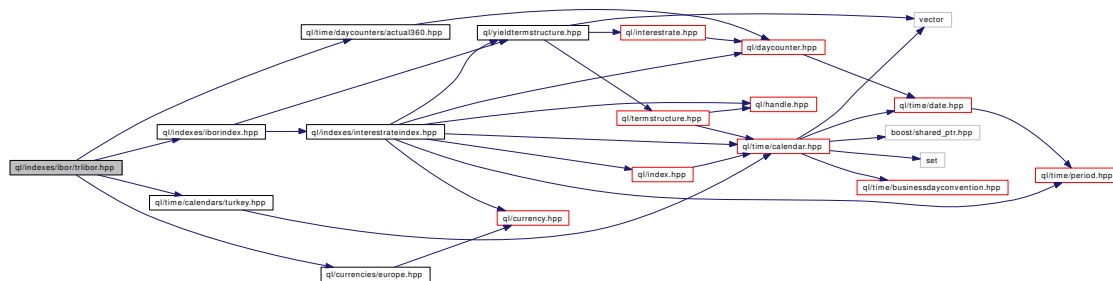
10.46 ql/indexes/ibor/trlibor.hpp File Reference

10.46.1 Detailed Description

TRY LIBOR rate

```
#include <ql/indexes/iborindex.hpp>
#include <ql/time/calendars/turkey.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for trlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TRLibor**
TRY LIBOR rate

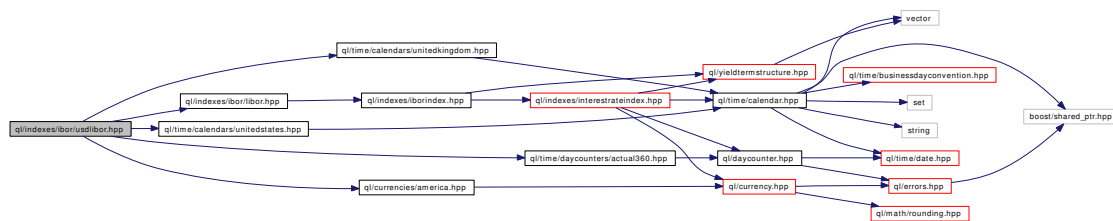
10.47 ql/indexes/ibor/usdlibor.hpp File Reference

10.47.1 Detailed Description

USD LIBOR rate

```
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/unitedkingdom.hpp>
#include <ql/time/calendars/unitedstates.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/america.hpp>
```

Include dependency graph for usdlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **USDLibor**
USD LIBOR rate

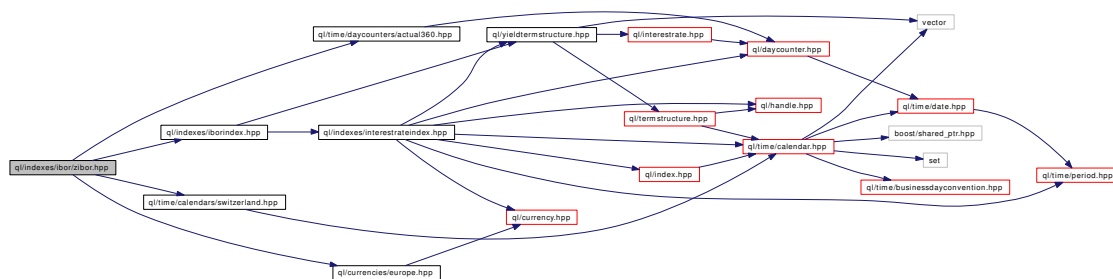
10.48 ql/indexes/ibor/zibor.hpp File Reference

10.48.1 Detailed Description

CHF ZIBOR rate

```
#include <ql/indexes/iborindex.hpp>
#include <ql/time/calendars/switzerland.hpp>
#include <ql/time/daycounters/actual360.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for zibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Zibor](#)
CHF ZIBOR rate

10.49 ql/indexes/iborindex.hpp File Reference

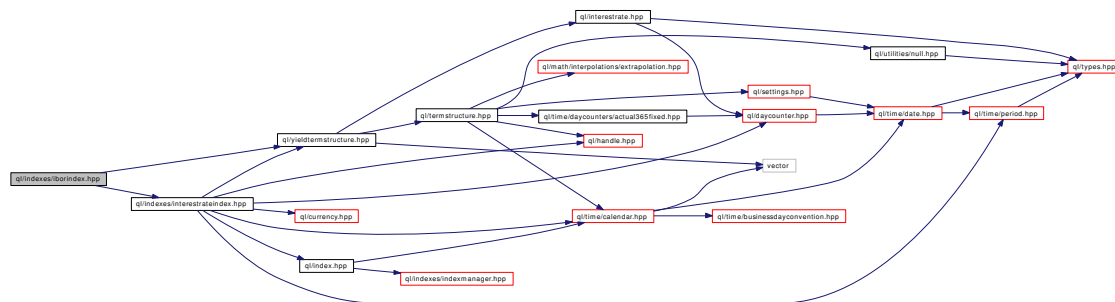
10.49.1 Detailed Description

base class for Inter-Bank-Offered-Rate indexes

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/indexes/interestrateindex.hpp>
```

Include dependency graph for iborindex.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **IborIndex**
base class for Inter-Bank-Offered-Rate indexes (e.g. Libor, etc.)

10.50 ql/indexes/indexmanager.hpp File Reference

10.50.1 Detailed Description

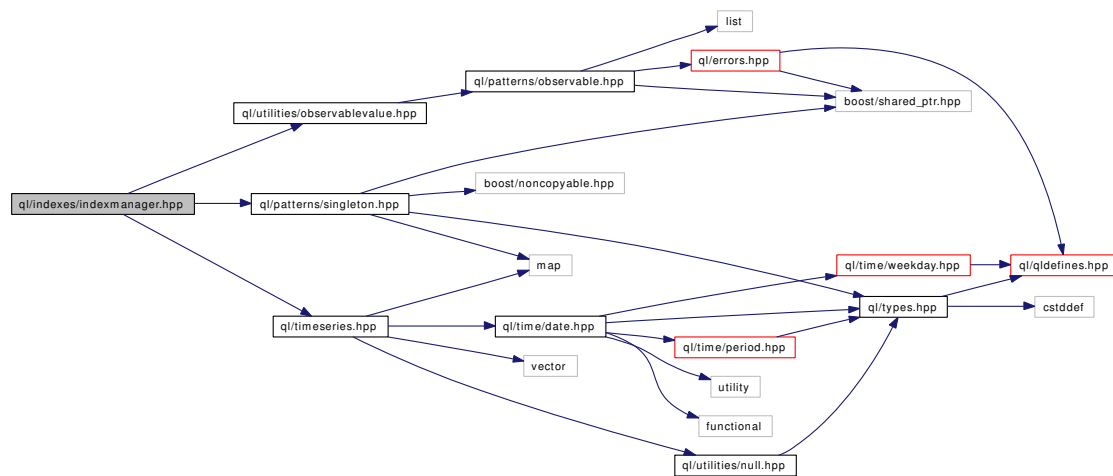
global repository for past index fixings

```
#include <ql/timeseries.hpp>
```

```
#include <ql/patterns/singleton.hpp>
```

```
#include <ql/utilities/observablevalue.hpp>
```

Include dependency graph for indexmanager.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [IndexManager](#)
global repository for past index fixings

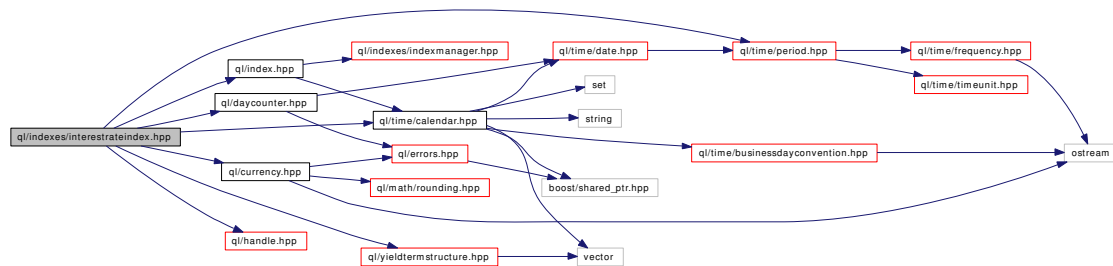
10.51 ql/indexes/interestraterateindex.hpp File Reference

10.51.1 Detailed Description

base class for interest rate indexes

```
#include <ql/index.hpp>
#include <ql/time/calendar.hpp>
#include <ql/currency.hpp>
#include <ql/daycounter.hpp>
#include <ql/time/period.hpp>
#include <ql/handle.hpp>
#include <ql/yieldtermstructure.hpp>
```

Include dependency graph for interestraterateindex.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InterestRateIndex](#)
base class for interest rate indexes

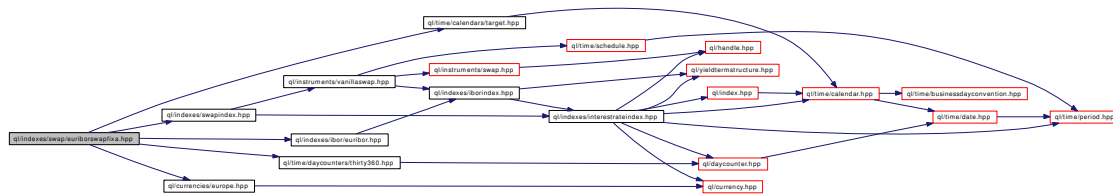
10.52 ql/indexes/swap/euriborswapfixa.hpp File Reference

10.52.1 Detailed Description

EuriborSwapFixA indexes

```
#include <ql/indexes/swapindex.hpp>
#include <ql/indexes/ibor/euribor.hpp>
#include <ql/time/calendars/target.hpp>
#include <ql/time/daycounters/thirty360.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for euriborswapfixa.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **EuriborSwapFixAvs3M**
EuriborSwapFixA vs 3M index base class
- class **EuriborSwapFixAvs6M**
EuriborSwapFixA vs 6M index base class
- class **EuriborSwapFixA1Y**
1-year EuriborSwapFixAvs3M index
- class **EuriborSwapFixA2Y**
2-year EuriborSwapFixAvs6M index
- class **EuriborSwapFixA3Y**
3-year EuriborSwapFixAvs6M index
- class **EuriborSwapFixA4Y**
4-year EuriborSwapFixAvs6M index
- class **EuriborSwapFixA5Y**
5-year EuriborSwapFixAvs6M index

- class [EuriborSwapFixA6Y](#)
6-year EuriborSwapFixAvs6M index
- class [EuriborSwapFixA7Y](#)
7-year EuriborSwapFixAvs6M index
- class [EuriborSwapFixA8Y](#)
8-year EuriborSwapFixAvs6M index
- class [EuriborSwapFixA9Y](#)
9-year EuriborSwapFixAvs6M index
- class [EuriborSwapFixA10Y](#)
10-year EuriborSwapFixAvs6M index
- class [EuriborSwapFixA12Y](#)
12-year EuriborSwapFixAvs6M index
- class [EuriborSwapFixA15Y](#)
15-year EuriborSwapFixAvs6M index
- class [EuriborSwapFixA20Y](#)
20-year EuriborSwapFixAvs6M index
- class [EuriborSwapFixA25Y](#)
25-year EuriborSwapFixAvs6M index
- class [EuriborSwapFixA30Y](#)
30-year EuriborSwapFixAvs6M index

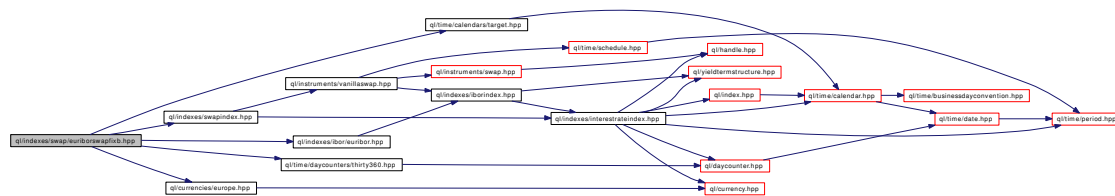
10.53 ql/indexes/swap/euriborswapfixb.hpp File Reference

10.53.1 Detailed Description

EuriborSwapFixB indexes

```
#include <ql/indexes/swapindex.hpp>
#include <ql/indexes/ibor/euribor.hpp>
#include <ql/time/calendars/target.hpp>
#include <ql/time/daycounters/thirty360.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for euriborswapfixb.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **EuriborSwapFixBvs3M**
EuriborSwapFixB vs 3M index base class
- class **EuriborSwapFixBvs6M**
EuriborSwapFixB vs 6M index base class
- class **EuriborSwapFixB1Y**
1-year EuriborSwapFixBvs3M index
- class **EuriborSwapFixB2Y**
2-year EuriborSwapFixBvs6M index
- class **EuriborSwapFixB3Y**
3-year EuriborSwapFixBvs6M index
- class **EuriborSwapFixB4Y**
4-year EuriborSwapFixBvs6M index
- class **EuriborSwapFixB5Y**
5-year EuriborSwapFixBvs6M index

- class [EuriborSwapFixB6Y](#)
6-year EuriborSwapFixBvs6M index
- class [EuriborSwapFixB7Y](#)
7-year EuriborSwapFixBvs6M index
- class [EuriborSwapFixB8Y](#)
8-year EuriborSwapFixBvs6M index
- class [EuriborSwapFixB9Y](#)
9-year EuriborSwapFixBvs6M index
- class [EuriborSwapFixB10Y](#)
10-year EuriborSwapFixBvs6M index
- class [EuriborSwapFixB12Y](#)
12-year EuriborSwapFixBvs6M index
- class [EuriborSwapFixB15Y](#)
15-year EuriborSwapFixBvs6M index
- class [EuriborSwapFixB20Y](#)
20-year EuriborSwapFixBvs6M index
- class [EuriborSwapFixB25Y](#)
25-year EuriborSwapFixBvs6M index
- class [EuriborSwapFixB30Y](#)
30-year EuriborSwapFixBvs6M index

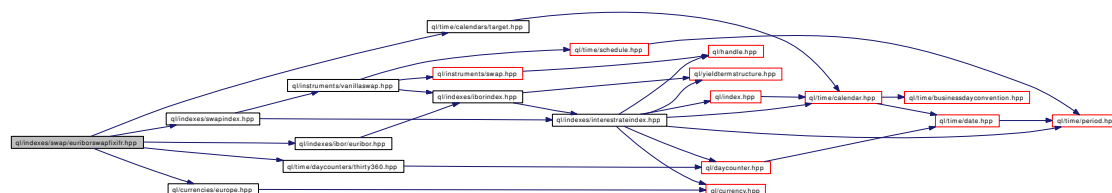
10.54 ql/indexes/swap/euriborswapfixifr.hpp File Reference

10.54.1 Detailed Description

EuriborSwapFixIFR indexes

```
#include <ql/indexes/swapindex.hpp>
#include <ql/indexes/ibor/euribor.hpp>
#include <ql/time/calendars/target.hpp>
#include <ql/time/daycounters/thirty360.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for euriborswapfixifr.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **EuriborSwapFixIFRvs3M**
EuriborSwapFixIFR vs 3M index base class
- class **EuriborSwapFixIFRvs6M**
EuriborSwapFixIFR vs 6M index base class
- class **EuriborSwapFixIFR1Y**
1-year EuriborSwapFixIFR3M index
- class **EuriborSwapFixIFR2Y**
2-year EuriborSwapFixIFRvs6M index
- class **EuriborSwapFixIFR3Y**
3-year EuriborSwapFixIFRvs6M index
- class **EuriborSwapFixIFR4Y**
4-year EuriborSwapFixIFRvs6M index
- class **EuriborSwapFixIFR5Y**
5-year EuriborSwapFixIFRvs6M index

- class [EuriborSwapFixIFR6Y](#)
6-year EuriborSwapFixIFRvs6M index
- class [EuriborSwapFixIFR7Y](#)
7-year EuriborSwapFixIFRvs6M index
- class [EuriborSwapFixIFR8Y](#)
8-year EuriborSwapFixIFRvs6M index
- class [EuriborSwapFixIFR9Y](#)
9-year EuriborSwapFixIFRvs6M index
- class [EuriborSwapFixIFR10Y](#)
10-year EuriborSwapFixIFRvs6M index
- class [EuriborSwapFixIFR12Y](#)
12-year EuriborSwapFixIFRvs6M index
- class [EuriborSwapFixIFR15Y](#)
15-year EuriborSwapFixIFRvs6M index
- class [EuriborSwapFixIFR20Y](#)
20-year EuriborSwapFixIFRvs6M index
- class [EuriborSwapFixIFR25Y](#)
25-year EuriborSwapFixIFRvs6M index
- class [EuriborSwapFixIFR30Y](#)
30-year EuriborSwapFixIFRvs6M index

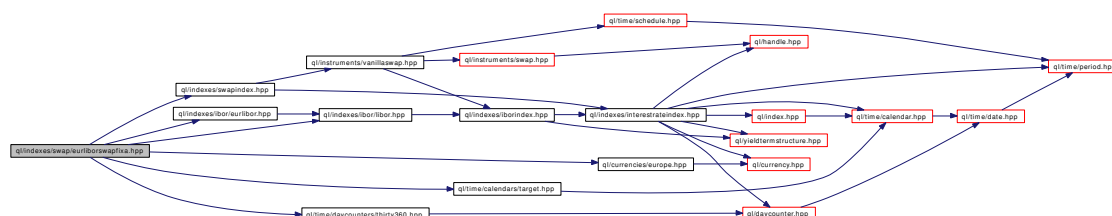
10.55 ql/indexes/swap/eurliborswapfixa.hpp File Reference

10.55.1 Detailed Description

EurliborSwapFixA indexes

```
#include <ql/indexes/swapindex.hpp>
#include <ql/indexes/ibor/eurlibor.hpp>
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/target.hpp>
#include <ql/time/daycounters/thirty360.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for `eurliborswapfixa.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [EurliborSwapFixAvs3M](#)
EurliborSwapFixA vs 3M index base class
- class [EurliborSwapFixAvs6M](#)
EurliborSwapFixA vs 6M index base class
- class [EurliborSwapFixA1Y](#)
1-year EurliborSwapFixAvs3M index
- class [EurliborSwapFixA2Y](#)
2-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA3Y](#)
3-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA4Y](#)
4-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA5Y](#)

5-year EurliborSwapFixAvs6M index

- class [EurliborSwapFixA6Y](#)
6-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA7Y](#)
7-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA8Y](#)
8-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA9Y](#)
9-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA10Y](#)
10-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA12Y](#)
12-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA15Y](#)
15-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA20Y](#)
20-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA25Y](#)
25-year EurliborSwapFixAvs6M index
- class [EurliborSwapFixA30Y](#)
30-year EurliborSwapFixAvs6M index

5-year EurliborSwapFixBvs6M index

- class [EurliborSwapFixB6Y](#)
6-year EurliborSwapFixBvs6M index
- class [EurliborSwapFixB7Y](#)
7-year EurliborSwapFixBvs6M index
- class [EurliborSwapFixB8Y](#)
8-year EurliborSwapFixBvs6M index
- class [EurliborSwapFixB9Y](#)
9-year EurliborSwapFixBvs6M index
- class [EurliborSwapFixB10Y](#)
10-year EurliborSwapFixBvs6M index
- class [EurliborSwapFixB12Y](#)
12-year EurliborSwapFixBvs6M index
- class [EurliborSwapFixB15Y](#)
15-year EurliborSwapFixBvs6M index
- class [EurliborSwapFixB20Y](#)
20-year EurliborSwapFixBvs6M index
- class [EurliborSwapFixB25Y](#)
25-year EurliborSwapFixBvs6M index
- class [EurliborSwapFixB30Y](#)
30-year EurliborSwapFixBvs6M index

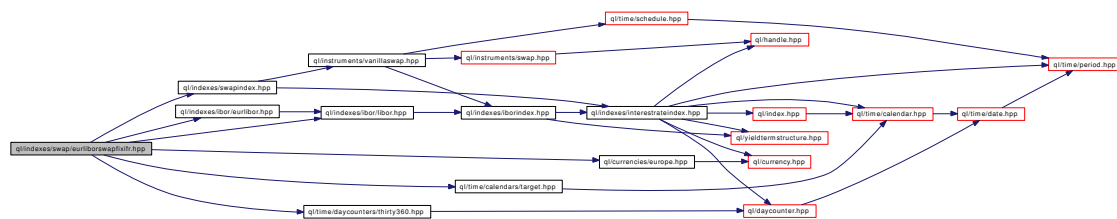
10.57 ql/indexes/swap/eurliborswapfixifr.hpp File Reference

10.57.1 Detailed Description

EurliborSwapFixIFR indexes

```
#include <ql/indexes/swapindex.hpp>
#include <ql/indexes/ibor/eurlibor.hpp>
#include <ql/indexes/ibor/libor.hpp>
#include <ql/time/calendars/target.hpp>
#include <ql/time/daycounters/thirty360.hpp>
#include <ql/currencies/europe.hpp>
```

Include dependency graph for eurliborswapfixifr.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **EurliborSwapFixIFRvs3M**
EurliborSwapFixIFR vs 3M index base class
- class **EurliborSwapFixIFRvs6M**
EurliborSwapFixIFR vs 6M index base class
- class **EurliborSwapFixIFR1Y**
1-year EurliborSwapFixIFRvs3M index
- class **EurliborSwapFixIFR2Y**
2-year EurliborSwapFixIFRvs6M index
- class **EurliborSwapFixIFR3Y**
3-year EurliborSwapFixIFRvs6M index
- class **EurliborSwapFixIFR4Y**
4-year EurliborSwapFixIFRvs6M index
- class **EurliborSwapFixIFR5Y**

5-year EurliborSwapFixIFRvs6M index

- class [EurliborSwapFixIFR6Y](#)
6-year EurliborSwapFixIFRvs6M index
- class [EurliborSwapFixIFR7Y](#)
7-year EurliborSwapFixIFRvs6M index
- class [EurliborSwapFixIFR8Y](#)
8-year EurliborSwapFixIFRvs6M index
- class [EurliborSwapFixIFR9Y](#)
9-year EurliborSwapFixIFRvs6M index
- class [EurliborSwapFixIFR10Y](#)
10-year EurliborSwapFixIFRvs6M index
- class [EurliborSwapFixIFR12Y](#)
12-year EurliborSwapFixIFRvs6M index
- class [EurliborSwapFixIFR15Y](#)
15-year EurliborSwapFixIFRvs6M index
- class [EurliborSwapFixIFR20Y](#)
20-year EurliborSwapFixIFRvs6M index
- class [EurliborSwapFixIFR25Y](#)
25-year EurliborSwapFixIFRvs6M index
- class [EurliborSwapFixIFR30Y](#)
30-year EurliborSwapFixIFRvs6M index

10.59 ql/instrument.hpp File Reference

10.59.1 Detailed Description

Abstract instrument class.

```
#include <ql/patterns/lazyobject.hpp>
```

```
#include <ql/pricingengine.hpp>
```

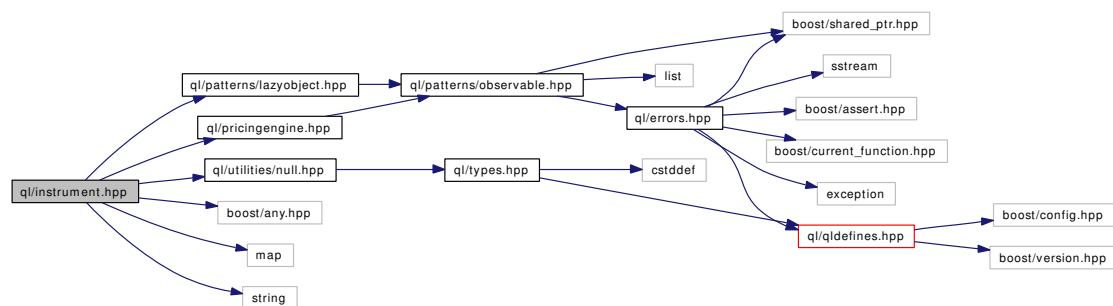
```
#include <ql/utilities/null.hpp>
```

```
#include <boost/any.hpp>
```

```
#include <map>
```

```
#include <string>
```

Include dependency graph for instrument.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Instrument](#)
Abstract instrument class.

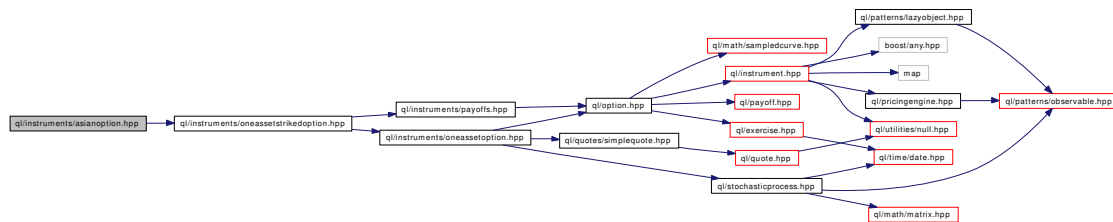
10.60 ql/instruments/asianoption.hpp File Reference

10.60.1 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.

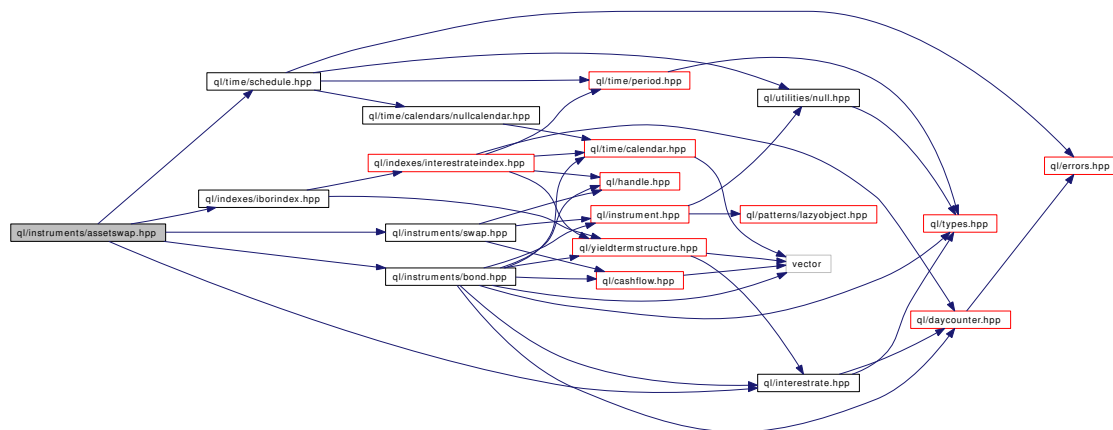
10.61 ql/instruments/assetswap.hpp File Reference

10.61.1 Detailed Description

Bullet bond vs Libor swap.

```
#include <ql/instruments/swap.hpp>
#include <ql/instruments/bond.hpp>
#include <ql/indexes/iborindex.hpp>
#include <ql/interestrate.hpp>
#include <ql/time/schedule.hpp>
```

Include dependency graph for assetswap.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AssetSwap](#)
Bullet bond vs Libor swap.
- class [AssetSwap::arguments](#)
Arguments for asset swap calculation
- class [AssetSwap::results](#)
Results from simple swap calculation

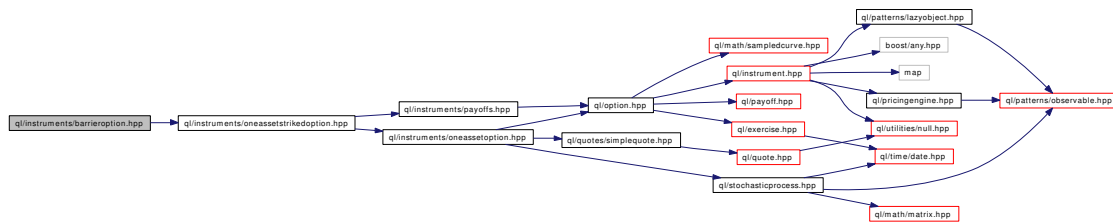
10.62 ql/instruments/barrieroption.hpp File Reference

10.62.1 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-option engine base class

10.63 ql/instruments/basketoption.hpp File Reference

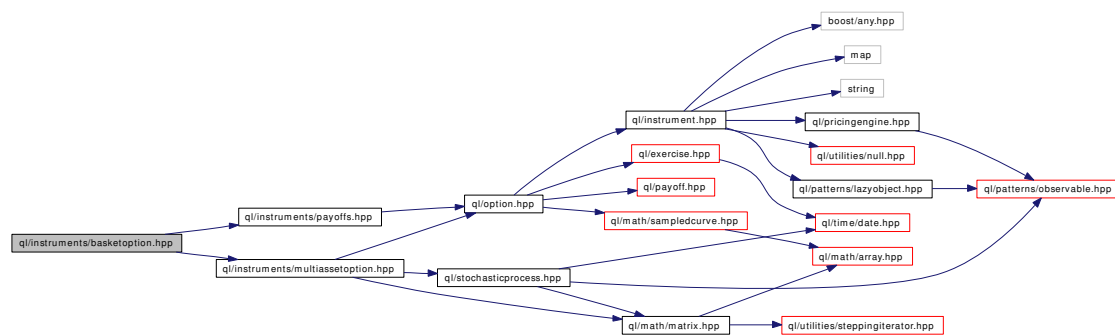
10.63.1 Detailed Description

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

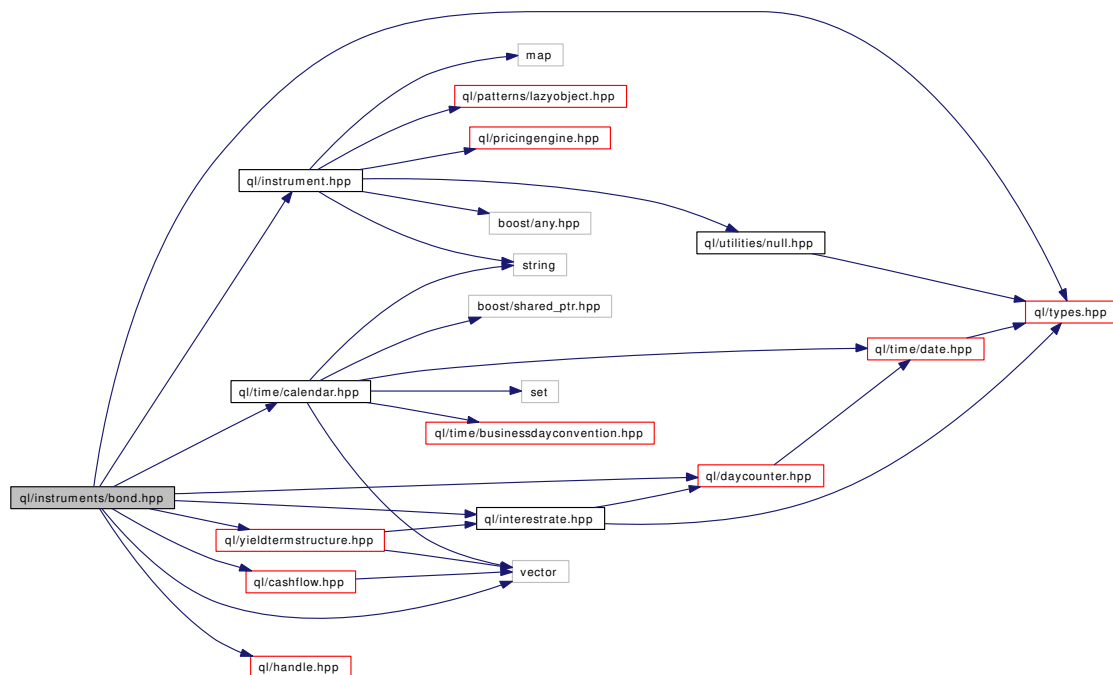
10.64 ql/instruments/bond.hpp File Reference

10.64.1 Detailed Description

concrete bond class

```
#include <ql/instrument.hpp>
#include <ql/time/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/interestrate.hpp>
#include <ql/types.hpp>
#include <ql/handle.hpp>
#include <ql/cashflow.hpp>
#include <ql/yieldtermstructure.hpp>
#include <vector>
```

Include dependency graph for bond.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `Bond`

Base bond class.

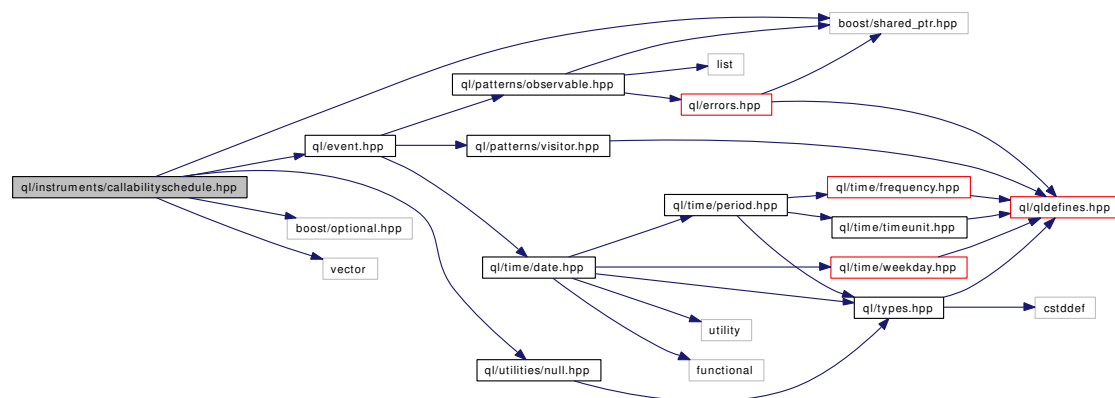
10.65 ql/instruments/callabilityschedule.hpp File Reference

10.65.1 Detailed Description

Schedule of put/call dates.

```
#include <ql/event.hpp>
#include <ql/utilities/null.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/optional.hpp>
#include <vector>
```

Include dependency graph for callabilityschedule.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Callability**
instrument callability
- class **Callability::Price**
amount to be paid upon callability

Typedefs

- typedef `std::vector< boost::shared_ptr< Callability > >` **CallabilitySchedule**

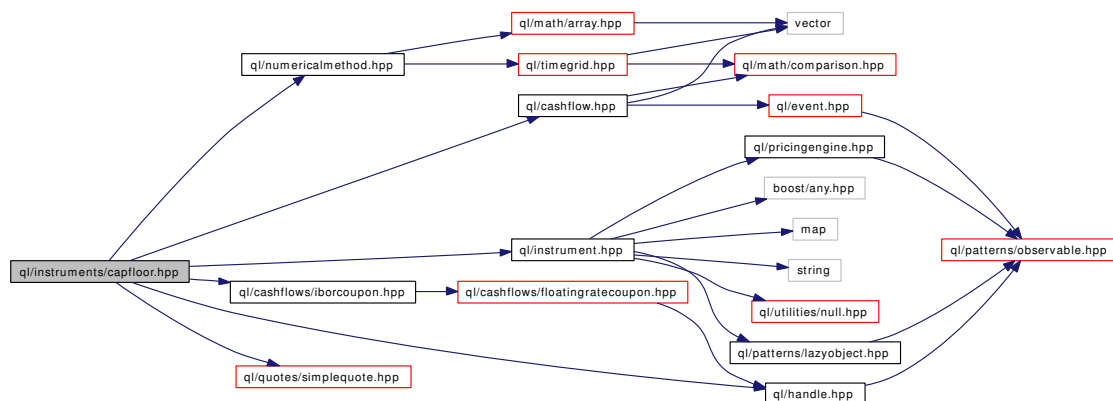
10.66 ql/instruments/capfloor.hpp File Reference

10.66.1 Detailed Description

cap and floor class

```
#include <ql/numericalmethod.hpp>
#include <ql/instrument.hpp>
#include <ql/cashflow.hpp>
#include <ql/cashflows/iborcoupon.hpp>
#include <ql/handle.hpp>
#include <ql/quotes/simplequote.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::engine](#)
base class for cap/floor engines

Functions

- `std::ostream & operator<< (std::ostream &, CapFloor::Type)`

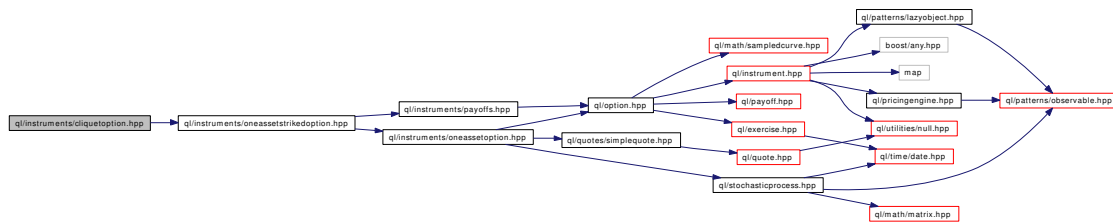
10.67 ql/instruments/cliquestoption.hpp File Reference

10.67.1 Detailed Description

Cliquet option.

```
#include <ql/instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for cliquestoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CliquetOption**
cliquet (Ratchet) option
- class **CliquetOption::arguments**
Arguments for cliquet option calculation
- class **CliquetOption::engine**
Cliquet engine base class.

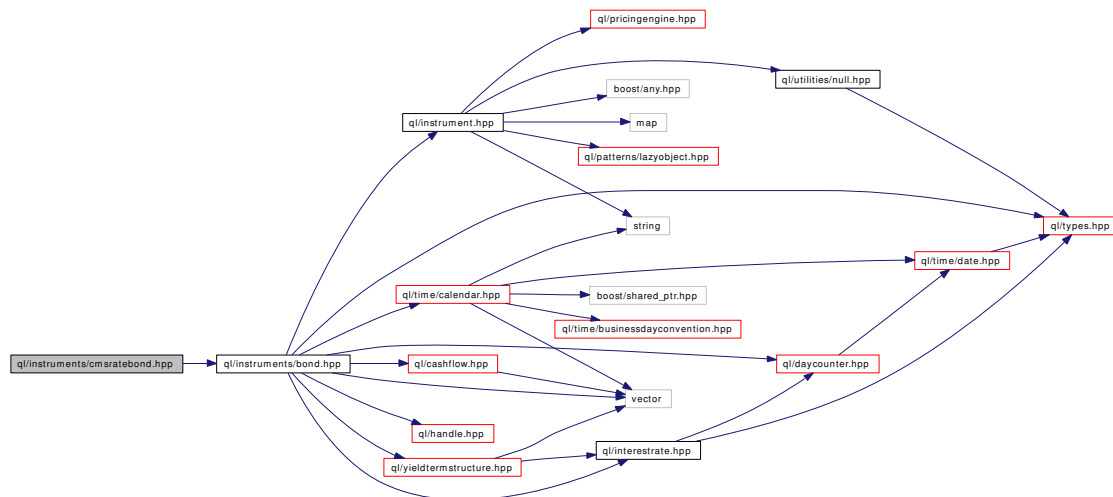
10.68 ql/instruments/cmsratebond.hpp File Reference

10.68.1 Detailed Description

CMS-rate bond.

```
#include <ql/instruments/bond.hpp>
```

Include dependency graph for cmsratebond.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CmsRateBond**
CMS-rate bond.

10.69 ql/instruments/compositeinstrument.hpp File Reference

10.69.1 Detailed Description

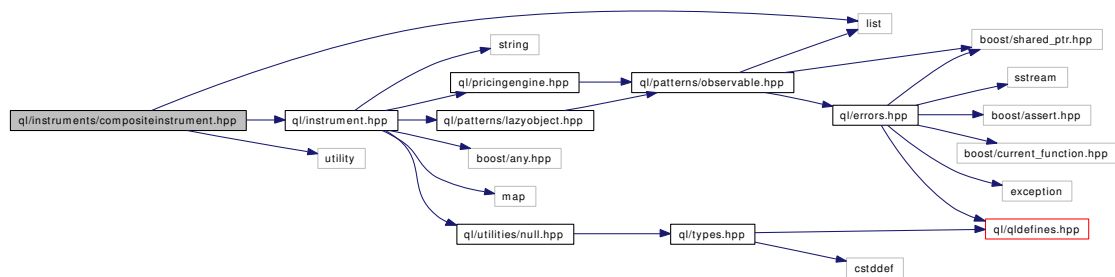
Composite instrument class.

```
#include <ql/instrument.hpp>
```

```
#include <list>
```

```
#include <utility>
```

Include dependency graph for compositeinstrument.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CompositeInstrument](#)
Composite instrument

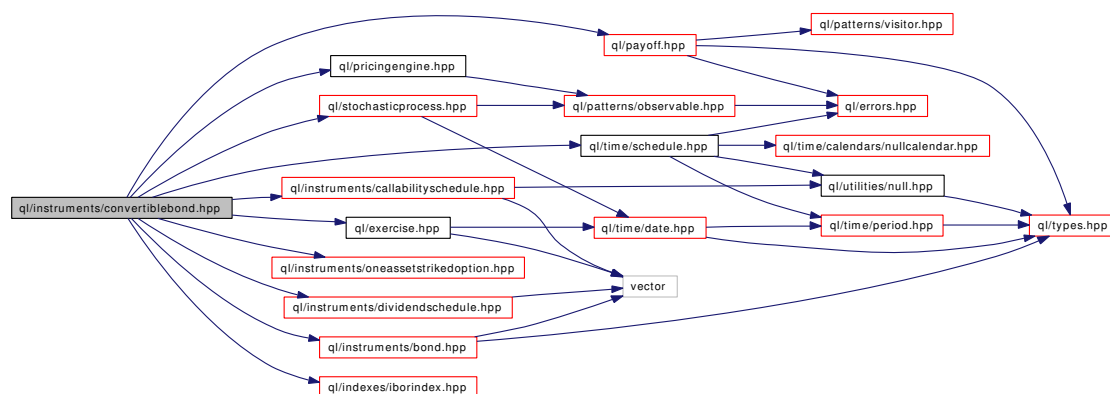
10.70 ql/instruments/convertiblebond.hpp File Reference

10.70.1 Detailed Description

convertible bond class

```
#include <ql/time/schedule.hpp>
#include <ql/exercise.hpp>
#include <ql/pricingengine.hpp>
#include <ql/payoff.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/instruments/bond.hpp>
#include <ql/instruments/oneassetstrikedoption.hpp>
#include <ql/instruments/dividendschedule.hpp>
#include <ql/instruments/callabilityschedule.hpp>
#include <ql/indexes/iborindex.hpp>
```

Include dependency graph for convertiblebond.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SoftCallability](#)
callability leaving to the holder the possibility to convert
- class [ConvertibleBond](#)
base class for convertible bonds
- class [ConvertibleZeroCouponBond](#)
convertible zero-coupon bond

- class [ConvertibleFixedCouponBond](#)
convertible fixed-coupon bond
- class [ConvertibleFloatingRateBond](#)
convertible floating-rate bond

10.71 ql/instruments/dividendschedule.hpp File Reference

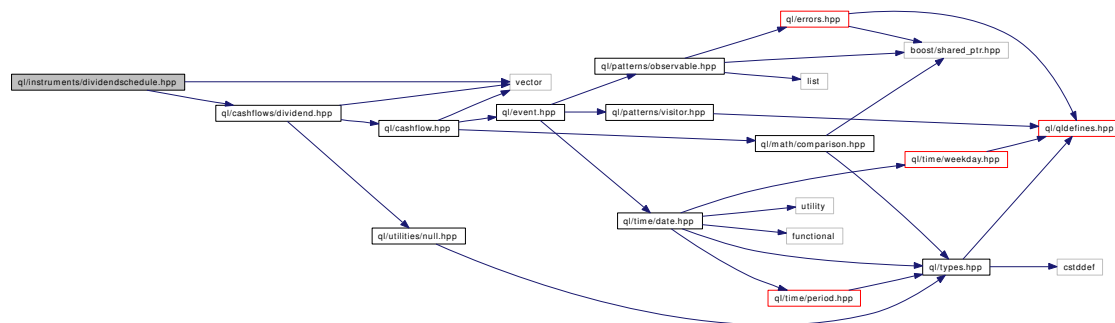
10.71.1 Detailed Description

Schedule of dividend dates.

```
#include <ql/cashflows/dividend.hpp>
```

```
#include <vector>
```

Include dependency graph for dividendschedule.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef std::vector< boost::shared_ptr< Dividend > > **DividendSchedule**

10.72 ql/instruments/dividendvanillaoption.hpp File Reference

10.72.1 Detailed Description

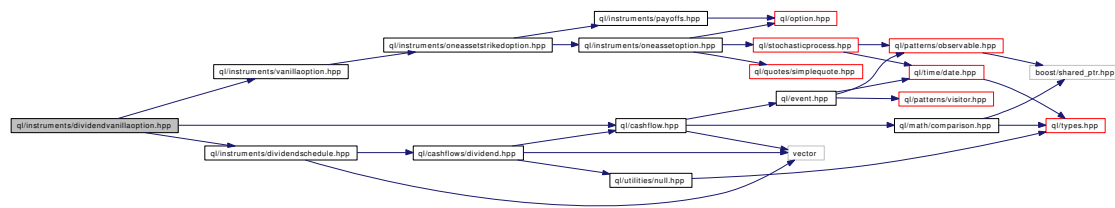
Vanilla option on a single asset with discrete dividends.

```
#include <ql/instruments/vanillaoption.hpp>
```

```
#include <ql/instruments/dividendschedule.hpp>
```

```
#include <ql/cashflow.hpp>
```

Include dependency graph for dividendvanillaoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **DividendVanillaOption**
Single-asset vanilla option (no barriers) with discrete dividends.
- class **DividendVanillaOption::arguments**
Arguments for dividend vanilla option calculation
- class **DividendVanillaOption::engine**
Dividend-vanilla-option engine base class

10.73 ql/instruments/europeanoption.hpp File Reference

10.73.1 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.

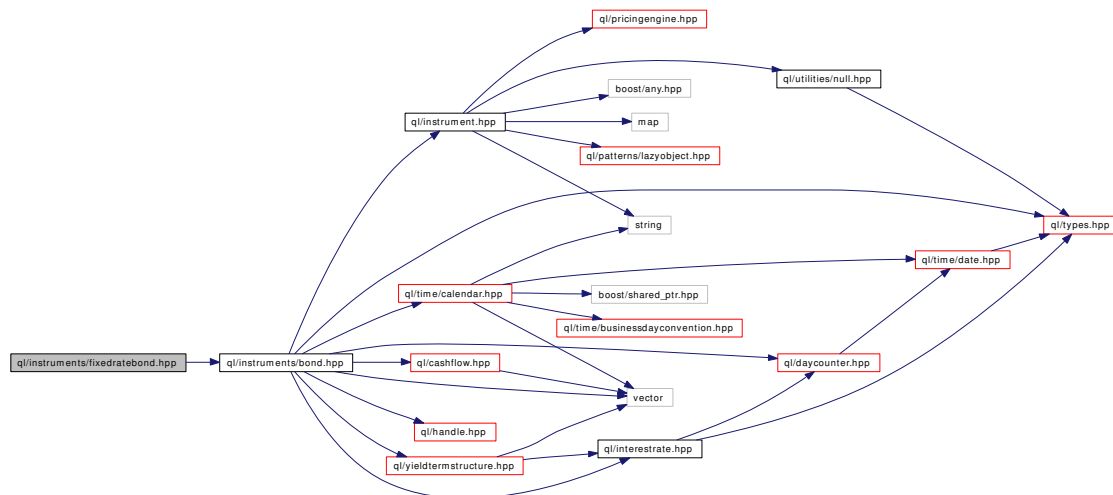
10.74 ql/instruments/fixedratebond.hpp File Reference

10.74.1 Detailed Description

fixed-rate bond

```
#include <ql/instruments/bond.hpp>
```

Include dependency graph for fixedratebond.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FixedRateBond**
fixed-rate bond

10.75 ql/instruments/fixedratebondforward.hpp File Reference

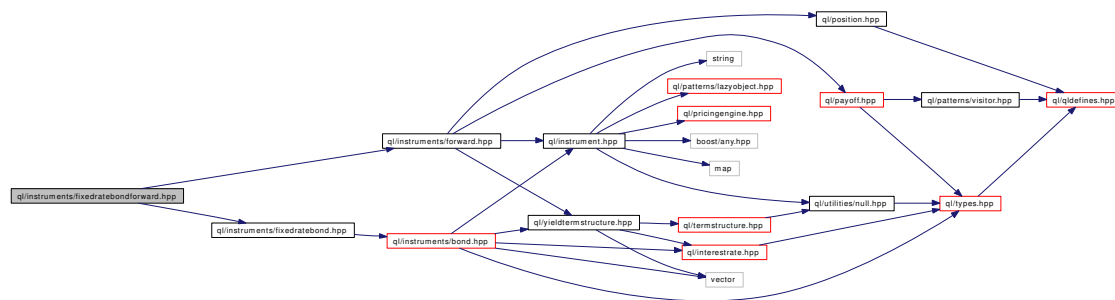
10.75.1 Detailed Description

forward contract on a fixed-rate bond

```
#include <ql/instruments/forward.hpp>
```

```
#include <ql/instruments/fixedratebond.hpp>
```

Include dependency graph for fixedratebondforward.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FixedRateBondForward**
Forward contract on a fixed-rate bond

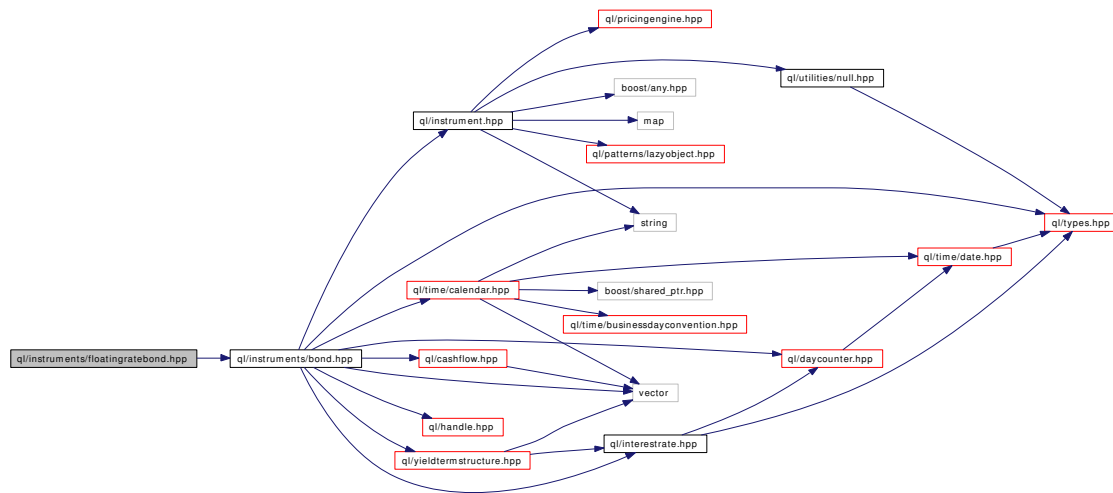
10.76 ql/instruments/floatingratebond.hpp File Reference

10.76.1 Detailed Description

floating-rate bond

```
#include <ql/instruments/bond.hpp>
```

Include dependency graph for floatingratebond.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FloatingRateBond**
floating-rate bond (possibly capped and/or floored)

10.77 ql/instruments/forward.hpp File Reference

10.77.1 Detailed Description

Base forward class.

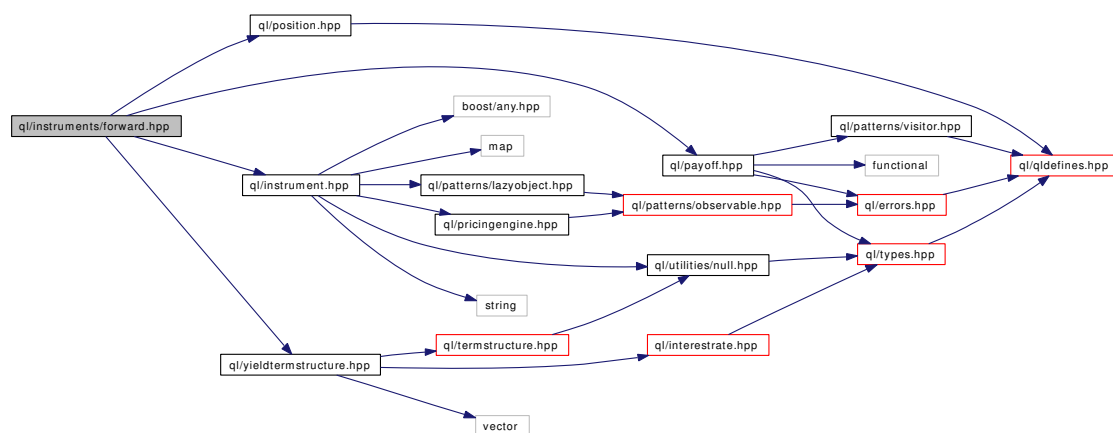
```
#include <ql/instrument.hpp>
```

```
#include <ql/position.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/payoff.hpp>
```

Include dependency graph for forward.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Forward**
Abstract base forward class.
- class **ForwardTypePayoff**
Class for forward type payoffs.

10.78 ql/instruments/forwardrateagreement.hpp File Reference

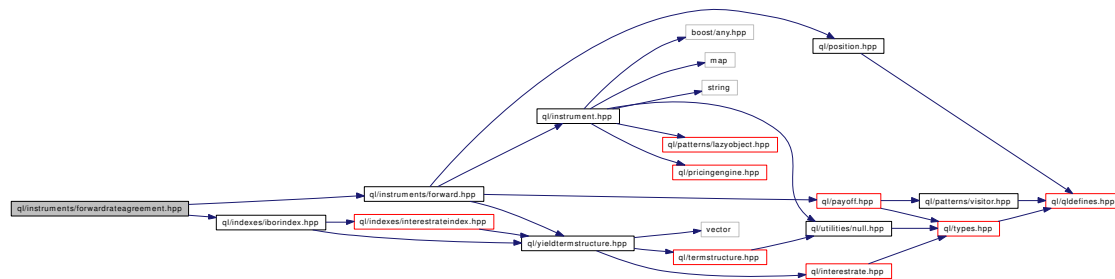
10.78.1 Detailed Description

forward rate agreement

```
#include <ql/instruments/forward.hpp>
```

```
#include <ql/indexes/iborindex.hpp>
```

Include dependency graph for forwardrateagreement.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ForwardRateAgreement](#)
Forward rate agreement (FRA) class

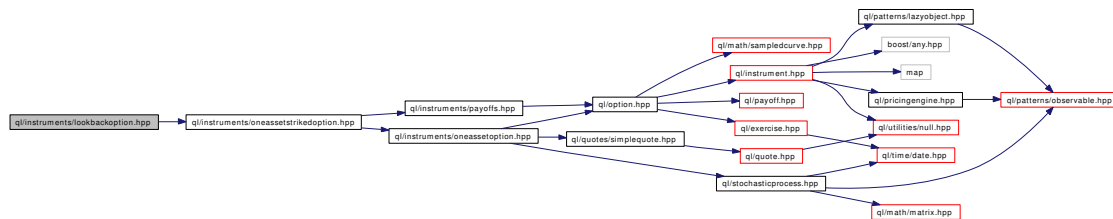
10.80 ql/instruments/lookbackoption.hpp File Reference

10.80.1 Detailed Description

Lookback option on a single asset.

```
#include <ql/instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for lookbackoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ContinuousFloatingLookbackOption](#)
Continuous-floating lookback option.
- class [ContinuousFixedLookbackOption](#)
Continuous-fixed lookback option.
- class [ContinuousFloatingLookbackOption::arguments](#)
Arguments for continuous floating lookback option calculation
- class [ContinuousFixedLookbackOption::arguments](#)
Arguments for continuous fixed lookback option calculation
- class [ContinuousFloatingLookbackOption::engine](#)
Continuous floating lookback engine base class
- class [ContinuousFixedLookbackOption::engine](#)
Continuous fixed lookback engine base class

10.81 ql/instruments/makecapfloor.hpp File Reference

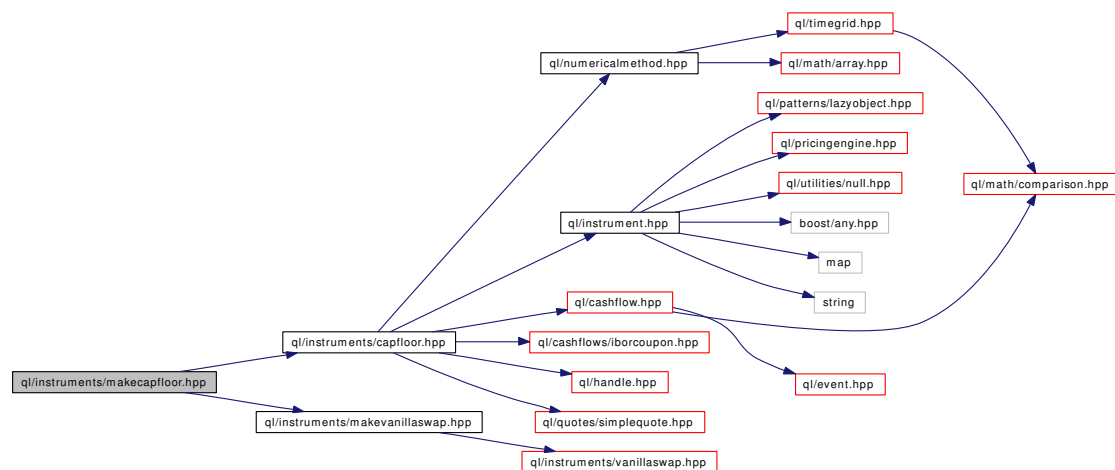
10.81.1 Detailed Description

Helper class to instantiate standard market cap/floor.

```
#include <ql/instruments/capfloor.hpp>
```

```
#include <ql/instruments/makevanillaswap.hpp>
```

Include dependency graph for makecapfloor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MakeCapFloor**
helper class

10.82 ql/instruments/makecms.hpp File Reference

10.82.1 Detailed Description

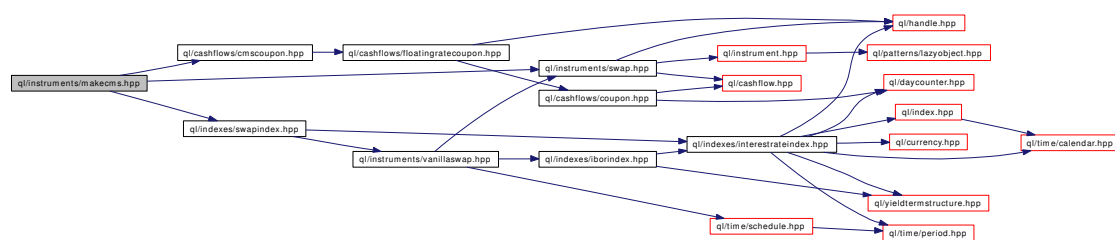
Helper class to instantiate standard market CMS.

```
#include <ql/cashflows/cmscoupon.hpp>
```

```
#include <ql/indexes/swapindex.hpp>
```

```
#include <ql/instruments/swap.hpp>
```

Include dependency graph for makecms.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MakeCms**
helper class

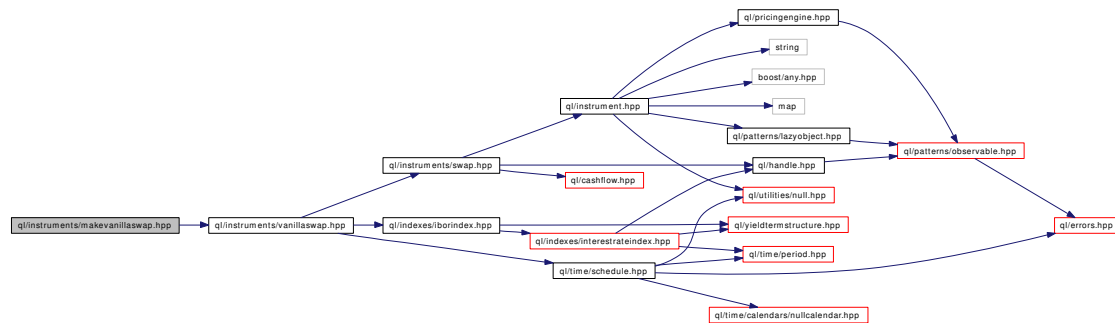
10.83 ql/instruments/makevanillaswap.hpp File Reference

10.83.1 Detailed Description

Helper class to instantiate standard market swaps.

```
#include <ql/instruments/vanillaswap.hpp>
```

Include dependency graph for makevanillaswap.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MakeVanillaSwap**
helper class

10.84 ql/instruments/multiassetoption.hpp File Reference

10.84.1 Detailed Description

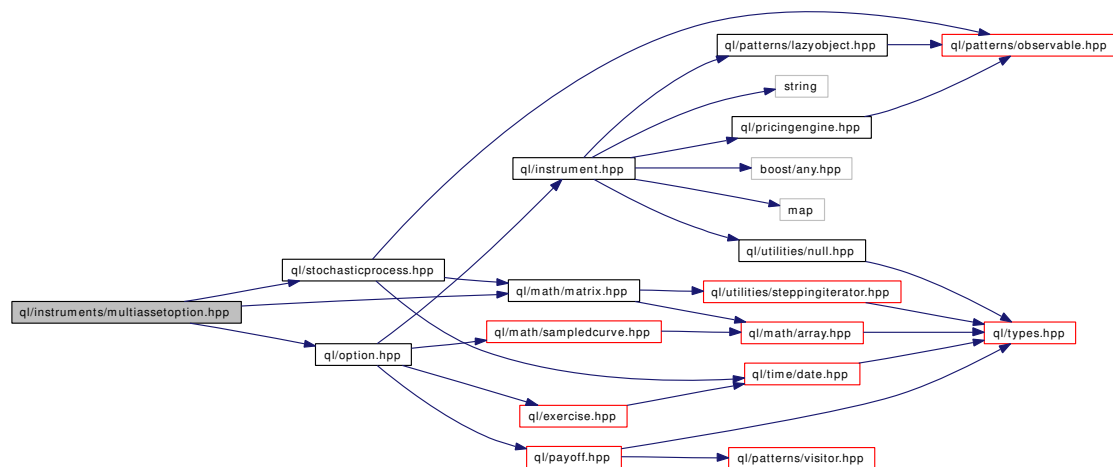
Option on multiple assets.

```
#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

10.85 ql/instruments/oneassetoption.hpp File Reference

10.85.1 Detailed Description

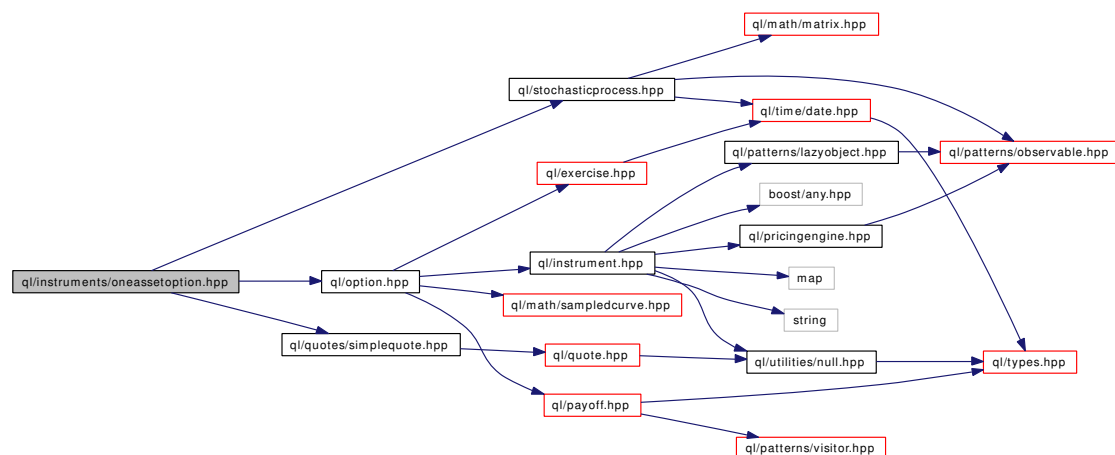
Option on a single asset.

```
#include <ql/option.hpp>
```

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/quotes/simplequote.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

10.86 ql/instruments/oneassetstrikedoption.hpp File Reference

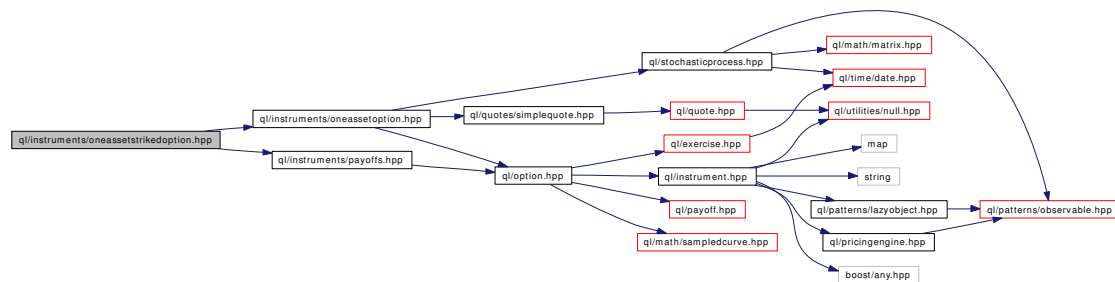
10.86.1 Detailed Description

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.

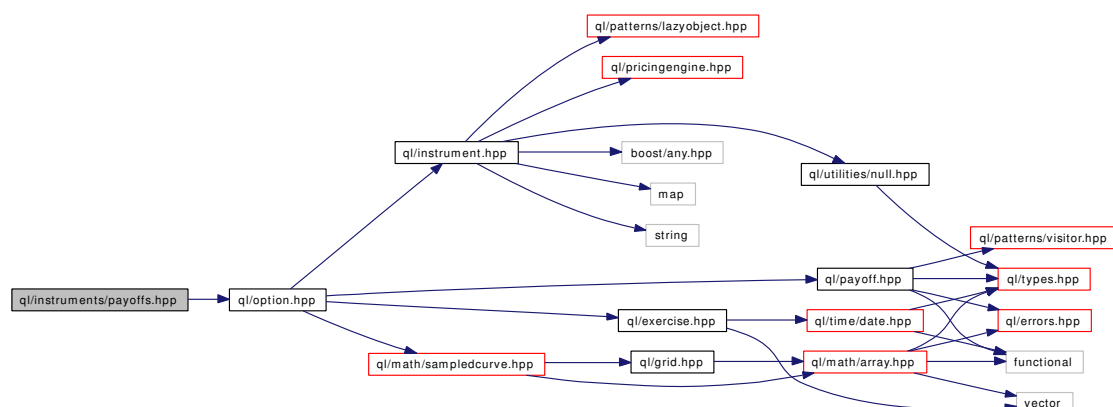
10.87 ql/instruments/payoffs.hpp File Reference

10.87.1 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 put/call payoffs.
- class **FloatingTypePayoff**
Payoff based on a floating strike
- 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 **AssetOrNothingPayoff**
Binary asset-or-nothing payoff.
- class **CashOrNothingPayoff**
Binary cash-or-nothing payoff.

- class [GapPayoff](#)
Binary gap payoff.
- class [SuperFundPayoff](#)
Binary superfund payoff.
- class [SuperSharePayoff](#)
Binary supershare payoff.

10.91 ql/instruments/stock.hpp File Reference

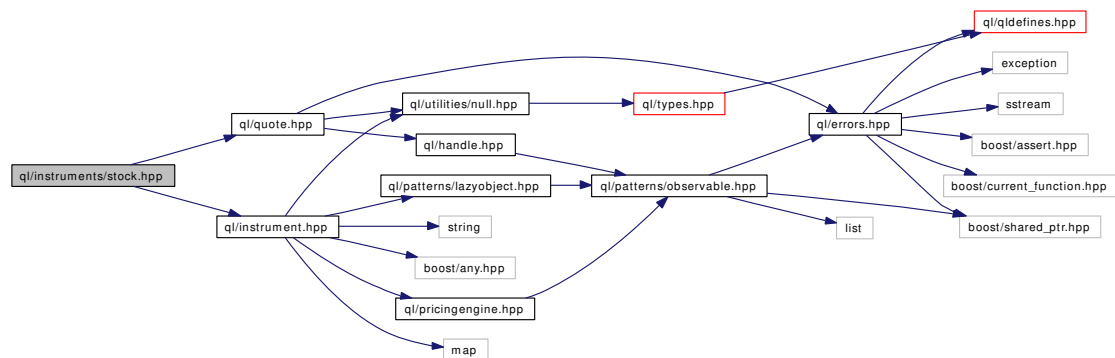
10.91.1 Detailed Description

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.

10.92 ql/instruments/swap.hpp File Reference

10.92.1 Detailed Description

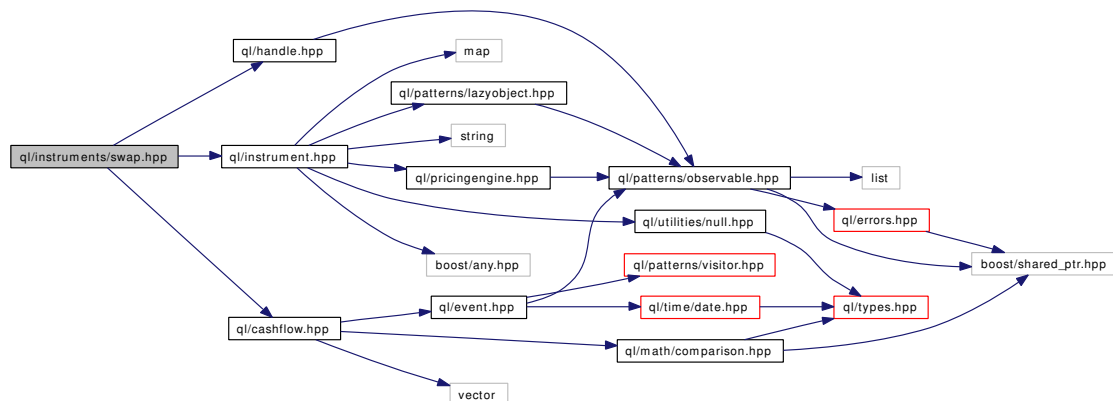
Interest rate swap.

```
#include <ql/instrument.hpp>
```

```
#include <ql/handle.hpp>
```

```
#include <ql/cashflow.hpp>
```

Include dependency graph for swap.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Swap](#)
Interest rate swap.

10.93 ql/instruments/swaption.hpp File Reference

10.93.1 Detailed Description

Swaption class.

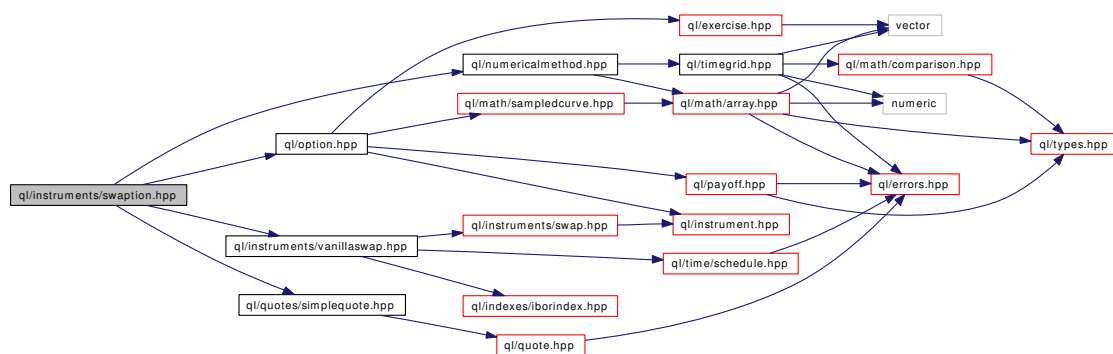
```
#include <ql/numericalmethod.hpp>
```

```
#include <ql/option.hpp>
```

```
#include <ql/instruments/vanillaswap.hpp>
```

```
#include <ql/quotes/simplequote.hpp>
```

Include dependency graph for swaption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct **Settlement**
settlement information
- class **Swaption**
Swaption class
- class **Swaption::arguments**
Arguments for swaption calculation
- class **Swaption::engine**
base class for swaption engines

Functions

- `std::ostream & operator<< (std::ostream &out, Settlement::Type type)`

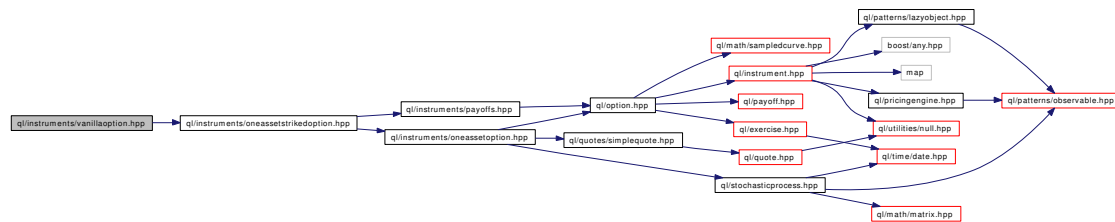
10.94 ql/instruments/vanillaoption.hpp File Reference

10.94.1 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.

10.95 ql/instruments/vanillaswap.hpp File Reference

10.95.1 Detailed Description

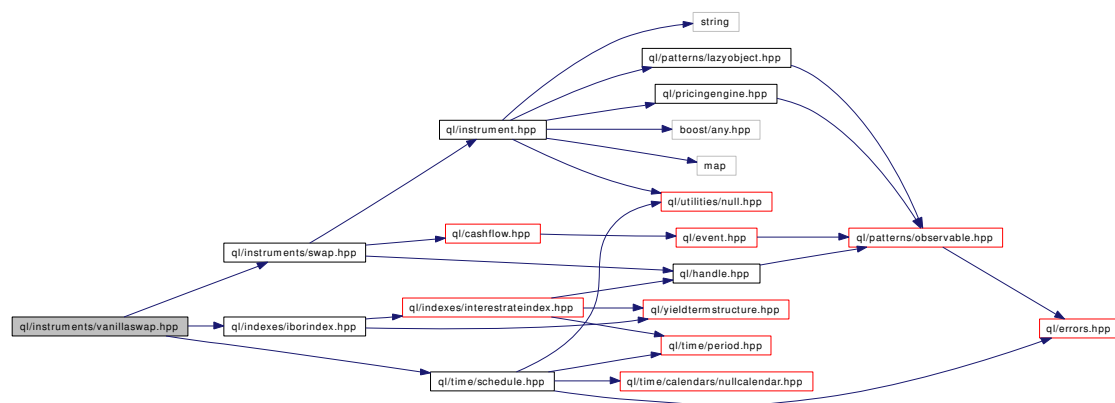
Simple fixed-rate vs Libor swap.

```
#include <ql/instruments/swap.hpp>
```

```
#include <ql/indexes/iborindex.hpp>
```

```
#include <ql/time/schedule.hpp>
```

Include dependency graph for vanillaswap.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [VanillaSwap::arguments](#)
Arguments for simple swap calculation
- class [VanillaSwap::results](#)
Results from simple swap calculation

Functions

- `std::ostream & operator<< (std::ostream &out, VanillaSwap::Type type)`

10.96 ql/instruments/varianceswap.hpp File Reference

10.96.1 Detailed Description

Variance swap.

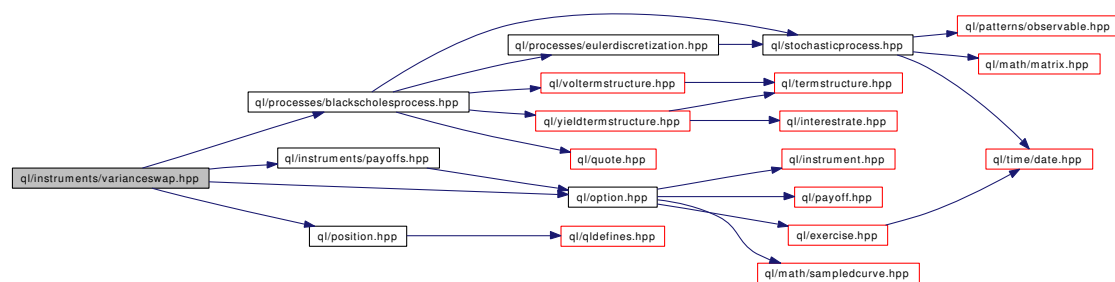
```
#include <ql/processes/blackscholesprocess.hpp>
```

```
#include <ql/instruments/payoffs.hpp>
```

```
#include <ql/option.hpp>
```

```
#include <ql/position.hpp>
```

Include dependency graph for varianceswap.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **VarianceSwap**
Variance swap.
- class **VarianceSwap::arguments**
Arguments for forward fair-variance calculation
- class **VarianceSwap::results**
Results from variance-swap calculation
- class **VarianceSwap::engine**
base class for variance-swap engines

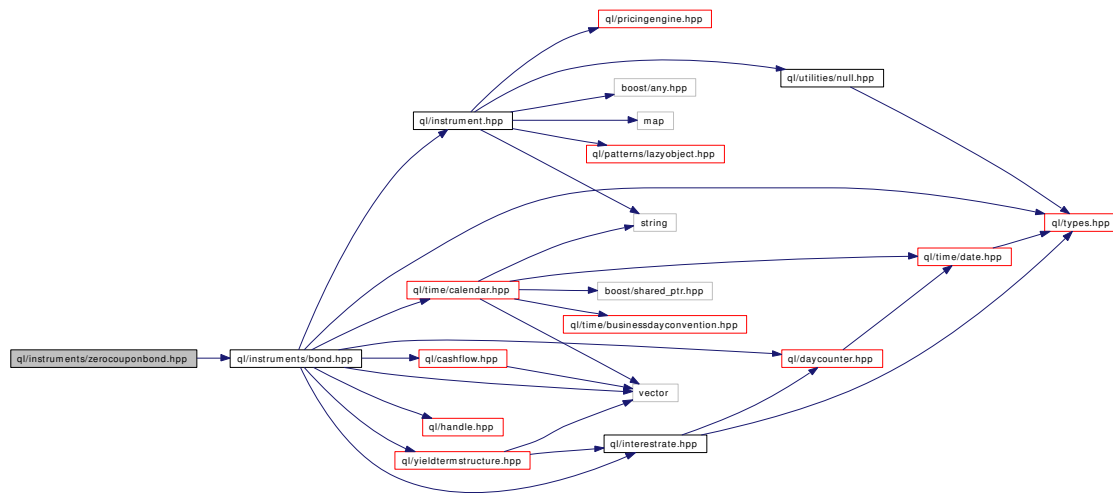
10.97 ql/instruments/zerocouponbond.hpp File Reference

10.97.1 Detailed Description

zero-coupon bond

```
#include <ql/instruments/bond.hpp>
```

Include dependency graph for zerocouponbond.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **ZeroCouponBond**
zero-coupon bond

10.98 ql/interestrate.hpp File Reference

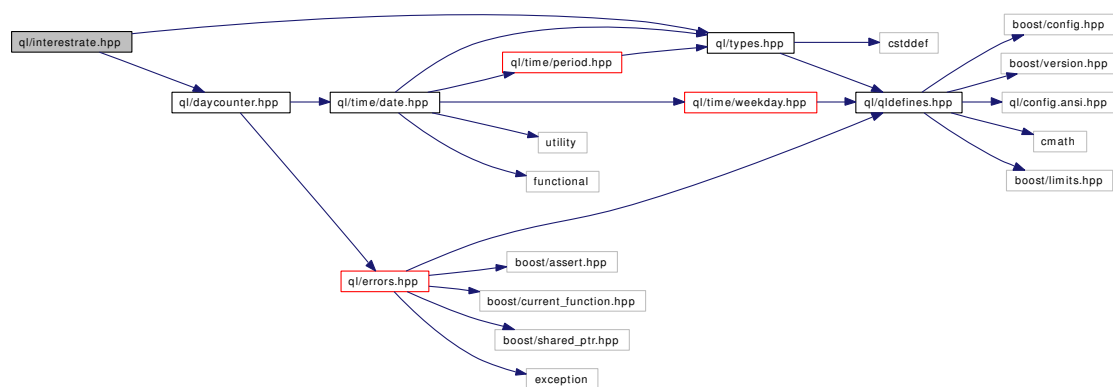
10.98.1 Detailed Description

Instrument rate class.

```
#include <ql/types.hpp>
```

```
#include <ql/daycounter.hpp>
```

Include dependency graph for interestrate.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InterestRate](#)
Concrete interest rate class.

Enumerations

- enum **Compounding** { [Simple](#) = 0, [Compounded](#) = 1, [Continuous](#) = 2, [SimpleThenCompounded](#) }
Interest rate compounding rule.

10.99 ql/legacy/libormarketmodels/lfmcovarproxy.hpp File Reference

10.99.1 Detailed Description

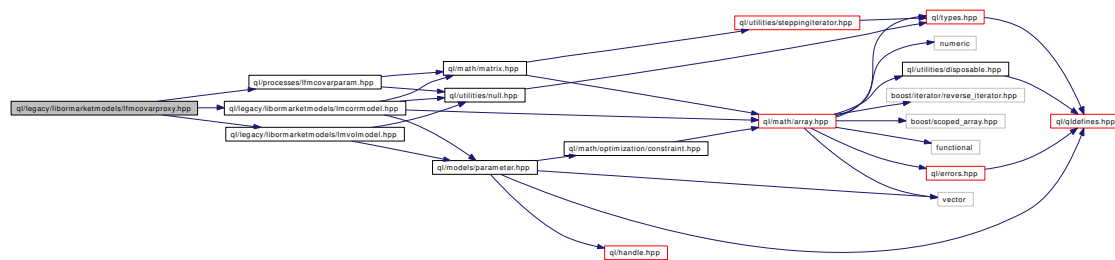
proxy for libor forward covariance parameterization

```
#include <ql/processes/lfmcovarparam.hpp>
```

```
#include <ql/legacy/libormarketmodels/lmvolmodel.hpp>
```

```
#include <ql/legacy/libormarketmodels/lmcorrmodel.hpp>
```

Include dependency graph for lfmcovarproxy.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **LfmCovarianceProxy**
proxy for a libor forward model covariance parameterization

10.100 ql/legacy/libormarketmodels/liborforwardmodel.hpp File Reference

10.100.1 Detailed Description

libor forward model incl. exact cap pricing Rebonato formula to approximate swaption prices.

```
#include <ql/processes/lfprocess.hpp>
```

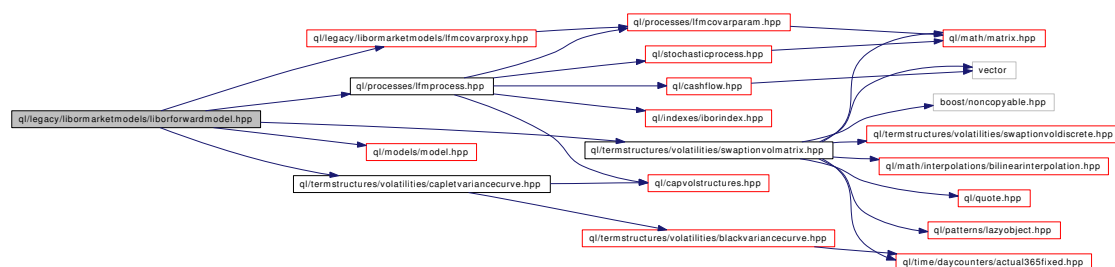
```
#include <ql/termstructures/volatilities/swaptionvolmatrix.hpp>
```

```
#include <ql/termstructures/volatilities/capletvariancecurve.hpp>
```

```
#include <ql/models/model.hpp>
```

```
#include <ql/legacy/libormarketmodels/lfmcovarproxy.hpp>
```

Include dependency graph for liborforwardmodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **LiborForwardModel**
Libor forward model

10.102.1 Detailed Description

```
#include <ql/legacy/libormarketmodels/lmvolmodel.hpp>
```

[illegible]

- namespace **QuantLib**

- class **LmConstWrapperVolatilityModel**
caplet const volatility model

10.103 ql/legacy/libormarketmodels/lmcorrmodel.hpp File Reference

10.103.1 Detailed Description

correlation model for libor market models

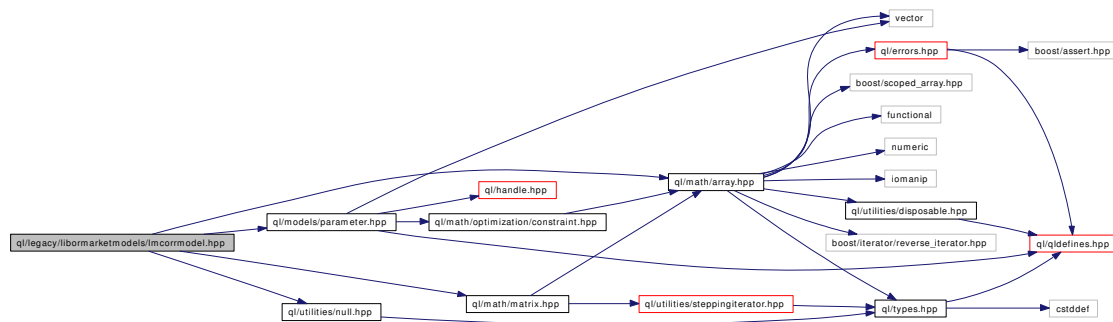
```
#include <ql/math/array.hpp>
```

```
#include <ql/math/matrix.hpp>
```

```
#include <ql/models/parameter.hpp>
```

```
#include <ql/utilities/null.hpp>
```

Include dependency graph for lmcorrmodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LmCorrelationModel](#)
libor forward correlation model

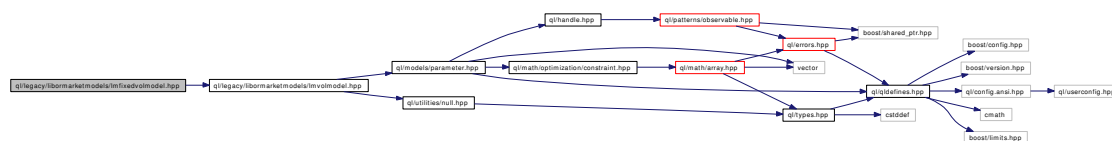
10.106 ql/legacy/libormarketmodels/lmfixedvolmodel.hpp File Reference

10.106.1 Detailed Description

model of constant volatilities for libor market models

```
#include <ql/legacy/libormarketmodels/lmvolmodel.hpp>
```

Include dependency graph for lmfixedvolmodel.hpp:



Namespaces

- namespace **QuantLib**

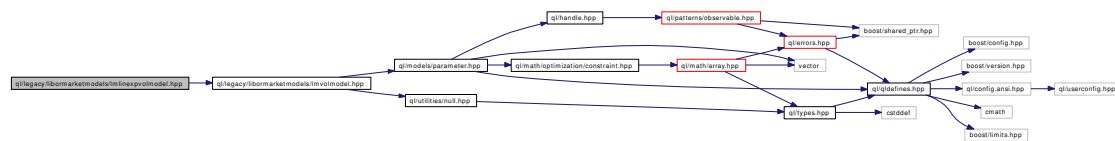
10.108 ql/legacy/libormarketmodels/lmlinearpvolmodel.hpp File Reference

10.108.1 Detailed Description

volatility model for libor market models

```
#include <ql/legacy/libormarketmodels/lmvolmodel.hpp>
```

Include dependency graph for lmlinearpvolmodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LmLinearExponentialVolatilityModel](#)
linear exponential volatility model

10.109 ql/legacy/libormarketmodels/lmvolmodel.hpp File Reference

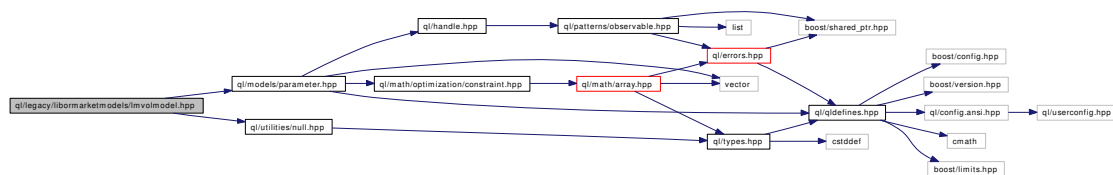
10.109.1 Detailed Description

volatility model for libor market models

```
#include <ql/models/parameter.hpp>
```

```
#include <ql/utilities/null.hpp>
```

Include dependency graph for lmvolmodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **LmVolatilityModel**
caplet volatility model

10.110 ql/legacy/pricers/discretegeometricaso.hpp File Reference

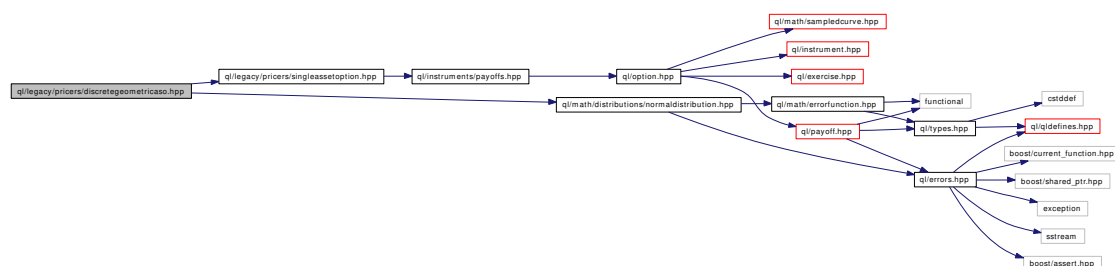
10.110.1 Detailed Description

Discrete geometric average-strike Asian option.

```
#include <ql/legacy/pricers/singleassetoption.hpp>
```

```
#include <ql/math/distributions/normaldistribution.hpp>
```

Include dependency graph for discretegeometricaso.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DiscreteGeometricASO](#)

Discrete geometric average-strike Asian option (European style).

10.111 ql/legacy/pricers/mccliquetooption.hpp File Reference

10.111.1 Detailed Description

Cliquet option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
```

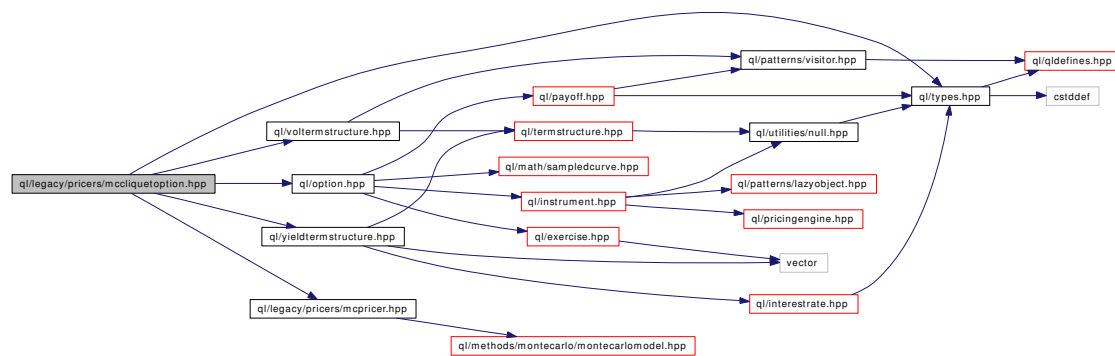
```
#include <ql/types.hpp>
```

```
#include <ql/legacy/pricers/mcpricer.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mccliquetooption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McCliquetOption**
simple example of Monte Carlo pricer

10.112 ql/legacy/pricers/mcdiscretearithmeticaso.hpp File Reference

10.112.1 Detailed Description

Discrete arithmetic average-strike Asian option.

```
#include <ql/option.hpp>
```

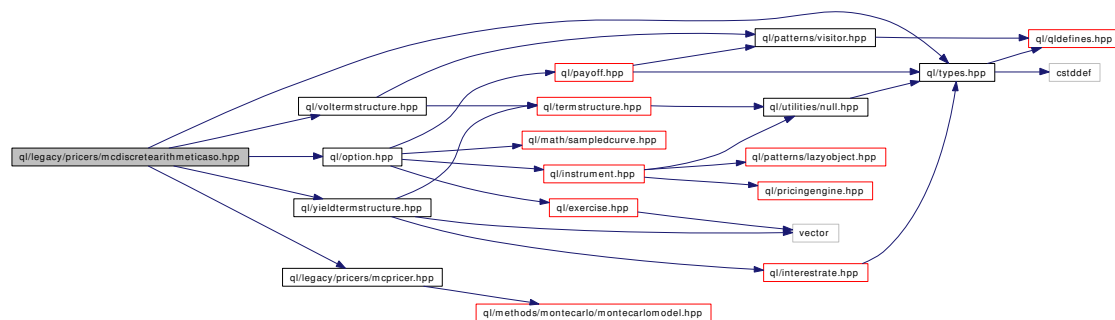
```
#include <ql/types.hpp>
```

```
#include <ql/legacy/pricers/mcpricer.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mcdiscretearithmeticaso.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McDiscreteArithmeticASO**

Discrete arithmetic average-strike Asian option.

10.113 ql/legacy/pricers/mceverest.hpp File Reference

10.113.1 Detailed Description

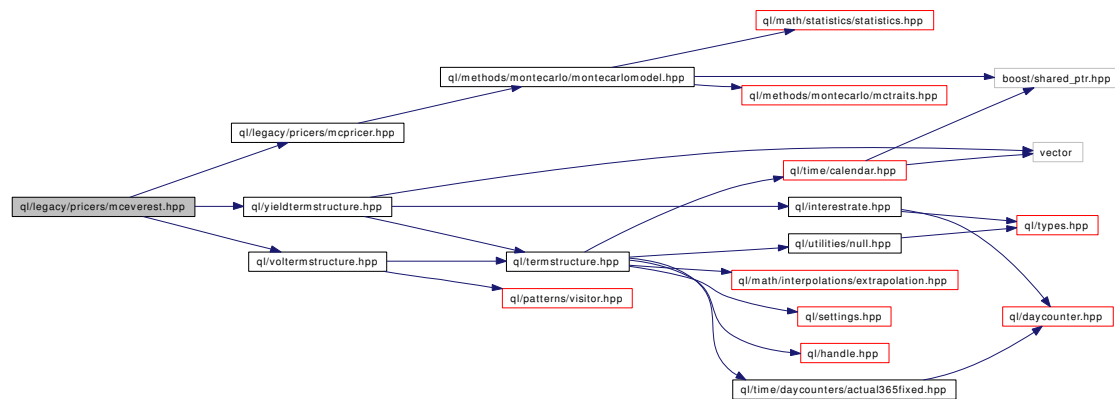
Everest-type option pricer

```
#include <ql/legacy/pricers/mcpricer.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mceverest.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McEverest**
Everest-type option pricer.

10.114 ql/legacy/pricers/mchimalaya.hpp File Reference

10.114.1 Detailed Description

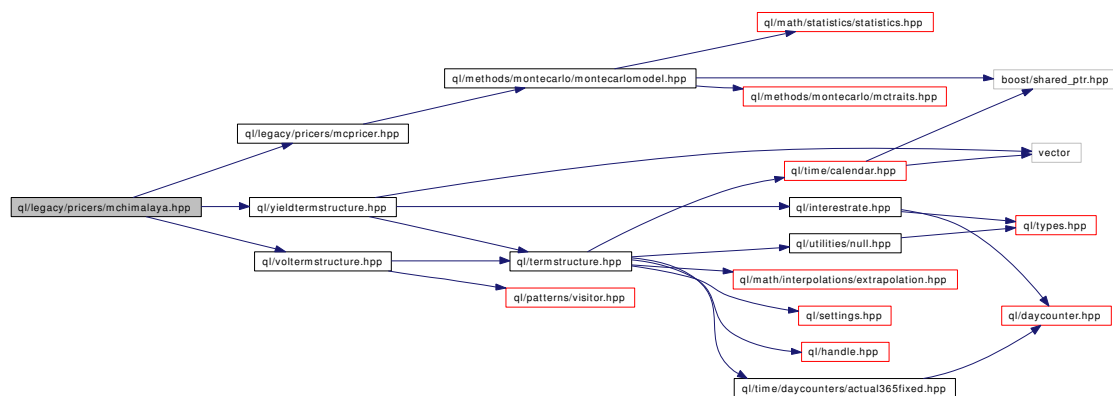
Himalayan-type option pricer.

```
#include <ql/legacy/pricers/mcpricer.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mchimalaya.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McHimalaya**
Himalayan-type option pricer.

10.115 ql/legacy/pricers/mcmaxbasket.hpp File Reference

10.115.1 Detailed Description

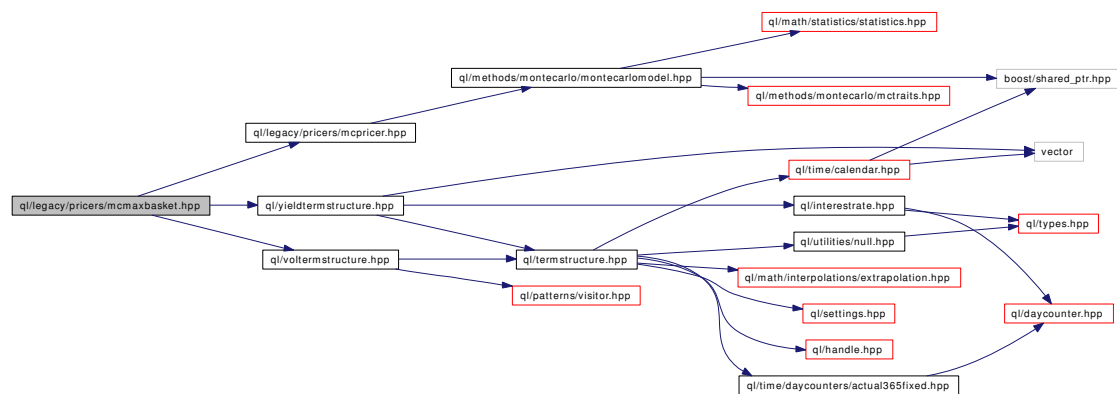
Max-basket Monte Carlo pricer.

```
#include <ql/legacy/pricers/mcpricer.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mcmaxbasket.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McMaxBasket**
Max-basket Monte Carlo pricer.

10.116 ql/legacy/pricers/mcpagoda.hpp File Reference

10.116.1 Detailed Description

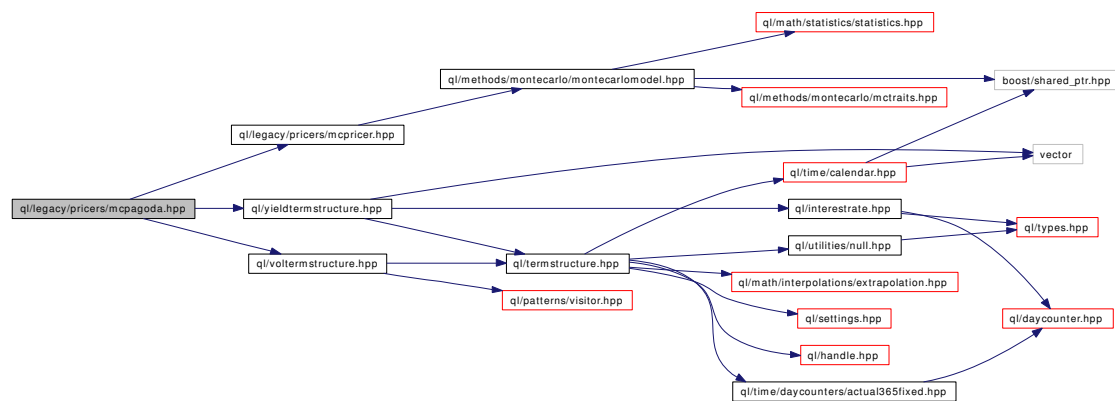
Roofed multi asset Asian option.

```
#include <ql/legacy/pricers/mcpricer.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mcpagoda.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McPagoda**
roofed Asian option

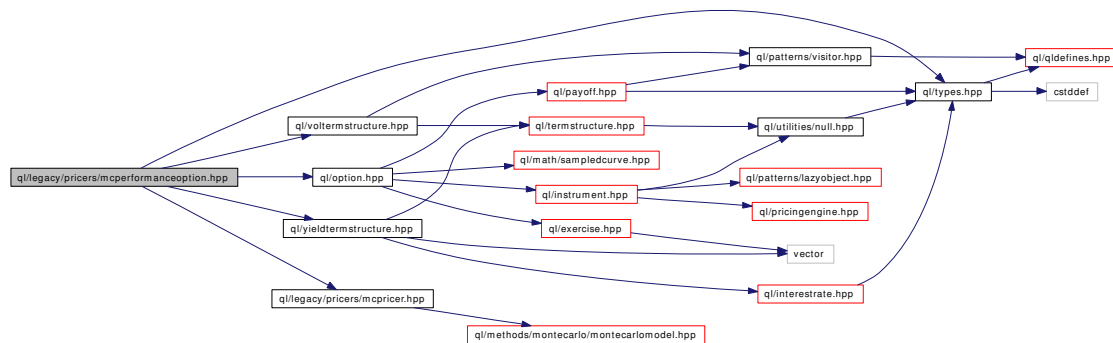
10.117 ql/legacy/pricers/mcperformanceoption.hpp File Reference

10.117.1 Detailed Description

Performance option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
#include <ql/types.hpp>
#include <ql/legacy/pricers/mcpricer.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mcperformanceoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McPerformanceOption**

Performance option computed using Monte Carlo simulation.

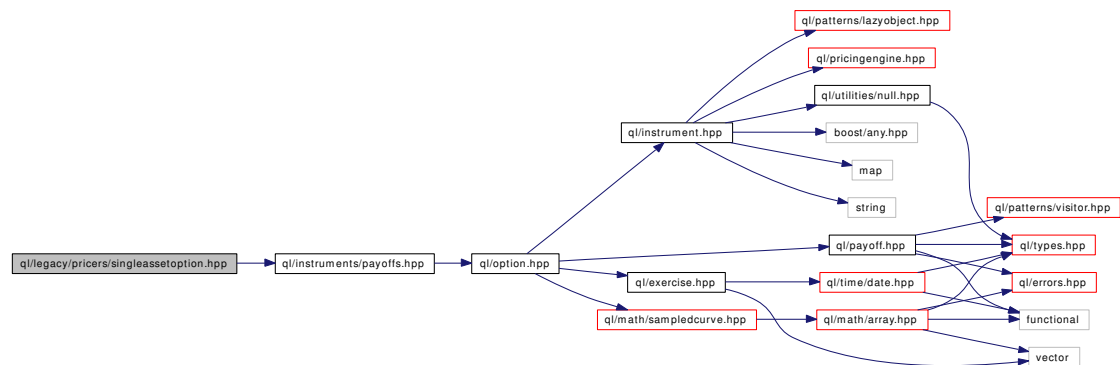
10.119 ql/legacy/pricers/singleassetoption.hpp File Reference

10.119.1 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.

10.120 ql/math/array.hpp File Reference

10.120.1 Detailed Description

1-D array used in linear algebra.

```
#include <ql/types.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <ql/utilities/disposable.hpp>
```

```
#include <boost/iterator/reverse_iterator.hpp>
```

```
#include <boost/scoped_array.hpp>
```

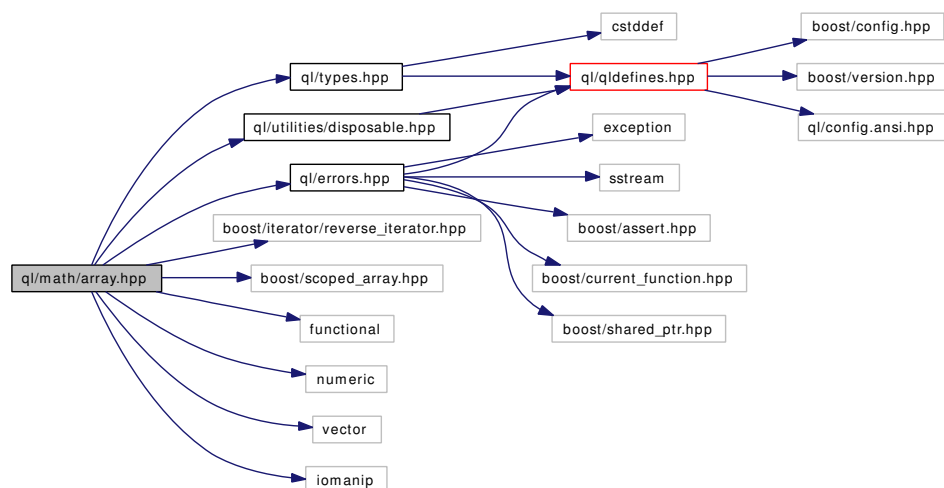
```
#include <functional>
```

```
#include <numeric>
```

```
#include <vector>
```

```
#include <iomanip>
```

Include dependency graph for array.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Array](#)
1-D array used in linear algebra.

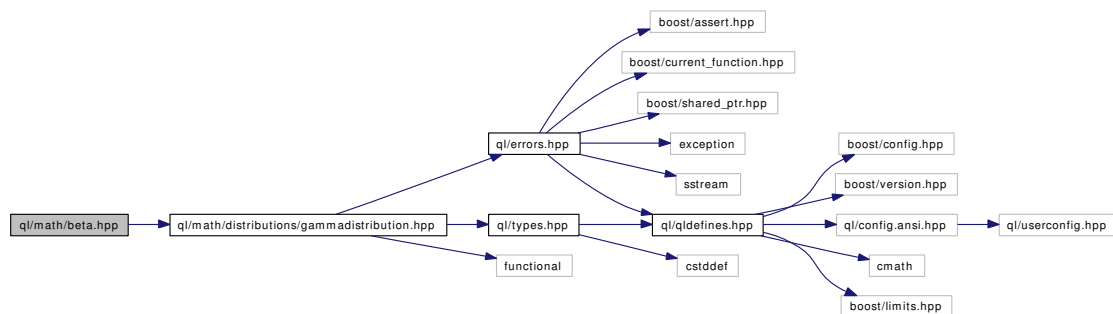
10.121 ql/math/beta.hpp File Reference

10.121.1 Detailed Description

Beta and beta incomplete functions.

```
#include <ql/math/distributions/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.

10.122 ql/math/comparison.hpp File Reference

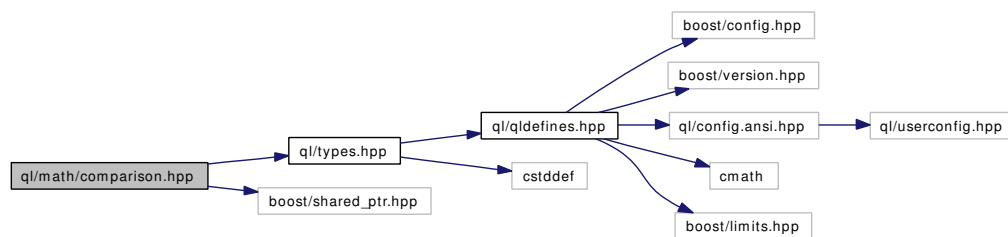
10.122.1 Detailed Description

floating-point comparisons

```
#include <ql/types.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for comparison.hpp:



Namespaces

- namespace **QuantLib**

Functions

- bool `close` (Real x, Real y)
- bool `close` (Real x, Real y, Size n)
- bool `close_enough` (Real x, Real y)
- bool `close_enough` (Real x, Real y, Size n)

10.123 ql/math/curve.hpp File Reference

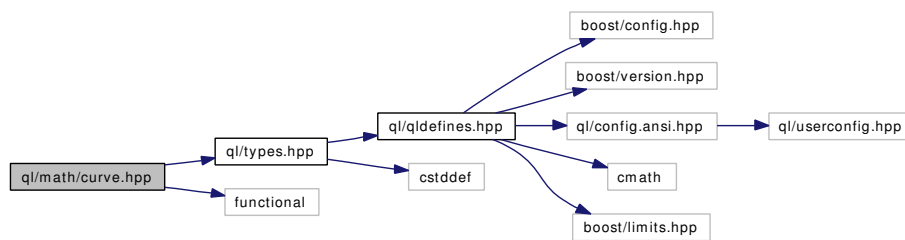
10.123.1 Detailed Description

Curve.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for curve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Curve**
abstract curve class

10.124 ql/math/distributions/binomialdistribution.hpp File Reference

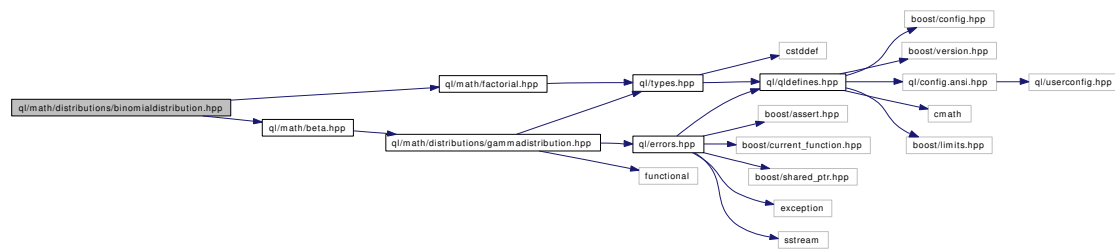
10.124.1 Detailed Description

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)

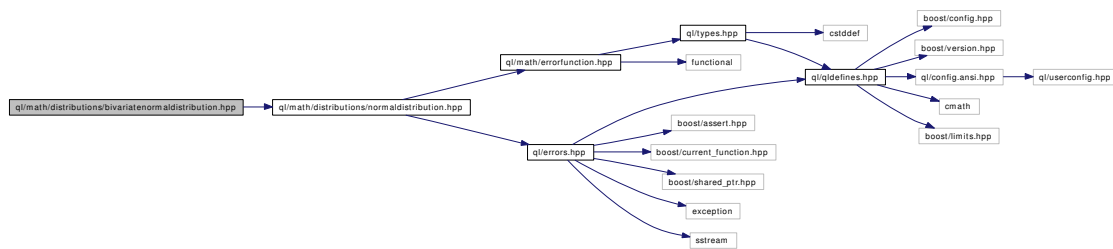
10.125 ql/math/distributions/bivariatenormaldistribution.hpp File Reference

10.125.1 Detailed Description

bivariate cumulative normal distribution

```
#include <ql/math/distributions/normaldistribution.hpp>
```

Include dependency graph for bivariatenormaldistribution.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BivariateCumulativeNormalDistributionDr78](#)
Cumulative bivariate normal distribution function.
- class [BivariateCumulativeNormalDistributionWe04DP](#)
Cumulative bivariate normal distribution function (West 2004).

Typedefs

- typedef [BivariateCumulativeNormalDistributionWe04DP](#) [BivariateCumulativeNormalDistribution](#)
default bivariate implementation

10.126 ql/math/distributions/chisquaredistribution.hpp File Reference

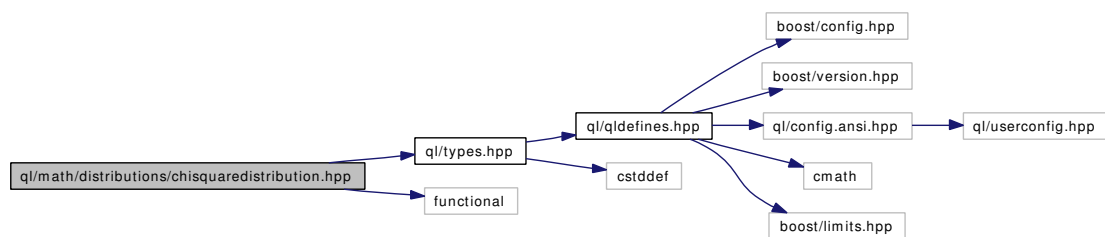
10.126.1 Detailed Description

Chi-square (central and non-central) distributions.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for chisquaredistribution.hpp:



Namespaces

- namespace **QuantLib**

10.127 ql/math/distributions/gammadistribution.hpp File Reference

10.127.1 Detailed Description

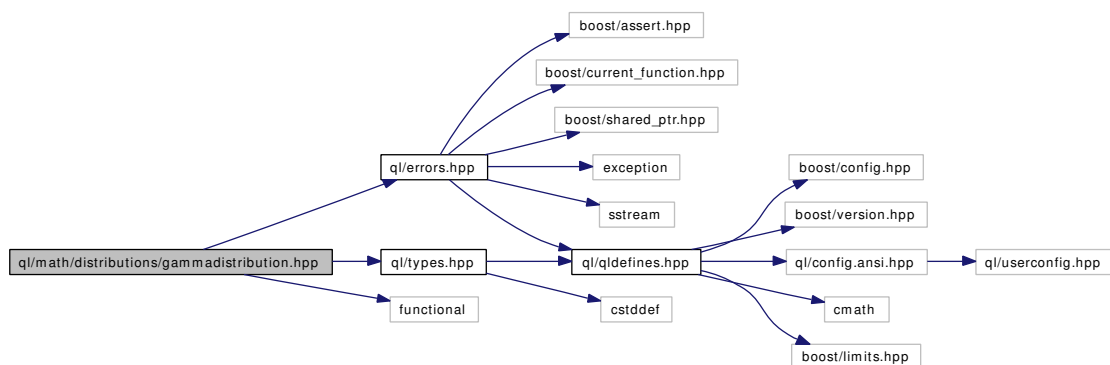
Gamma distribution.

```
#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.

10.128 ql/math/distributions/normaldistribution.hpp File Reference

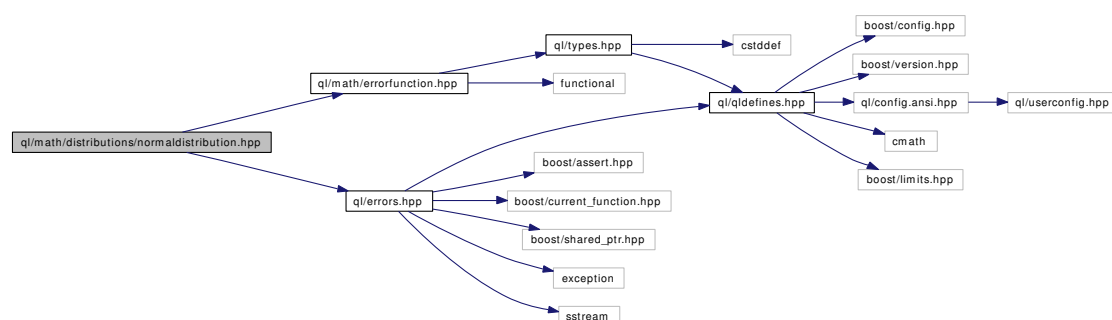
10.128.1 Detailed Description

normal, cumulative and inverse cumulative distributions

```
#include <ql/math/errorfunction.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for normaldistribution.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [NormalDistribution](#)
Normal distribution function.
- class [CumulativeNormalDistribution](#)
Cumulative normal distribution function.
- class [InverseCumulativeNormal](#)
Inverse cumulative normal distribution function.
- class [MoroInverseCumulativeNormal](#)
Moro Inverse cumulative normal distribution class.

Typedefs

- typedef NormalDistribution **GaussianDistribution**
- typedef InverseCumulativeNormal **InvCumulativeNormalDistribution**

10.129 ql/math/distributions/poissondistribution.hpp File Reference

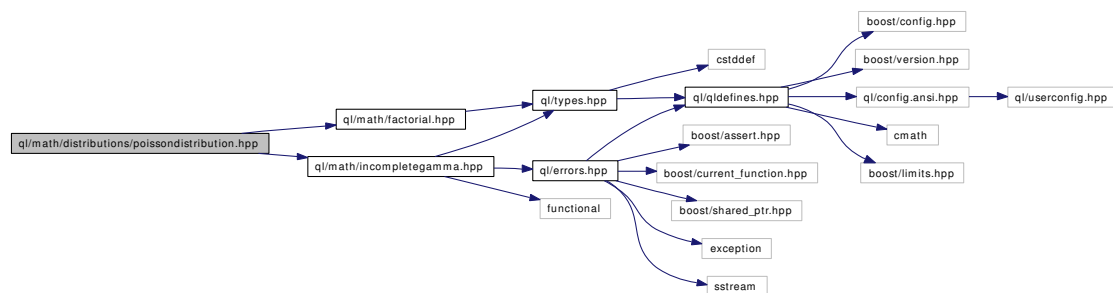
10.129.1 Detailed Description

Poisson distribution.

```
#include <ql/math/factorial.hpp>
```

```
#include <ql/math/incompletegammamma.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.

10.130 ql/math/domain.hpp File Reference

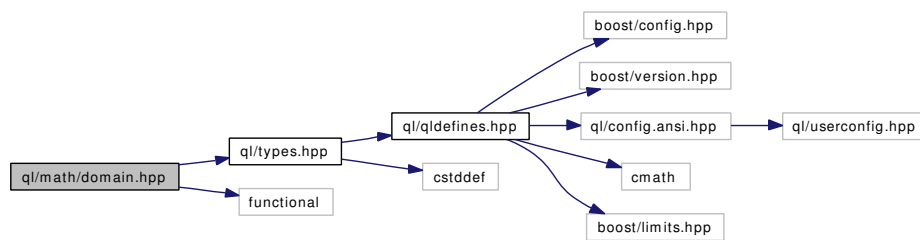
10.130.1 Detailed Description

domain

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for domain.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Domain](#)
domain abstract lclass

10.131 ql/math/errorfunction.hpp File Reference

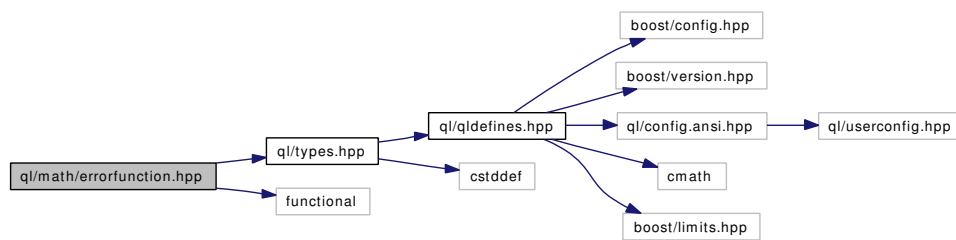
10.131.1 Detailed Description

Error function.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for errorfunction.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **ErrorFunction**
Error function

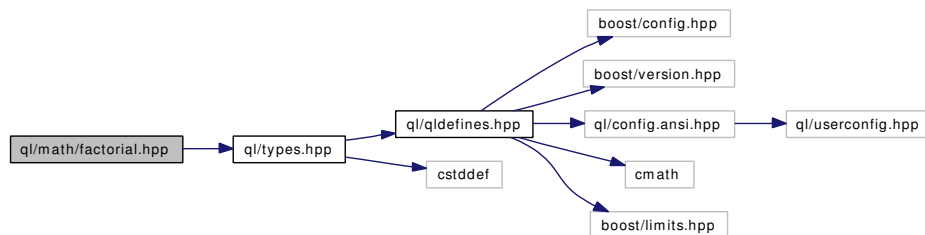
10.132 ql/math/factorial.hpp File Reference

10.132.1 Detailed Description

Factorial numbers calculator.

```
#include <ql/types.hpp>
```

Include dependency graph for factorial.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Factorial](#)
Factorial numbers calculator

10.133 ql/math/functional.hpp File Reference

10.133.1 Detailed Description

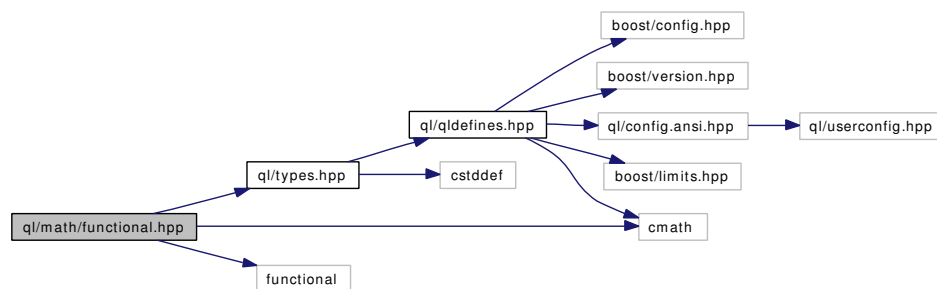
functionals and combinators not included in the STL

```
#include <ql/types.hpp>
```

```
#include <cmath>
```

```
#include <functional>
```

Include dependency graph for functional.hpp:



Namespaces

- namespace **QuantLib**

Functions

- `template<class F, class R>`
`clipped_function< F, R > clip (const F &f, const R &r)`
- `template<class F, class G>`
`composed_function< F, G > compose (const F &f, const G &g)`
- `template<class F, class G, class H>`
`binary_compose3_function< F, G, H > compose3 (const F &f, const G &g, const H &h)`

10.134 ql/math/incompletegamma.hpp File Reference

10.134.1 Detailed Description

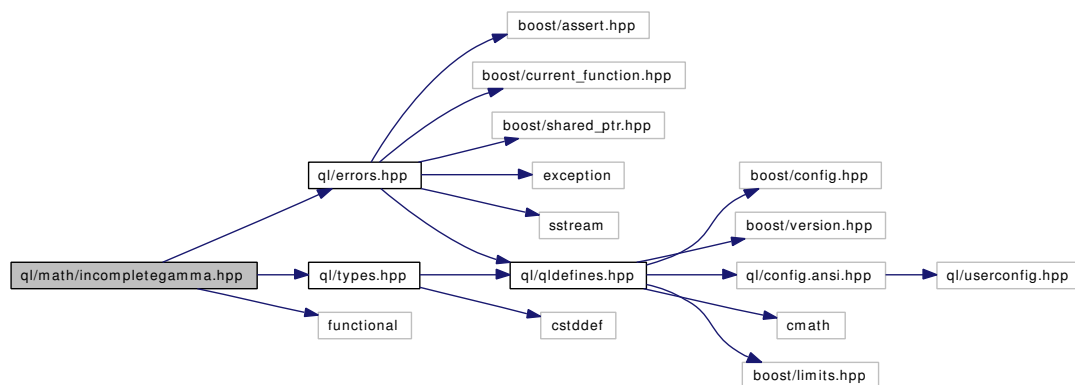
Incomplete Gamma function.

```
#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)

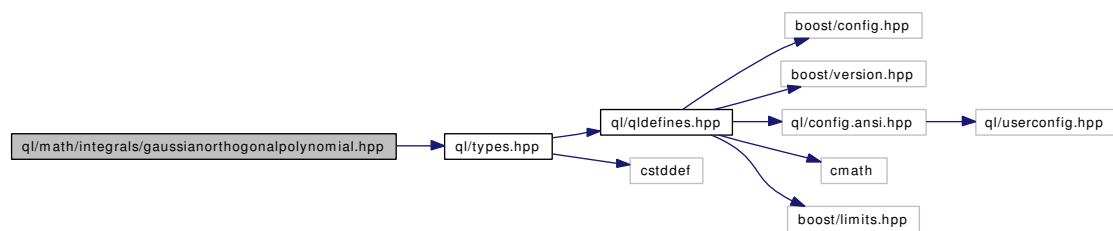
10.135 ql/math/integrals/gaussianorthogonalpolynomial.hpp File Reference

10.135.1 Detailed Description

orthogonal polynomials for gaussian quadratures

```
#include <ql/types.hpp>
```

Include dependency graph for gaussianorthogonalpolynomial.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GaussianOrthogonalPolynomial](#)
orthogonal polynomial for Gaussian quadratures
- class [GaussLaguerrePolynomial](#)
Gauss-Laguerre polynomial.
- class [GaussHermitePolynomial](#)
Gauss-Hermite polynomial.
- class [GaussJacobiPolynomial](#)
Gauss-Jacobi polynomial.
- class [GaussLegendrePolynomial](#)
Gauss-Legendre polynomial.
- class [GaussChebyshevPolynomial](#)
Gauss-Chebyshev polynomial.
- class [GaussChebyshev2thPolynomial](#)
Gauss-Chebyshev polynomial (second kind).
- class [GaussGegenbauerPolynomial](#)
Gauss-Gegenbauer polynomial.

- class [GaussHyperbolicPolynomial](#)
Gauss hyperbolic polynomial.

10.136 ql/math/integrals/gaussianquadratures.hpp File Reference

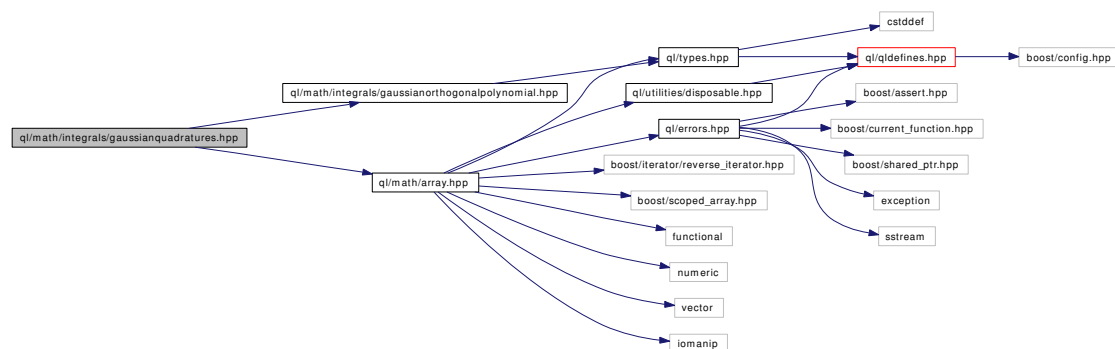
10.136.1 Detailed Description

Integral of a 1-dimensional function using the Gauss quadratures.

```
#include <ql/math/array.hpp>
```

```
#include <ql/math/integrals/gaussianorthogonalpolynomial.hpp>
```

Include dependency graph for gaussianquadratures.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GaussianQuadrature](#)
Integral of a 1-dimensional function using the Gauss quadratures method.
- class [GaussLaguerreIntegration](#)
generalized Gauss-Laguerre integration
- class [GaussHermiteIntegration](#)
generalized Gauss-Hermite integration
- class [GaussJacobiIntegration](#)
Gauss-Jacobi integration.
- class [GaussHyperbolicIntegration](#)
Gauss-Hyperbolic integration.
- class [GaussLegendreIntegration](#)
Gauss-Legendre integration.
- class [GaussChebyshevIntegration](#)

Gauss-Chebyshev integration.

- class [GaussChebyshev2thIntegration](#)
Gauss-Chebyshev integration (second kind).
- class [GaussGegenbauerIntegration](#)
Gauss-Gegenbauer integration.
- class [TabulatedGaussLegendre](#)
tabulated Gauss-Legendre quadratures

10.137 ql/math/integrals/integral.hpp File Reference

10.137.1 Detailed Description

Integrators base class definition.

```
#include <ql/errors.hpp>
```

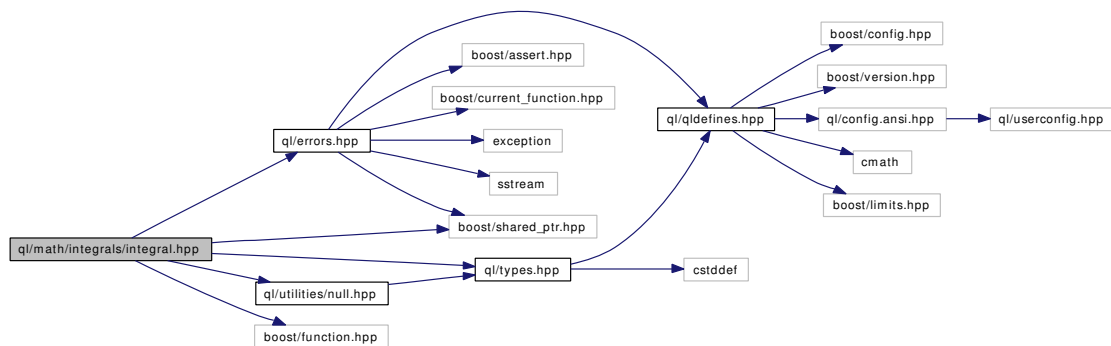
```
#include <ql/types.hpp>
```

```
#include <ql/utilities/null.hpp>
```

```
#include <boost/function.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for integral.hpp:



Namespaces

- namespace **QuantLib**

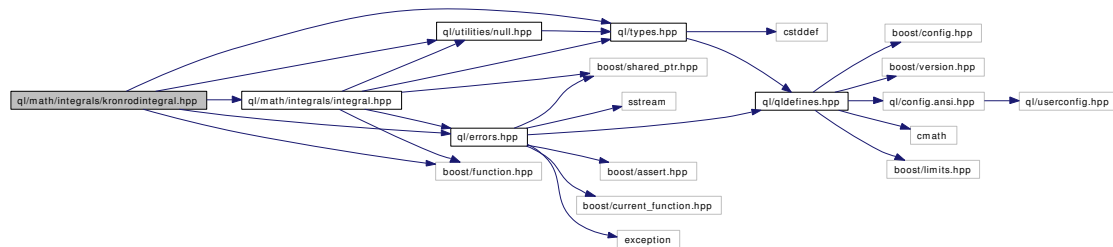
10.138 ql/math/integrals/kronrodintegral.hpp File Reference

10.138.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod method.

```
#include <ql/errors.hpp>
#include <ql/types.hpp>
#include <ql/utilities/null.hpp>
#include <ql/math/integrals/integral.hpp>
#include <boost/function.hpp>
```

Include dependency graph for kronrodintegral.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GaussKronrodNonAdaptive](#)
Integral of a 1-dimensional function using the Gauss-Kronrod methods.
- class [GaussKronrodAdaptive](#)
Integral of a 1-dimensional function using the Gauss-Kronrod methods.

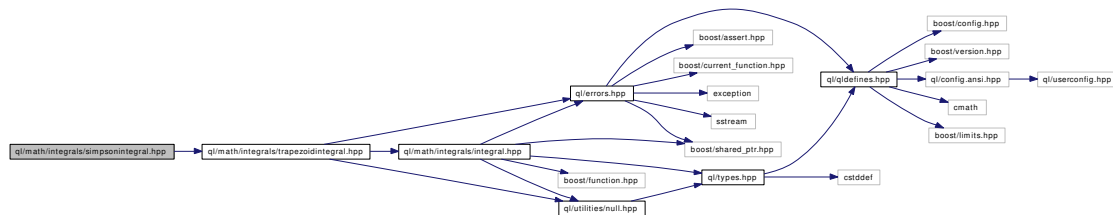
10.140 ql/math/integrals/simpsonintegral.hpp File Reference

10.140.1 Detailed Description

integral of a one-dimensional function using Simpson formula

```
#include <ql/math/integrals/trapezoidintegral.hpp>
```

Include dependency graph for `simpsonintegral.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [SimpsonIntegral](#)
Integral of a one-dimensional function.

10.141 ql/math/integrals/trapezoidintegral.hpp File Reference

10.141.1 Detailed Description

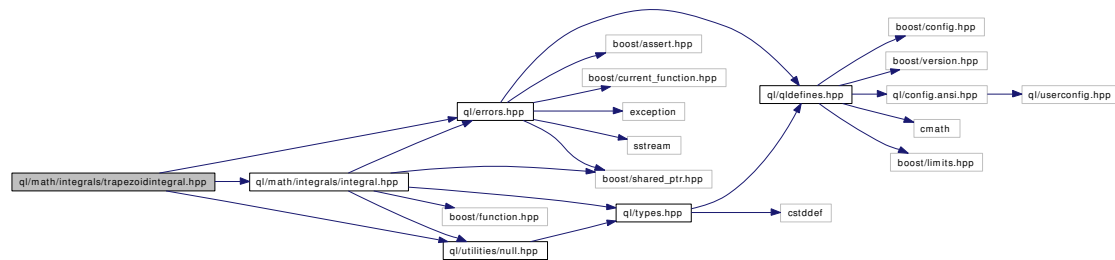
integral of a one-dimensional function using the trapezoid formula

```
#include <ql/math/integrals/integral.hpp>
```

```
#include <ql/utilities/null.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for trapezoidintegral.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TrapezoidIntegral](#)
Integral of a one-dimensional function.

10.142 ql/math/interpolation.hpp File Reference

10.142.1 Detailed Description

base class for 1-D interpolations

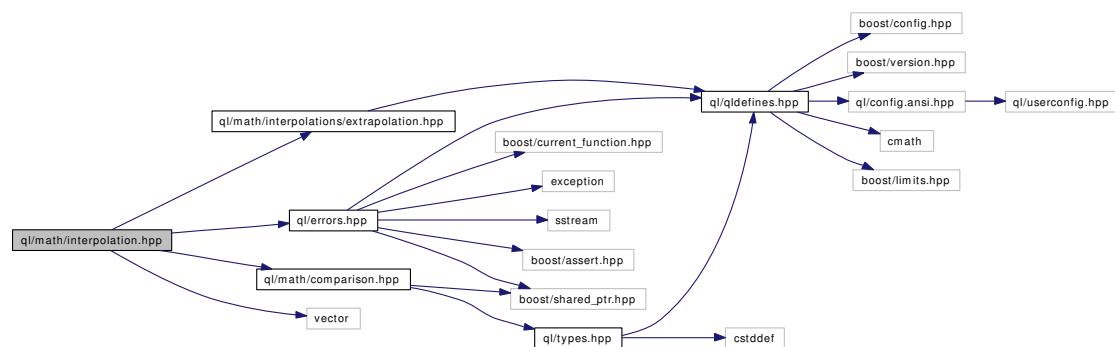
```
#include <ql/math/interpolations/extrapolation.hpp>
```

```
#include <ql/math/comparison.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <vector>
```

Include dependency graph for interpolation.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Interpolation](#)
base class for 1-D interpolations.
- class [Interpolation::Impl](#)
abstract base class for interpolation implementations
- class [Interpolation::templateImpl](#)
basic template implementation

10.143 ql/math/interpolations/backwardflatinterpolation.hpp File Reference

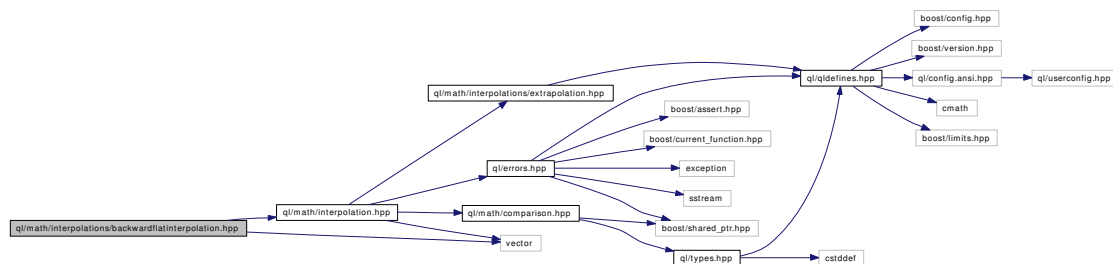
10.143.1 Detailed Description

backward-flat interpolation between discrete points

```
#include <ql/math/interpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for backwardflatinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [BackwardFlatInterpolation](#)
Backward-flat interpolation between discrete points.
- class [BackwardFlat](#)
Backward-flat interpolation factory and traits.

10.144 ql/math/interpolations/bicubicsplineinterpolation.hpp File Reference

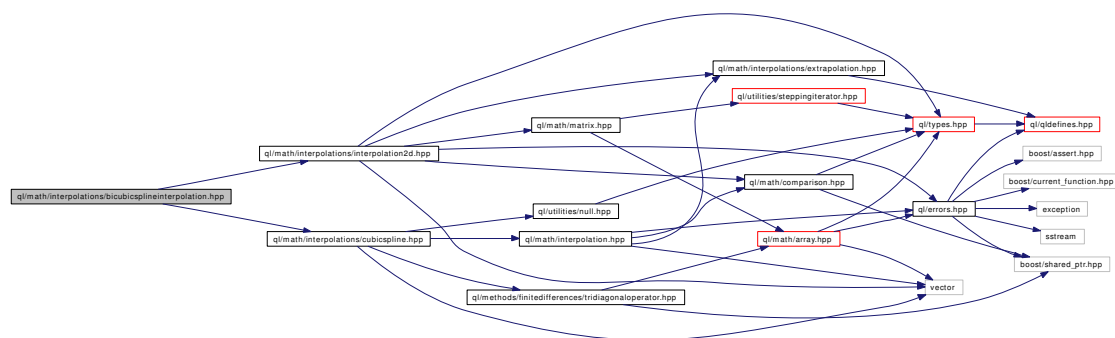
10.144.1 Detailed Description

bicubic spline interpolation between discrete points

```
#include <ql/math/interpolations/interpolation2d.hpp>
```

```
#include <ql/math/interpolations/cubicspline.hpp>
```

Include dependency graph for bicubicsplineinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class **BicubicSpline**
bicubic-spline interpolation between discrete points
- class **Bicubic**
bicubic-spline-interpolation factory

10.146 ql/math/interpolations/cubicspline.hpp File Reference

10.146.1 Detailed Description

cubic spline interpolation between discrete points

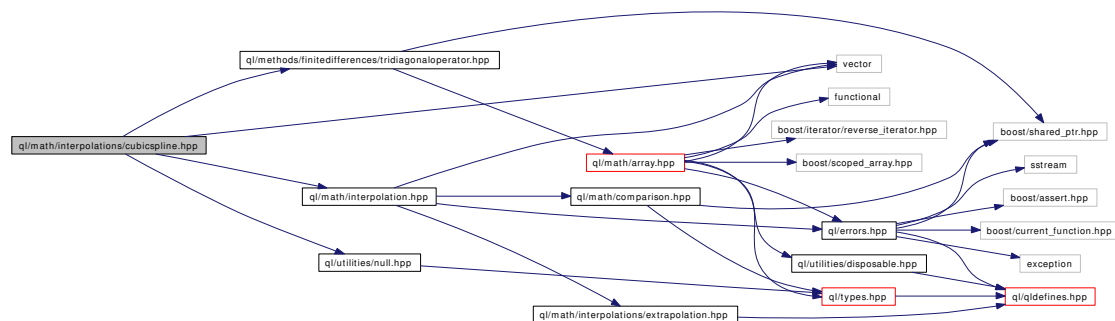
```
#include <ql/math/interpolation.hpp>
```

```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

```
#include <ql/utilities/null.hpp>
```

```
#include <vector>
```

Include dependency graph for cubicspline.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class **CubicSpline**
Cubic spline interpolation between discrete points.
- class **MonotonicCubicSpline**
Cubic spline with monotonicity constraint
- class **NaturalCubicSpline**
Cubic spline with null second derivative at end points
- class **NaturalMonotonicCubicSpline**
Natural cubic spline with monotonicity constraint.
- class **Cubic**
cubic-spline interpolation factory and traits

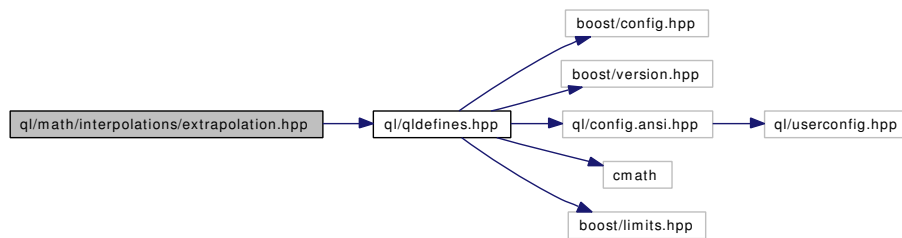
10.147 ql/math/interpolations/extrapolation.hpp File Reference

10.147.1 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

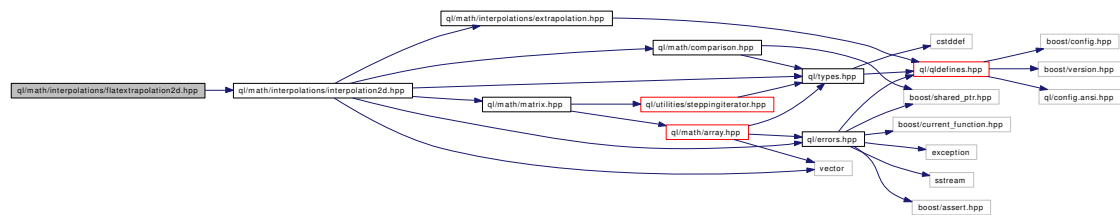
10.148 ql/math/interpolations/flatextrapolation2d.hpp File Reference

10.148.1 Detailed Description

abstract base classes for 2-D flat extrapolations

```
#include <ql/math/interpolations/interpolation2d.hpp>
```

Include dependency graph for flatextrapolation2d.hpp:



Namespaces

- namespace **QuantLib**

10.149 ql/math/interpolations/forwardflatinterpolation.hpp File Reference

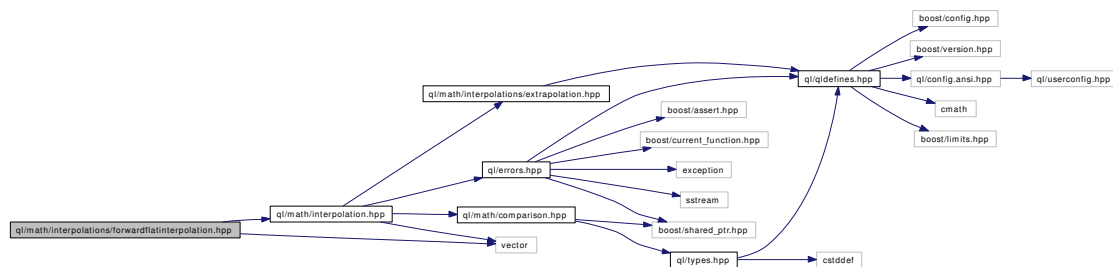
10.149.1 Detailed Description

forward-flat interpolation between discrete points

```
#include <ql/math/interpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for forwardflatinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [ForwardFlatInterpolation](#)
Forward-flat interpolation between discrete points.
- class [ForwardFlat](#)
Forward-flat interpolation factory and traits.

10.150 ql/math/interpolations/interpolation2d.hpp File Reference

10.150.1 Detailed Description

abstract base classes for 2-D interpolations

```
#include <ql/math/interpolations/extrapolation.hpp>
```

```
#include <ql/math/comparison.hpp>
```

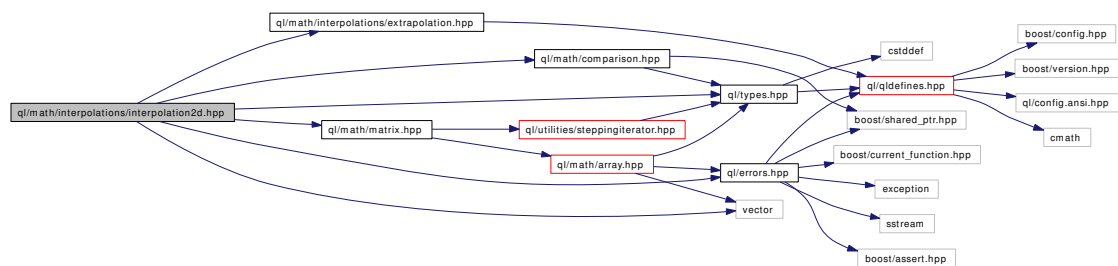
```
#include <ql/math/matrix.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <vector>
```

Include dependency graph for interpolation2d.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Interpolation2D](#)
base class for 2-D interpolations.
- class [Interpolation2D::Impl](#)
abstract base class for 2-D interpolation implementations
- class [Interpolation2D::templateImpl](#)
basic template implementation

10.151 ql/math/interpolations/linearinterpolation.hpp File Reference

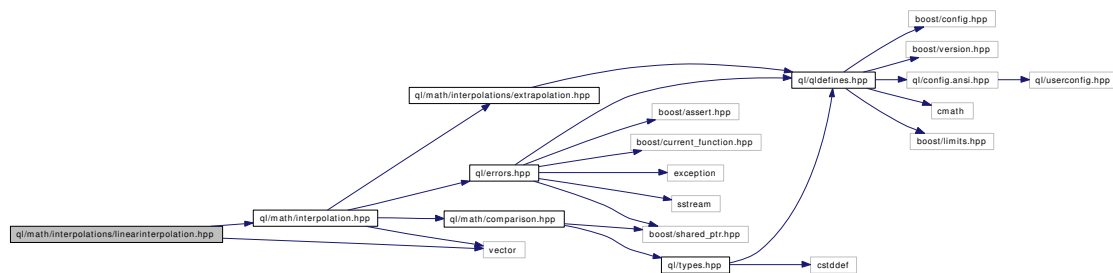
10.151.1 Detailed Description

linear interpolation between discrete points

```
#include <ql/math/interpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for linearinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [LinearInterpolation](#)
Linear interpolation between discrete points
- class [Linear](#)
Linear-interpolation factory and traits

10.152 ql/math/interpolations/loglinearinterpolation.hpp File Reference

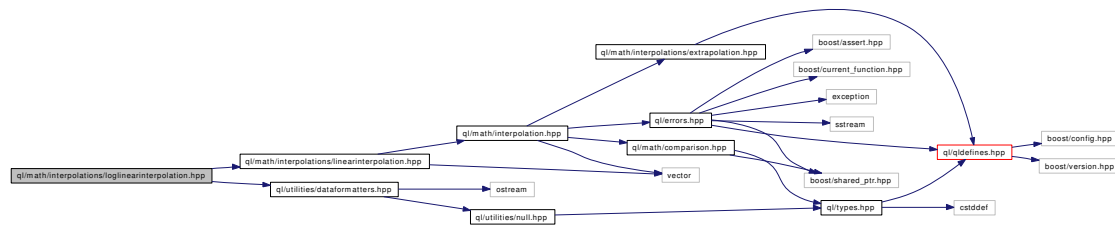
10.152.1 Detailed Description

log-linear interpolation between discrete points

```
#include <ql/math/interpolations/linearinterpolation.hpp>
```

```
#include <ql/utilities/dataformatters.hpp>
```

Include dependency graph for loglinearinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [LogLinearInterpolation](#)
log-linear interpolation between discrete points
- class [LogLinear](#)
log-linear interpolation factory and traits

10.153 ql/math/interpolations/multicubicspline.hpp File Reference

10.153.1 Detailed Description

N-dimensional cubic spline interpolation between discrete points.

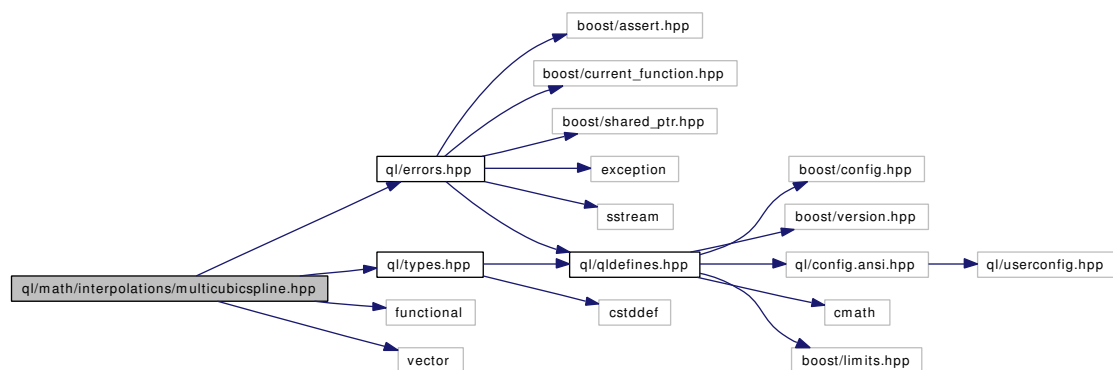
```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <functional>
```

```
#include <vector>
```

Include dependency graph for multicubicspline.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [MultiCubicSpline](#)

N-dimensional cubic spline interpolation between discrete points.

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**

10.155 ql/math/lexicographicalview.hpp File Reference

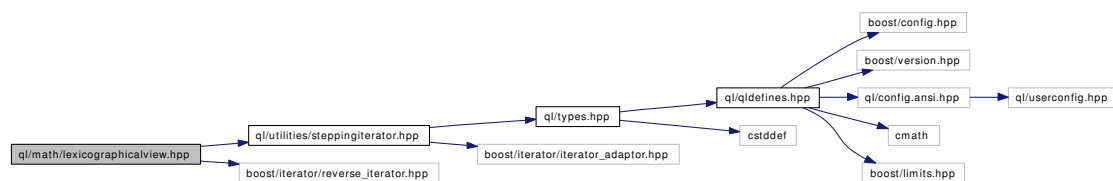
10.155.1 Detailed Description

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.

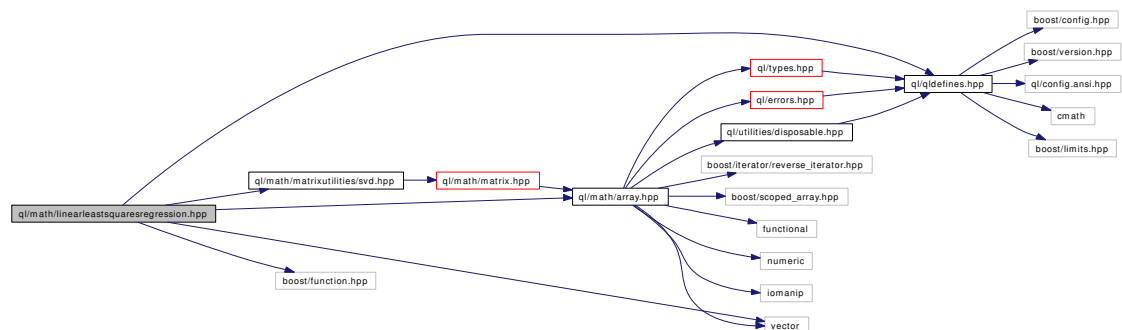
10.156 ql/math/linearleastsquaresregression.hpp File Reference

10.156.1 Detailed Description

general linear least square regression

```
#include <ql/qldefines.hpp>
#include <ql/math/matrixutilities/svd.hpp>
#include <ql/math/array.hpp>
#include <boost/function.hpp>
#include <vector>
```

Include dependency graph for linearleastsquaresregression.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LinearLeastSquaresRegression](#)
general linear least squares regression

10.157 ql/math/matrix.hpp File Reference

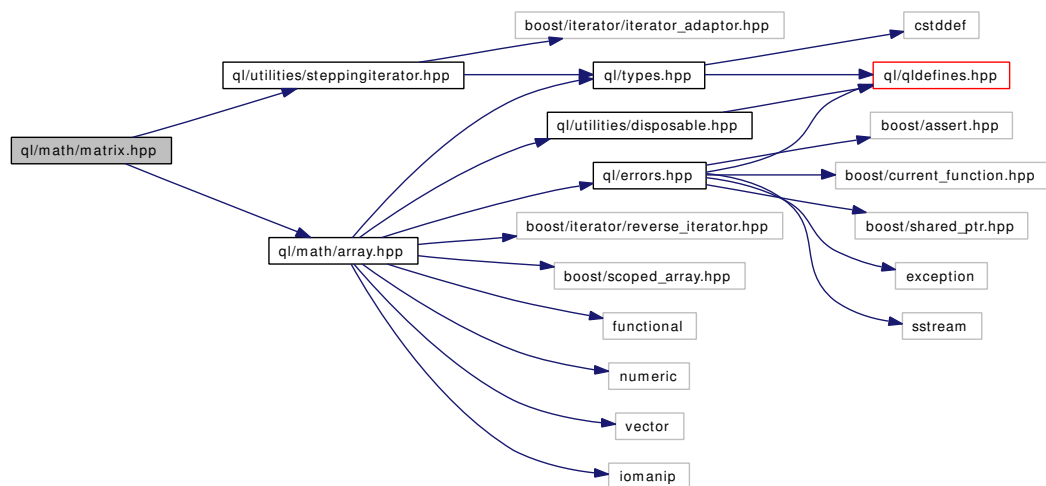
10.157.1 Detailed Description

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.

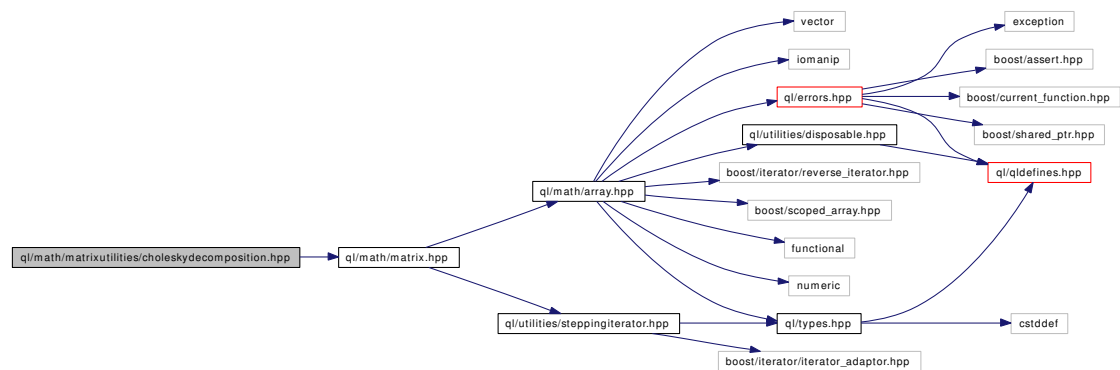
10.158 ql/math/matrixutilities/choleskydecomposition.hpp File Reference

10.158.1 Detailed Description

Cholesky decomposition.

```
#include <ql/math/matrix.hpp>
```

Include dependency graph for choleskydecomposition.hpp:



Namespaces

- namespace **QuantLib**

10.159 ql/math/matrixutilities/getcovariance.hpp File Reference

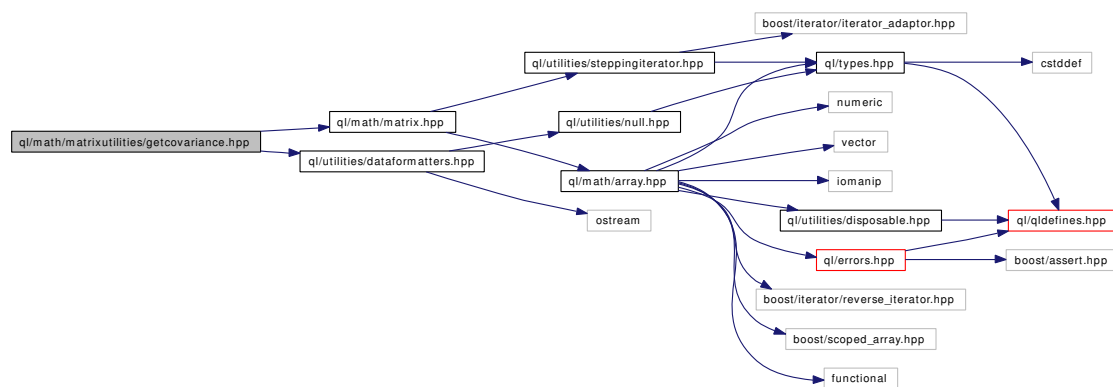
10.159.1 Detailed Description

Covariance matrix calculation.

```
#include <ql/math/matrix.hpp>
```

```
#include <ql/utilities/dataformatters.hpp>
```

Include dependency graph for getcovariance.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CovarianceDecomposition](#)
Covariance decomposition into correlation and variances.

Functions

- template<class DataIterator>
Disposable< Matrix > [getCovariance](#) (DataIterator stdDevBegin, DataIterator stdDevEnd,
const Matrix &corr, Real tolerance=1.0e-12)
Calculation of covariance from correlation and standard deviations.

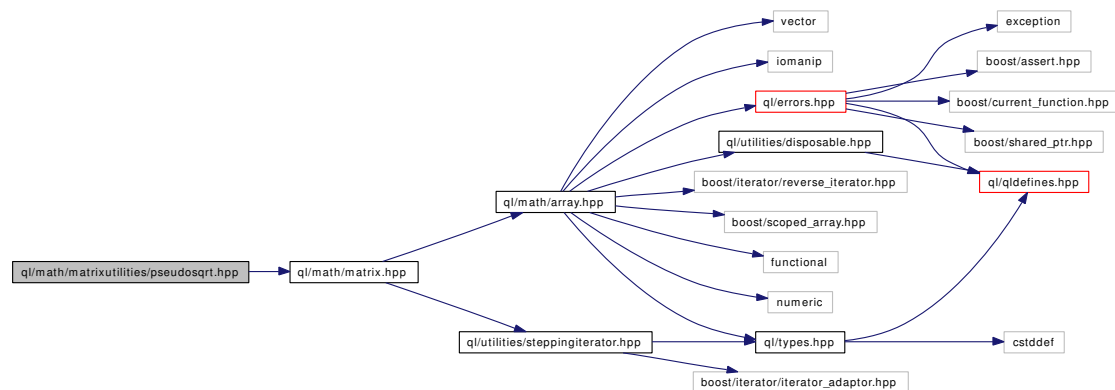
10.160 ql/math/matrixutilities/pseudosqrt.hpp File Reference

10.160.1 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

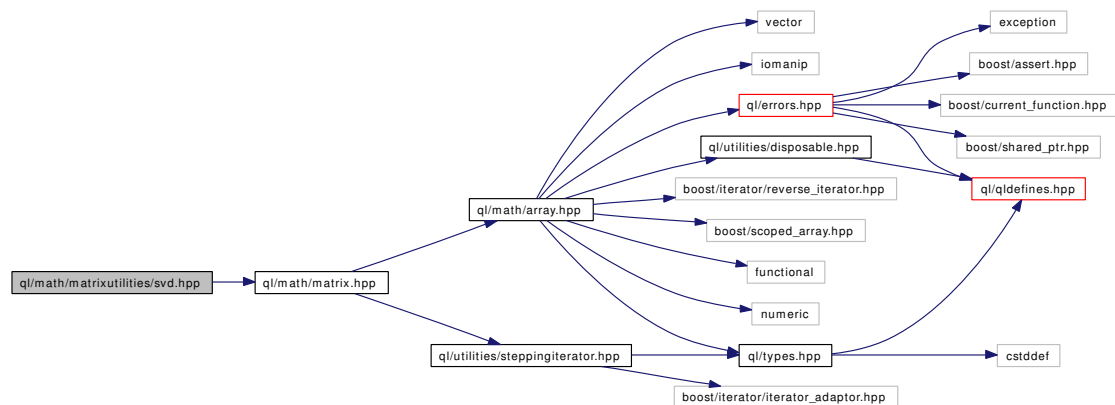
10.161 ql/math/matrixutilities/svd.hpp File Reference

10.161.1 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.

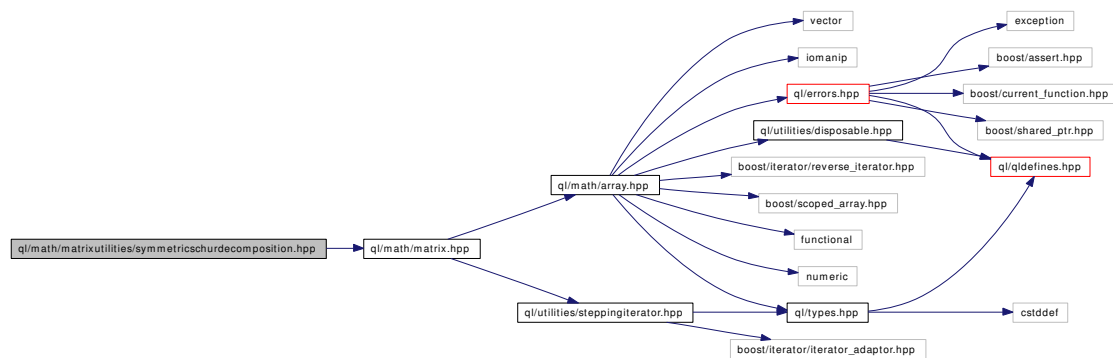
10.162 ql/math/matrixutilities/symmetricschurdecomposition.hpp File Reference

10.162.1 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.

10.163 ql/math/matrixutilities/tqreigendecomposition.hpp File Reference

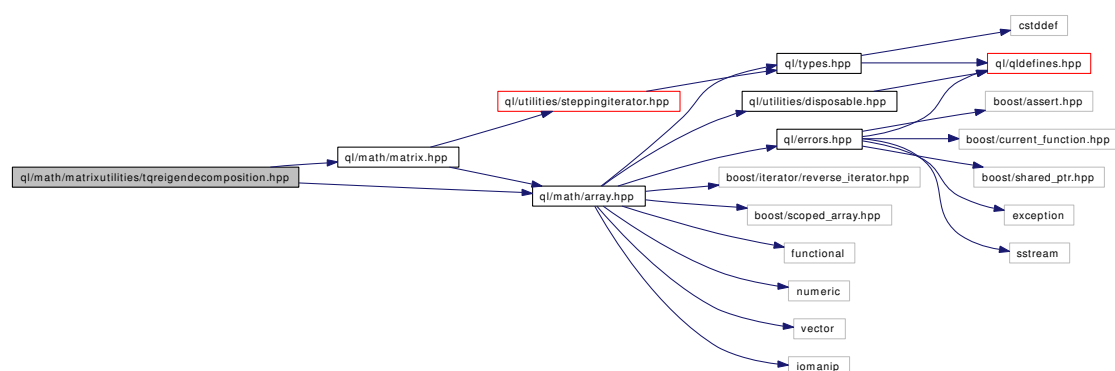
10.163.1 Detailed Description

tridiag. QR eigen decomposition with explicite shift aka Wilkinson

```
#include <ql/math/array.hpp>
```

```
#include <ql/math/matrix.hpp>
```

Include dependency graph for tqreigendecomposition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TqrEigenDecomposition**
tridiag. QR eigen decomposition with explicite shift aka Wilkinson

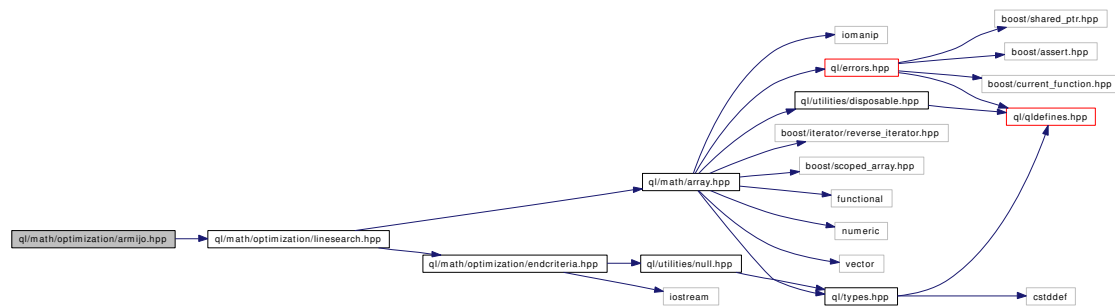
10.164 ql/math/optimization/armijo.hpp File Reference

10.164.1 Detailed Description

Armijo line-search class.

```
#include <ql/math/optimization/linesearch.hpp>
```

Include dependency graph for armijo.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ArmijoLineSearch](#)
Armijo line search.

10.165.1 Detailed Description

```
#include <ql/math/optimization/linesearchbasedmethod.hpp>
```

```

graph LR
    A[qml/math/optimization/conjugategradient.hpp] --> B[qml/math/optimization/linesearchbasedmethod.hpp]
    B --> C[qml/math/optimization/method.hpp]
    C --> D[qml/math/optimization/constraint.hpp]
    C --> E[qml/math/optimization/costfunction.hpp]
    C --> F[qml/math/optimization/endoriteria.hpp]
    C --> G[qml/math/array.hpp]
    G --> H[qml/math/optimization/sum.hpp]
    G --> I[qml/math/optimization/qerrors.hpp]
    G --> J[qml/math/optimization/qutililities/disposable.hpp]
    G --> K[boost/iterator/reverse_iterator.hpp]
    G --> L[boost/scoped_array.hpp]
    G --> M[functional]
    G --> N[numeric]
    G --> O[vector]
    G --> P[qml/math/optimization/qutililities/multi.hpp]
    G --> Q[qml/math/optimization/qtypes.hpp]
    G --> R[ostream]
    Q --> S[cstdint]
    I --> T[qldefines.hpp]
    J --> T
    P --> T
    Qtypes[qml/math/optimization/qtypes.hpp] --> T
  
```

- namespace **QuantLib**

- class **ConjugateGradient**
Multi-dimensional Conjugate Gradient class.

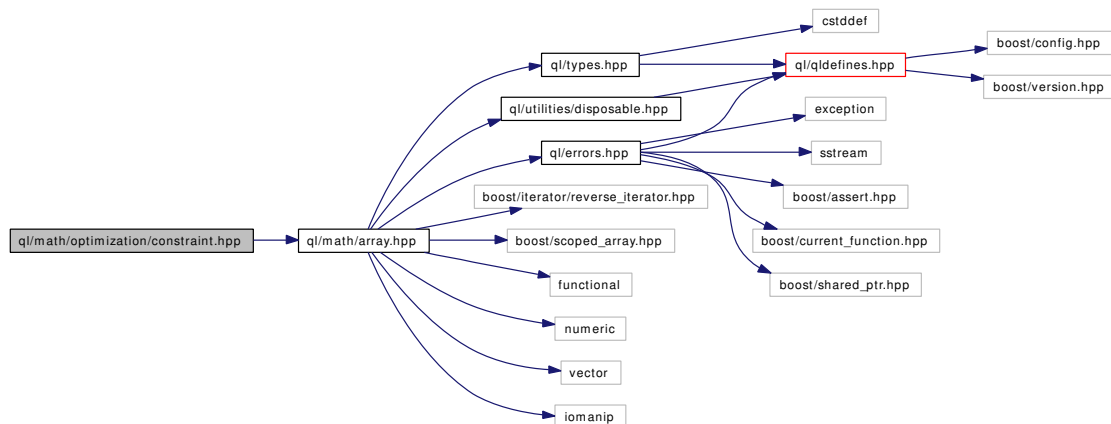
10.166 ql/math/optimization/constraint.hpp File Reference

10.166.1 Detailed Description

Abstract constraint class.

```
#include <ql/math/array.hpp>
```

Include dependency graph for constraint.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Constraint**
Base constraint class.
- class **Constraint::Impl**
Base class for constraint implementations.
- 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

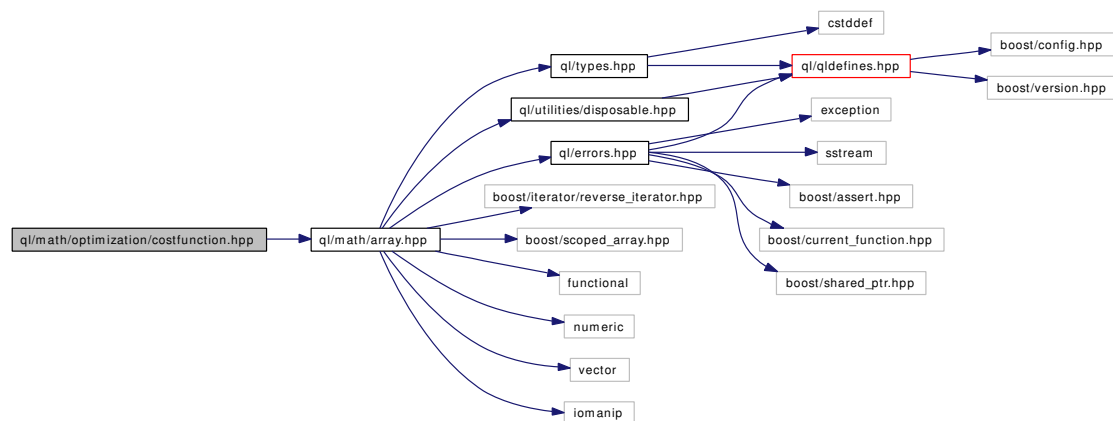
10.167 ql/math/optimization/costfunction.hpp File Reference

10.167.1 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.

10.168 ql/math/optimization/endcriteria.hpp File Reference

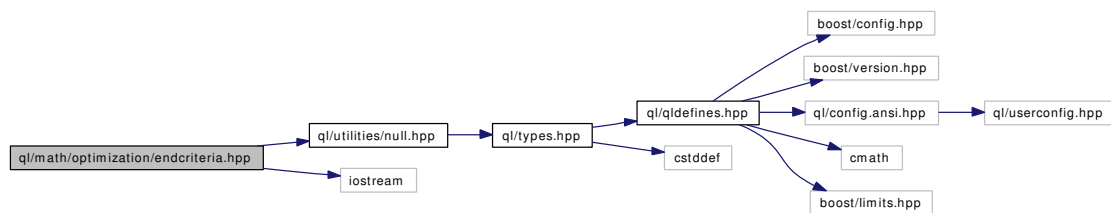
10.168.1 Detailed Description

Optimization criteria class.

```
#include <ql/utilities/null.hpp>
```

```
#include <iostream>
```

Include dependency graph for endcriteria.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [EndCriteria](#)
Criteria to end optimization process:.

Functions

- `std::ostream & operator<< (std::ostream &out, EndCriteria::Type ecType)`

10.171 ql/math/optimization/linesearch.hpp File Reference

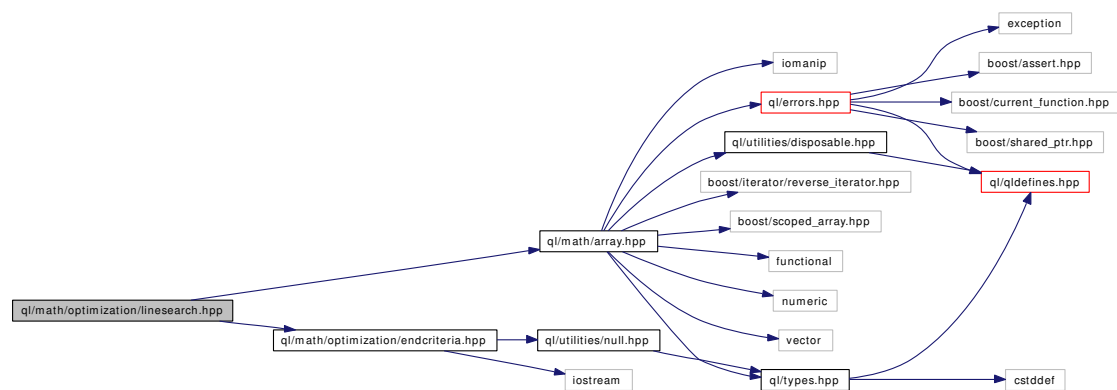
10.171.1 Detailed Description

Line search abstract class.

```
#include <ql/math/array.hpp>
```

```
#include <ql/math/optimization/endcriteria.hpp>
```

Include dependency graph for linesearch.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LineSearch](#)
Base class for line search.

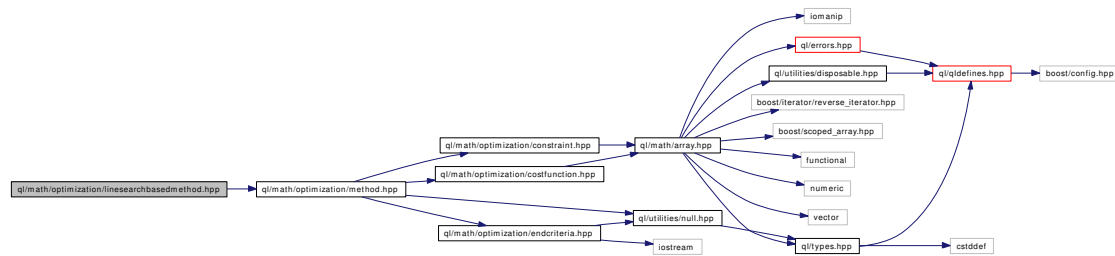
10.172 ql/math/optimization/linebasedmethod.hpp File Reference

10.172.1 Detailed Description

Abstract optimization method class.

```
#include <ql/math/optimization/method.hpp>
```

Include dependency graph for linebasedmethod.hpp:



Namespaces

- namespace **QuantLib**

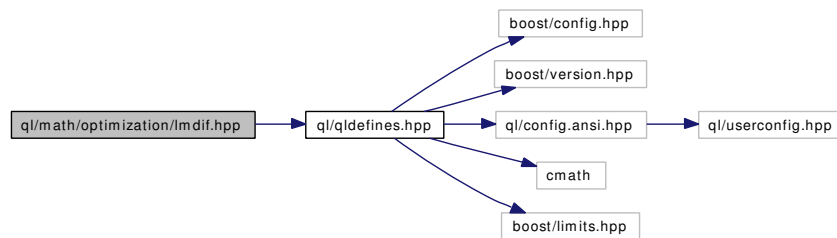
10.173 ql/math/optimization/lmdif.hpp File Reference

10.173.1 Detailed Description

wrapper for MINPACK minimization routine

```
#include <ql/qldefines.hpp>
```

Include dependency graph for lmdif.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MINPACK**

Functions

- void **lmdif** (int m, int n, double *x, double *fvec, double ftol, double xtol, double gtol, int maxfev, double epsfcn, double *diag, int mode, double factor, int nprint, int *info, int *nfev, double *fjac, int ldfjac, int *ipvt, double *qtf, double *wa1, double *wa2, double *wa3, double *wa4)

10.174 ql/math/optimization/method.hpp File Reference

10.174.1 Detailed Description

Abstract optimization method class.

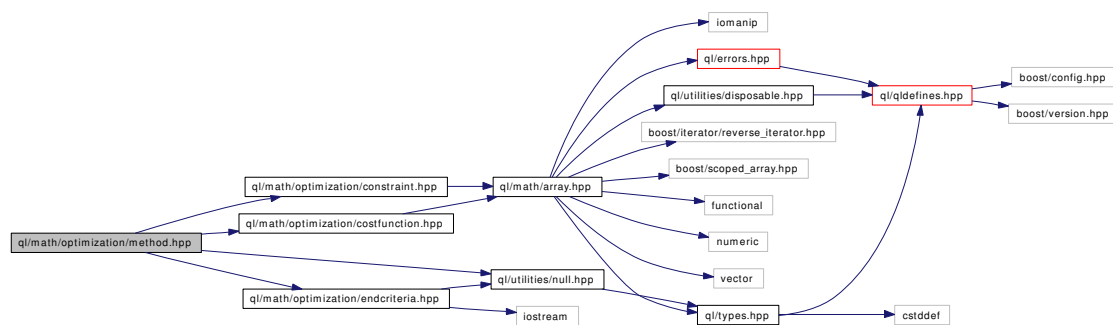
```
#include <ql/utilities/null.hpp>
```

```
#include <ql/math/optimization/constraint.hpp>
```

```
#include <ql/math/optimization/costfunction.hpp>
```

```
#include <ql/math/optimization/endcriteria.hpp>
```

Include dependency graph for method.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [OptimizationMethod](#)
Abstract class for constrained optimization method.

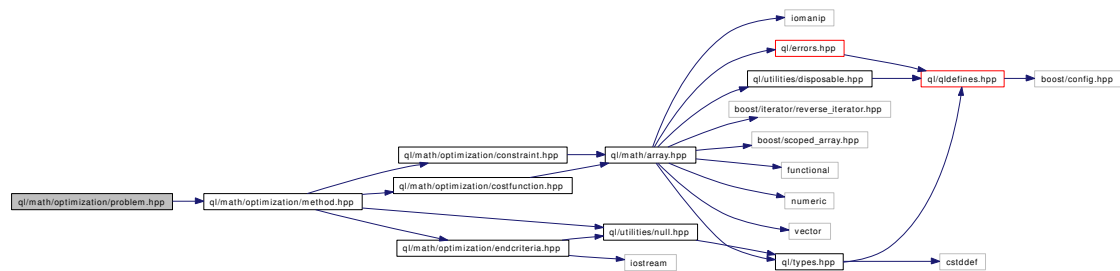
10.175 ql/math/optimization/problem.hpp File Reference

10.175.1 Detailed Description

Abstract optimization problem class.

```
#include <ql/math/optimization/method.hpp>
```

Include dependency graph for problem.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Problem**
Constrained optimization problem.

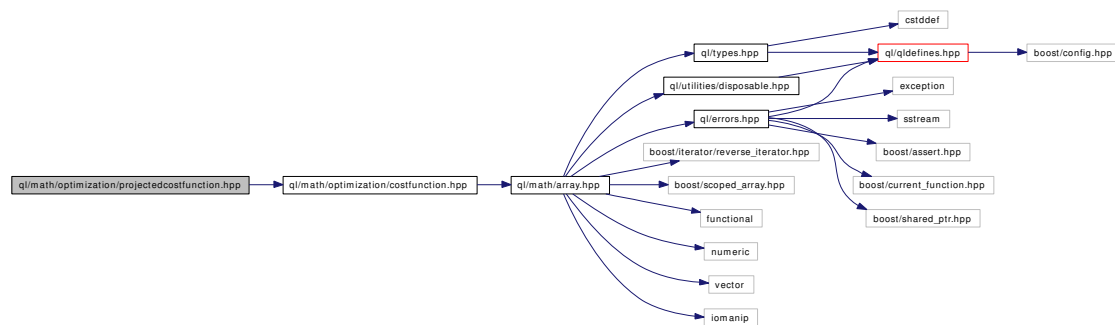
10.176 ql/math/optimization/projectedcostfunction.hpp File Reference

10.176.1 Detailed Description

Cost function utility.

```
#include <ql/math/optimization/costfunction.hpp>
```

Include dependency graph for projectedcostfunction.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ProjectedCostFunction](#)
Parameterized cost function.

10.177 ql/math/optimization/simplex.hpp File Reference

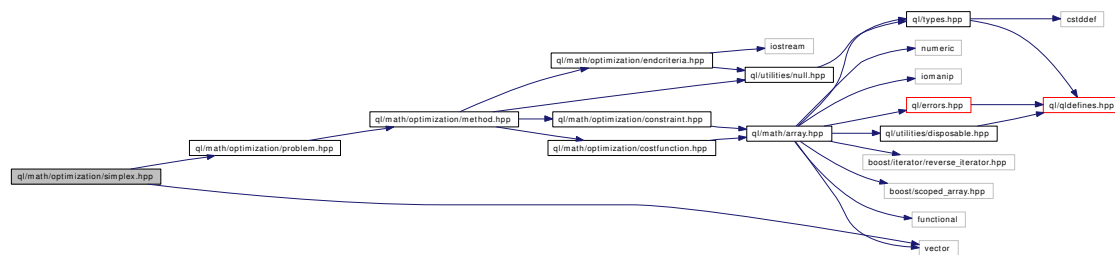
10.177.1 Detailed Description

Simplex optimization method.

```
#include <ql/math/optimization/problem.hpp>
```

```
#include <vector>
```

Include dependency graph for simplex.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Simplex**
Multi-dimensional simplex class.

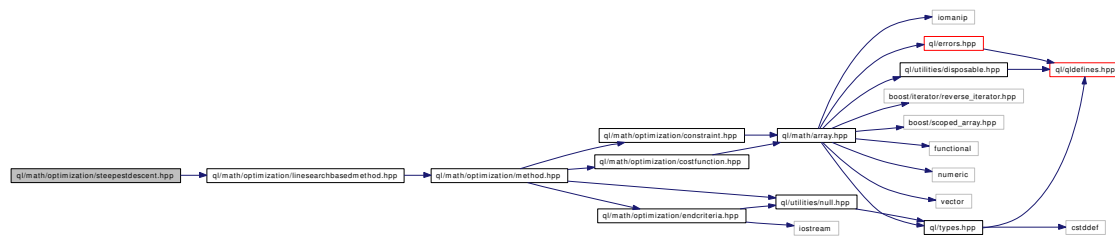
10.178 ql/math/optimization/steepestdescent.hpp File Reference

10.178.1 Detailed Description

Steepest descent optimization method.

```
#include <ql/math/optimization/linesearchbasedmethod.hpp>
```

Include dependency graph for steepestdescent.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SteepestDescent](#)
Multi-dimensional steepest-descent class.

10.179 ql/math/primenumbers.hpp File Reference

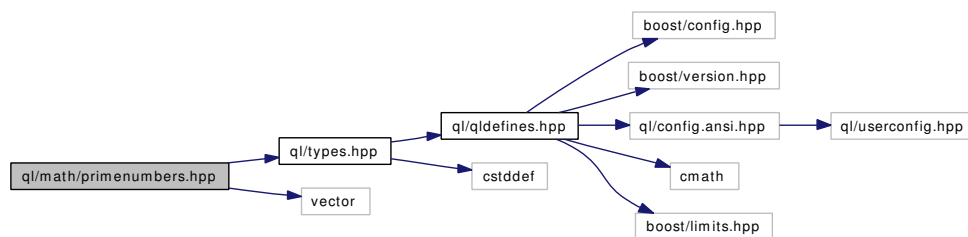
10.179.1 Detailed Description

Prime numbers calculator.

```
#include <ql/types.hpp>
```

```
#include <vector>
```

Include dependency graph for primenumbers.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PrimeNumbers](#)
Prime numbers calculator.

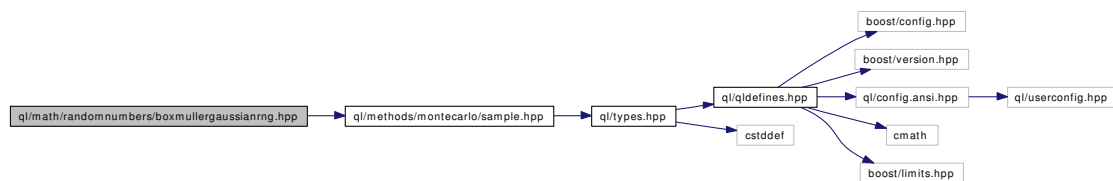
10.180 ql/math/randomnumbers/boxmullergaussianrng.hpp File Reference

10.180.1 Detailed Description

Box-Muller Gaussian random-number generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

Include dependency graph for boxmullergaussianrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BoxMullerGaussianRng](#)
Gaussian random number generator.

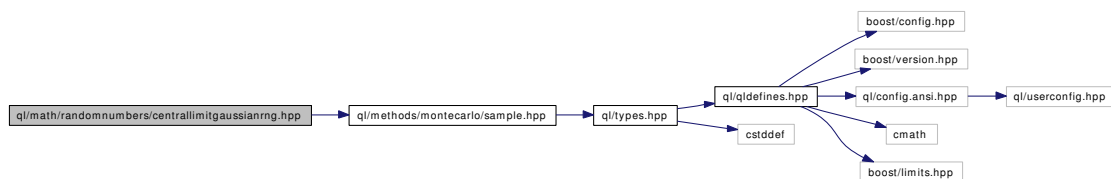
10.181 ql/math/randomnumbers/centrallimitgaussianrng.hpp File Reference

10.181.1 Detailed Description

Central limit Gaussian random-number generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

Include dependency graph for centrallimitgaussianrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CLGaussianRng](#)
Gaussian random number generator.

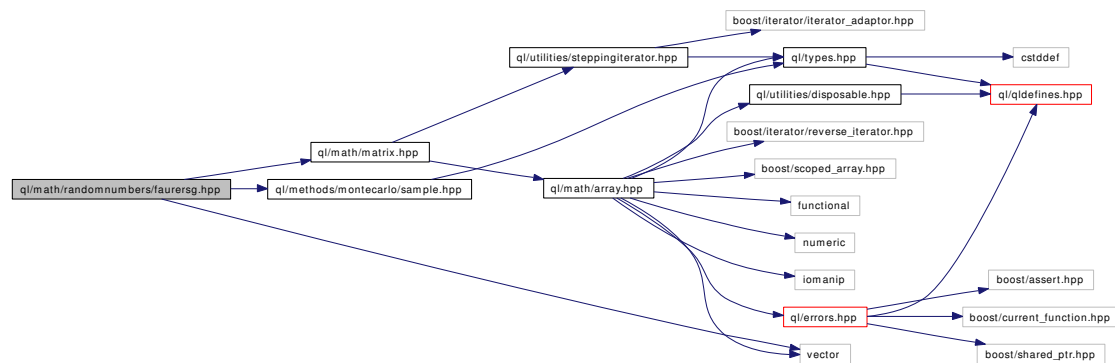
10.182 ql/math/randomnumbers/faurersg.hpp File Reference

10.182.1 Detailed Description

Faure low-discrepancy sequence generator.

```
#include <ql/math/matrix.hpp>
#include <ql/methods/montecarlo/sample.hpp>
#include <vector>
```

Include dependency graph for faurersg.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FaureRsg](#)
Faure low-discrepancy sequence generator.

10.183 ql/math/randomnumbers/haltonrsg.hpp File Reference

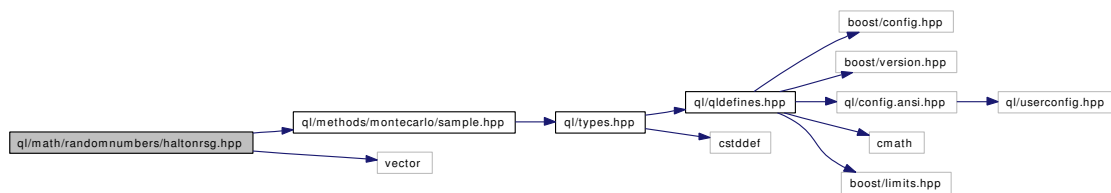
10.183.1 Detailed Description

Halton low-discrepancy sequence generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for haltonrsg.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [HaltonRsg](#)
Halton low-discrepancy sequence generator.

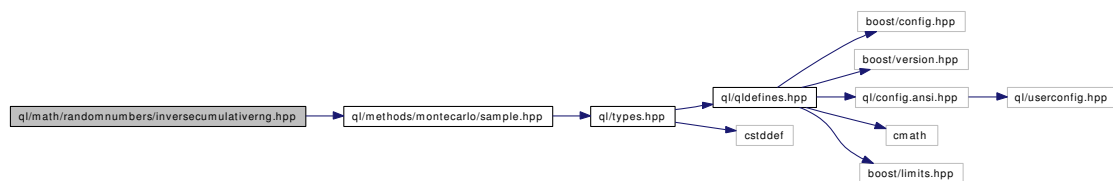
10.184 ql/math/randomnumbers/inversecumulativrng.hpp File Reference

10.184.1 Detailed Description

Inverse cumulative Gaussian random-number generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

Include dependency graph for inversecumulativrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InverseCumulativeRng](#)
Inverse cumulative random number generator.

10.185 ql/math/randomnumbers/inversecumulativergs.hpp File Reference

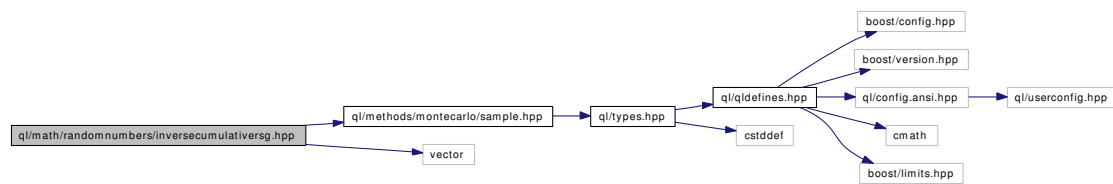
10.185.1 Detailed Description

Inverse cumulative random sequence generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for inversecumulativergs.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InverseCumulativeRsg](#)
Inverse cumulative random sequence generator.

10.186 `ql/math/randomnumbers/knuthuniformrng.hpp` File Reference

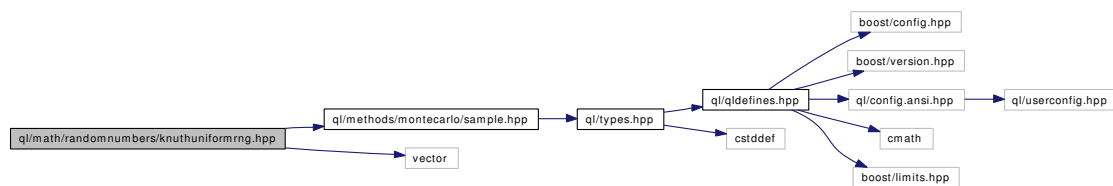
10.186.1 Detailed Description

Knuth uniform random number generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for `knuthuniformrng.hpp`:



Namespaces

- namespace `QuantLib`

Classes

- class `KnuthUniformRng`
Uniform random number generator.

10.187 ql/math/randomnumbers/lecuyeruniformrng.hpp File Reference

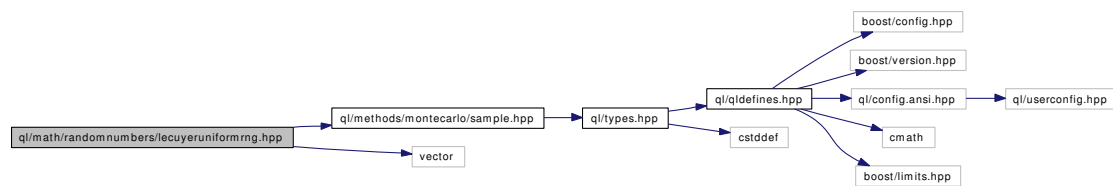
10.187.1 Detailed Description

L'Ecuyer uniform random number generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for lecuyeruniformrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LecuyerUniformRng](#)
Uniform random number generator.

10.188 `ql/math/randomnumbers/mt19937uniformrng.hpp` File Reference

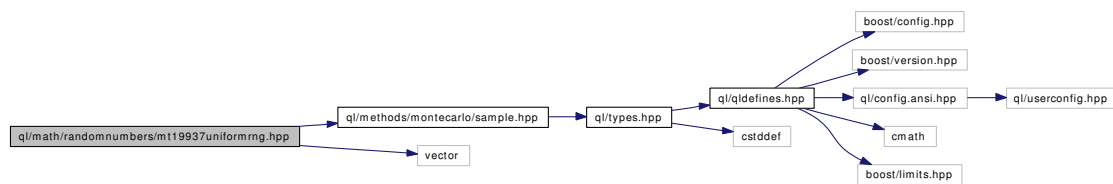
10.188.1 Detailed Description

Mersenne Twister uniform random number generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for `mt19937uniformrng.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [MersenneTwisterUniformRng](#)
Uniform random number generator.

10.189 ql/math/randomnumbers/randomizedlds.hpp File Reference

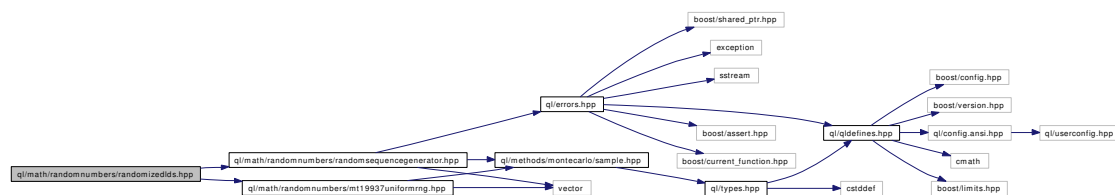
10.189.1 Detailed Description

Randomized low-discrepancy sequence.

```
#include <ql/math/randomnumbers/randomsequencegenerator.hpp>
```

```
#include <ql/math/randomnumbers/mt19937uniformrng.hpp>
```

Include dependency graph for randomizedlds.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [RandomizedLDS](#)

Randomized (random shift) low-discrepancy sequence.

10.190 ql/math/randomnumbers/randomsequencegenerator.hpp File Reference

10.190.1 Detailed Description

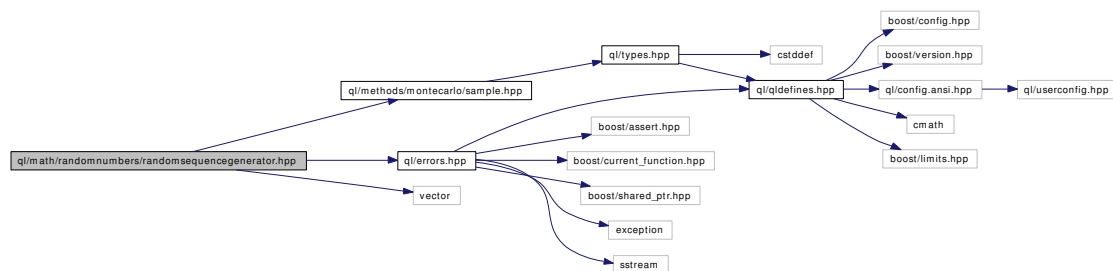
Random sequence generator based on a pseudo-random number generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <vector>
```

Include dependency graph for randomsequencegenerator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [RandomSequenceGenerator](#)

Random sequence generator based on a pseudo-random number generator.

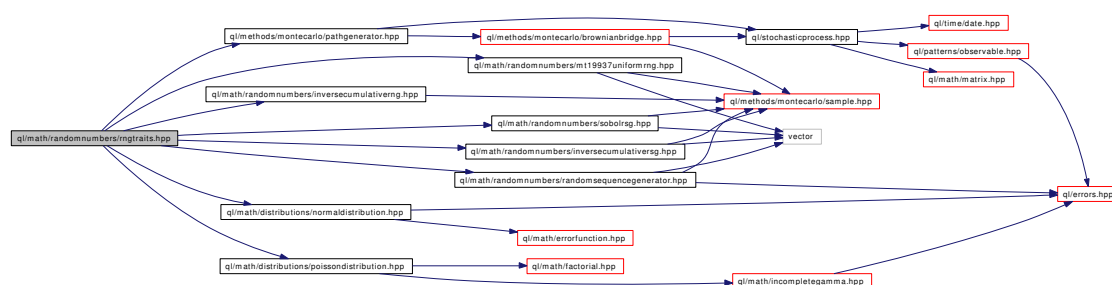
10.191 ql/math/randomnumbers/rngtraits.hpp File Reference

10.191.1 Detailed Description

random-number generation policies

```
#include <ql/methods/montecarlo/pathgenerator.hpp>
#include <ql/math/randomnumbers/mt19937uniformrng.hpp>
#include <ql/math/randomnumbers/inversecumulativrng.hpp>
#include <ql/math/randomnumbers/randomsequencegenerator.hpp>
#include <ql/math/randomnumbers/sobolrsg.hpp>
#include <ql/math/randomnumbers/inversecumulativsrg.hpp>
#include <ql/math/distributions/normaldistribution.hpp>
#include <ql/math/distributions/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 > [LowDiscrepancy](#)
default traits for low-discrepancy sequence generation

10.192 ql/math/randomnumbers/seedgenerator.hpp File Reference

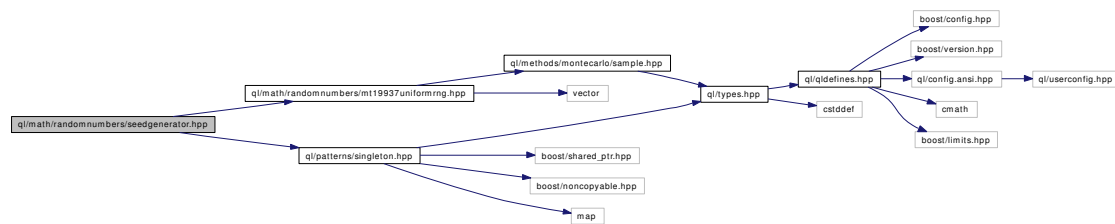
10.192.1 Detailed Description

Random seed generator.

```
#include <ql/math/randomnumbers/mt19937uniformrng.hpp>
```

```
#include <ql/patterns/singleton.hpp>
```

Include dependency graph for seedgenerator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **SeedGenerator**
Random seed generator.

10.193 ql/math/randomnumbers/sobolrsg.hpp File Reference

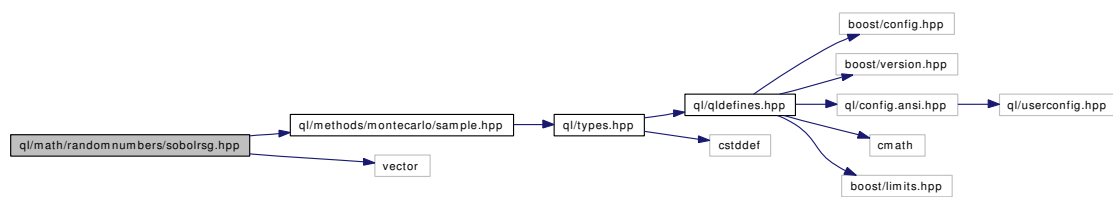
10.193.1 Detailed Description

Sobol low-discrepancy sequence generator.

```
#include <ql/methods/montecarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for sobolrsg.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SobolRsg](#)
Sobol low-discrepancy sequence generator.

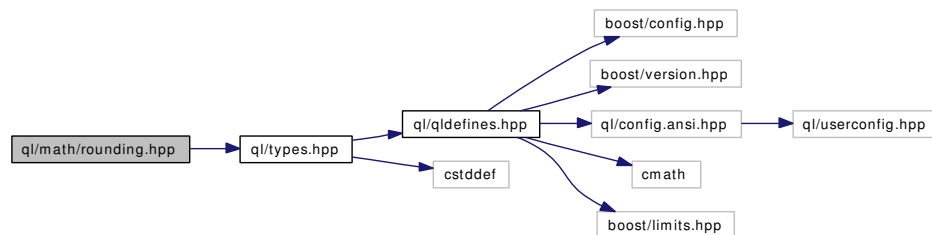
10.194 ql/math/rounding.hpp File Reference

10.194.1 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.

10.195 ql/math/sampledcurve.hpp File Reference

10.195.1 Detailed Description

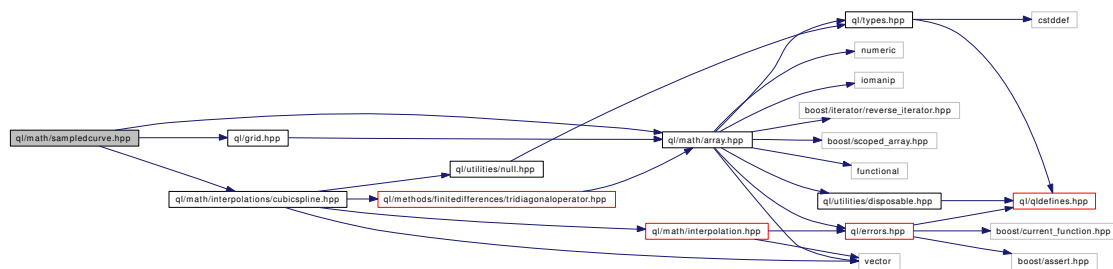
a class that contains a sampled curve

```
#include <ql/math/array.hpp>
```

```
#include <ql/grid.hpp>
```

```
#include <ql/math/interpolations/cubicspline.hpp>
```

Include dependency graph for sampledcurve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SampledCurve](#)
This class contains a sampled curve.

Typedefs

- typedef `SampledCurve` **SampledCurveSet**

Functions

- void **swap** (SampledCurve &, SampledCurve &)
- std::ostream & **operator<<** (std::ostream &out, const SampledCurve &a)

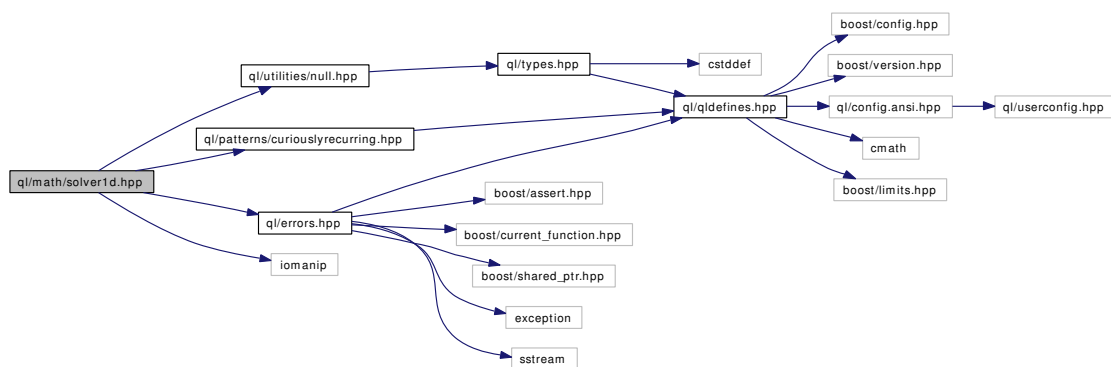
10.196 ql/math/solver1d.hpp File Reference

10.196.1 Detailed Description

Abstract 1-D solver class.

```
#include <ql/utilities/null.hpp>
#include <ql/patterns/curiouslyrecurring.hpp>
#include <ql/errors.hpp>
#include <iomanip>
```

Include dependency graph for solver1d.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Solver1D](#)
Base class for 1-D solvers.

Defines

- `#define MAX_FUNCTION_EVALUATIONS 100`

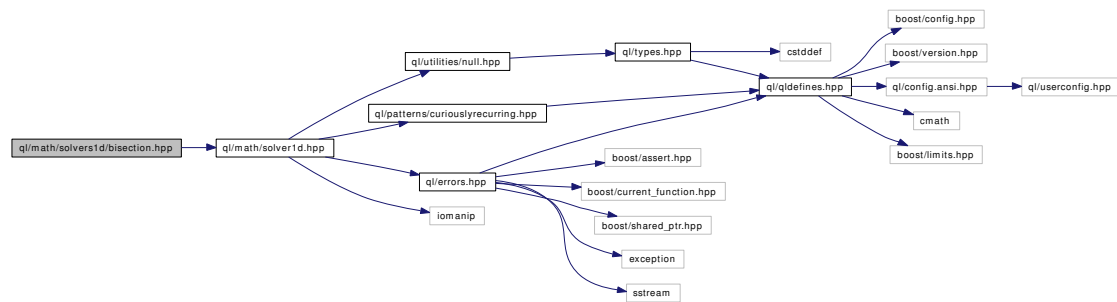
10.197 ql/math/solvers1d/bisection.hpp File Reference

10.197.1 Detailed Description

bisection 1-D solver

```
#include <ql/math/solver1d.hpp>
```

Include dependency graph for bisection.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Bisection**
Bisection 1-D solver

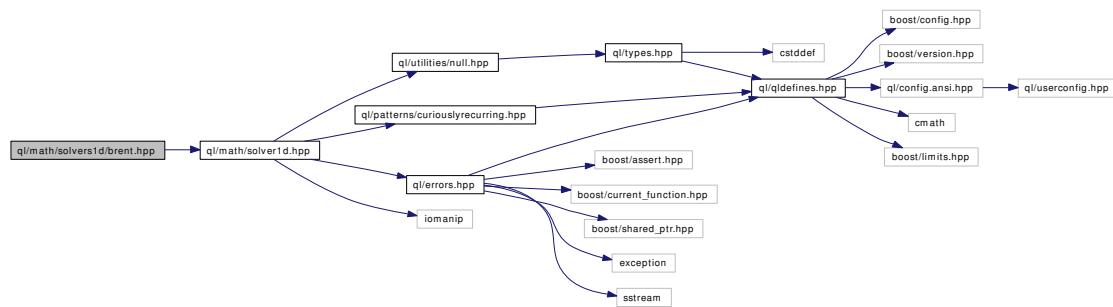
10.198 ql/math/solvers1d/brent.hpp File Reference

10.198.1 Detailed Description

Brent 1-D solver.

```
#include <ql/math/solver1d.hpp>
```

Include dependency graph for brent.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Brent**
Brent 1-D solver

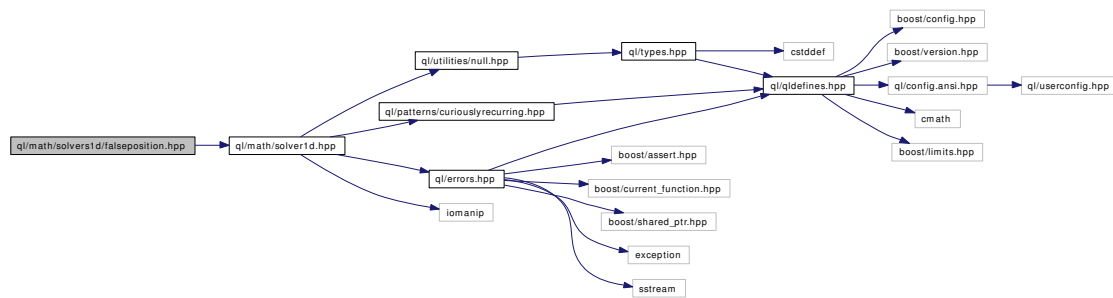
10.199 ql/math/solvers1d/falseposition.hpp File Reference

10.199.1 Detailed Description

false-position 1-D solver

```
#include <ql/math/solver1d.hpp>
```

Include dependency graph for falseposition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FalsePosition](#)
False position 1-D solver.

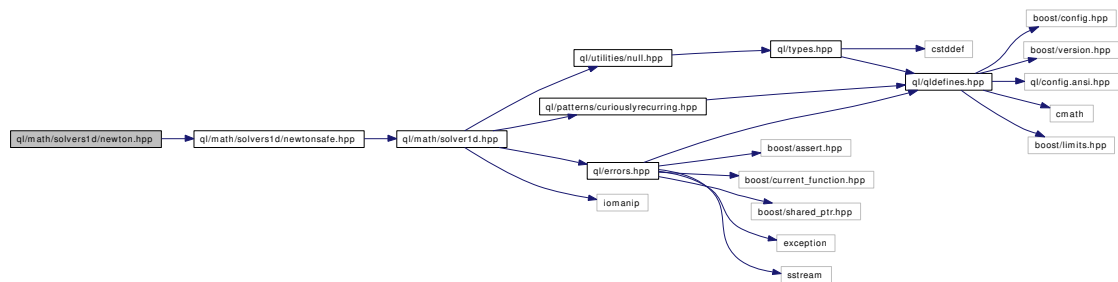
10.200 ql/math/solvers1d/newton.hpp File Reference

10.200.1 Detailed Description

Newton 1-D solver.

```
#include <ql/math/solvers1d/newtonsafe.hpp>
```

Include dependency graph for newton.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Newton](#)
Newton 1-D solver

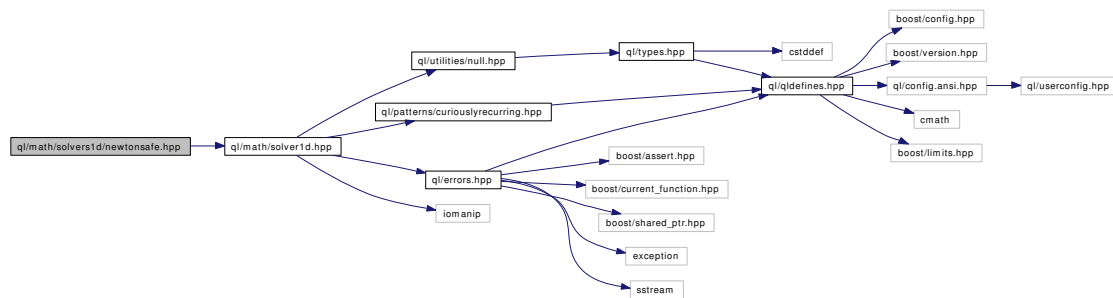
10.201 ql/math/solvers1d/newtonsafe.hpp File Reference

10.201.1 Detailed Description

Safe (bracketed) Newton 1-D solver.

```
#include <ql/math/solver1d.hpp>
```

Include dependency graph for newtonsafe.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [NewtonSafe](#)
safe Newton 1-D solver

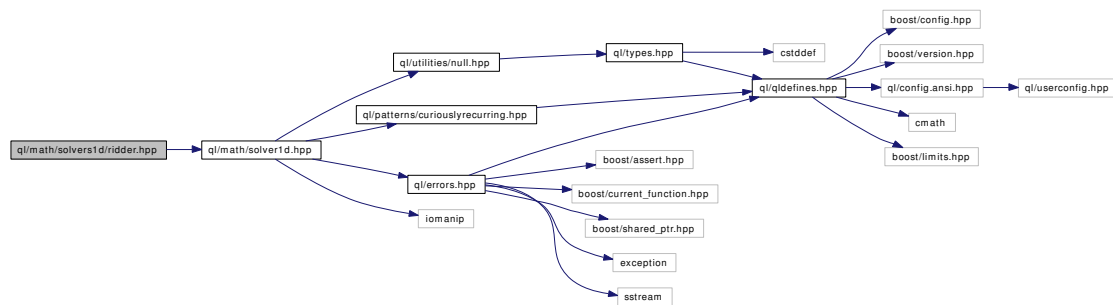
10.202 ql/math/solvers1d/ridder.hpp File Reference

10.202.1 Detailed Description

Ridder 1-D solver.

```
#include <ql/math/solver1d.hpp>
```

Include dependency graph for ridder.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Ridder**
Ridder 1-D solver

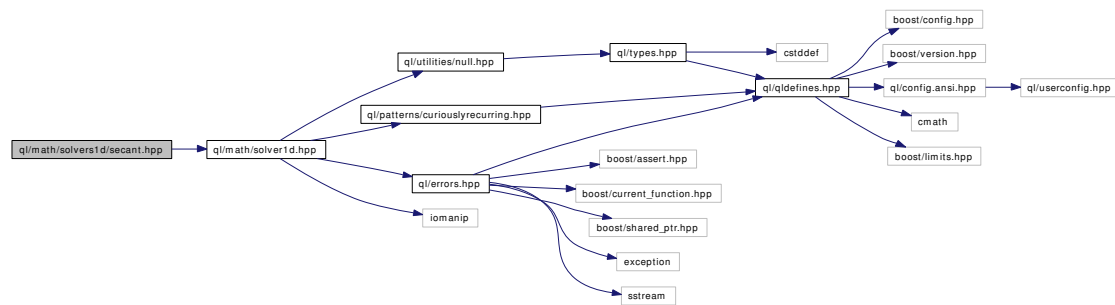
10.203 ql/math/solvers1d/secant.hpp File Reference

10.203.1 Detailed Description

secant 1-D solver

```
#include <ql/math/solver1d.hpp>
```

Include dependency graph for secant.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Secant](#)
Secant 1-D solver

10.204 ql/math/statistics/convergencestatistics.hpp File Reference

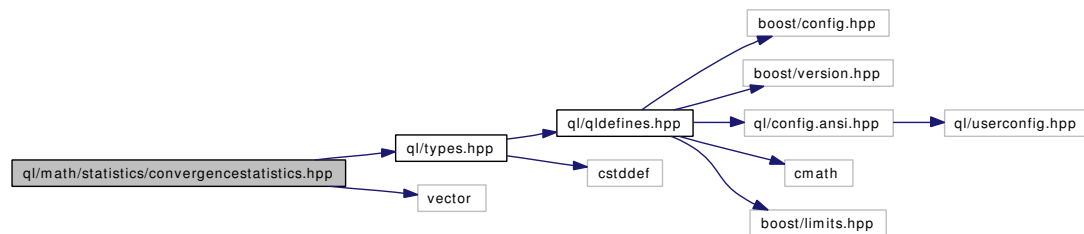
10.204.1 Detailed Description

statistics tool with risk measures

```
#include <ql/types.hpp>
```

```
#include <vector>
```

Include dependency graph for convergencestatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ConvergenceStatistics](#)
statistics class with convergence table

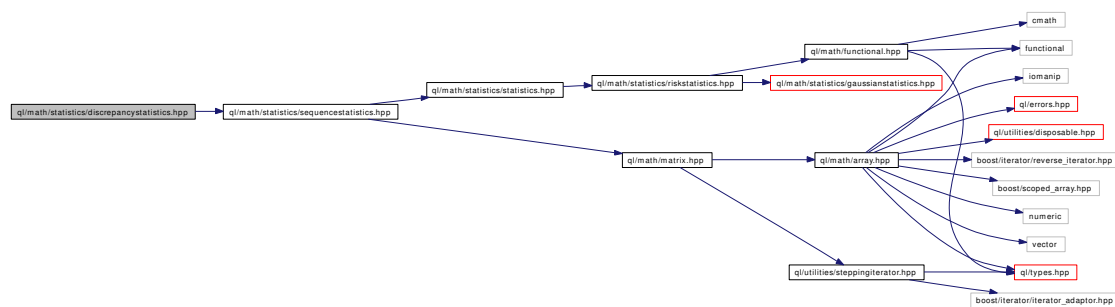
10.205 ql/math/statistics/discrepancystatistics.hpp File Reference

10.205.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

```
#include <ql/math/statistics/sequencestatistics.hpp>
```

Include dependency graph for discrepandystatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DiscrepancyStatistics](#)
Statistic tool for sequences with discrepancy calculation.

10.206 ql/math/statistics/gaussianstatistics.hpp File Reference

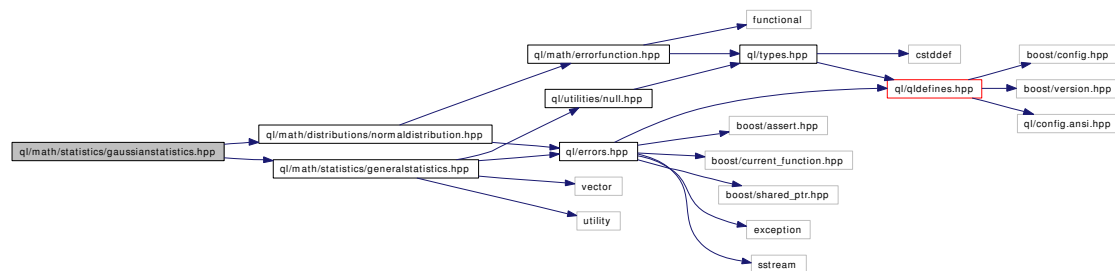
10.206.1 Detailed Description

statistics tool for gaussian-assumption risk measures

```
#include <ql/math/distributions/normaldistribution.hpp>
```

```
#include <ql/math/statistics/generalstatistics.hpp>
```

Include dependency graph for gaussianstatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GenericGaussianStatistics](#)
Statistics tool for gaussian-assumption risk measures.
- class [StatsHolder](#)
Helper class for precomputed distributions.

Typedefs

- typedef `GenericGaussianStatistics< GeneralStatistics >` [GaussianStatistics](#)
default gaussian statistic tool

10.207 ql/math/statistics/generalstatistics.hpp File Reference

10.207.1 Detailed Description

statistics tool

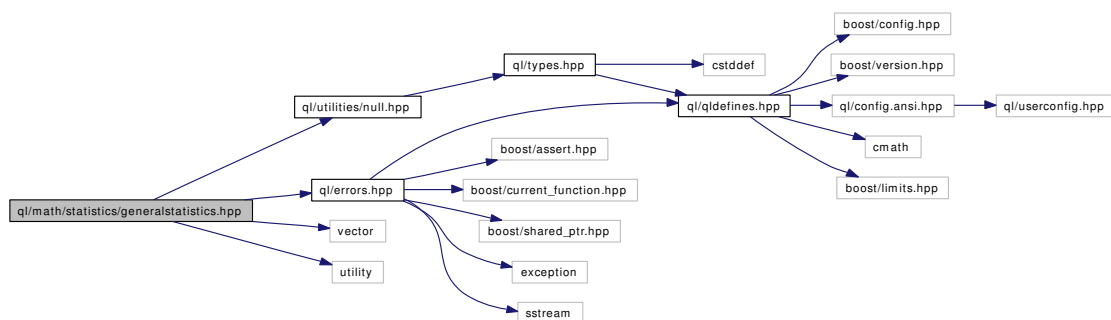
```
#include <ql/utilities/null.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <vector>
```

```
#include <utility>
```

Include dependency graph for generalstatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GeneralStatistics](#)
Statistics tool.

10.208 ql/math/statistics/incrementalstatistics.hpp File Reference

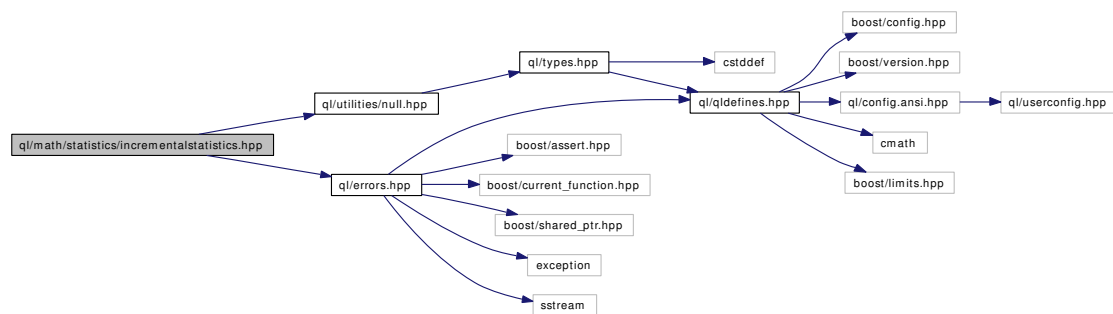
10.208.1 Detailed Description

statistics tool based on incremental accumulation

```
#include <ql/utilities/null.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for incrementalstatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [IncrementalStatistics](#)

Statistics tool based on incremental accumulation.

10.209 ql/math/statistics/riskstatistics.hpp File Reference

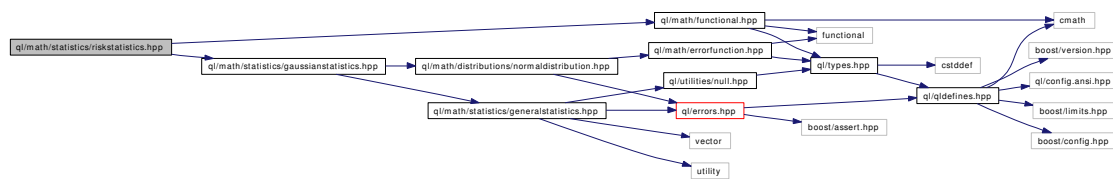
10.209.1 Detailed Description

empirical-distribution risk measures

```
#include <ql/math/functional.hpp>
```

```
#include <ql/math/statistics/gaussianstatistics.hpp>
```

Include dependency graph for riskstatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GenericRiskStatistics](#)
empirical-distribution risk measures

Typedefs

- typedef `GenericRiskStatistics< GaussianStatistics >` [RiskStatistics](#)
default risk measures tool

10.210 ql/math/statistics/sequencestatistics.hpp File Reference

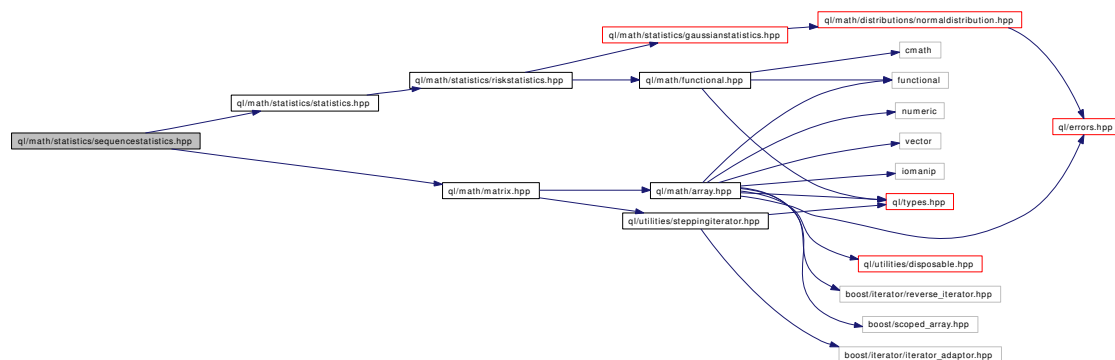
10.210.1 Detailed Description

Statistics tools for sequence (vector, list, array) samples.

```
#include <ql/math/statistics/statistics.hpp>
```

```
#include <ql/math/matrix.hpp>
```

Include dependency graph for sequencestatistics.hpp:



Namespaces

- namespace [QuantLib](#)

Classes

- class [GenericSequenceStatistics](#)
Statistics analysis of N-dimensional (sequence) data.

Defines

- #define `DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID(METHOD)`
- #define `DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE(METHOD)`

Typedefs

- typedef `GenericSequenceStatistics` [SequenceStatistics](#)
default multi-dimensional statistics tool

10.210.2 Define Documentation

10.210.2.1 #define DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID(METHOD)

Value:

```
template <class Stat> \
    std::vector<Real> \
    GenericSequenceStatistics<Stat>::METHOD() const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(); \
        return results_; \
    }
```

10.210.2.2 #define DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE(METHOD)

Value:

```
template <class Stat> \
    std::vector<Real> \
    GenericSequenceStatistics<Stat>::METHOD(Real x) const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(x); \
        return results_; \
    }
```

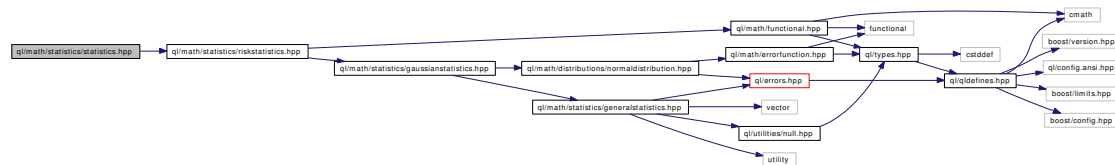
10.211 ql/math/statistics/statistics.hpp File Reference

10.211.1 Detailed Description

statistics tool with risk measures

```
#include <ql/math/statistics/riskstatistics.hpp>
```

Include dependency graph for statistics.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef RiskStatistics [Statistics](#)
default statistics tool

10.212 ql/math/surface.hpp File Reference

10.212.1 Detailed Description

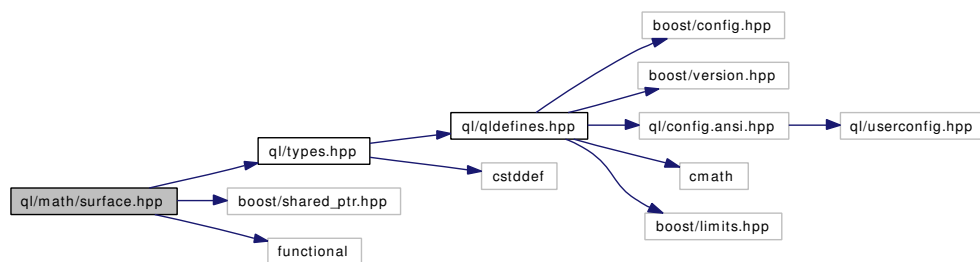
Surface.

```
#include <ql/types.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

```
#include <functional>
```

Include dependency graph for surface.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Surface](#)
Surface abstract class

10.213 ql/math/transformedgrid.hpp File Reference

10.213.1 Detailed Description

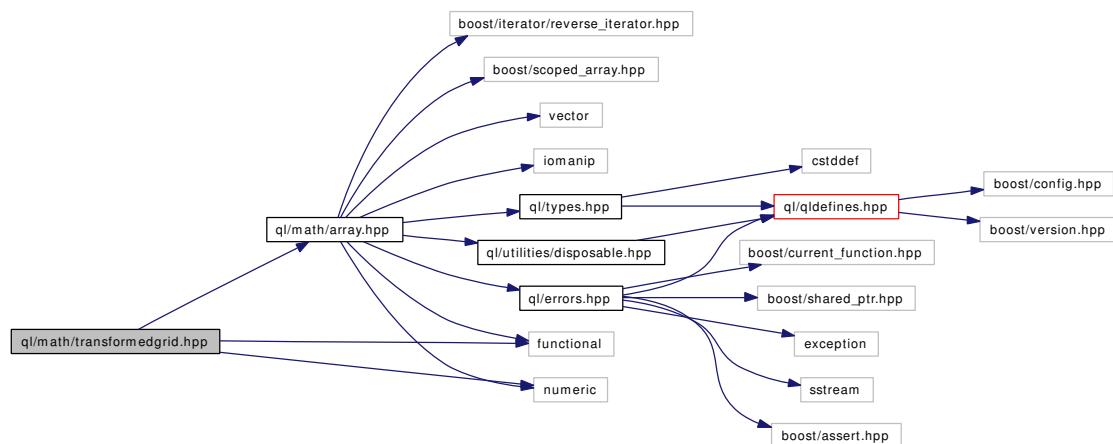
encapsulates a grid

```
#include <ql/math/array.hpp>
```

```
#include <functional>
```

```
#include <numeric>
```

Include dependency graph for transformedgrid.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TransformedGrid](#)
transformed grid

10.214 ql/methods/finitedifferences/americancondition.hpp File Reference

10.214.1 Detailed Description

american option exercise condition

```
#include <ql/methods/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.

10.215 ql/methods/finitedifferences/boundarycondition.hpp

File Reference

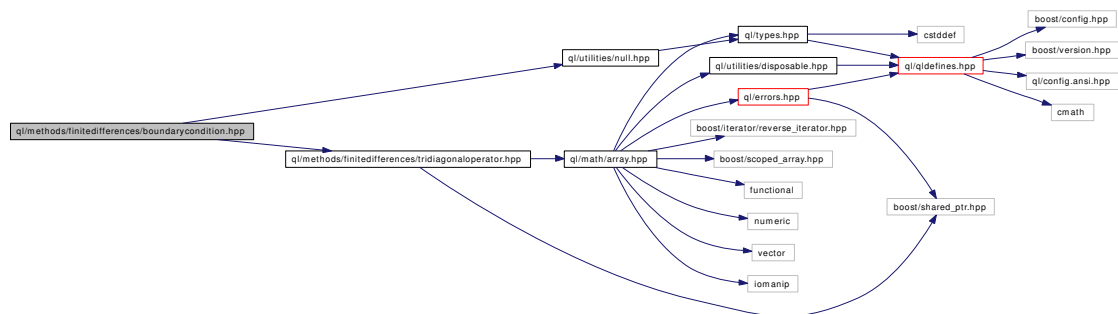
10.215.1 Detailed Description

boundary conditions for differential operators

```
#include <ql/utilities/null.hpp>
```

```
#include <ql/methods/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).

10.216 ql/methods/finitedifferences/bsmoperator.hpp File Reference

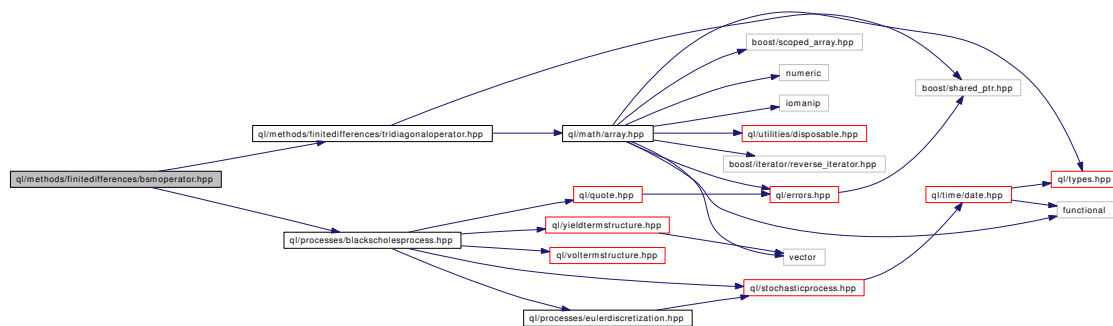
10.216.1 Detailed Description

differential operator for Black-Scholes-Merton equation

```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

Include dependency graph for bsmoperator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BSMOperator**
Black-Scholes-Merton differential operator.

10.217 ql/methods/finitedifferences/bsmtermoperator.hpp File Reference

10.217.1 Detailed Description

differential operator for Black-Scholes-Merton equation

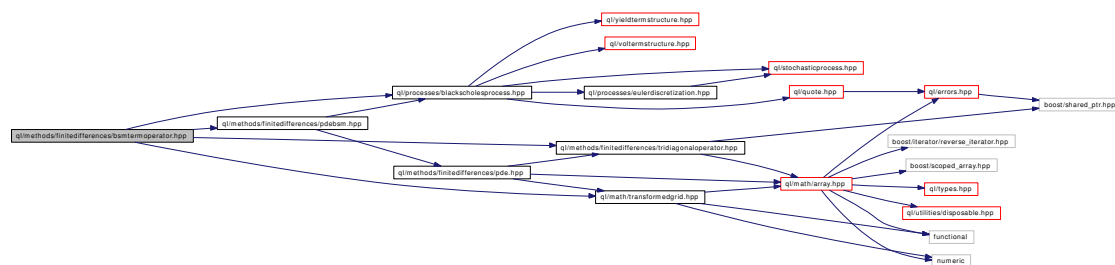
```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

```
#include <ql/math/transformedgrid.hpp>
```

```
#include <ql/methods/finitedifferences/pdebsm.hpp>
```

Include dependency graph for bsmtermoperator.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef PdeOperator< PdeBSM > **BSMTermOperator**
Black-Scholes-Merton differential operator.

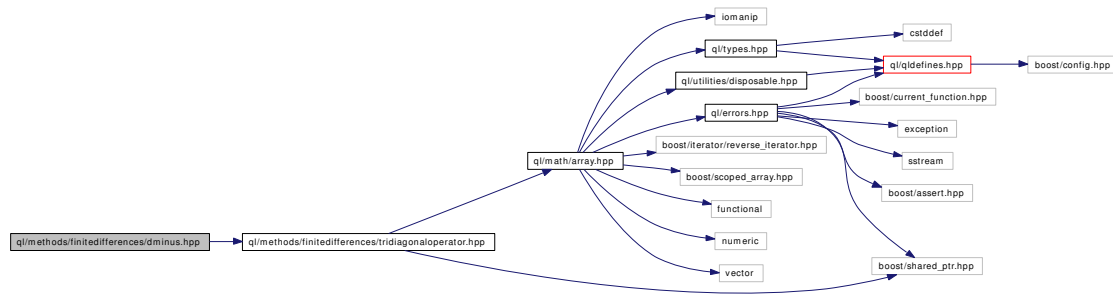
10.219 ql/methods/finitedifferences/dminus.hpp File Reference

10.219.1 Detailed Description

D_- matricial representation

```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dminus.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **DMinus**
 D_- matricial representation

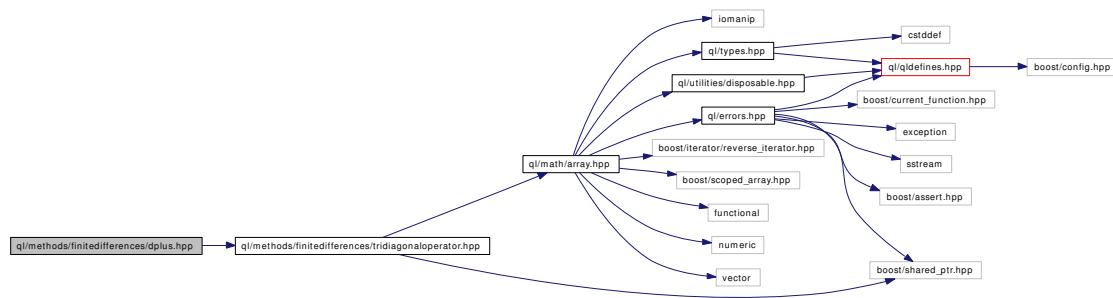
10.220 ql/methods/finitedifferences/dplus.hpp File Reference

10.220.1 Detailed Description

D_+ matricial representation

```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dplus.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **DPlus**
 D_+ matricial representation

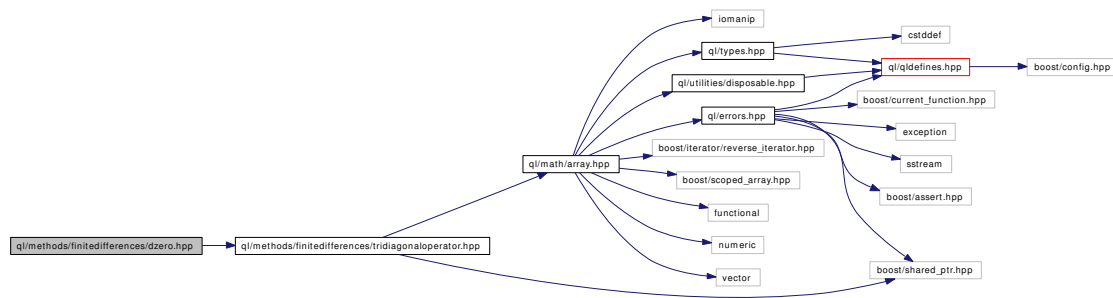
10.222 ql/methods/finitedifferences/dzero.hpp File Reference

10.222.1 Detailed Description

D_0 matricial representation

```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dzero.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **DZero**
 D_0 matricial representation

10.223 `ql/methods/finitedifferences/expliciteuler.hpp` File Reference

10.223.1 Detailed Description

explicit Euler scheme for finite difference methods

```
#include <ql/methods/finitedifferences/mixedscheme.hpp>
```

Include dependency graph for expliciteuler.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class `ExplicitEuler`
Forward Euler scheme for finite difference methods

10.224 ql/methods/finitedifferences/fdtypedefs.hpp File Reference

10.224.1 Detailed Description

default choices for template instantiations

```
#include <ql/methods/finitedifferences/cranknicolson.hpp>
```

```
#include <ql/methods/finitedifferences/parallelevolver.hpp>
```

Include dependency graph for fdtypedefs.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef FiniteDifferenceModel< CrankNicolson< TridiagonalOperator > > [StandardFiniteDifferenceModel](#)
default choice for finite-difference model
- typedef FiniteDifferenceModel< ParallelEvolver< CrankNicolson< TridiagonalOperator > > > [StandardSystemFiniteDifferenceModel](#)
default choice for parallel finite-difference model
- typedef StepCondition< Array > [StandardStepCondition](#)
default choice for step condition
- typedef CurveDependentStepCondition< Array > **StandardCurveDependentStepCondition**

10.228 ql/methods/finitedifferences/onefactoroperator.hpp File Reference

10.228.1 Detailed Description

general differential operator for one-factor interest rate models

```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

```
#include <ql/math/transformedgrid.hpp>
```

```
#include <ql/methods/finitedifferences/pdeshortrate.hpp>
```

Include dependency graph for onefactoroperator.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef PdeOperator< PdeShortRate > **OneFactorOperator**
Interest-rate single factor model differential operator.

10.230.1 Detailed Description

```
#include <ql/methods/finitedifferences/boundarycondition.hpp>
#include <ql/methods/finitedifferences/stepcondition.hpp>
#include <vector>
```

```

graph LR
    q_math_array_hpp["q/math/array.hpp"]
    q_method_finite_differences_operator_traits_hpp["q/methods/finite_differences/operator_traits.hpp"]
    q_method_finite_differences_boundary_condition_hpp["q/methods/finite_differences/boundary_condition.hpp"]
    q_method_finite_differences_tridiagonal_operator_hpp["q/methods/finite_differences/tridiagonal_operator.hpp"]
    q_method_finite_differences_step_condition_hpp["q/methods/finite_differences/step_condition.hpp"]
    q_instruments_payoffs_hpp["q/instruments/payoffs.hpp"]
    q_option_hpp["q/option.hpp"]
    q_utilities_null_hpp["q/utilities/null.hpp"]
    q_errors_hpp["q/errors.hpp"]
    q_utilities_disposable_hpp["q/utilities/disposable.hpp"]
    q_types_hpp["q/types.hpp"]
    q_defines_hpp["q/defines.hpp"]
    cstddef["cstddef"]
    vector["vector"]
    io_manip["io_manip"]
    boost_iterator_reverse_iterator_hpp["boost/iterator/reverse_iterator.hpp"]
    boost_scoped_array_hpp["boost/scoped_array.hpp"]
    functional["functional"]
    numeric["numeric"]
    boost_shared_ptr_hpp["boost/shared_ptr.hpp"]

    q_math_array_hpp --> vector
    q_math_array_hpp --> io_manip
    q_math_array_hpp --> boost_iterator_reverse_iterator_hpp
    q_math_array_hpp --> boost_scoped_array_hpp
    q_math_array_hpp --> functional
    q_math_array_hpp --> numeric
    q_math_array_hpp --> boost_shared_ptr_hpp
    q_math_array_hpp --> q_errors_hpp
    q_math_array_hpp --> q_utilities_disposable_hpp
    q_math_array_hpp --> q_types_hpp
    q_math_array_hpp --> q_defines_hpp
    q_math_array_hpp --> cstddef

    q_method_finite_differences_operator_traits_hpp --> q_math_array_hpp
    q_method_finite_differences_boundary_condition_hpp --> q_math_array_hpp
    q_method_finite_differences_tridiagonal_operator_hpp --> q_math_array_hpp
    q_method_finite_differences_step_condition_hpp --> q_math_array_hpp
    q_instruments_payoffs_hpp --> q_option_hpp
    q_option_hpp --> q_math_array_hpp
    q_utilities_null_hpp --> q_math_array_hpp
  
```

- namespace **QuantLib**

10.232 ql/methods/finitedifferences/pde.hpp File Reference

10.232.1 Detailed Description

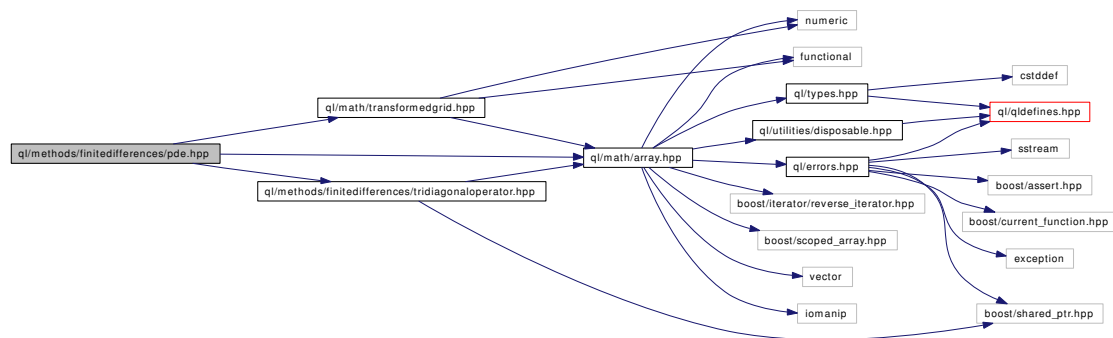
General class for one dimensional PDE's.

```
#include <ql/math/array.hpp>
```

```
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
```

```
#include <ql/math/transformedgrid.hpp>
```

Include dependency graph for pde.hpp:



Namespaces

- namespace **QuantLib**

10.233 ql/methods/finitedifferences/pdebsm.hpp File Reference

10.233.1 Detailed Description

Black-Scholes-Merton PDE.

```
#include <ql/methods/finitedifferences/pde.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

Include dependency graph for pdebsm.hpp:



Namespaces

- namespace **QuantLib**

10.234 ql/methods/finitedifferences/pdeshortrate.hpp File Reference

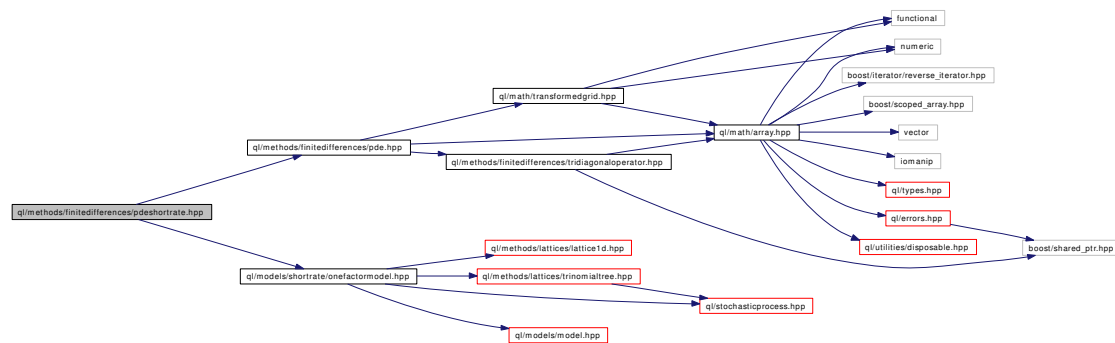
10.234.1 Detailed Description

adapter to short rate

```
#include <ql/methods/finitedifferences/pde.hpp>
```

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

Include dependency graph for pdeshortrate.hpp:



Namespaces

- namespace **QuantLib**

10.235 ql/methods/finitedifferences/shoutcondition.hpp File Reference

10.235.1 Detailed Description

shout option exercise condition

```
#include <ql/methods/finitedifferences/fdtypedefs.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

```
#include <ql/instruments/payoffs.hpp>
```

Include dependency graph for shoutcondition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ShoutCondition](#)
Shout option condition.

10.236 ql/methods/finitedifferences/stepcondition.hpp File Reference

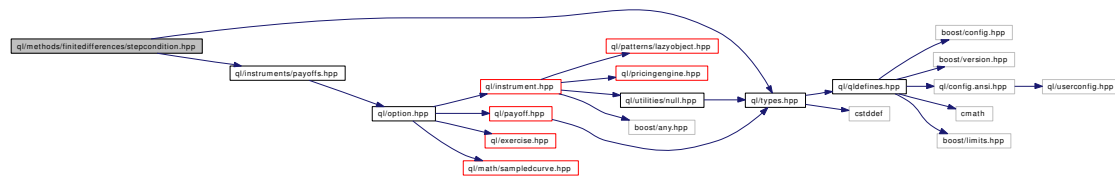
10.236.1 Detailed Description

conditions to be applied at every time step

```
#include <ql/types.hpp>
```

```
#include <ql/instruments/payoffs.hpp>
```

Include dependency graph for stepcondition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **StepCondition**
condition to be applied at every time step
- class **NullCondition**
null step condition

10.237 ql/methods/finitedifferences/tridiagonaloperator.hpp

File Reference

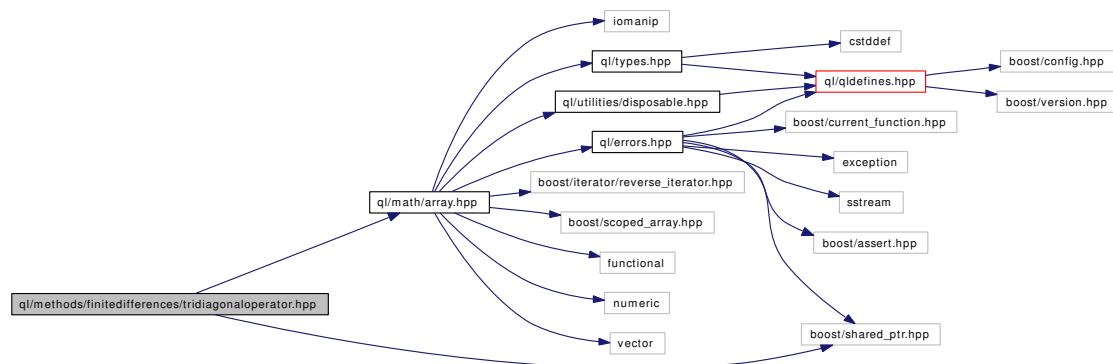
10.237.1 Detailed Description

tridiagonal operator

```
#include <ql/math/array.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for tridiagonaloperator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TridiagonalOperator**
Base implementation for tridiagonal operator.
- class **TridiagonalOperator::TimeSetter**
encapsulation of time-setting logic

Functions

- void **swap** (TridiagonalOperator &, TridiagonalOperator &)
- Disposable< TridiagonalOperator > **operator+** (const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **operator-** (const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **operator+** (const TridiagonalOperator &D1, const TridiagonalOperator &D2)
- Disposable< TridiagonalOperator > **operator-** (const TridiagonalOperator &D1, const TridiagonalOperator &D2)
- Disposable< TridiagonalOperator > **operator *** (Real a, const TridiagonalOperator &D)

- Disposable< TridiagonalOperator > **operator** * (const TridiagonalOperator &D, Real a)
- Disposable< TridiagonalOperator > **operator** / (const TridiagonalOperator &D, Real a)

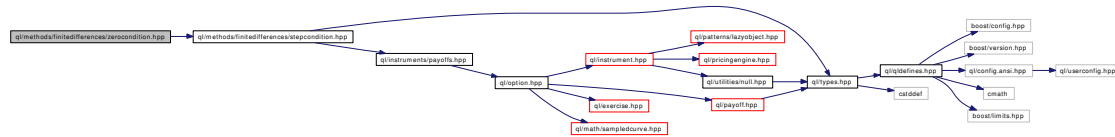
10.238 ql/methods/finitedifferences/zerocondition.hpp File Reference

10.238.1 Detailed Description

zero option exercise condition

```
#include <ql/methods/finitedifferences/stepcondition.hpp>
```

Include dependency graph for zerocondition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ZeroCondition](#)
Zero exercise condition.

Trigeorgis (additive equal jumps) binomial tree

- class [Tian](#)

Tian tree: third moment matching, multiplicative approach

- class [LeisenReimer](#)

Leisen & Reimer tree: multiplicative approach.

10.240 ql/methods/lattices/bsmlattice.hpp File Reference

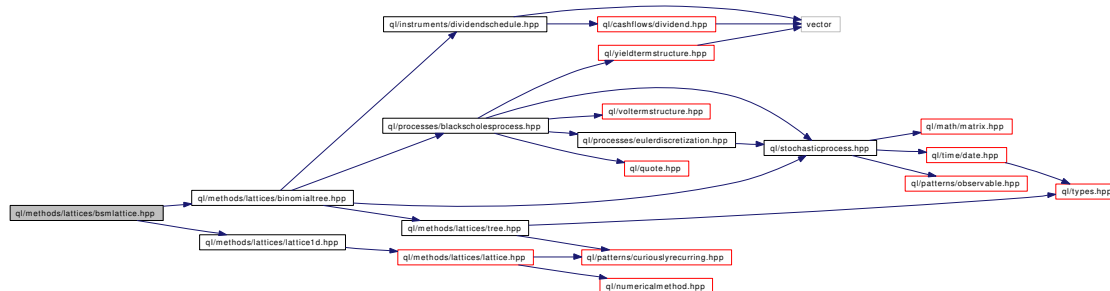
10.240.1 Detailed Description

Binomial trees under the BSM model.

```
#include <ql/methods/lattices/binomialtree.hpp>
```

```
#include <ql/methods/lattices/lattice1d.hpp>
```

Include dependency graph for bsmlattice.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackScholesLattice**

Simple binomial lattice approximating the Black-Scholes model.

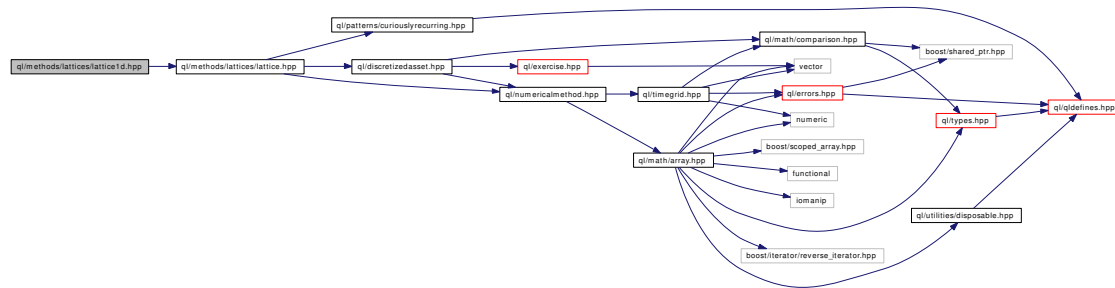
10.242 ql/methods/lattices/lattice1d.hpp File Reference

10.242.1 Detailed Description

One-dimensional lattice class.

```
#include <ql/methods/lattices/lattice.hpp>
```

Include dependency graph for lattice1d.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TreeLattice1D](#)
One-dimensional tree-based lattice.

10.243 ql/methods/lattices/lattice2d.hpp File Reference

10.243.1 Detailed Description

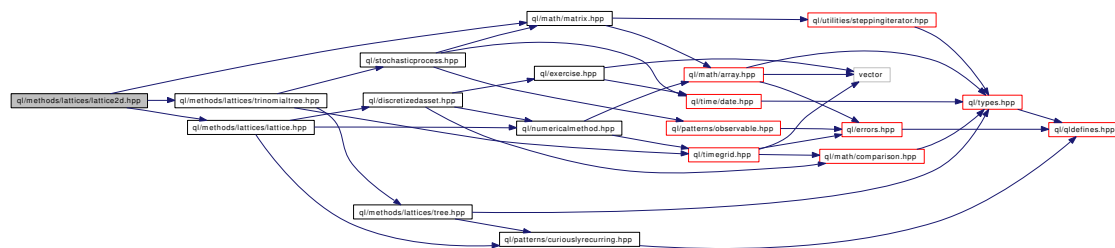
Two-dimensional lattice class.

```
#include <ql/methods/lattices/lattice.hpp>
```

```
#include <ql/methods/lattices/trinomialtree.hpp>
```

```
#include <ql/math/matrix.hpp>
```

Include dependency graph for lattice2d.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TreeLattice2D**
Two-dimensional tree-based lattice.

10.244 ql/methods/lattices/tflattice.hpp File Reference

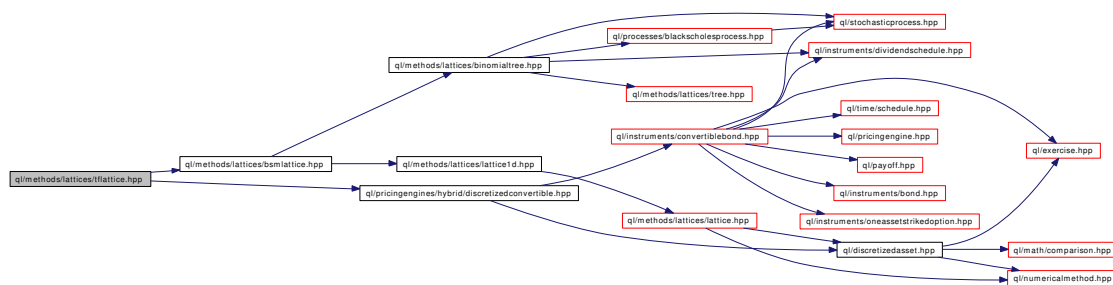
10.244.1 Detailed Description

Binomial Tsiveriotis-Fernandes tree model.

```
#include <ql/methods/lattices/bsmlattice.hpp>
```

```
#include <ql/pricingengines/hybrid/discretizedconvertible.hpp>
```

Include dependency graph for tflattice.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TsiveriotisFernandesLattice**
Binomial lattice approximating the Tsiveriotis-Fernandes model.

10.245 ql/methods/lattices/tree.hpp File Reference

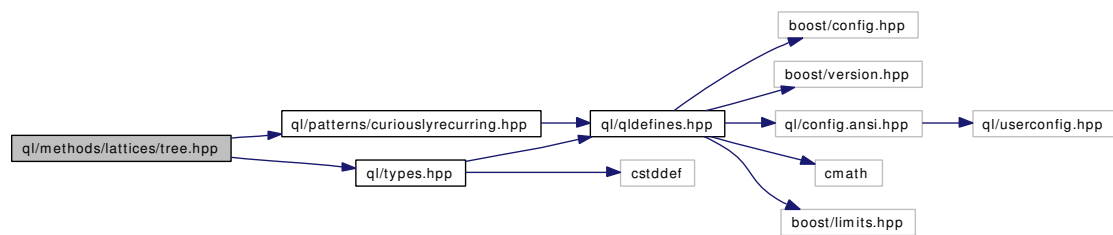
10.245.1 Detailed Description

Tree class.

```
#include <ql/types.hpp>
```

```
#include <ql/patterns/curiouslyrecurring.hpp>
```

Include dependency graph for tree.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Tree](#)

Tree approximating a single-factor diffusion

10.246 ql/methods/lattices/trinomialtree.hpp File Reference

10.246.1 Detailed Description

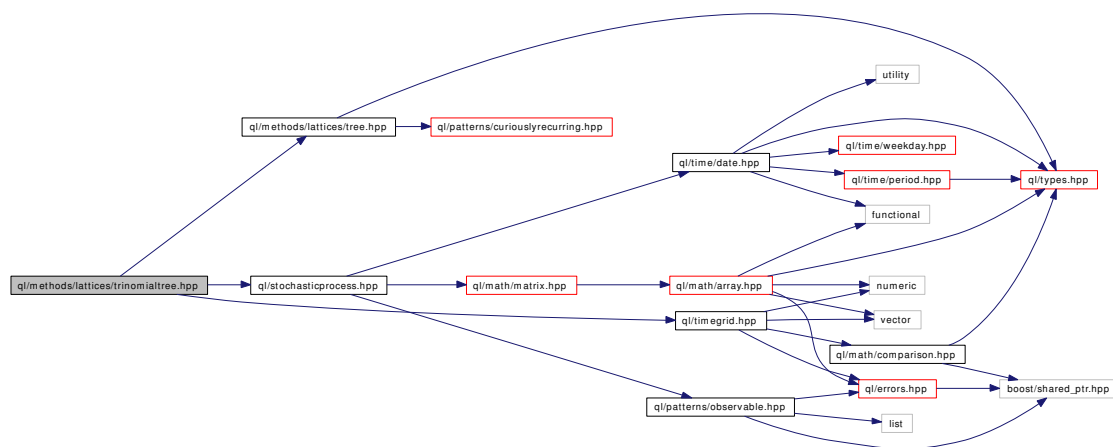
Trinomial tree class.

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/methods/lattices/tree.hpp>
```

```
#include <ql/timegrid.hpp>
```

Include dependency graph for trinomialtree.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TrinomialTree**
Recombining trinomial tree class.

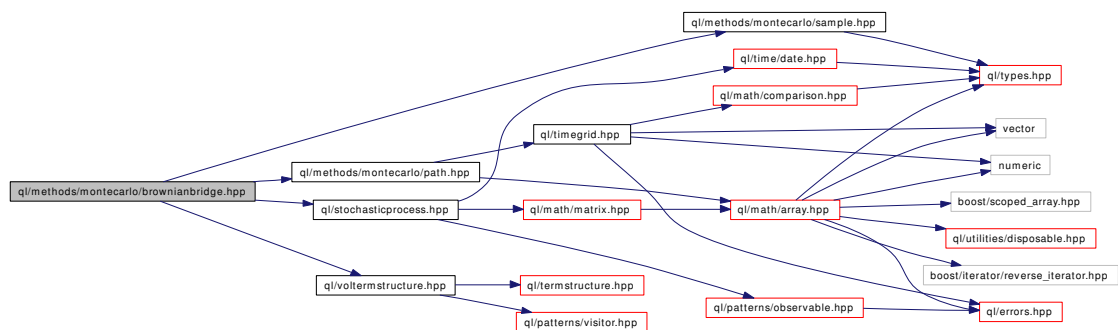
10.247 ql/methods/montecarlo/brownianbridge.hpp File Reference

10.247.1 Detailed Description

Browian bridge.

```
#include <ql/methods/montecarlo/path.hpp>
#include <ql/methods/montecarlo/sample.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/voltermstructure.hpp>
```

Include dependency graph for brownianbridge.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BrownianBridge](#)
Builds Wiener process paths using Gaussian variates.

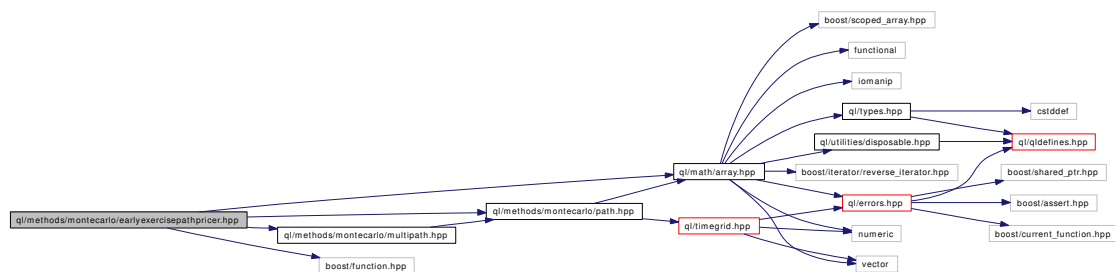
10.248 ql/methods/montecarlo/earlyexercisepathpricer.hpp File Reference

10.248.1 Detailed Description

base class for early exercise single-path pricers

```
#include <ql/math/array.hpp>
#include <ql/methods/montecarlo/path.hpp>
#include <ql/methods/montecarlo/multipath.hpp>
#include <boost/function.hpp>
```

Include dependency graph for earlyexercisepathpricer.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **EarlyExercisePathPricer**
base class for early exercise path pricers

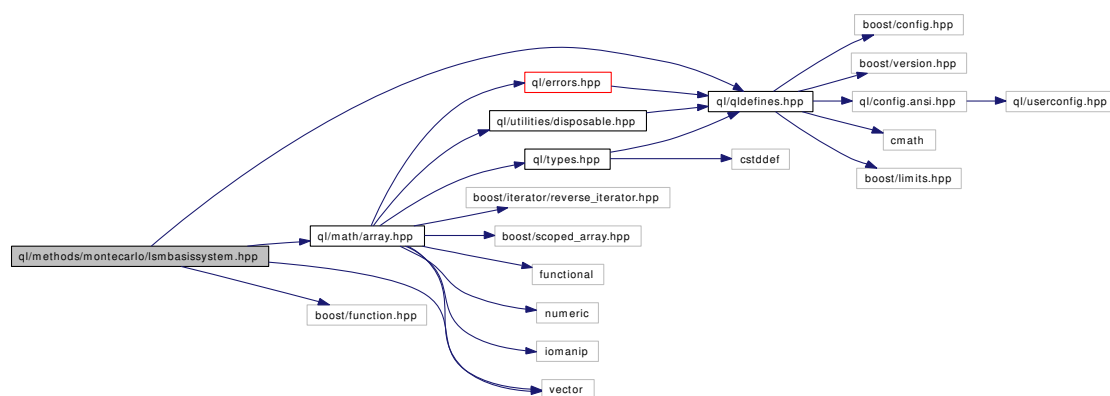
10.250 ql/methods/montecarlo/lsmbasissystem.hpp File Reference

10.250.1 Detailed Description

utility classes for Longstaff-Schwartz early-exercise Monte Carlo

```
#include <ql/qldefines.hpp>
#include <ql/math/array.hpp>
#include <boost/function.hpp>
#include <vector>
```

Include dependency graph for lsmbasissystem.hpp:



Namespaces

- namespace **QuantLib**

10.252 ql/methods/montecarlo/montecarlomodel.hpp File Reference

10.252.1 Detailed Description

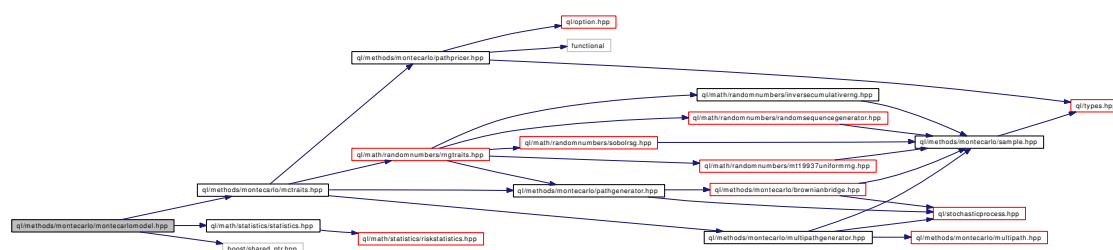
General-purpose Monte Carlo model.

```
#include <ql/methods/montecarlo/mctraits.hpp>
```

```
#include <ql/math/statistics/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.

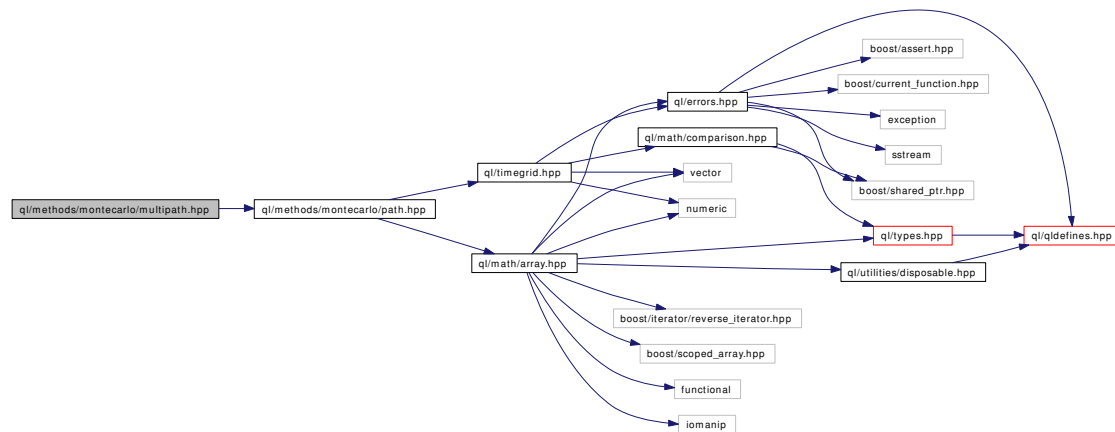
10.253 ql/methods/montecarlo/multipath.hpp File Reference

10.253.1 Detailed Description

Correlated multiple asset paths.

```
#include <ql/methods/montecarlo/path.hpp>
```

Include dependency graph for multipath.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MultiPath](#)
Correlated multiple asset paths.

10.255 ql/methods/montecarlo/path.hpp File Reference

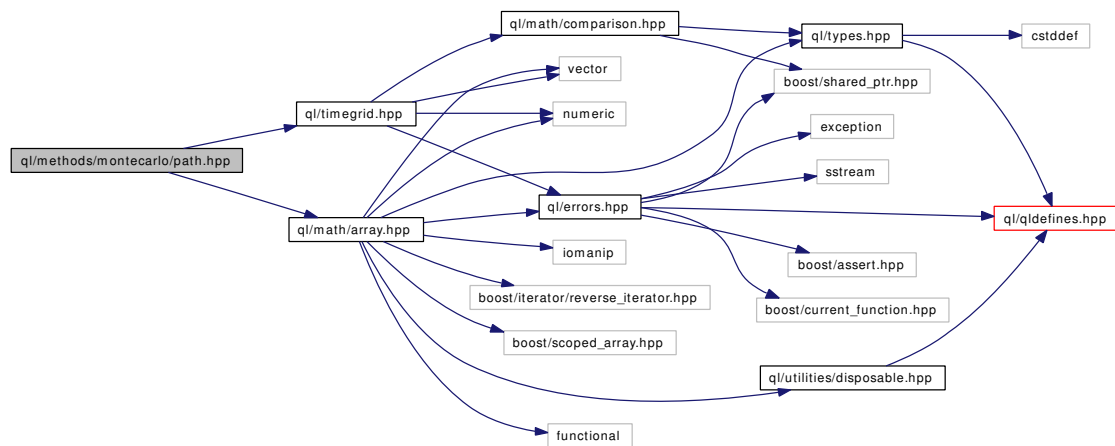
10.255.1 Detailed Description

single factor random walk

```
#include <ql/timegrid.hpp>
```

```
#include <ql/math/array.hpp>
```

Include dependency graph for path.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Path](#)
single-factor random walk

10.256 ql/methods/montecarlo/pathgenerator.hpp File Reference

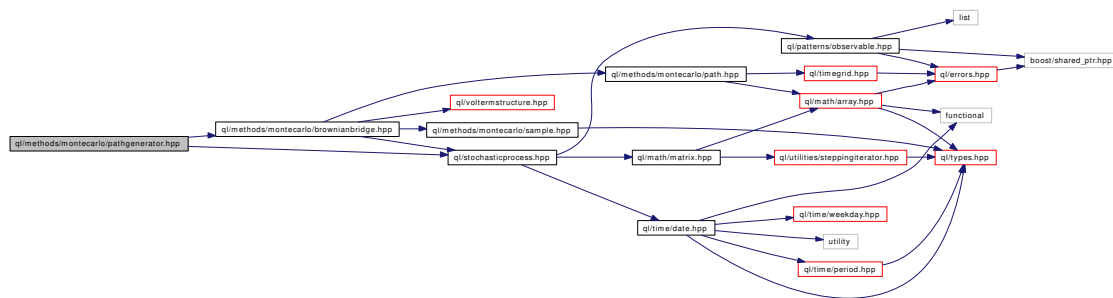
10.256.1 Detailed Description

Generates random paths using a sequence generator.

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/methods/montecarlo/brownianbridge.hpp>
```

Include dependency graph for pathgenerator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PathGenerator](#)
Generates random paths using a sequence generator.

10.257 ql/methods/montecarlo/pathpricer.hpp File Reference

10.257.1 Detailed Description

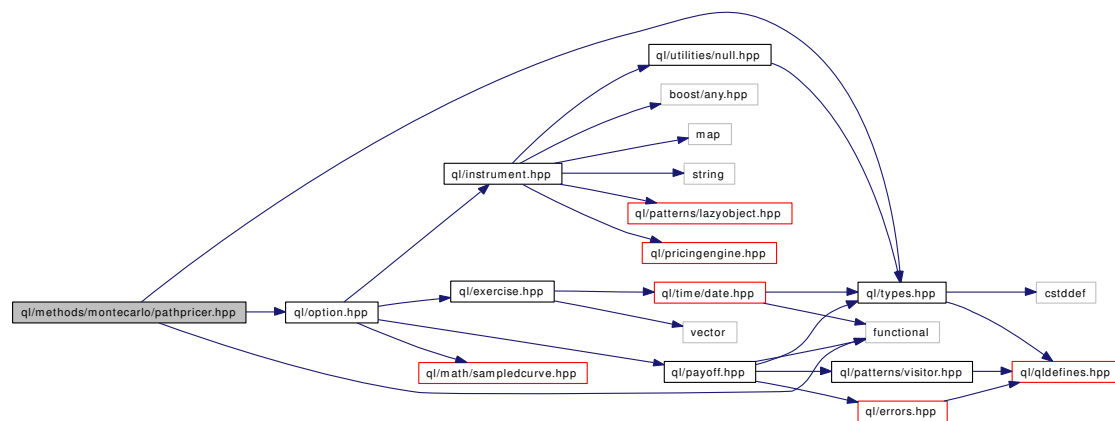
base class for single-path pricers

```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for pathpricer.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PathPricer](#)
base class for path pricers

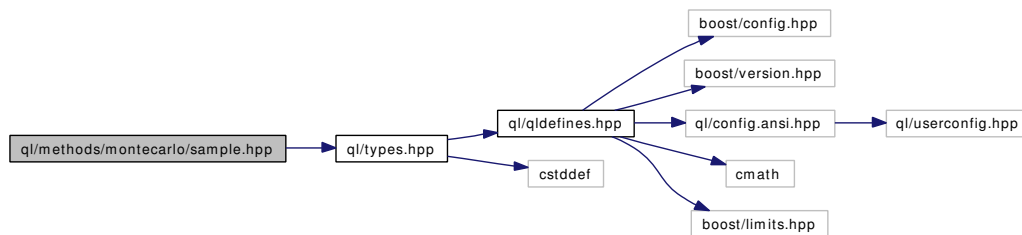
10.258 ql/methods/montecarlo/sample.hpp File Reference

10.258.1 Detailed Description

weighted sample

```
#include <ql/types.hpp>
```

Include dependency graph for sample.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct [Sample](#)
weighted sample

10.259 ql/models/calibrationhelper.hpp File Reference

10.259.1 Detailed Description

Calibration helper class.

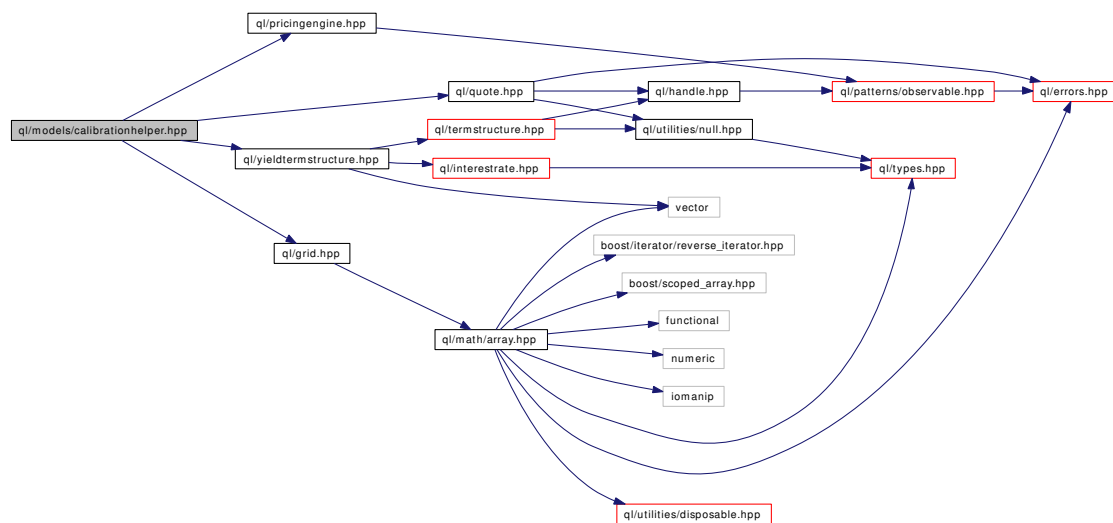
```
#include <ql/grid.hpp>
```

```
#include <ql/quote.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/pricingengine.hpp>
```

Include dependency graph for calibrationhelper.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CalibrationHelper**
liquid market instrument used during calibration

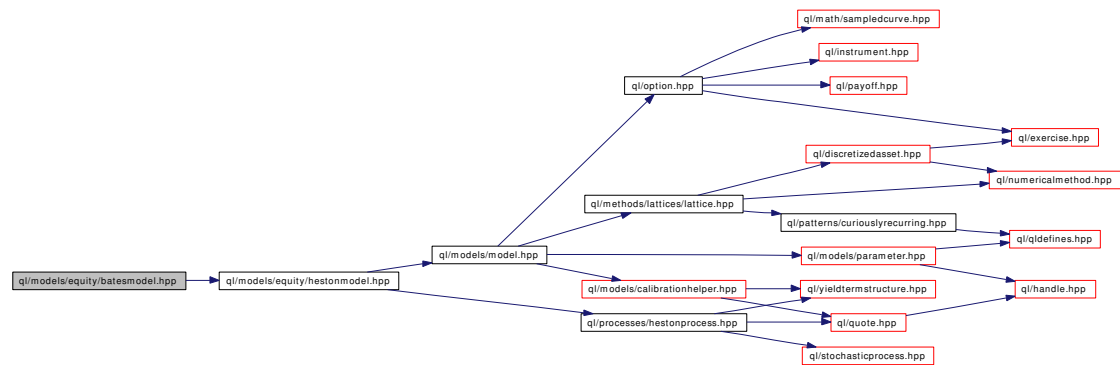
10.260 ql/models/equity/batesmodel.hpp File Reference

10.260.1 Detailed Description

extended versions of the Heston model

```
#include <ql/models/equity/hestonmodel.hpp>
```

Include dependency graph for batesmodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BatesModel](#)
Bates stochastic-volatility model.

10.261 ql/models/equity/hestonmodel.hpp File Reference

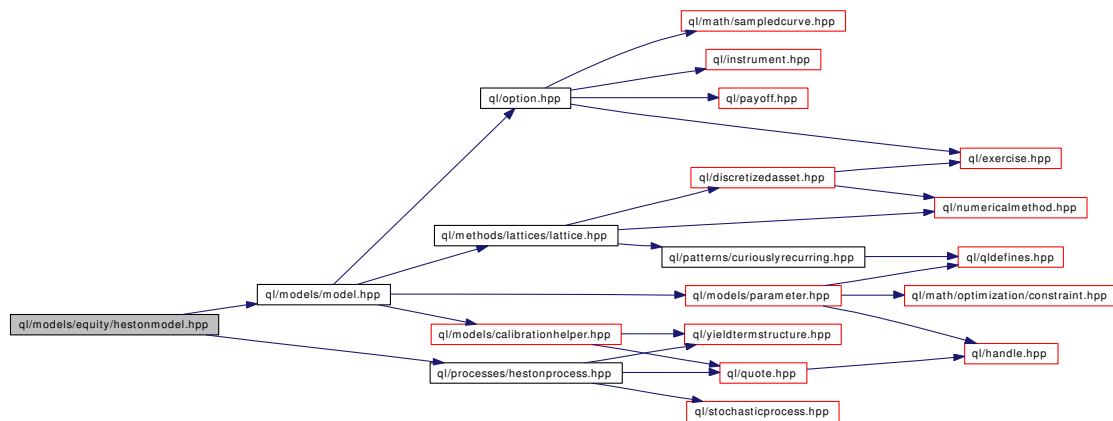
10.261.1 Detailed Description

Heston model for the stochastic volatility of an asset.

```
#include <ql/models/model.hpp>
```

```
#include <ql/processes/hestonprocess.hpp>
```

Include dependency graph for hestonmodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [HestonModel](#)
Heston model for the stochastic volatility of an asset.

10.262 ql/models/equity/hestonmodelhelper.hpp File Reference

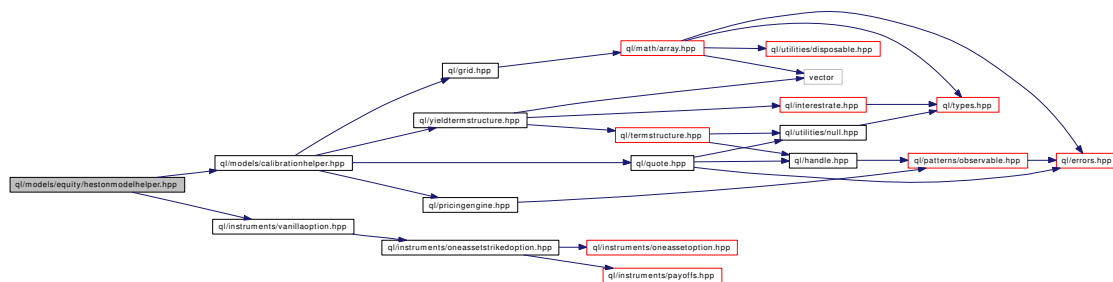
10.262.1 Detailed Description

Heston-model calibration helper.

```
#include <ql/models/calibrationhelper.hpp>
```

```
#include <ql/instruments/vanillaoption.hpp>
```

Include dependency graph for hestonmodelhelper.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **HestonModelHelper**
calibration helper for Heston model

10.263 ql/models/marketmodels/driftcomputation/cmsmmdriftcalculator.hpp File Reference

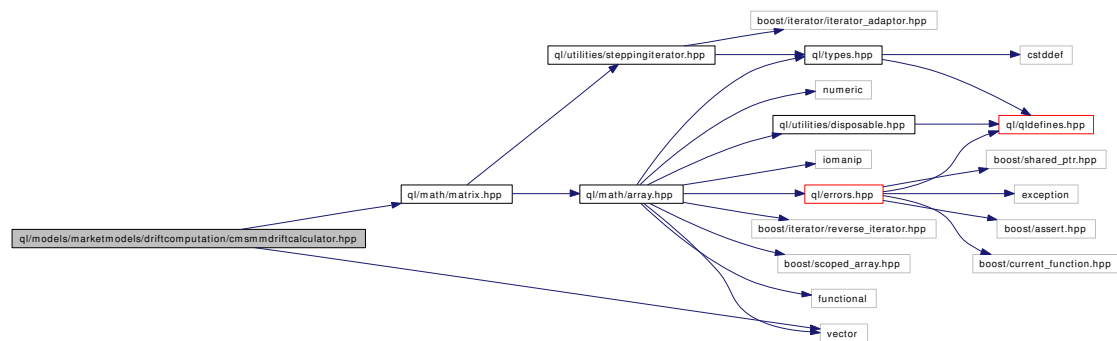
10.263.1 Detailed Description

Drift computation for CMS market model.

```
#include <ql/math/matrix.hpp>
```

```
#include <vector>
```

Include dependency graph for cmsmmdriftcalculator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CMSMMDriftCalculator**
Drift computation for CMS market models.

10.266 ql/models/marketmodels/driftcomputation/smmdriftcalculator.hpp File Reference

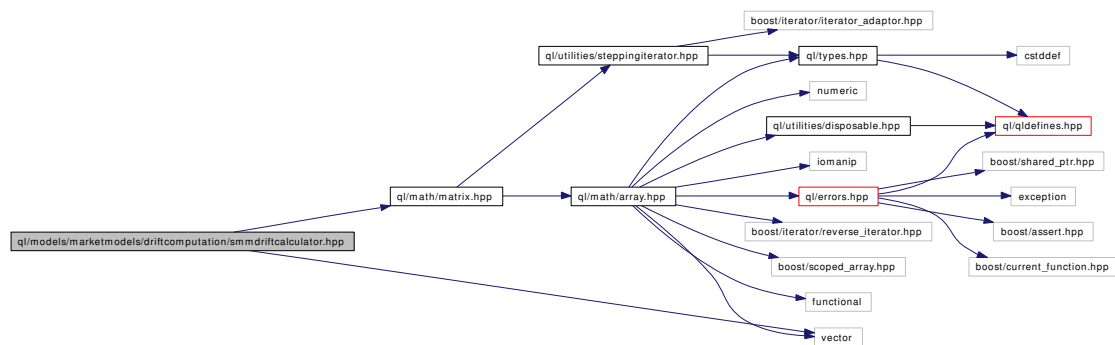
10.266.1 Detailed Description

Drift computation for coterminal-swap market model.

```
#include <ql/math/matrix.hpp>
```

```
#include <vector>
```

Include dependency graph for smmdriftcalculator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SMMDriftCalculator](#)
Drift computation for coterminal swap market models.

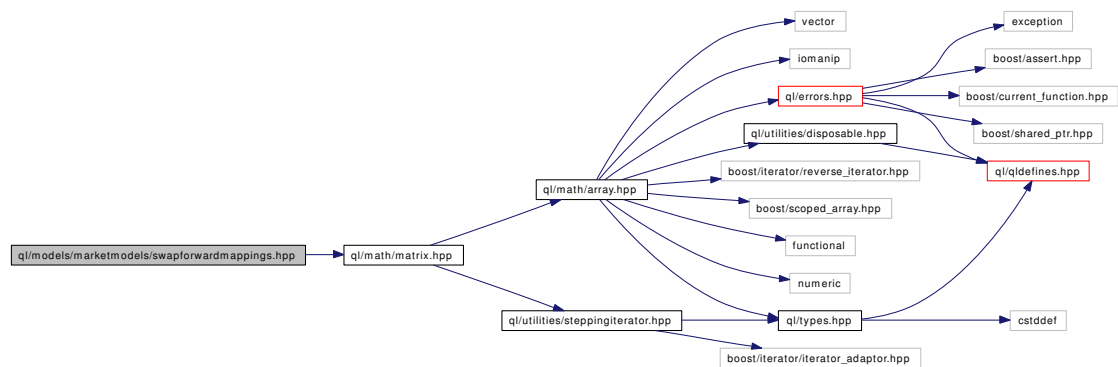
10.267 ql/models/marketmodels/swapforwardmappings.hpp File Reference

10.267.1 Detailed Description

Utility functions for mapping between swap rate and forward rate.

```
#include <ql/math/matrix.hpp>
```

Include dependency graph for swapforwardmappings.hpp:



Namespaces

- namespace **QuantLib**

10.268 ql/models/model.hpp File Reference

10.268.1 Detailed Description

Abstract interest rate model class.

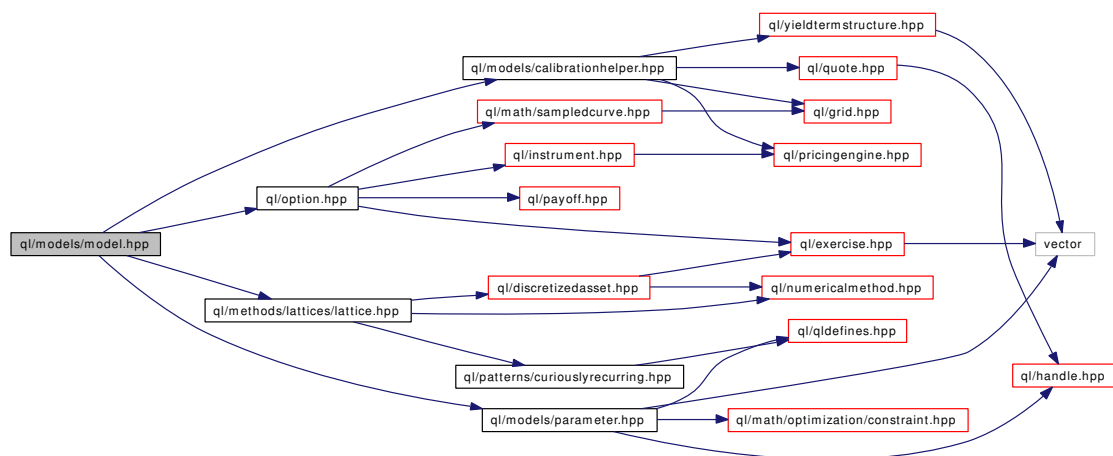
```
#include <ql/option.hpp>
```

```
#include <ql/methods/lattices/lattice.hpp>
```

```
#include <ql/models/parameter.hpp>
```

```
#include <ql/models/calibrationhelper.hpp>
```

Include dependency graph for model.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AffineModel](#)
Affine model class.
- class [TermStructureConsistentModel](#)
Term-structure consistent model class.
- class [CalibratedModel](#)
Calibrated model class.
- class [ShortRateModel](#)
Abstract short-rate model class.

10.269 ql/models/parameter.hpp File Reference

10.269.1 Detailed Description

Model parameter classes.

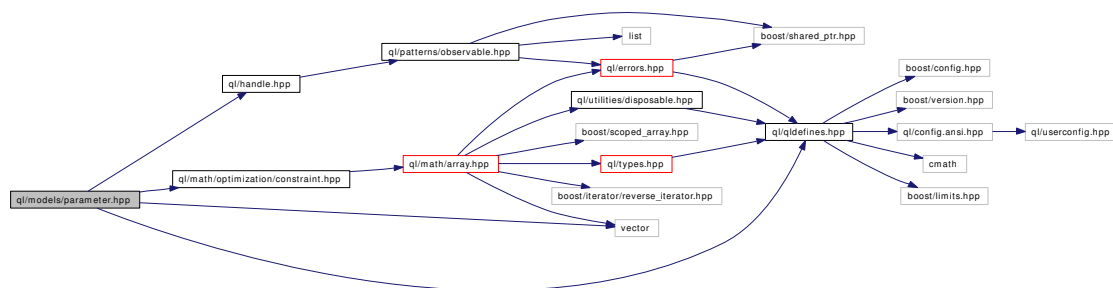
```
#include <ql/qldefines.hpp>
```

```
#include <ql/handle.hpp>
```

```
#include <ql/math/optimization/constraint.hpp>
```

```
#include <vector>
```

Include dependency graph for parameter.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Parameter**
Base class for model arguments.
- class **Parameter::Impl**
Base class for model parameter implementation.
- 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.

10.270 ql/models/shortrate/calibrationhelpers/caphelper.hpp File Reference

10.270.1 Detailed Description

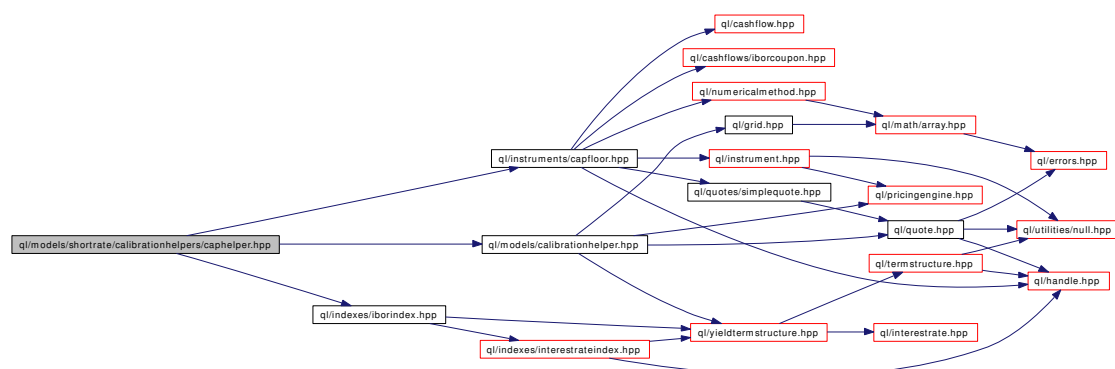
CapHelper calibration helper.

```
#include <ql/models/calibrationhelper.hpp>
```

```
#include <ql/instruments/capfloor.hpp>
```

```
#include <ql/indexes/iborindex.hpp>
```

Include dependency graph for caphelper.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CapHelper**
calibration helper for ATM cap

10.271 ql/models/shortrate/calibrationhelpers/swaptionhelper.hpp File Reference

10.271.1 Detailed Description

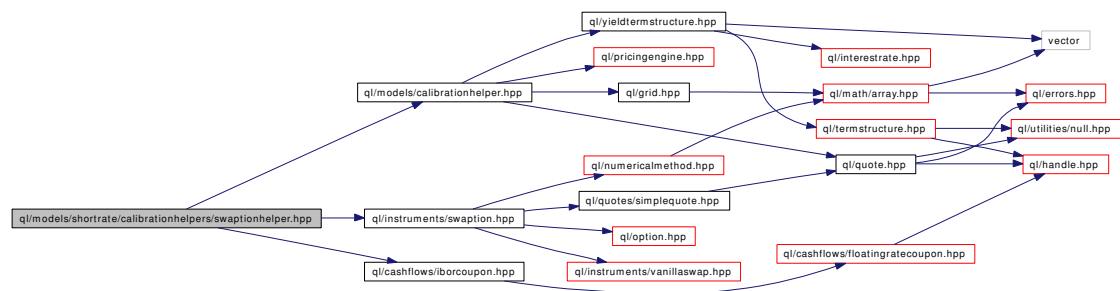
Swaption calibration helper.

```
#include <ql/models/calibrationhelper.hpp>
```

```
#include <ql/instruments/swaption.hpp>
```

```
#include <ql/cashflows/iborcoupon.hpp>
```

Include dependency graph for swaptionhelper.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **SwaptionHelper**
calibration helper for ATM swaption

10.272 ql/models/shortrate/onefactormodel.hpp File Reference

10.272.1 Detailed Description

Abstract one-factor interest rate model class.

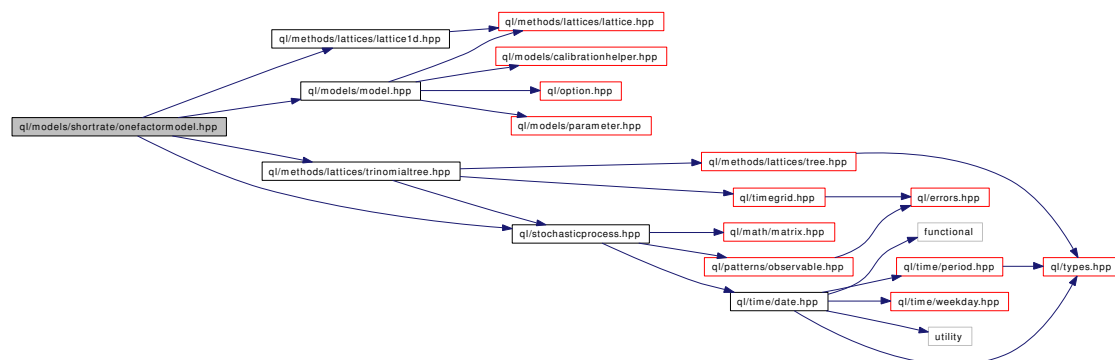
```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/models/model.hpp>
```

```
#include <ql/methods/lattices/lattice1d.hpp>
```

```
#include <ql/methods/lattices/trinomialtree.hpp>
```

Include dependency graph for onefactormodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [OneFactorModel](#)
Single-factor short-rate model abstract class.
- class [OneFactorModel::ShortRateDynamics](#)
Base class describing the short-rate dynamics.
- class [OneFactorModel::ShortRateTree](#)
Recombining trinomial tree discretizing the state variable.
- class [OneFactorAffineModel](#)
Single-factor affine base class.

10.273 ql/models/shortrate/onefactormodels/blackkarasinski.hpp File Reference

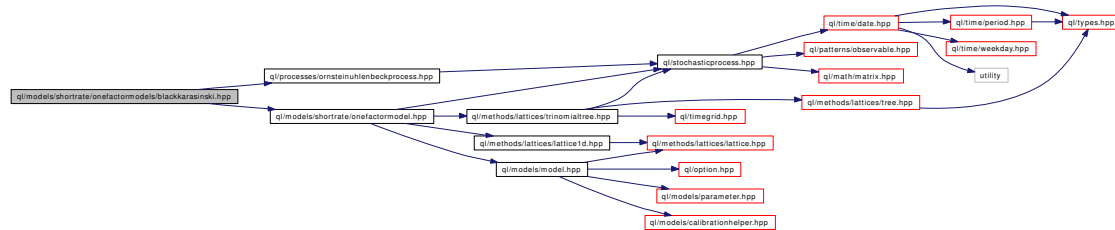
10.273.1 Detailed Description

Black-Karasinski model.

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

```
#include <ql/processes/ornsteinuhlenbeckprocess.hpp>
```

Include dependency graph for blackkarasinski.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackKarasinski**
Standard Black-Karasinski model class.
- class **BlackKarasinski::Dynamics**
Short-rate dynamics in the Black-Karasinski model.

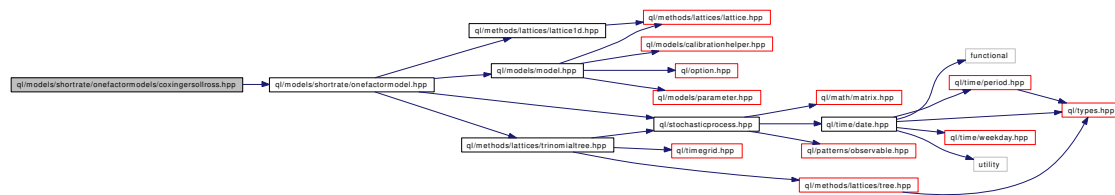
10.274 ql/models/shortrate/onefactormodels/coxingersollross.hpp File Reference

10.274.1 Detailed Description

Cox-Ingersoll-Ross model.

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

Include dependency graph for coxingersollross.hpp:



Namespaces

- namespace **QuantLib**

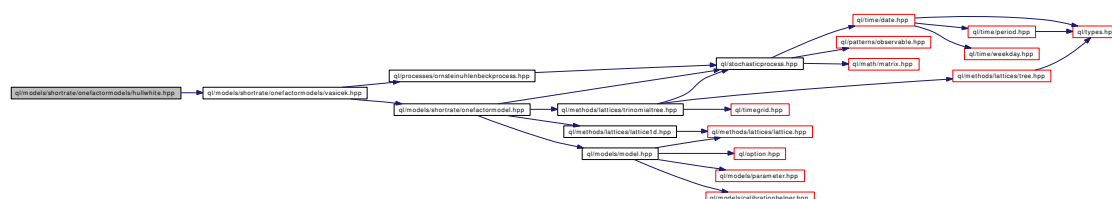
Classes

- class [CoxIngersollRoss](#)
Cox-Ingersoll-Ross model class.
- class [CoxIngersollRoss::Dynamics](#)
Dynamics of the short-rate under the Cox-Ingersoll-Ross model

10.276.1 Detailed Description

```
#include <ql/models/shortrate/onefactormodels/vasicek.hpp>
```

Include dependency graph for hullwhite.hpp:



- namespace **QuantLib**

- 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 $\varphi(t)$.

10.277 ql/models/shortrate/onefactormodels/vasicek.hpp File Reference

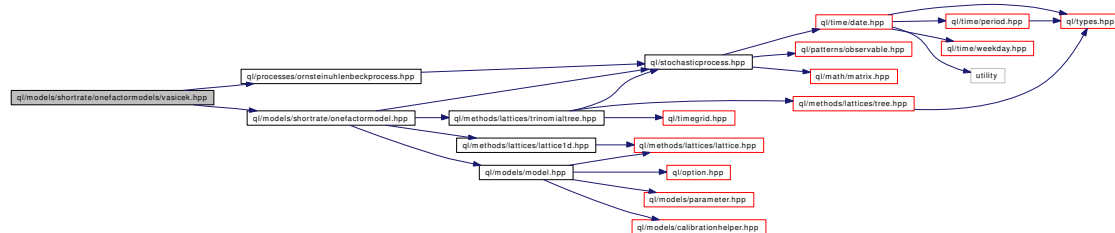
10.277.1 Detailed Description

Vasicek model class.

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

```
#include <ql/processes/ornsteinuhlenbeckprocess.hpp>
```

Include dependency graph for vasicek.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Vasicek**
Vasicek model class
- class **Vasicek::Dynamics**
Short-rate dynamics in the Vasicek model.

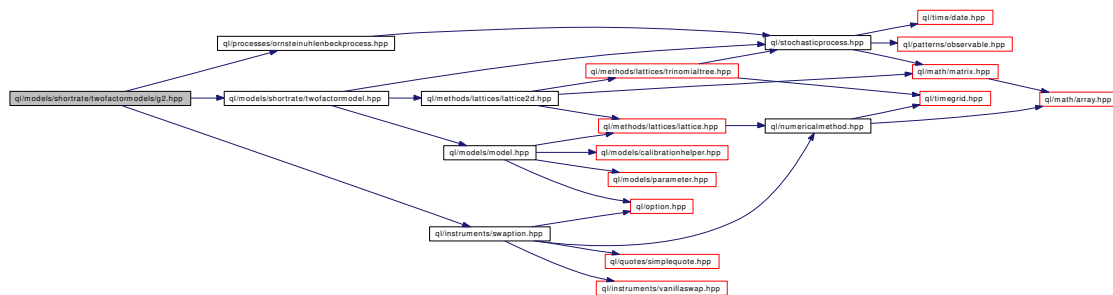
10.279 ql/models/shortrate/twofactormodels/g2.hpp File Reference

10.279.1 Detailed Description

Two-factor additive Gaussian Model G2++.

```
#include <ql/models/shortrate/twofactormodel.hpp>
#include <ql/processes/ornsteinuhlenbeckprocess.hpp>
#include <ql/instruments/swaption.hpp>
```

Include dependency graph for g2.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **G2**
Two-additive-factor gaussian model class.
- class **G2::FittingParameter**
Analytical term-structure fitting parameter $\varphi(t)$.

10.280 ql/models/volatility/constantestimator.hpp File Reference

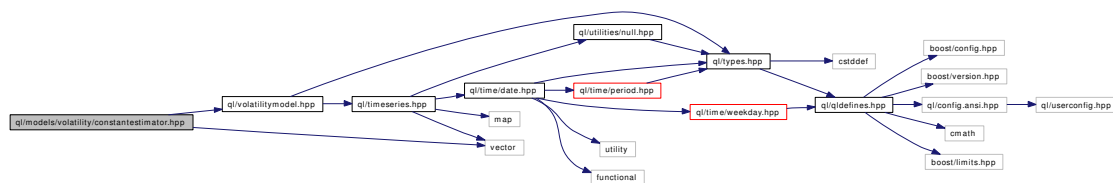
10.280.1 Detailed Description

Constant volatility estimator.

```
#include <ql/volatilitymodel.hpp>
```

```
#include <vector>
```

Include dependency graph for constantestimator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ConstantEstimator](#)
Constant-estimator volatility model.

10.281 ql/models/volatility/garch.hpp File Reference

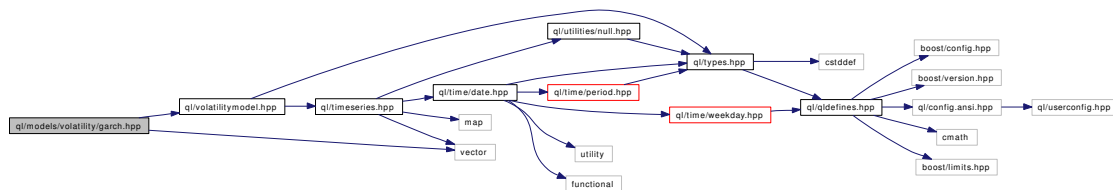
10.281.1 Detailed Description

GARCH volatility model.

```
#include <ql/volatilitymodel.hpp>
```

```
#include <vector>
```

Include dependency graph for garch.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Garch11](#)
GARCH volatility model.

10.282 ql/models/volatility/garmanklass.hpp File Reference

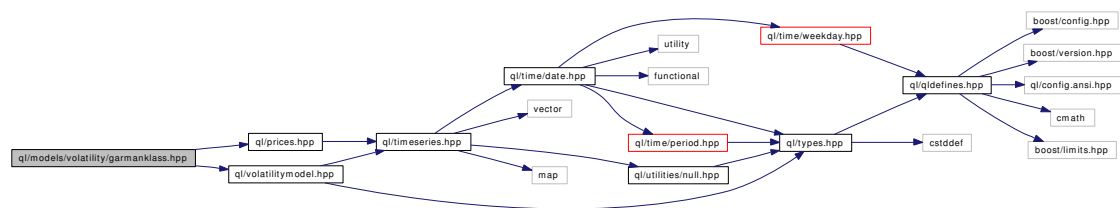
10.282.1 Detailed Description

Volatility estimators using high low data.

```
#include <ql/volatilitymodel.hpp>
```

```
#include <ql/prices.hpp>
```

Include dependency graph for garmanklass.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GarmanKlassAbstract](#)
Garman-Klass volatility model.

10.283 ql/models/volatility/simplelocalestimator.hpp File Reference

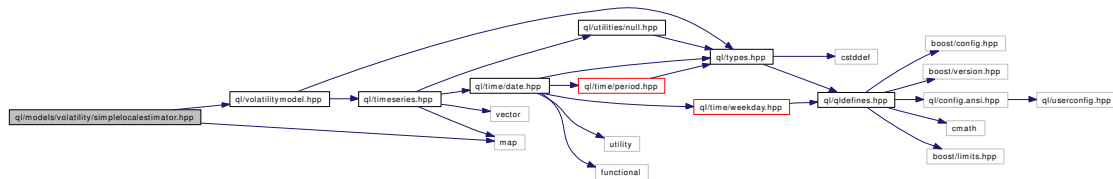
10.283.1 Detailed Description

Constant volatility estimator.

```
#include <ql/volatilitymodel.hpp>
```

```
#include <map>
```

Include dependency graph for simplelocalestimator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SimpleLocalEstimator](#)
Local-estimator volatility model.

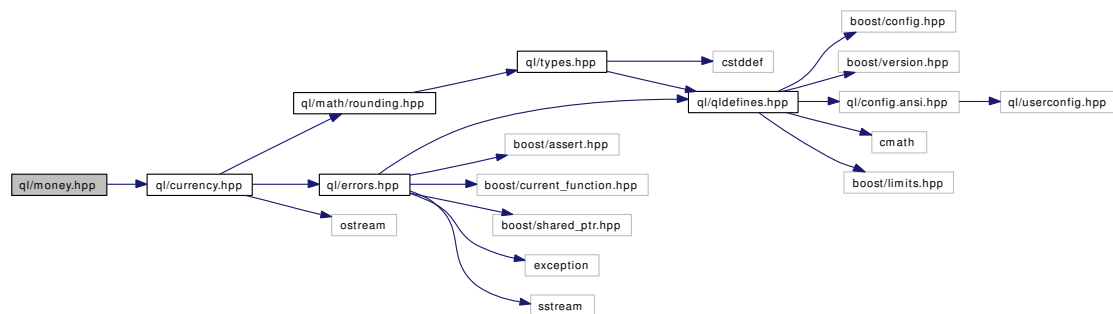
10.284 ql/money.hpp File Reference

10.284.1 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

10.285 ql/numericalmethod.hpp File Reference

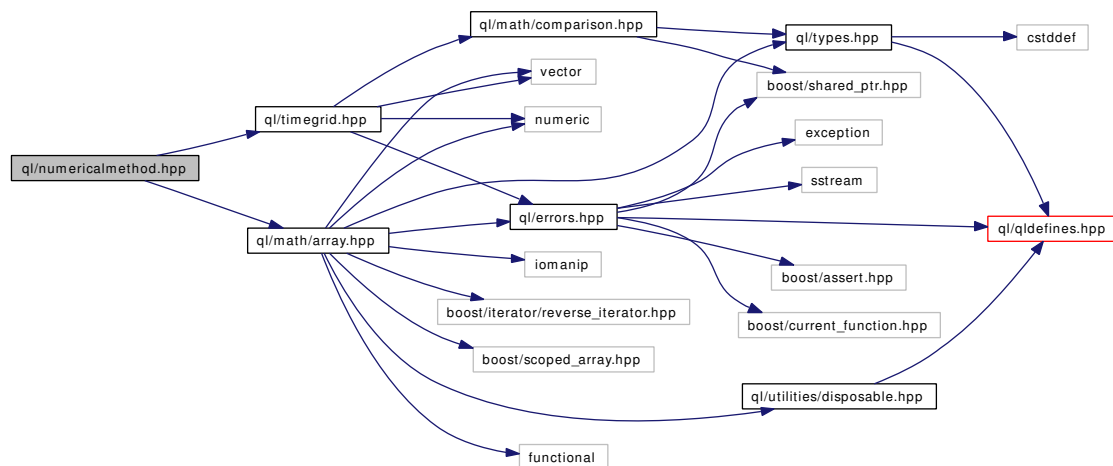
10.285.1 Detailed Description

Numerical method class.

```
#include <ql/timegrid.hpp>
```

```
#include <ql/math/array.hpp>
```

Include dependency graph for numericalmethod.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Lattice](#)
Lattice (tree, finite-differences) base class

10.286 ql/option.hpp File Reference

10.286.1 Detailed Description

Base option class.

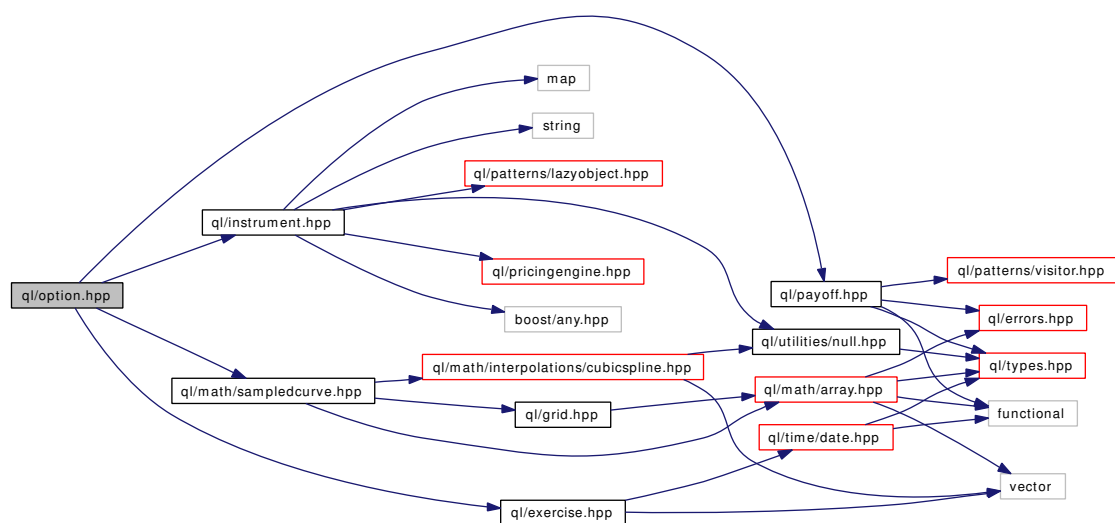
```
#include <ql/instrument.hpp>
```

```
#include <ql/payoff.hpp>
```

```
#include <ql/exercise.hpp>
```

```
#include <ql/math/sampledcurve.hpp>
```

Include dependency graph for option.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Option**
base option class
- class **Option::arguments**
basic option arguments
- class **Greeks**
additional option results
- class **MoreGreeks**
more additional option results

10.287 ql/patterns/composite.hpp File Reference

10.287.1 Detailed Description

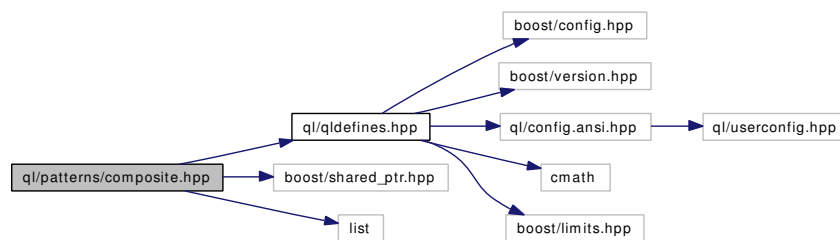
composite pattern

```
#include <ql/qldefines.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

```
#include <list>
```

Include dependency graph for composite.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Composite](#)
Composite pattern.

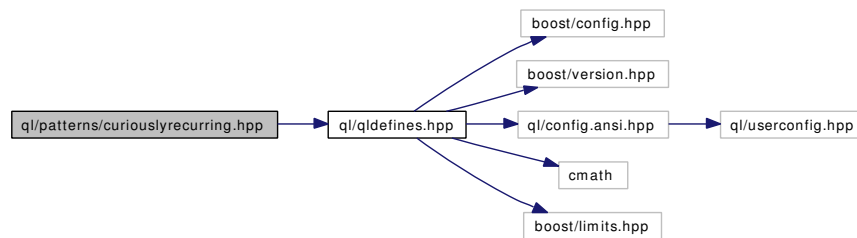
10.288 ql/patterns/curiouslyrecurring.hpp File Reference

10.288.1 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.

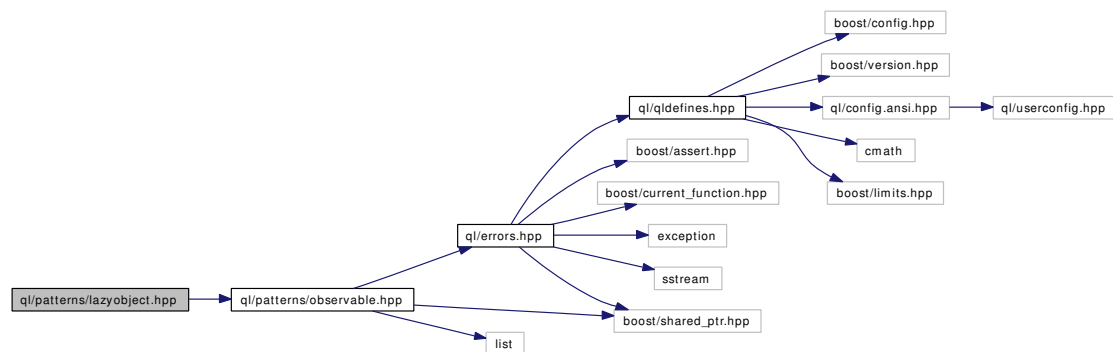
10.289 ql/patterns/lazyobject.hpp File Reference

10.289.1 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.

10.290 ql/patterns/observable.hpp File Reference

10.290.1 Detailed Description

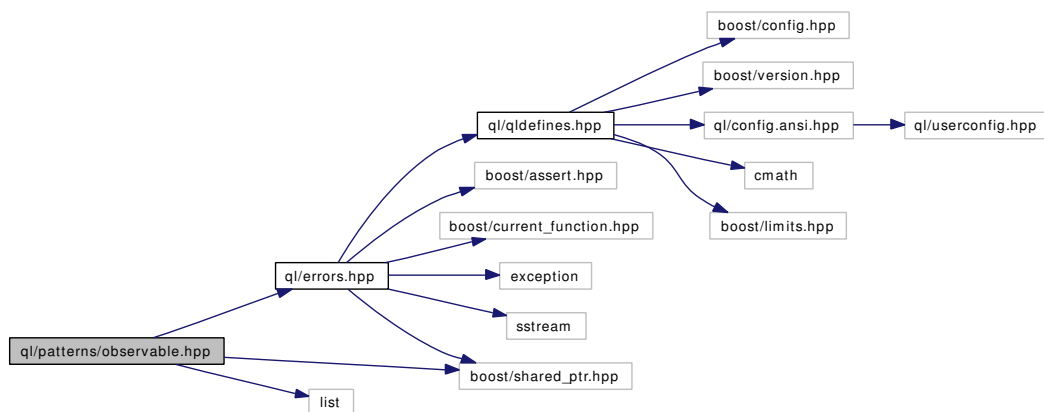
observer/observable pattern

```
#include <ql/errors.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

```
#include <list>
```

Include dependency graph for observable.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Observable**
Object that notifies its changes to a set of observables.
- class **Observer**
Object that gets notified when a given observable changes.

10.291 ql/patterns/singleton.hpp File Reference

10.291.1 Detailed Description

basic support for the singleton pattern

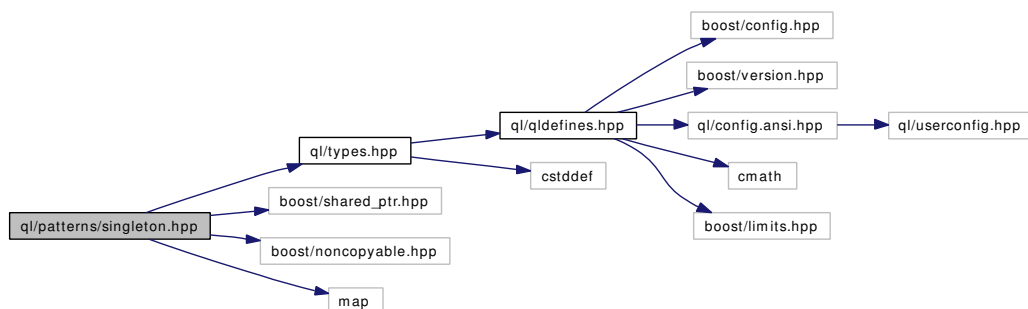
```
#include <ql/types.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

```
#include <boost/noncopyable.hpp>
```

```
#include <map>
```

Include dependency graph for singleton.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Singleton](#)
Basic support for the singleton pattern.

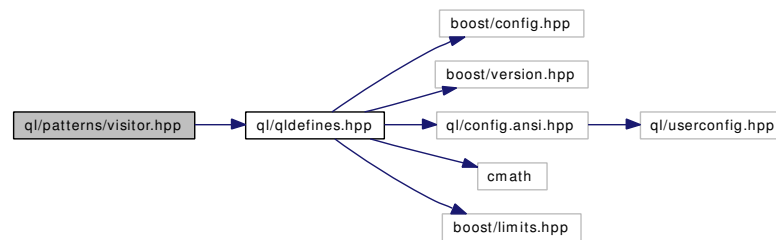
10.292 ql/patterns/visitor.hpp File Reference

10.292.1 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

10.293 ql/payoff.hpp File Reference

10.293.1 Detailed Description

Option payoff classes.

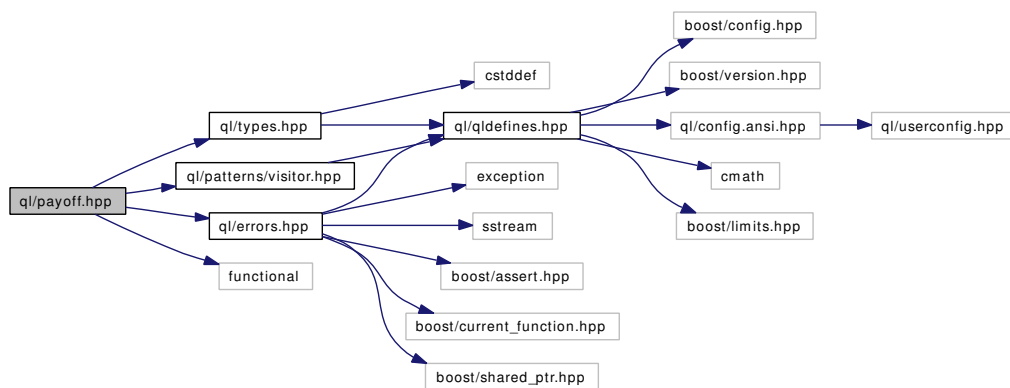
```
#include <ql/types.hpp>
```

```
#include <ql/patterns/visitor.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <functional>
```

Include dependency graph for payoff.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Payoff](#)
Abstract base class for option payoffs.

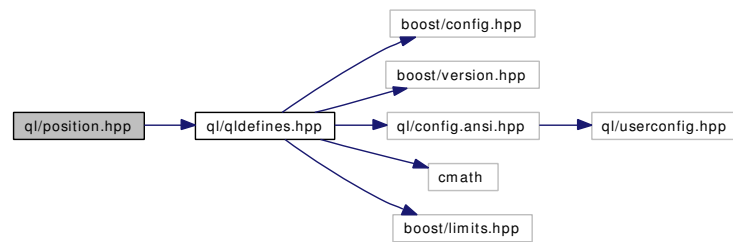
10.294 ql/position.hpp File Reference

10.294.1 Detailed Description

Short or long position.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for position.hpp:



Namespaces

- namespace **QuantLib**

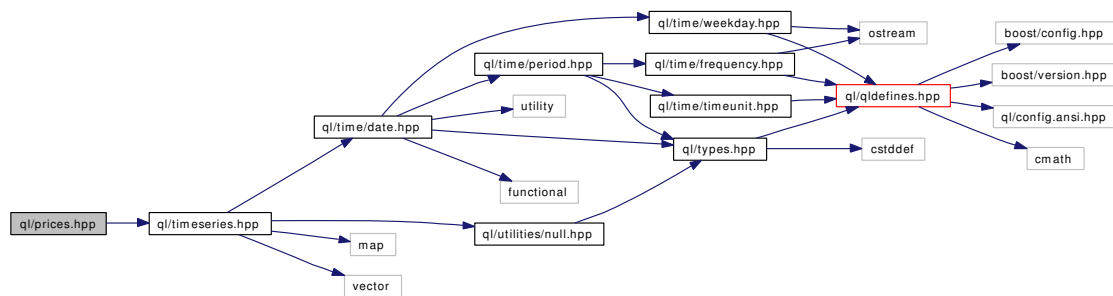
10.295 ql/prices.hpp File Reference

10.295.1 Detailed Description

price classes

```
#include <ql/timeseries.hpp>
```

Include dependency graph for prices.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [IntervalPrice](#)
interval price

Enumerations

- enum **PriceType** {
 [Bid](#), [Ask](#), [Last](#), [Close](#),
 [Mid](#), [MidEquivalent](#), [MidSafe](#) }
Price types.

Functions

- Real [midEquivalent](#) (const Real bid, const Real ask, const Real last, const Real close)
- Real [midSafe](#) (const Real bid, const Real ask)

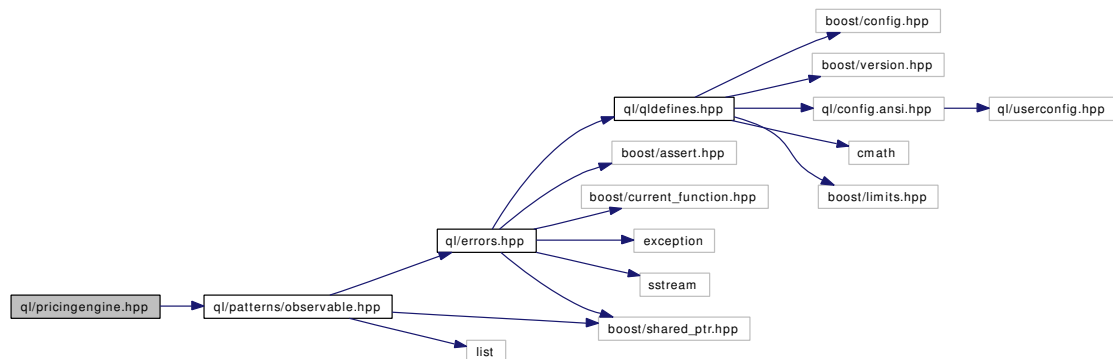
10.296 ql/pricingengine.hpp File Reference

10.296.1 Detailed Description

Base class for pricing engines.

```
#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

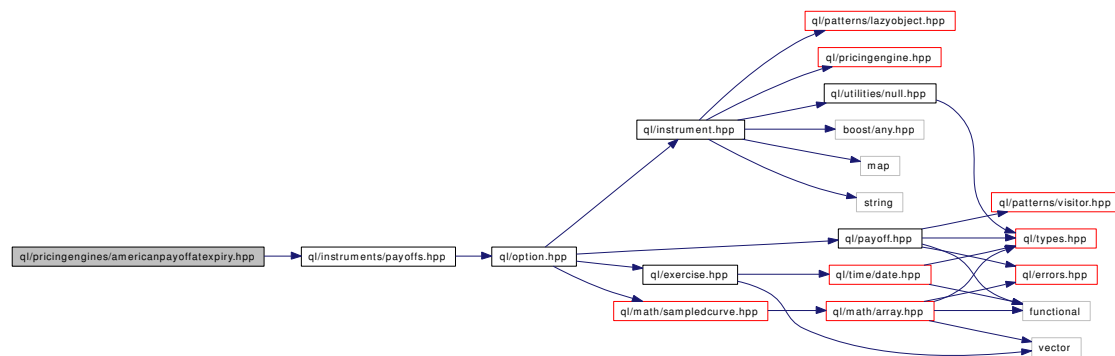
10.297 ql/pricingengines/americanpayoffatexpiry.hpp File Reference

10.297.1 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](#)

Analytic formula for American exercise payoff at-expiry options.

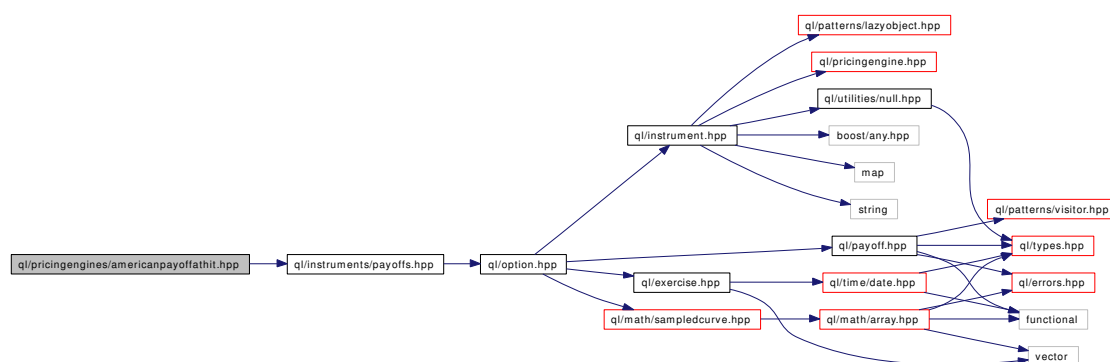
10.298 ql/pricingengines/americanpayoffathit.hpp File Reference

10.298.1 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](#)
Analytic formula for American exercise payoff at-hit options.

10.301 ql/pricingengines/asian/mc_discr_arith_av_price.hpp File Reference

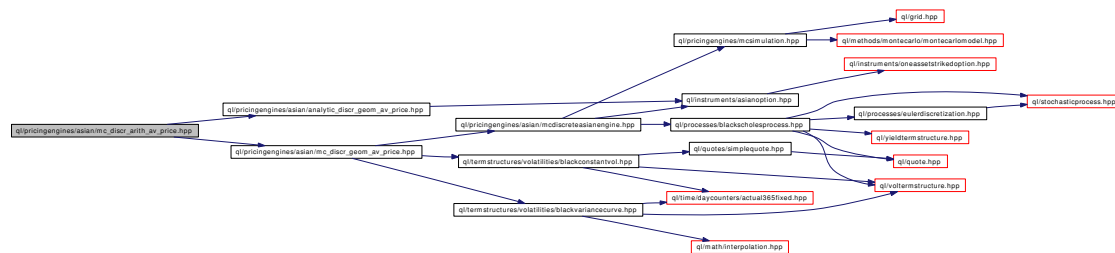
10.301.1 Detailed Description

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.

10.302.1 Detailed Description

```
#include <ql/pricingengines/asian/mcdiscreteasianengine.hpp>
#include <ql/termstructures/volatilities/blackconstantvol.hpp>
#include <ql/termstructures/volatilities/blackvariancecurve.hpp>
```

[illegible]

- namespace **QuantLib**

- class `MCDiscreteGeometricAPEngine`

Monte Carlo pricing engine for discrete geometric average price Asian.

10.303 ql/pricingengines/asian/mcdiscreteasianengine.hpp File Reference

10.303.1 Detailed Description

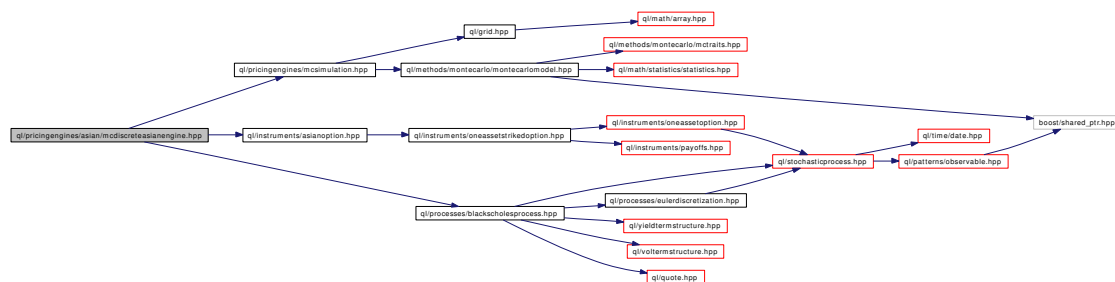
Monte Carlo pricing engine for discrete average Asians.

```
#include <ql/pricingengines/mcsimulation.hpp>
```

```
#include <ql/instruments/asianoption.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

Include dependency graph for mcdiscreteasianengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCDiscreteAveragingAsianEngine**
Pricing engine for discrete average Asians using Monte Carlo simulation.

10.304 ql/pricingengines/barrier/analyticbarrierengine.hpp File Reference

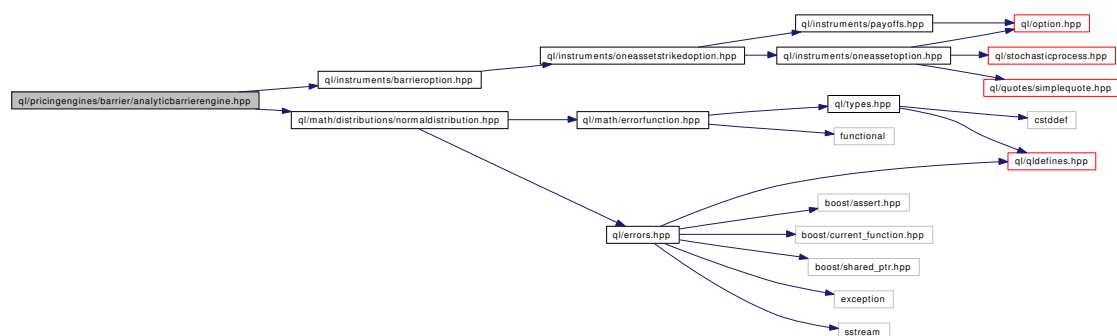
10.304.1 Detailed Description

Analytic barrier option engines.

```
#include <ql/instruments/barrieroption.hpp>
```

```
#include <ql/math/distributions/normaldistribution.hpp>
```

Include dependency graph for analyticbarrierengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticBarrierEngine](#)
Pricing engine for barrier options using analytical formulae.

10.305 ql/pricingengines/barrier/mcbarrierengine.hpp File Reference

10.305.1 Detailed Description

Monte Carlo barrier option engines.

```
#include <ql/instruments/barrieroption.hpp>
```

```
#include <ql/pricingengines/mcsimulation.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

Include dependency graph for mcbarrierengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCBarrierEngine**

Pricing engine for barrier options using Monte Carlo simulation.

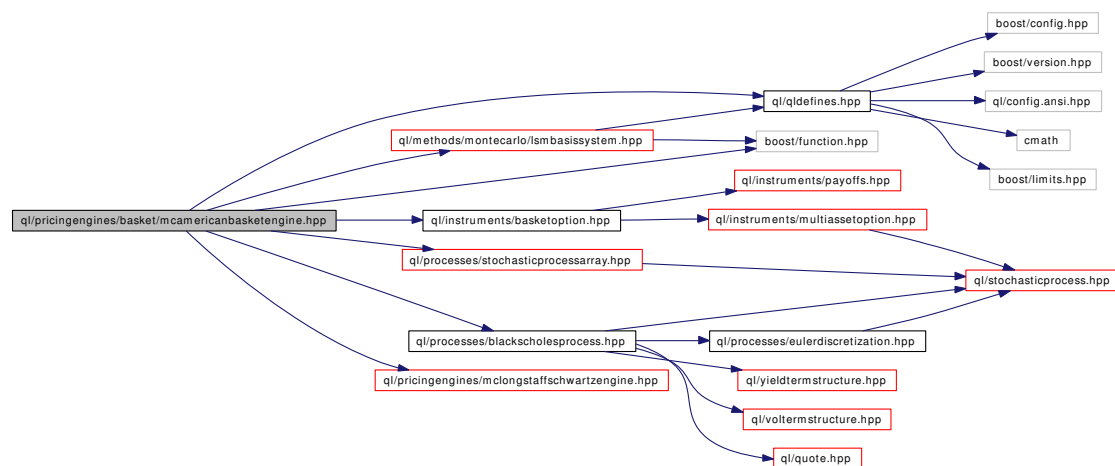
10.306 ql/pricingengines/basket/mcamericanbasketengine.hpp File Reference

10.306.1 Detailed Description

Least-square Monte Carlo engines.

```
#include <ql/qldefines.hpp>
#include <ql/instruments/basketoption.hpp>
#include <ql/processes/blackscholesprocess.hpp>
#include <ql/processes/stochasticprocessarray.hpp>
#include <ql/methods/montecarlo/lsmbasissystem.hpp>
#include <ql/pricingengines/mclongstaffschwartzengine.hpp>
#include <boost/function.hpp>
```

Include dependency graph for mcamericanbasketengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCAmericanBasketEngine**
least-square Monte Carlo engine

10.307 ql/pricingengines/basket/mcbasketengine.hpp File Reference

10.307.1 Detailed Description

European basket MC Engine.

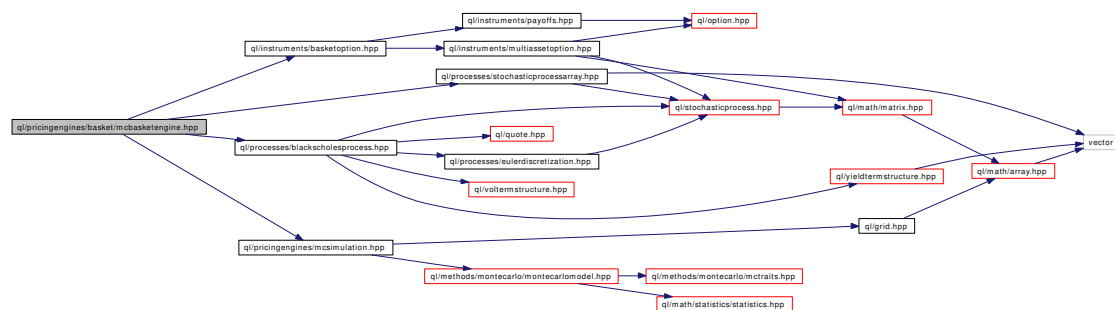
```
#include <ql/instruments/basketoption.hpp>
```

```
#include <ql/pricingengines/mcsimulation.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

```
#include <ql/processes/stochasticprocessarray.hpp>
```

Include dependency graph for mcbasketengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCBasketEngine**

Pricing engine for basket options using Monte Carlo simulation.

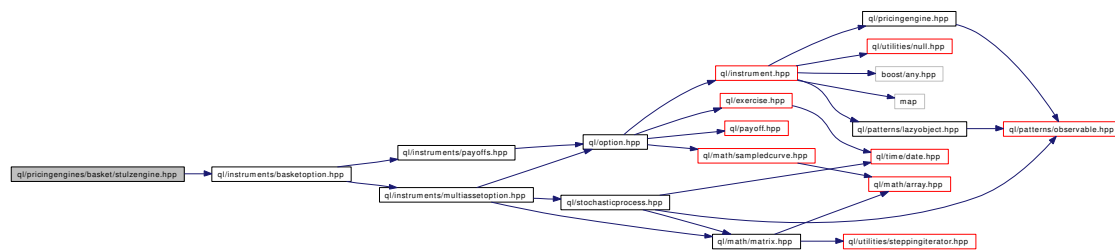
10.308 ql/pricingengines/basket/stulzengine.hpp File Reference

10.308.1 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.

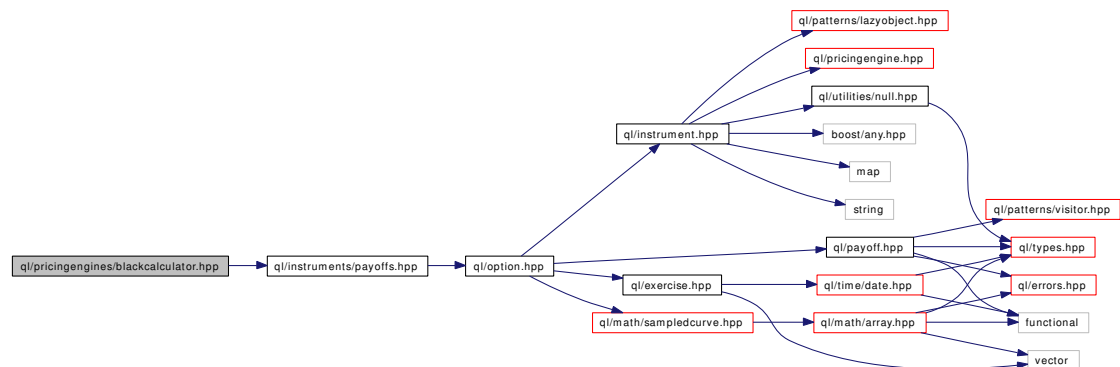
10.309 ql/pricingengines/blackcalculator.hpp File Reference

10.309.1 Detailed Description

Black-formula calculator class.

```
#include <ql/instruments/payoffs.hpp>
```

Include dependency graph for blackcalculator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackCalculator**
Black 1976 calculator class.

10.310 ql/pricingengines/blackformula.hpp File Reference

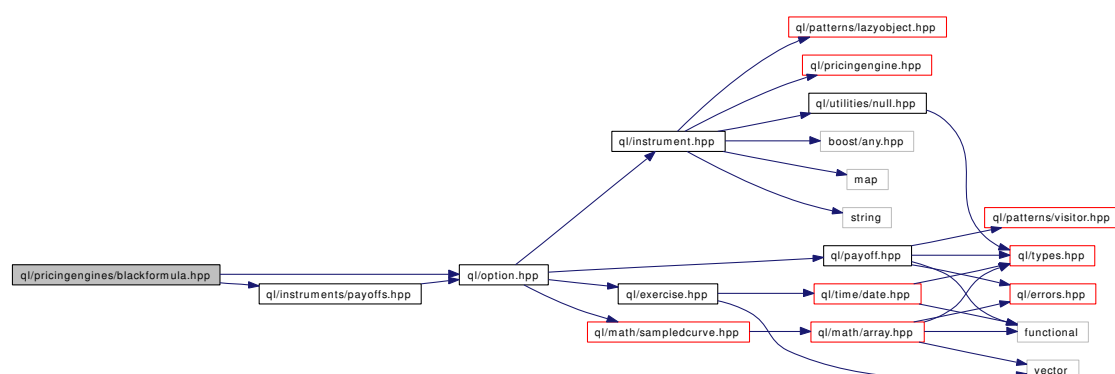
10.310.1 Detailed Description

Black formula.

```
#include <ql/option.hpp>
```

```
#include <ql/instruments/payoffs.hpp>
```

Include dependency graph for blackformula.hpp:



Namespaces

- namespace **QuantLib**

Functions

- Real [blackFormula](#) (Option::Type optionType, Real strike, Real forward, Real stdDev, Real discount=1.0, Real displacement=0.0)
- Real [blackFormula](#) (const boost::shared_ptr< PlainVanillaPayoff > &payoff, Real forward, Real stdDev, Real discount=1.0, Real displacement=0.0)
- Real [blackFormulaImpliedStdDevApproximation](#) (Option::Type optionType, Real strike, Real forward, Real blackPrice, Real discount=1.0, Real displacement=0.0)
- Real [blackFormulaImpliedStdDevApproximation](#) (const boost::shared_ptr< PlainVanillaPayoff > &payoff, Real forward, Real blackPrice, Real discount=1.0, Real displacement=0.0)
- Real [blackFormulaImpliedStdDev](#) (Option::Type optionType, Real strike, Real forward, Real blackPrice, Real discount=1.0, Real guess=Null< Real >(), Real accuracy=1.0e-6, Real displacement=0.0)
- Real [blackFormulaImpliedStdDev](#) (const boost::shared_ptr< PlainVanillaPayoff > &payoff, Real forward, Real blackPrice, Real discount=1.0, Real guess=Null< Real >(), Real accuracy=1.0e-6, Real displacement=0.0)
- Real [blackFormulaCashItmProbability](#) (Option::Type optionType, Real strike, Real forward, Real stdDev, Real displacement=0.0)
- Real [blackFormulaCashItmProbability](#) (const boost::shared_ptr< PlainVanillaPayoff > &payoff, Real forward, Real stdDev, Real displacement=0.0)
- Real [blackFormulaStdDevDerivative](#) (Real strike, Real forward, Real stdDev, Real discount=1.0, Real displacement=0.0)

- Real [blackFormulaStdDevDerivative](#) (const boost::shared_ptr< PlainVanillaPayoff > &payoff, Real forward, Real stdDev, Real discount=1.0, Real displacement=0.0)
- Real [bachelierBlackFormula](#) (Option::Type optionType, Real strike, Real forward, Real stdDev, Real discount=1.0)
- Real [bachelierBlackFormula](#) (const boost::shared_ptr< PlainVanillaPayoff > &payoff, Real forward, Real stdDev, Real discount=1.0)

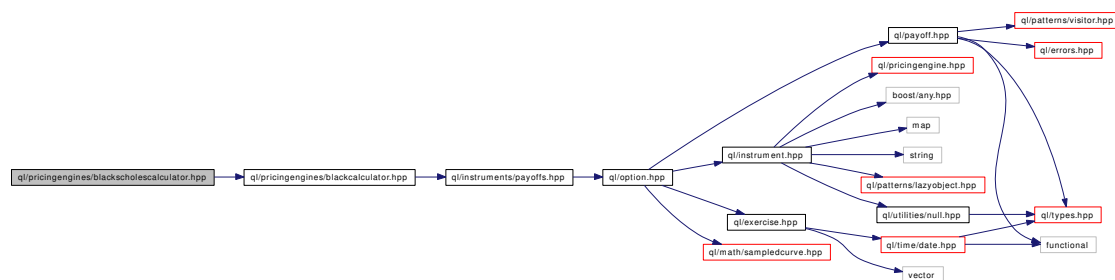
10.311 ql/pricingengines/blackscholescalculator.hpp File Reference

10.311.1 Detailed Description

Black-Scholes formula calculator class.

```
#include <ql/pricingengines/blackcalculator.hpp>
```

Include dependency graph for blackscholescalculator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackScholesCalculator**
Black-Scholes 1973 calculator class.

10.312 ql/pricingengines/capfloor/analyticcapfloorengine.hpp File Reference

10.312.1 Detailed Description

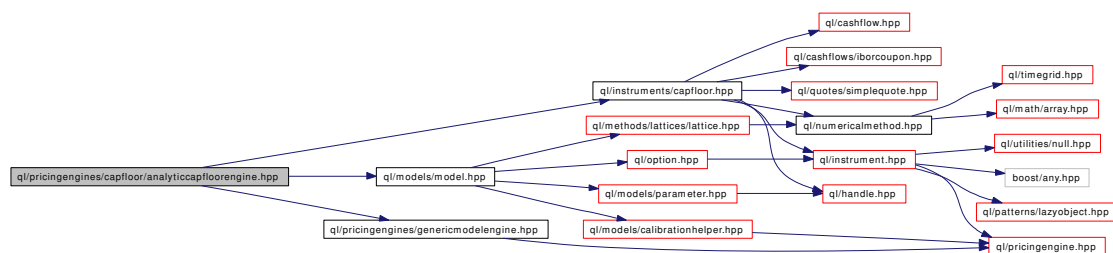
Analytic engine for caps/floors.

```
#include <ql/instruments/capfloor.hpp>
```

```
#include <ql/pricingengines/genericmodelengine.hpp>
```

```
#include <ql/models/model.hpp>
```

Include dependency graph for analyticcapfloorengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **AnalyticCapFloorEngine**
Analytic engine for cap/floor.

10.313 ql/pricingengines/capfloor/blackcapfloorengine.hpp File Reference

10.313.1 Detailed Description

Black-formula cap/floor engine.

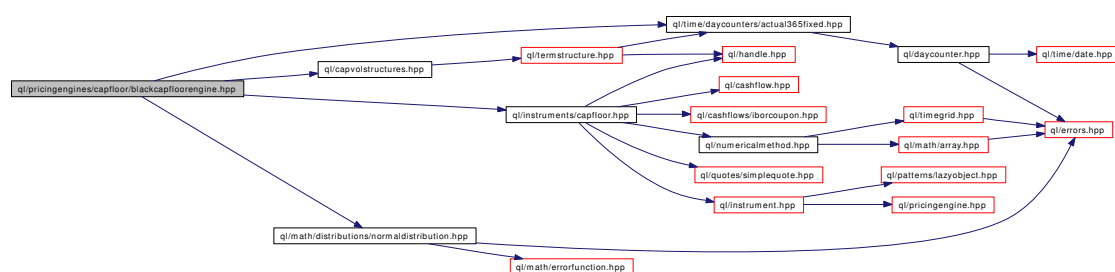
```
#include <ql/time/daycounters/actual365fixed.hpp>
```

```
#include <ql/instruments/capfloor.hpp>
```

```
#include <ql/math/distributions/normaldistribution.hpp>
```

```
#include <ql/capvolstructures.hpp>
```

Include dependency graph for blackcapfloorengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackCapFloorEngine**
Black-formula cap/floor engine.

10.314 ql/pricingengines/capfloor/discretizedcapfloor.hpp File Reference

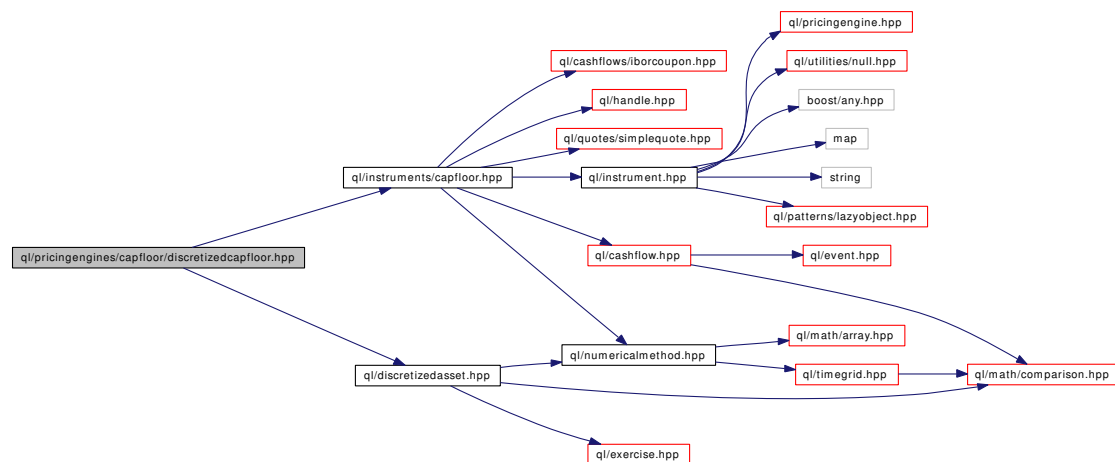
10.314.1 Detailed Description

discretized cap/floor

```
#include <ql/instruments/capfloor.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for discretizedcapfloor.hpp:



Namespaces

- namespace **QuantLib**

10.315 ql/pricingengines/capfloor/marketmodelcapfloorengine.hpp File Reference

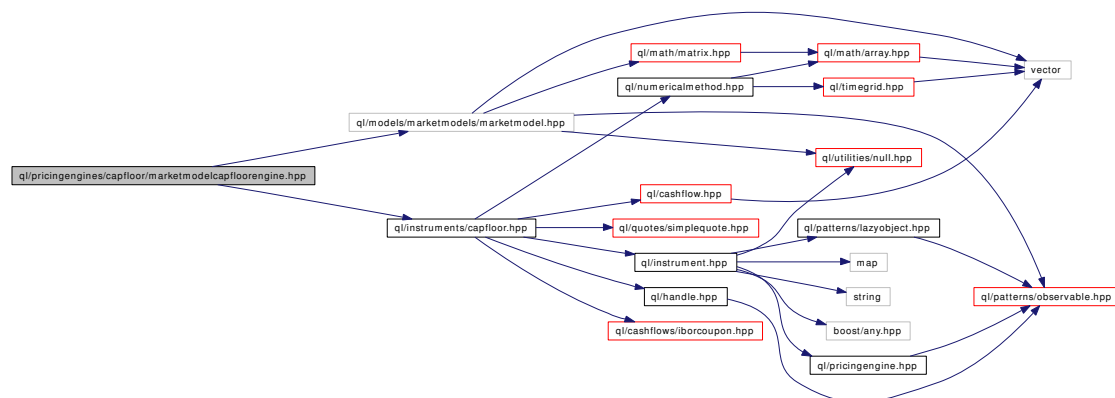
10.315.1 Detailed Description

Market-model cap/floor engine.

```
#include <ql/instruments/capfloor.hpp>
```

```
#include <ql/models/marketmodels/marketmodel.hpp>
```

Include dependency graph for marketmodelcapfloorengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MarketModelCapFloorEngine](#)
Market-model cap/floor engine.

10.316 ql/pricingengines/capfloor/mchullwhiteengine.hpp File Reference

10.316.1 Detailed Description

Monte Carlo Hull-White engine for cap/floors.

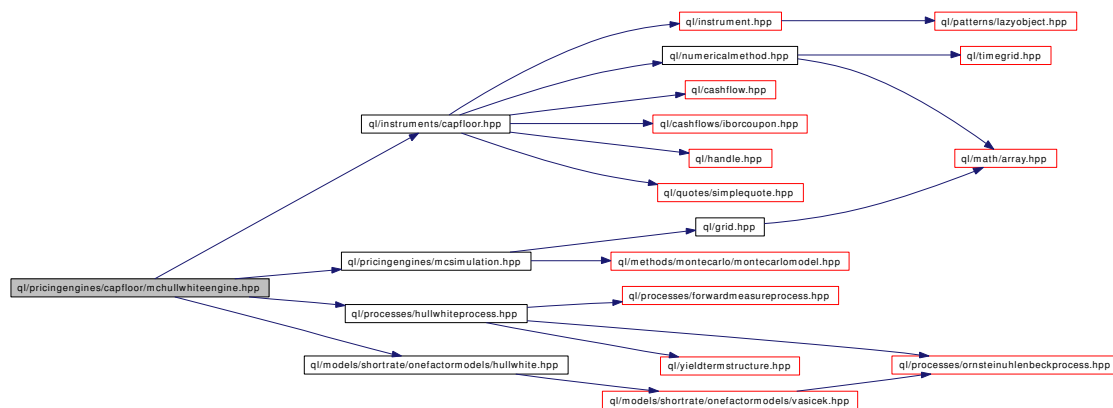
```
#include <ql/instruments/capfloor.hpp>
```

```
#include <ql/pricingengines/mcsimulation.hpp>
```

```
#include <ql/processes/hullwhiteprocess.hpp>
```

```
#include <ql/models/shortrate/onefactormodels/hullwhite.hpp>
```

Include dependency graph for mchullwhiteengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCHullWhiteCapFloorEngine**
Monte Carlo Hull-White engine for cap/floors.
- class **MakeMCHullWhiteCapFloorEngine**
Monte Carlo Hull-White cap-floor engine factory.

10.317 ql/pricingengines/capfloor/treecapfloorengine.hpp File Reference

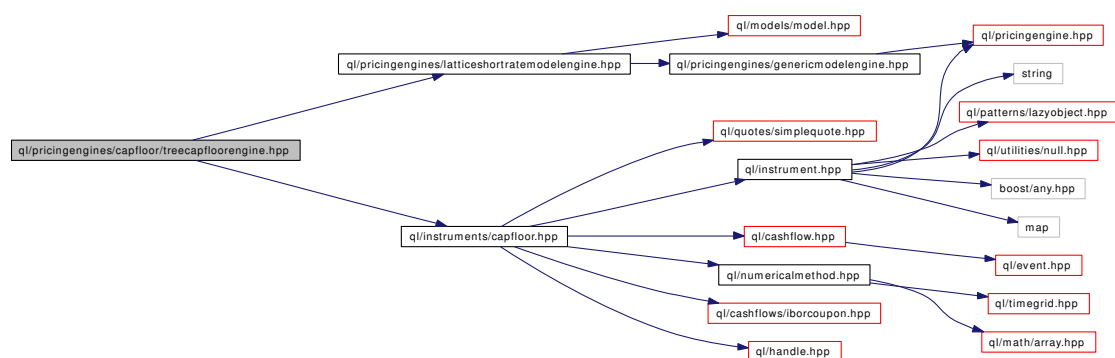
10.317.1 Detailed Description

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.

10.319.1 Detailed Description

```
#include <ql/instruments/cliquestoption.hpp>
```

[illegible]

- namespace **QuantLib**

- class `AnalyticPerformanceEngine`

Pricing engine for performance options using analytical formulae.

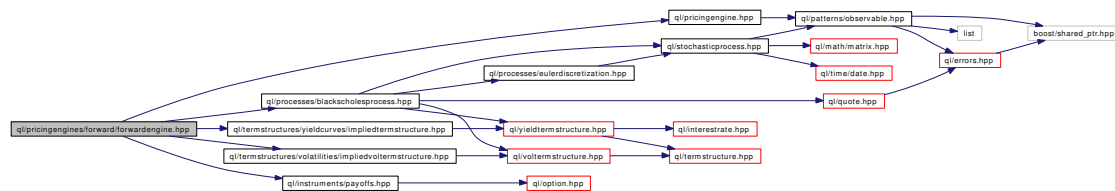
10.320 ql/pricingengines/forward/forwardengine.hpp File Reference

10.320.1 Detailed Description

Forward (strike-resetting) option engine.

```
#include <ql/pricingengine.hpp>
#include <ql/processes/blackscholesprocess.hpp>
#include <ql/termstructures/volatilities/IMPLIEDVOLTERMSTRUCTURE.hpp>
#include <ql/termstructures/yieldcurves/IMPLIEDTERMSTRUCTURE.hpp>
#include <ql/instruments/payoffs.hpp>
```

Include dependency graph for forwardengine.hpp:



Namespaces

- namespace **QuantLib**

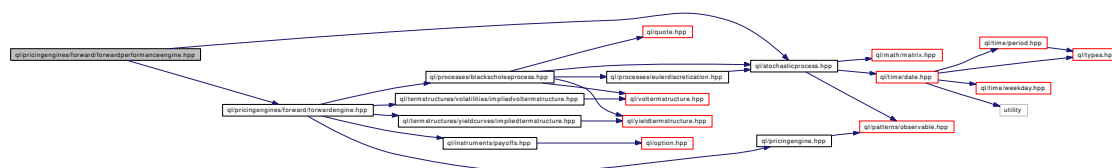
Classes

- class **ForwardOptionArguments**
Arguments for forward (strike-resetting) option calculation
- class **ForwardEngine**
Forward-engine base class

10.321.1 Detailed Description

```
#include <ql/pricingengines/forward/forwardengine.hpp>
```

Include dependency graph for forwardperformanceengine.hpp:



- namespace **QuantLib**

- class `ForwardPerformanceEngine`

Forward performance engine

10.322 ql/pricingengines/forward/mcvarianceswapengine.hpp File Reference

10.322.1 Detailed Description

Monte Carlo variance-swap engine.

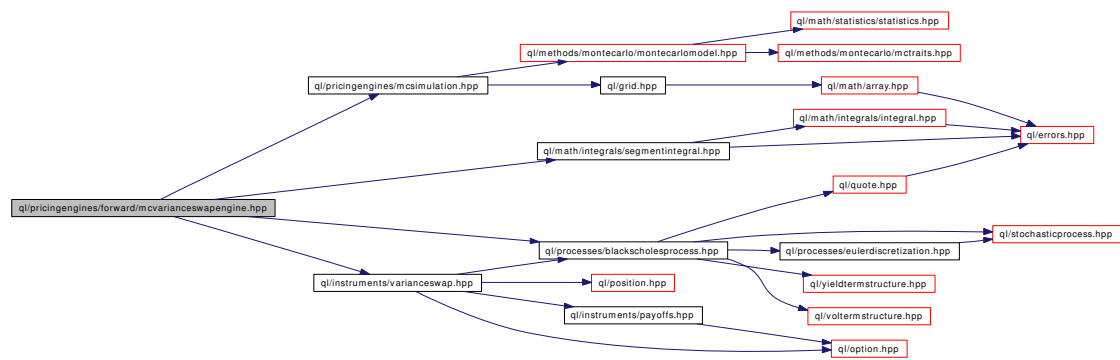
```
#include <ql/pricingengines/mcsimulation.hpp>
```

```
#include <ql/math/integrals/segmentintegral.hpp>
```

```
#include <ql/instruments/varianceswap.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

Include dependency graph for mcvarianceswapengine.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [MCVarianceSwapEngine](#)
Variance-swap pricing engine using Monte Carlo simulation.
- class [MakeMCVarianceSwapEngine](#)
Monte Carlo variance-swap engine factory.

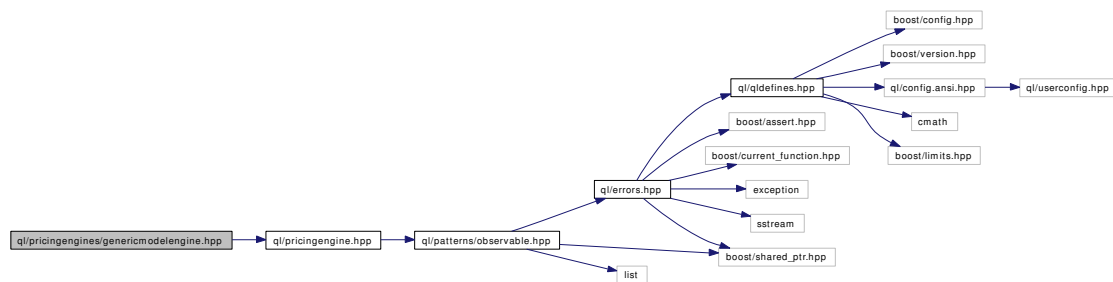
10.324 ql/pricingengines/genericmodelengine.hpp File Reference

10.324.1 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.

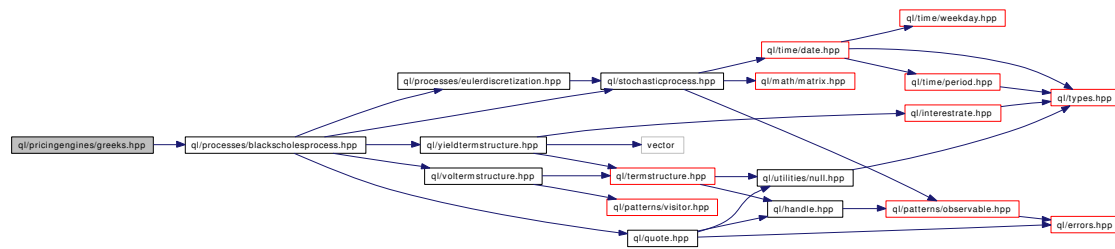
10.325 ql/pricingengines/greeks.hpp File Reference

10.325.1 Detailed Description

default greek calculations

```
#include <ql/processes/blackscholesprocess.hpp>
```

Include dependency graph for greeks.hpp:



Namespaces

- namespace **QuantLib**

Functions

- Real [blackScholesTheta](#) (const boost::shared_ptr< GeneralizedBlackScholesProcess > &, Real value, Real delta, Real gamma)
default theta calculation for Black-Scholes options
- Real [defaultThetaPerDay](#) (Real theta)
default theta-per-day calculation

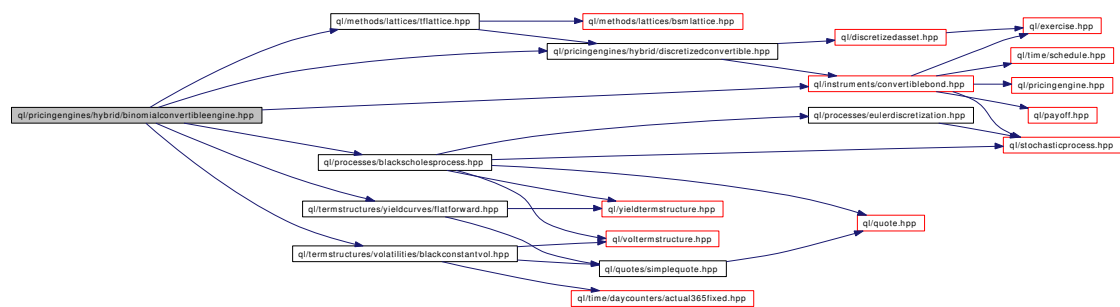
10.326 ql/pricingengines/hybrid/binomialconvertibleengine.hpp File Reference

10.326.1 Detailed Description

binomial engine for convertible bonds

```
#include <ql/methods/lattices/tflattice.hpp>
#include <ql/pricingengines/hybrid/discretizedconvertible.hpp>
#include <ql/processes/blackscholesprocess.hpp>
#include <ql/termstructures/yieldcurves/flatforward.hpp>
#include <ql/termstructures/volatilities/blackconstantvol.hpp>
#include <ql/instruments/convertiblebond.hpp>
```

Include dependency graph for binomialconvertibleengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BinomialConvertibleEngine**
Binomial Tsiveriotis-Fernandes engine for convertible bonds.

10.327 ql/pricingengines/hybrid/discretizedconvertible.hpp File Reference

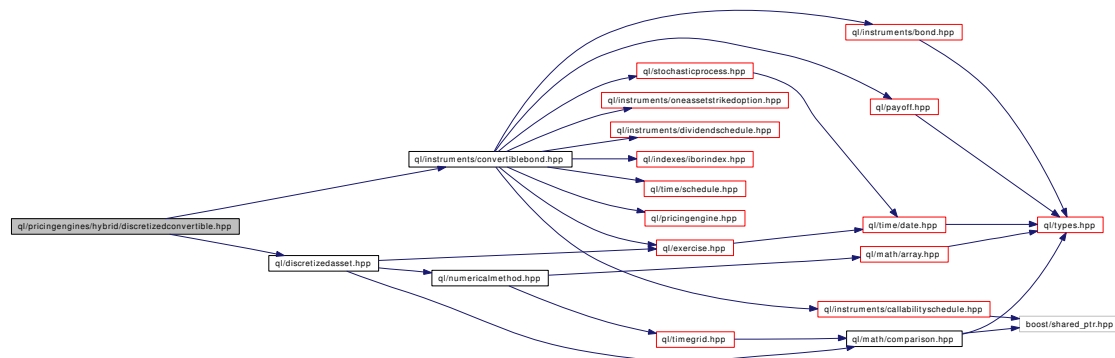
10.327.1 Detailed Description

discretized convertible

```
#include <ql/discretizedasset.hpp>
```

```
#include <ql/instruments/convertiblebond.hpp>
```

Include dependency graph for discretizedconvertible.hpp:



Namespaces

- namespace **QuantLib**

10.328 ql/pricingengines/latticeshortratemodelengine.hpp File Reference

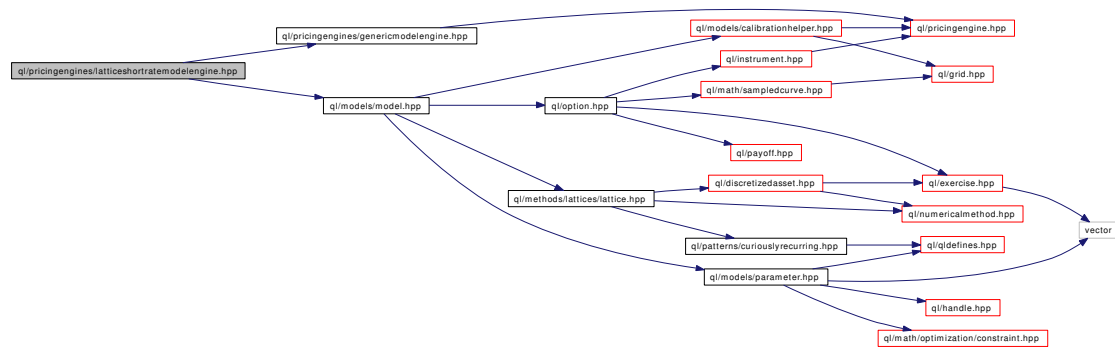
10.328.1 Detailed Description

Engine for a short-rate model specialized on a lattice.

```
#include <ql/models/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.

10.329 ql/pricingengines/lookback/analyticcontinuousfixedlookback.hpp File Reference

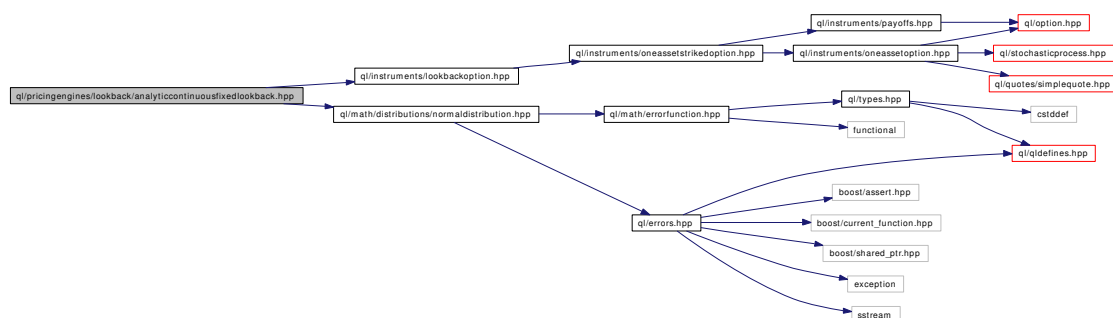
10.329.1 Detailed Description

Analytic engine for continuous fixed-strike lookback.

```
#include <ql/instruments/lookbackoption.hpp>
```

```
#include <ql/math/distributions/normaldistribution.hpp>
```

Include dependency graph for analyticcontinuousfixedlookback.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticContinuousFixedLookbackEngine](#)
Pricing engine for European continuous fixed-strike lookback.

10.330 ql/pricingengines/lookback/analyticcontinuousfloatinglookback.hpp File Reference

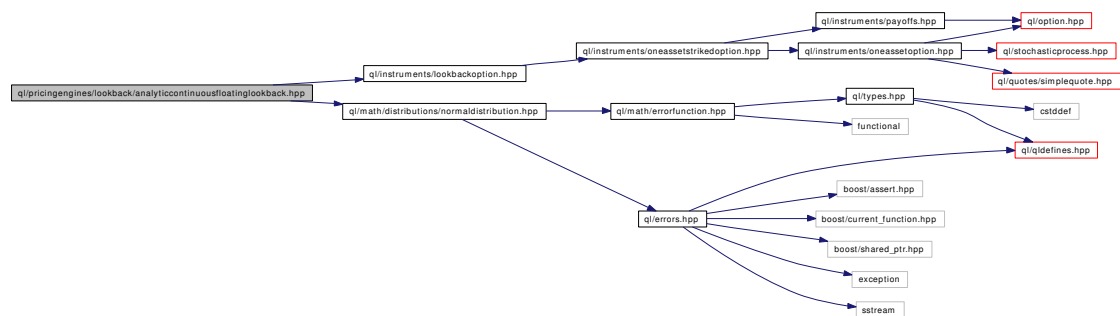
10.330.1 Detailed Description

Analytic engine for continuous floating-strike lookback.

```
#include <ql/instruments/lookbackoption.hpp>
```

```
#include <ql/math/distributions/normaldistribution.hpp>
```

Include dependency graph for analyticcontinuousfloatinglookback.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticContinuousFloatingLookbackEngine](#)
Pricing engine for European continuous floating-strike lookback.

10.331 ql/pricingengines/mclongstaffschwartzengine.hpp File Reference

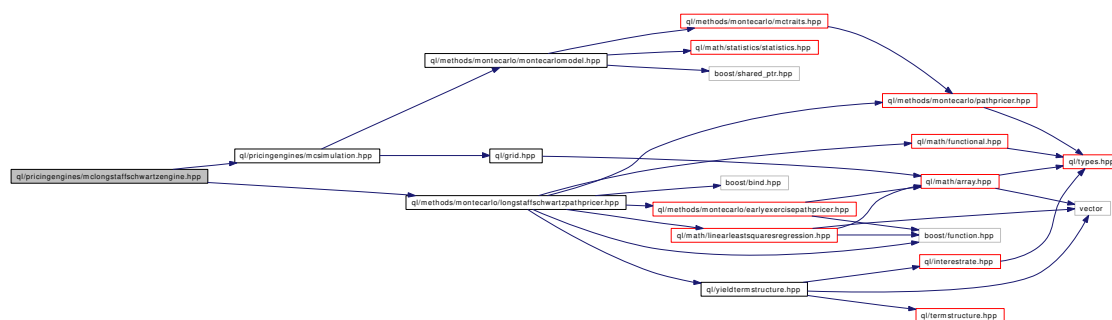
10.331.1 Detailed Description

Longstaff Schwartz Monte Carlo engine for early exercise options.

```
#include <ql/pricingengines/mcsimulation.hpp>
```

```
#include <ql/methods/montecarlo/longstaffschwartzpathpricer.hpp>
```

Include dependency graph for mclongstaffschwartzengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCLongstaffSchwartzEngine**
Longstaff-Schwarz Monte Carlo engine for early exercise options.

10.332 ql/pricingengines/mcsimulation.hpp File Reference

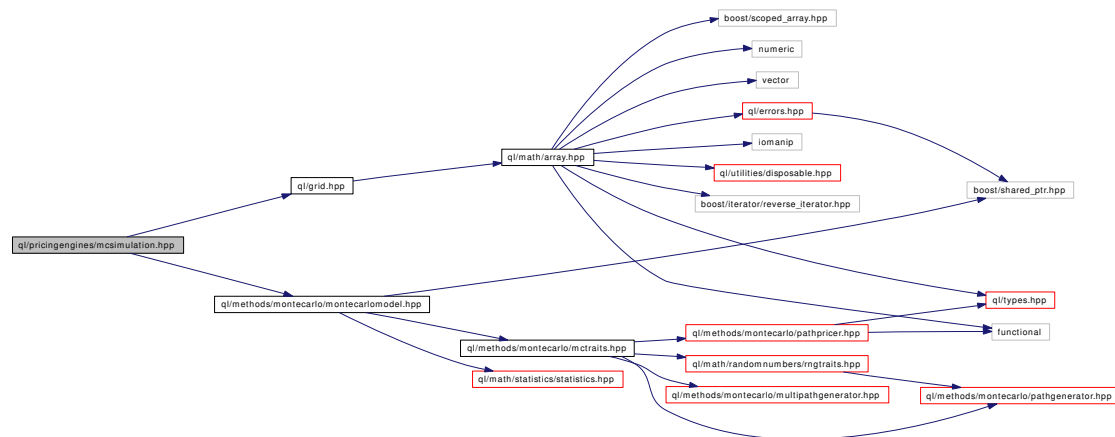
10.332.1 Detailed Description

framework for Monte Carlo engines

```
#include <ql/grid.hpp>
```

```
#include <ql/methods/montecarlo/montecarломodel.hpp>
```

Include dependency graph for mcsimulation.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [McSimulation](#)
base class for Monte Carlo engines

10.333 ql/pricingengines/quanto/quantoengine.hpp File Reference

10.333.1 Detailed Description

Quanto option engine.

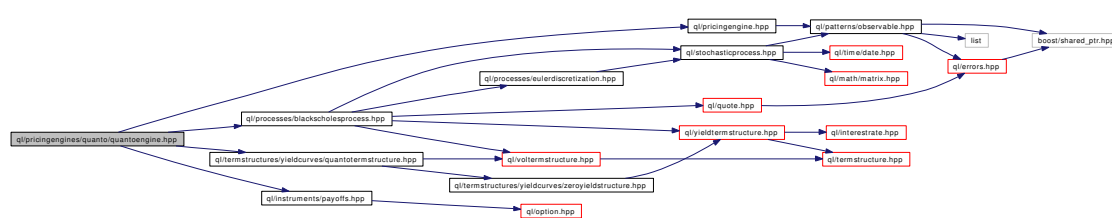
```
#include <ql/pricingengine.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

```
#include <ql/termstructures/yieldcurves/quantotermstructure.hpp>
```

```
#include <ql/instruments/payoffs.hpp>
```

Include dependency graph for quantoengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **QuantoOptionArguments**
Arguments for quanto option calculation
- class **QuantoOptionResults**
Results from quanto option calculation
- class **QuantoEngine**
Quanto engine base class.

10.334 ql/pricingengines/swaption/blackswaptionengine.hpp File Reference

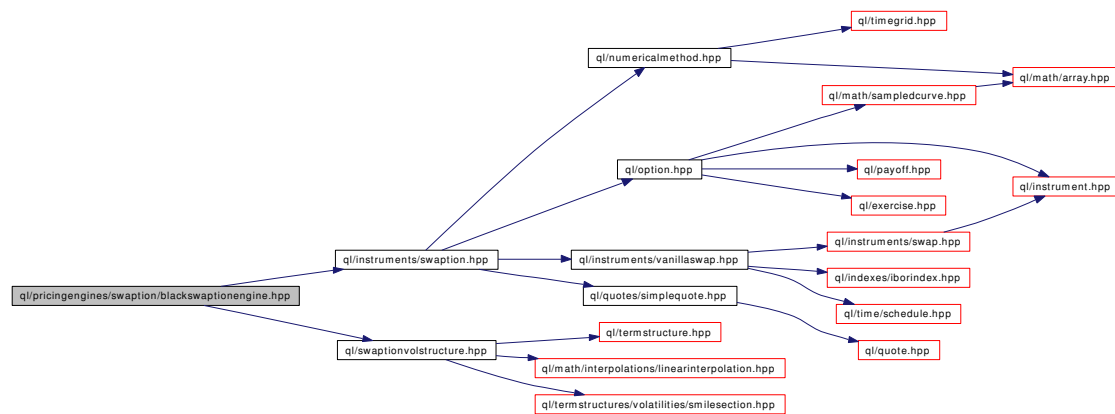
10.334.1 Detailed Description

Black-formula swaption engine.

```
#include <ql/instruments/swaption.hpp>
```

```
#include <ql/swaptionvolstructure.hpp>
```

Include dependency graph for blackswaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackSwaptionEngine**
Black-formula swaption engine.

10.335 ql/pricingengines/swaption/discretizedswaption.hpp File Reference

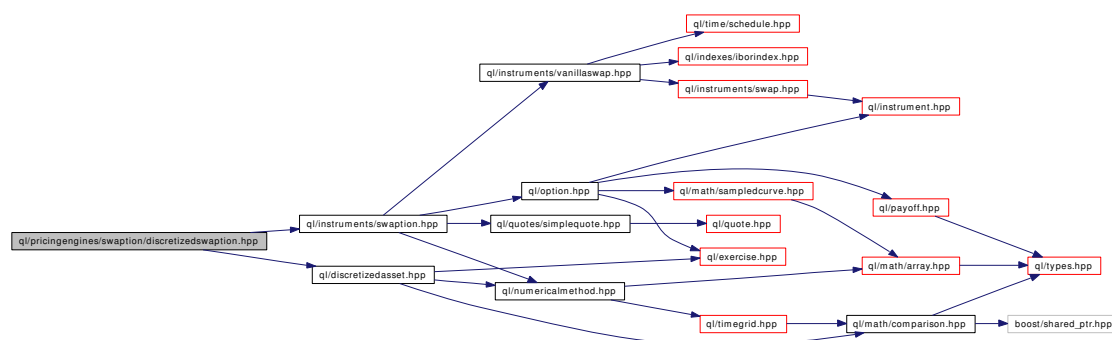
10.335.1 Detailed Description

Discretized swaption class.

```
#include <ql/instruments/swaption.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for discretizedswaption.hpp:



Namespaces

- namespace QuantLib

10.336 ql/pricingengines/swaption/g2swaptionengine.hpp File Reference

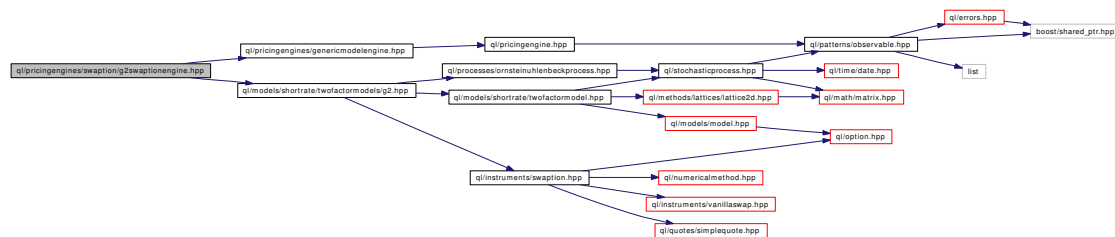
10.336.1 Detailed Description

Swaption pricing engine for two-factor additive Gaussian Model G2++.

```
#include <ql/pricingengines/genericmodelengine.hpp>
```

```
#include <ql/models/shortrate/twofactormodels/g2.hpp>
```

Include dependency graph for g2swaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **G2SwaptionEngine**
Swaption priced by means of the Black formula

10.337 ql/pricingengines/swaption/jamshidianswaptionengine.hpp File Reference

10.337.1 Detailed Description

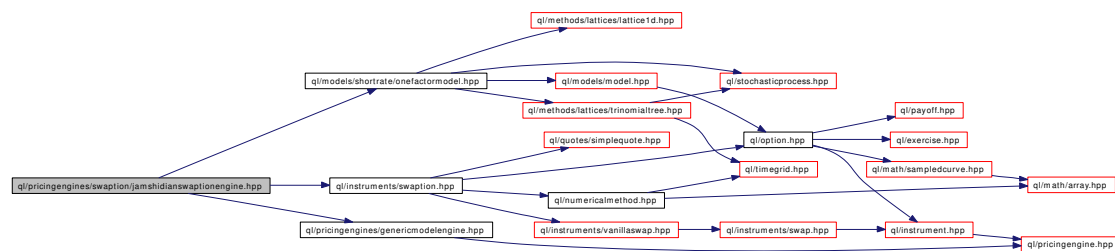
Swaption engine using Jamshidian's decomposition.

```
#include <ql/instruments/swaption.hpp>
```

```
#include <ql/models/shortrate/onefactormodel.hpp>
```

```
#include <ql/pricingengines/genericmodelengine.hpp>
```

Include dependency graph for jamshidianswaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **JamshidianSwaptionEngine**
Jamshidian swaption engine.

10.338 ql/pricingengines/swaption/lfmswaptionengine.hpp File Reference

10.338.1 Detailed Description

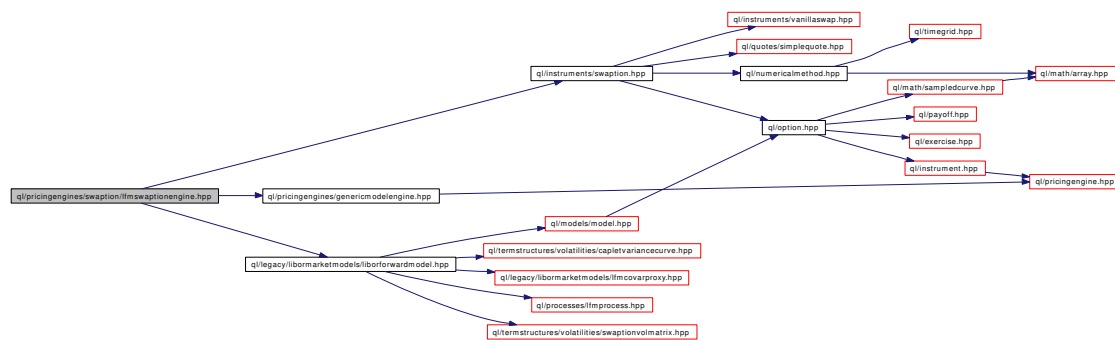
libor forward model swaption engine based on black formula

```
#include <ql/instruments/swaption.hpp>
```

```
#include <ql/pricingengines/genericmodelengine.hpp>
```

```
#include <ql/legacy/libormarketmodels/liborforwardmodel.hpp>
```

Include dependency graph for lfmswaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **LfmSwaptionEngine**

Libor forward model swaption engine based on Black formula

10.339 ql/pricingengines/swaption/treeswaptionengine.hpp File Reference

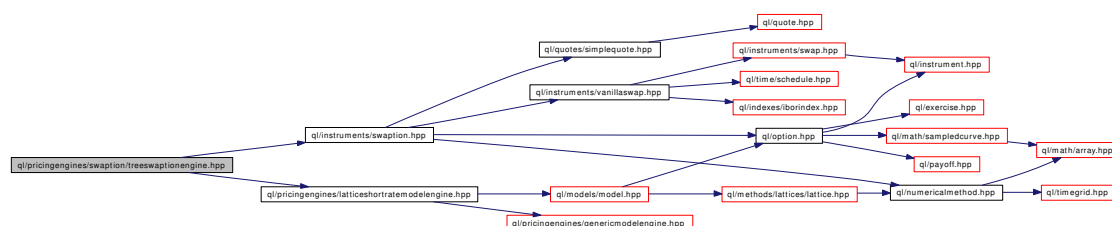
10.339.1 Detailed Description

Numerical lattice engines for swaps and swaptions.

```
#include <ql/instruments/swaption.hpp>
```

```
#include <ql/pricingengines/latticeshortratemodelengine.hpp>
```

Include dependency graph for treeswaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TreeVanillaSwapEngine**
Numerical lattice engine for simple swaps.
- class **TreeSwaptionEngine**
Numerical lattice engine for swaptions.

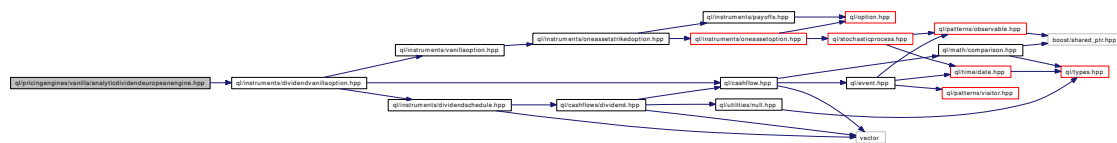
10.341 ql/pricingengines/vanilla/analyticdividendeuropeanengine.hpp
File Reference

10.341.1 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.

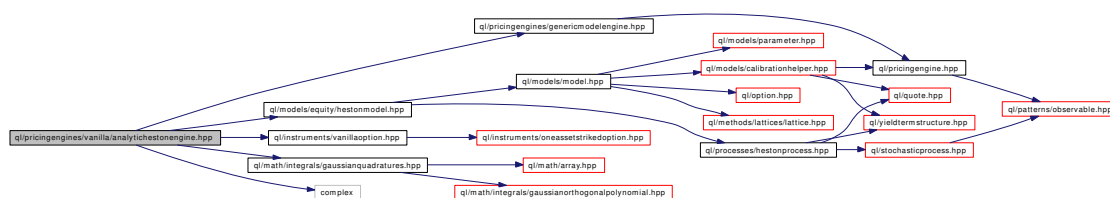
10.343 ql/pricingengines/vanilla/analytichestonengine.hpp File Reference

10.343.1 Detailed Description

analytic Heston-model engine

```
#include <ql/pricingengines/genericmodelengine.hpp>
#include <ql/models/equity/hestonmodel.hpp>
#include <ql/instruments/vanillaoption.hpp>
#include <ql/math/integrals/gaussianquadratures.hpp>
#include <complex>
```

Include dependency graph for analytichestonengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticHestonEngine](#)
analytic Heston-model engine based on Fourier transform

10.345 ql/pricingengines/vanilla/batesengine.hpp File Reference

10.345.1 Detailed Description

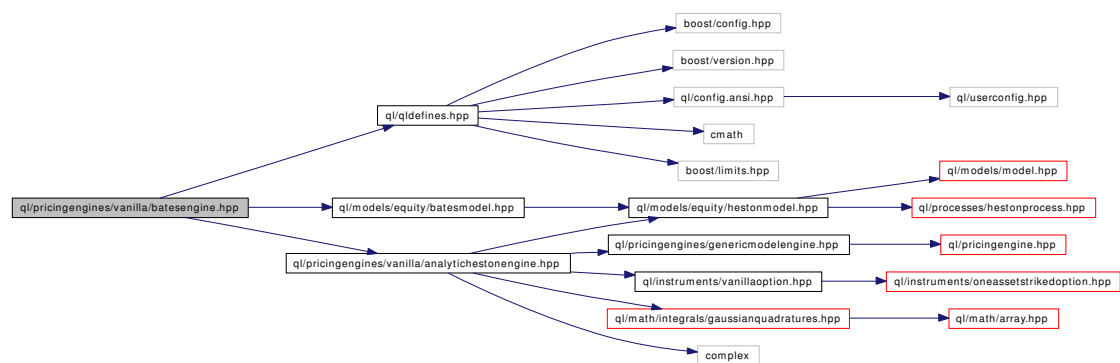
analytic Bates model engine

```
#include <ql/qldefines.hpp>
```

```
#include <ql/models/equity/batesmodel.hpp>
```

```
#include <ql/pricingengines/vanilla/analytichestonengine.hpp>
```

Include dependency graph for batesengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BatesEngine](#)
Bates model engines based on Fourier transform.

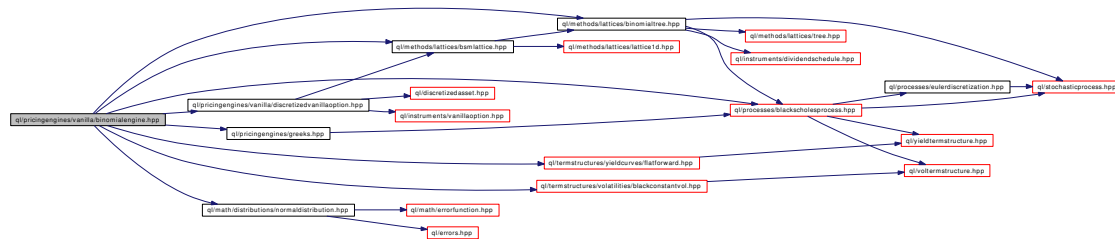
10.346 ql/pricingengines/vanilla/binomialengine.hpp File Reference

10.346.1 Detailed Description

Binomial option engine.

```
#include <ql/methods/lattices/binomialtree.hpp>
#include <ql/methods/lattices/bsmlattice.hpp>
#include <ql/math/distributions/normaldistribution.hpp>
#include <ql/pricingengines/vanilla/discretizedvanillaoption.hpp>
#include <ql/pricingengines/greeks.hpp>
#include <ql/processes/blackscholesprocess.hpp>
#include <ql/termstructures/yieldcurves/flatforward.hpp>
#include <ql/termstructures/volatilities/blackconstantvol.hpp>
```

Include dependency graph for binomialengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BinomialVanillaEngine**
Pricing engine for vanilla options using binomial trees.

10.347.1 Detailed Description

```
#include <ql/instruments/vanillaoption.hpp>
```

[illegible]

- namespace **QuantLib**

- class `BjersundStenslandApproximationEngine`
Bjersund and Stensland pricing engine for American options (1993).

10.348 ql/pricingengines/vanilla/discretizedvanillaoption.hpp

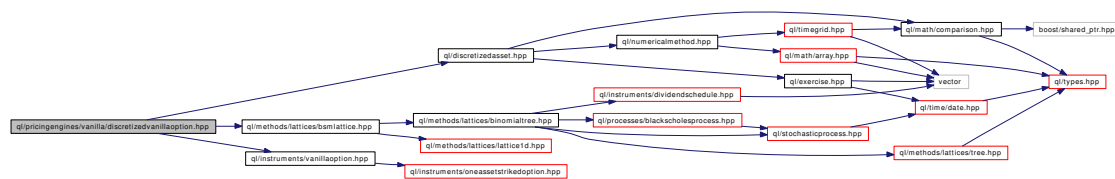
File Reference

10.348.1 Detailed Description

discretized vanilla option

```
#include <ql/discretizedasset.hpp>
#include <ql/methods/lattices/bsmllattice.hpp>
#include <ql/instruments/vanillaoption.hpp>
```

Include dependency graph for discretizedvanillaoption.hpp:



Namespaces

- namespace QuantLib

10.349 ql/pricingengines/vanilla/fdamericanengine.hpp File Reference

10.349.1 Detailed Description

Finite-differences American option engine.

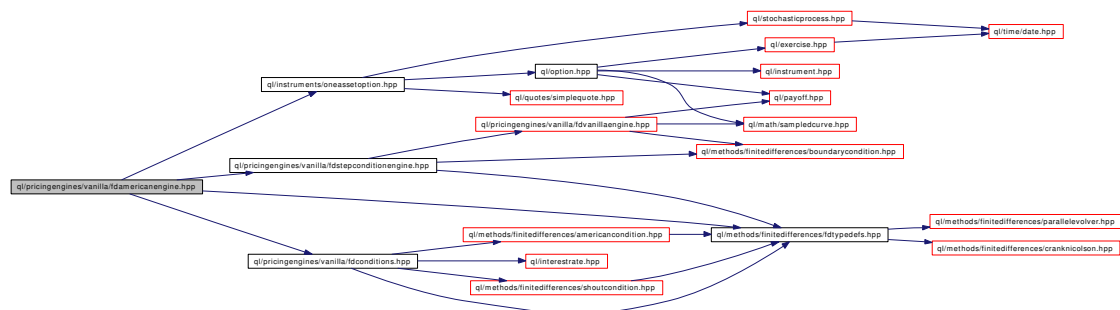
```
#include <ql/instruments/oneassetoption.hpp>
```

```
#include <ql/pricingengines/vanilla/fdstepconditionengine.hpp>
```

```
#include <ql/pricingengines/vanilla/fdconditions.hpp>
```

```
#include <ql/methods/finitedifferences/fdtypedefs.hpp>
```

Include dependency graph for fdamericanengine.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef FDEngineAdapter< FDAmericanCondition< FDStepConditionEngine >, OneAssetOption::engine > **FDAmericanEngine**

Finite-differences pricing engine for American one asset options.

10.350 ql/pricingengines/vanilla/fdbermudanengine.hpp File Reference

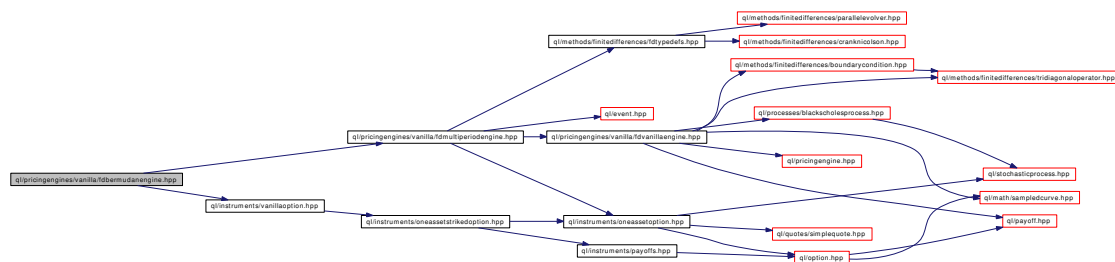
10.350.1 Detailed Description

finite-difference Bermudan engine

```
#include <ql/instruments/vanillaoption.hpp>
```

```
#include <ql/pricingengines/vanilla/fdmultipleriodengine.hpp>
```

Include dependency graph for fdbermudanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FDBermudanEngine**
Finite-differences Bermudan engine.

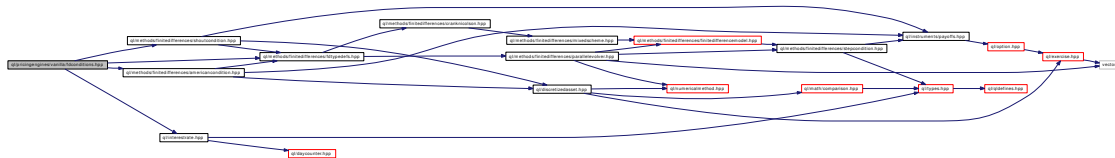
10.351 ql/pricingengines/vanilla/fdconditions.hpp File Reference

10.351.1 Detailed Description

Finite-difference templates to generate engines.

```
#include <ql/methods/finitedifferences/fdtypedefs.hpp>
#include <ql/methods/finitedifferences/americancondition.hpp>
#include <ql/methods/finitedifferences/shoutcondition.hpp>
#include <ql/interestrate.hpp>
```

Include dependency graph for fdconditions.hpp:



Namespaces

- namespace **QuantLib**

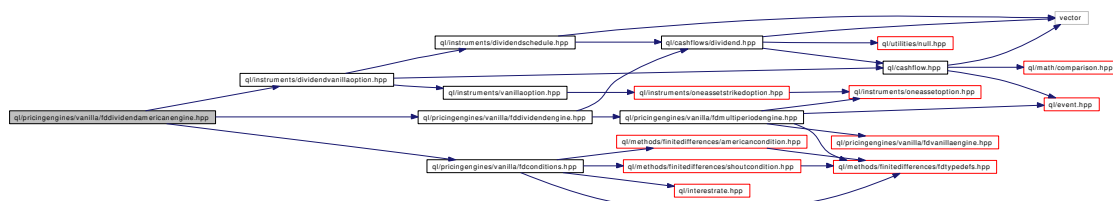
10.352 ql/pricingengines/vanilla/fddividendamericanengine.hpp File Reference

10.352.1 Detailed Description

american engine with discrete deterministic dividends

```
#include <ql/instruments/dividendvanillaoption.hpp>
#include <ql/pricingengines/vanilla/fddividendengine.hpp>
#include <ql/pricingengines/vanilla/fdconditions.hpp>
```

Include dependency graph for fddividendamericanengine.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef FDEngineAdapter< FDAmericanCondition< FDDividendEngine >, DividendVanillaOption::engine > **FDDividendAmericanEngine**
Finite-differences pricing engine for dividend American options.
- typedef FDEngineAdapter< FDAmericanCondition< FDDividendEngineMerton73 >, DividendVanillaOption::engine > **FDDividendAmericanEngineMerton73**
- typedef FDEngineAdapter< FDAmericanCondition< FDDividendEngineShiftScale >, DividendVanillaOption::engine > **FDDividendAmericanEngineShiftScale**

10.353	ql/pricingengines/vanilla/fddividendengine.hpp	File Reference
--------	--	----------------

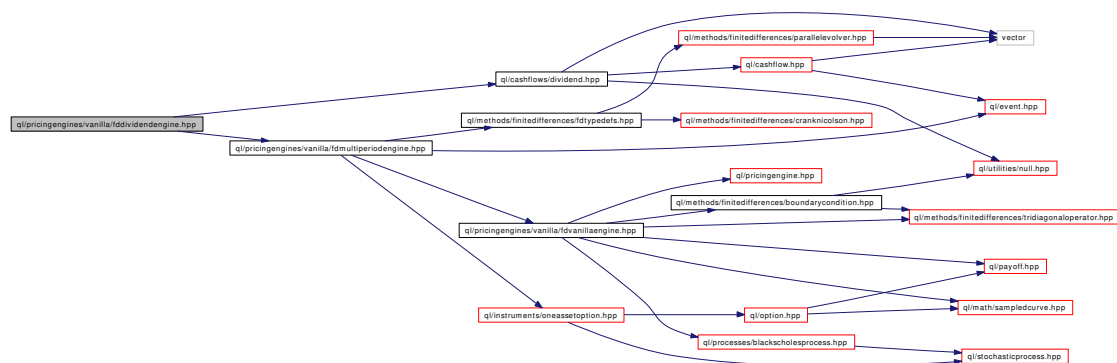
10.353.1 Detailed Description

base engine for option with dividends

```
#include <ql/pricingengines/vanilla/fdmultiengine.hpp>
```

```
#include <ql/cashflows/dividend.hpp>
```

Include dependency graph for fddividengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class `FDDividendEngineMerton73`
Finite-differences pricing engine for dividend options using.
- class `FDDividendEngineShiftScale`
Finite-differences pricing engine for dividend options using.

Typedefs

- `typedef FDDividendEngineMerton73 FDDividendEngine`

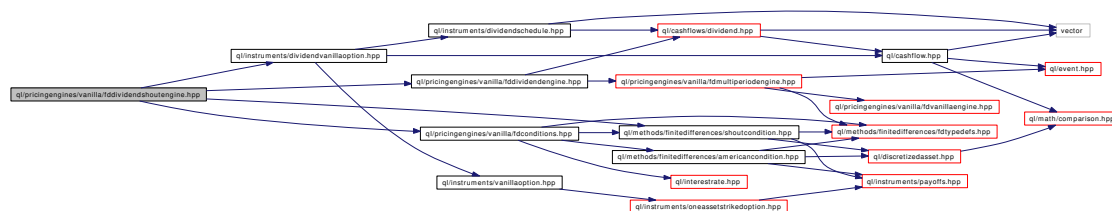
10.355 ql/pricingengines/vanilla/fddividendshoutengine.hpp File Reference

10.355.1 Detailed Description

base class for shout engine with dividends

```
#include <ql/instruments/dividendvanillaoption.hpp>
#include <ql/pricingengines/vanilla/fddividendengine.hpp>
#include <ql/pricingengines/vanilla/fdconditions.hpp>
#include <ql/methods/finitedifferences/shoutcondition.hpp>
```

Include dependency graph for fddividendshoutengine.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef FDEngineAdapter< FDS shoutCondition< FDDividendEngine >, DividendVanillaOption::engine > **FDDividendShoutEngine**
Finite-differences shout engine with dividends.
- typedef FDEngineAdapter< FDS shoutCondition< FDDividendEngineMerton73 >, DividendVanillaOption::engine > **FDDividendShoutEngineMerton73**
- typedef FDEngineAdapter< FDS shoutCondition< FDDividendEngineShiftScale >, DividendVanillaOption::engine > **FDDividendShoutEngineShiftScale**

10.356 ql/pricingengines/vanilla/fdeuropeanengine.hpp File Reference

10.356.1 Detailed Description

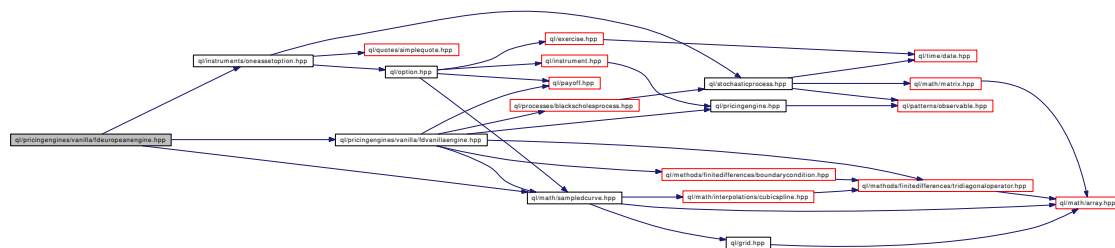
Finite-difference European engine.

```
#include <ql/instruments/oneassetoption.hpp>
```

```
#include <ql/pricingengines/vanilla/fdvanillaengine.hpp>
```

```
#include <ql/math/sampledcurve.hpp>
```

Include dependency graph for fdeuropeanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FDEuropeanEngine**
Pricing engine for European options using finite-differences.

10.357 ql/pricingengines/vanilla/fdmultiperiodengine.hpp File Reference

10.357.1 Detailed Description

base engine for options with events happening at specific times

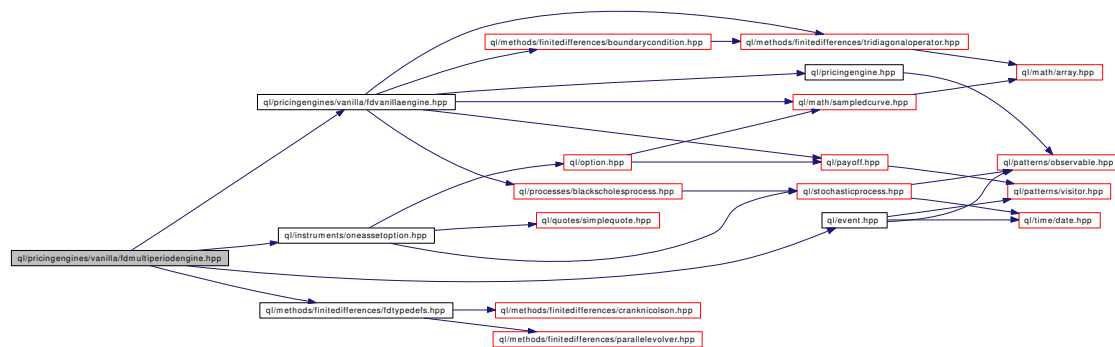
```
#include <ql/pricingengines/vanilla/fdvanillaengine.hpp>
```

```
#include <ql/methods/finitedifferences/fdtypedefs.hpp>
```

```
#include <ql/instruments/oneassetoption.hpp>
```

```
#include <ql/event.hpp>
```

Include dependency graph for fdmultiperiodengine.hpp:



Namespaces

- namespace QuantLib

10.358 ql/pricingengines/vanilla/fdshoutengine.hpp File Reference

10.358.1 Detailed Description

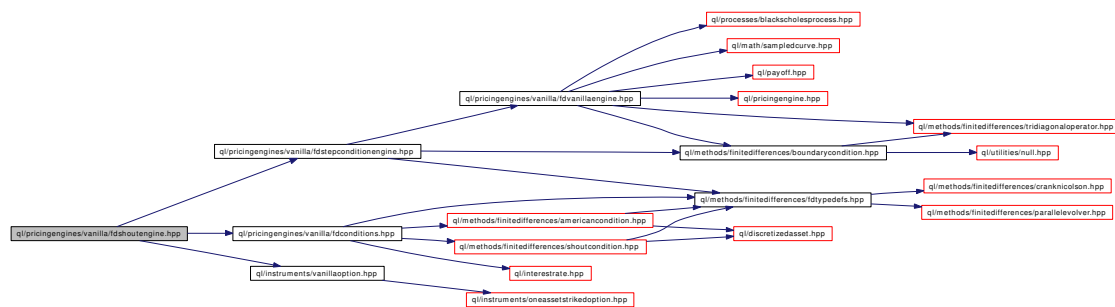
Finite-differences shout engine.

```
#include <ql/pricingengines/vanilla/fdstepconditionengine.hpp>
```

```
#include <ql/pricingengines/vanilla/fdconditions.hpp>
```

```
#include <ql/instruments/vanillaoption.hpp>
```

Include dependency graph for fdshoutengine.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef `FDEngineAdapter< FDShoutCondition< FDStepConditionEngine >, VanillaOption::engine >` **FDShoutEngine**

Finite-differences pricing engine for shout vanilla options.

10.359 ql/pricingengines/vanilla/fdstepconditionengine.hpp File Reference

10.359.1 Detailed Description

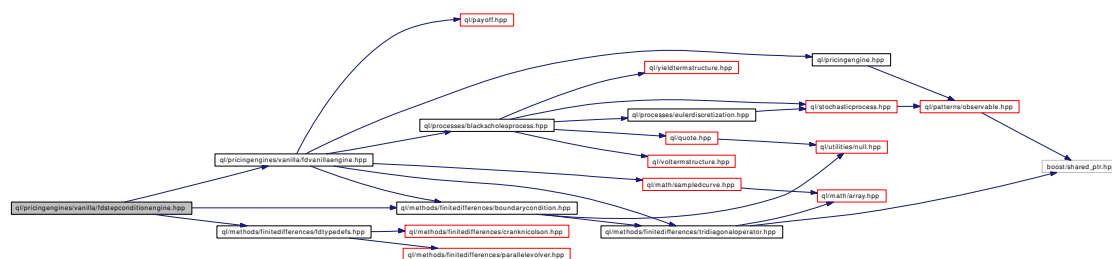
Finite-differences step-condition engine.

```
#include <ql/pricingengines/vanilla/fdvanillaengine.hpp>
```

```
#include <ql/methods/finitedifferences/fdtypedefs.hpp>
```

```
#include <ql/methods/finitedifferences/boundarycondition.hpp>
```

Include dependency graph for fdstepconditionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FDStepConditionEngine**

Finite-differences pricing engine for American-style vanilla options.

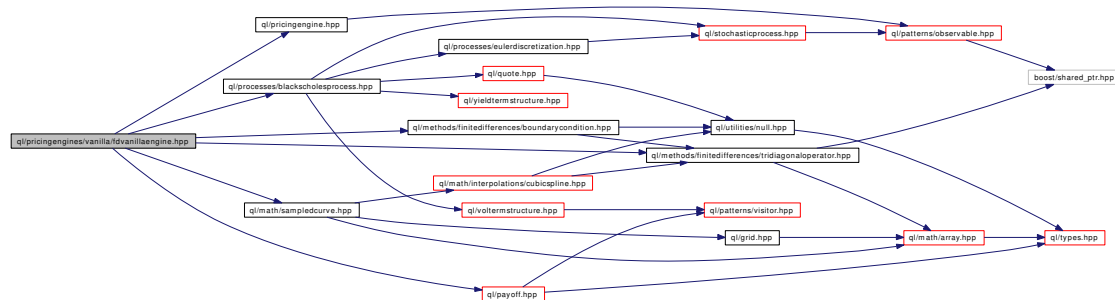
10.360 ql/pricingengines/vanilla/fdvanillaengine.hpp File Reference

10.360.1 Detailed Description

Finite-differences vanilla-option engine.

```
#include <ql/pricingengine.hpp>
#include <ql/methods/finitedifferences/tridiagonaloperator.hpp>
#include <ql/methods/finitedifferences/boundarycondition.hpp>
#include <ql/processes/blackscholesprocess.hpp>
#include <ql/math/sampledcurve.hpp>
#include <ql/payoff.hpp>
```

Include dependency graph for fdvanillaengine.hpp:



Namespaces

- namespace **QuantLib**

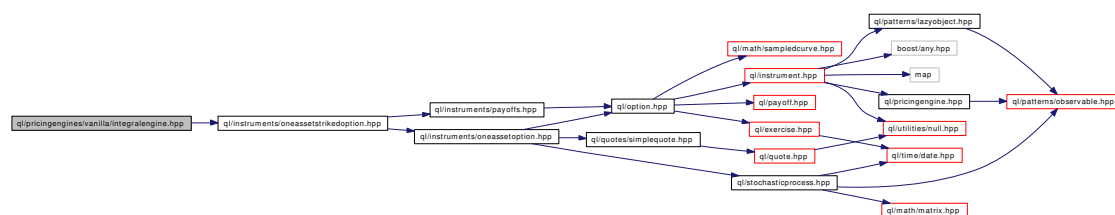
10.361 ql/pricingengines/vanilla/integralengine.hpp File Reference

10.361.1 Detailed Description

Integral option engine.

```
#include <ql/instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for integralengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [IntegralEngine](#)
Pricing engine for European vanilla options using integral approach.

10.363.1 Detailed Description

```
#include <ql/instruments/vanillaoption.hpp>
```

[illegible]

- namespace **QuantLib**

- class `JuQuadraticApproximationEngine`
Pricing engine for American options with Ju quadratic approximation.

10.364 ql/pricingengines/vanilla/mcamericanengine.hpp File Reference

10.364.1 Detailed Description

American Monte Carlo engine.

```
#include <ql/qldefines.hpp>
```

```
#include <ql/payoff.hpp>
```

```
#include <ql/methods/montecarlo/lsmbasissystem.hpp>
```

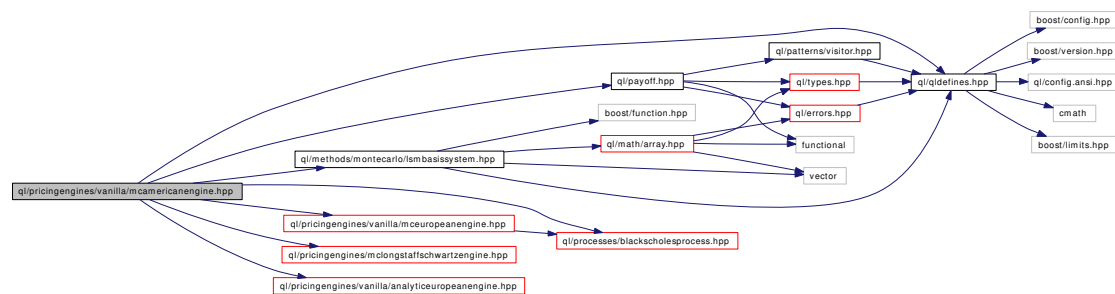
```
#include <ql/processes/blackscholesprocess.hpp>
```

```
#include <ql/pricingengines/mclongstaffschwartzengine.hpp>
```

```
#include <ql/pricingengines/vanilla/mceuropeanengine.hpp>
```

```
#include <ql/pricingengines/vanilla/analyticeuropeanengine.hpp>
```

Include dependency graph for mcamericanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCAmericanEngine**
American Monte Carlo engine.
- class **MakeMCAmericanEngine**
Monte Carlo American engine factory.

10.365 ql/pricingengines/vanilla/mcdigitalengine.hpp File Reference

10.365.1 Detailed Description

digital option Monte Carlo engine

```
#include <ql/exercise.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

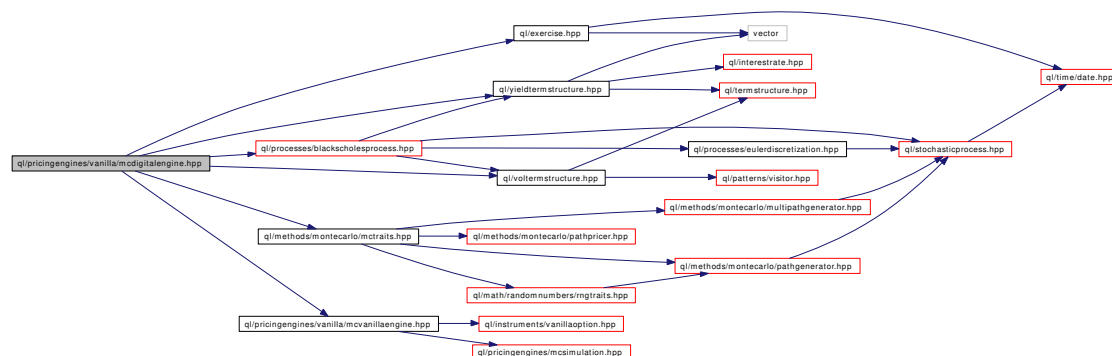
```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/methods/montecarlo/mctraits.hpp>
```

```
#include <ql/pricingengines/vanilla/mcvanillaengine.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

Include dependency graph for mcdigitalengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCDigitalEngine**
Pricing engine for digital options using Monte Carlo simulation.
- class **MakeMCDigitalEngine**
Monte Carlo digital engine factory.

10.366 ql/pricingengines/vanilla/mceuropeanengine.hpp File Reference

10.366.1 Detailed Description

Monte Carlo European option engine.

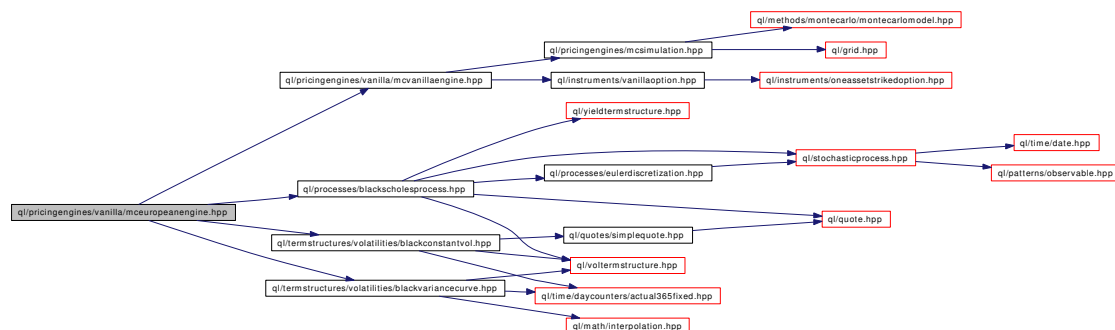
```
#include <ql/pricingengines/vanilla/mcvanillaengine.hpp>
```

```
#include <ql/processes/blackscholesprocess.hpp>
```

```
#include <ql/termstructures/volatilities/blackconstantvol.hpp>
```

```
#include <ql/termstructures/volatilities/blackvariancecurve.hpp>
```

Include dependency graph for mceuropeanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MCEuropeanEngine](#)
European option pricing engine using Monte Carlo simulation.
- class [MakeMCEuropeanEngine](#)
Monte Carlo European engine factory.

10.367 ql/pricingengines/vanilla/mceuropeanhestonengine.hpp File Reference

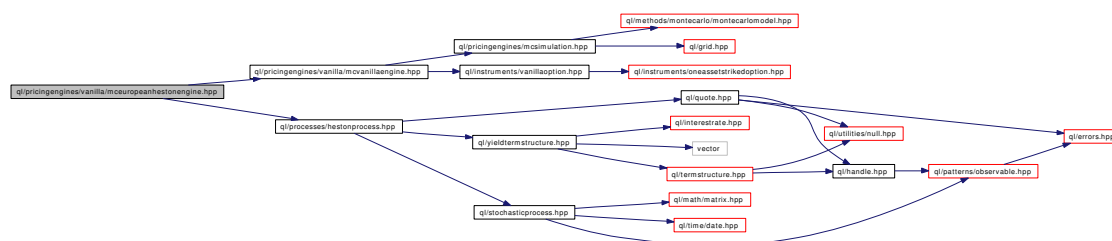
10.367.1 Detailed Description

Monte Carlo Heston-model engine for European options.

```
#include <ql/pricingengines/vanilla/mcvanillaengine.hpp>
```

```
#include <ql/processes/hestonprocess.hpp>
```

Include dependency graph for mceuropeanhestonengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCEuropeanHestonEngine**
Monte Carlo Heston-model engine for European options.
- class **MakeMCEuropeanHestonEngine**
Monte Carlo Heston European engine factory.

10.368 ql/pricingengines/vanilla/mcvanillaengine.hpp File Reference

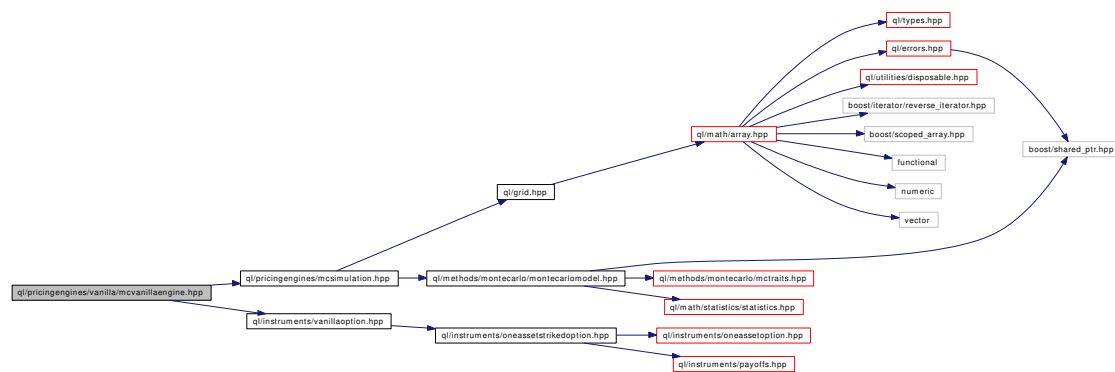
10.368.1 Detailed Description

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.

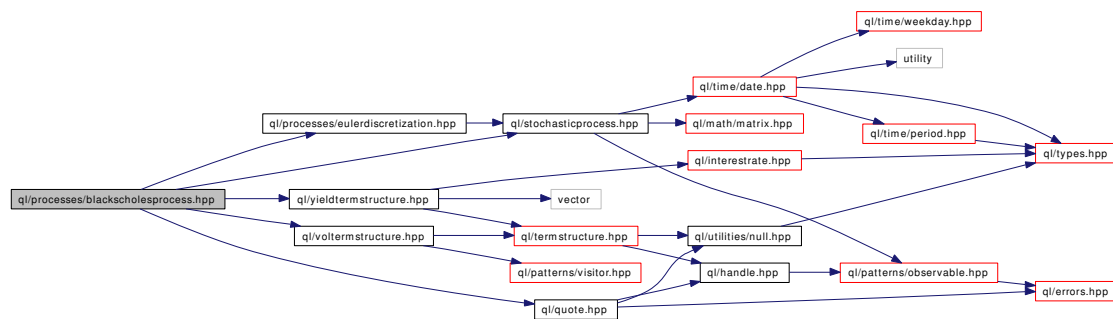
10.369 ql/processes/blackscholesprocess.hpp File Reference

10.369.1 Detailed Description

Black-Scholes processes.

```
#include <ql/stochasticprocess.hpp>
#include <ql/processes/eulerdiscrretization.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/voltermstructure.hpp>
#include <ql/quote.hpp>
```

Include dependency graph for blackscholesprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GeneralizedBlackScholesProcess](#)
Generalized Black-Scholes stochastic process.
- class [BlackScholesProcess](#)
Black-Scholes (1973) stochastic process.
- class [BlackScholesMertonProcess](#)
Merton (1973) extension to the Black-Scholes stochastic process.
- class [BlackProcess](#)
Black (1976) stochastic process.
- class [GarmanKohlhagenProcess](#)
Garman-Kohlhagen (1983) stochastic process.

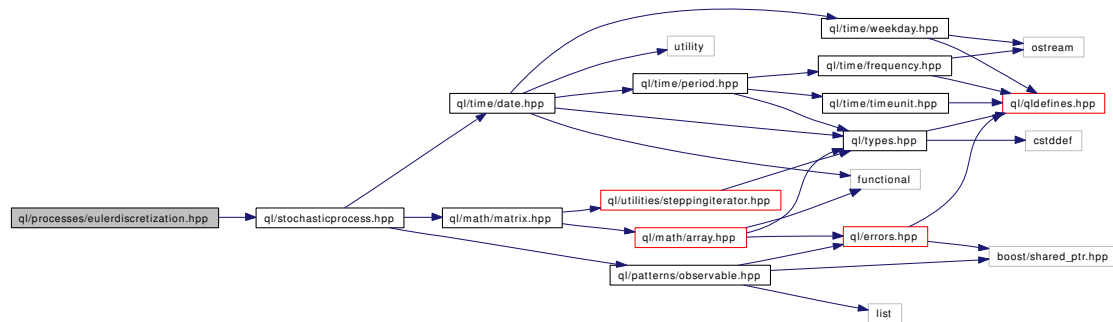
10.370 ql/processes/euldiscretization.hpp File Reference

10.370.1 Detailed Description

Euler discretization for stochastic processes.

```
#include <ql/stochasticprocess.hpp>
```

Include dependency graph for euldiscretization.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [EulerDiscretization](#)
Euler discretization for stochastic processes.

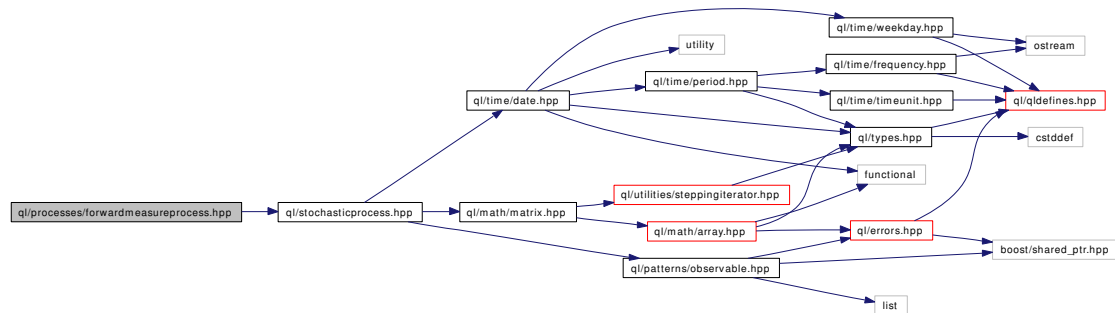
10.371 ql/processes/forwardmeasureprocess.hpp File Reference

10.371.1 Detailed Description

forward-measure stochastic processes

```
#include <ql/stochasticprocess.hpp>
```

Include dependency graph for forwardmeasureprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **ForwardMeasureProcess**
forward-measure stochastic process
- class **ForwardMeasureProcess1D**
forward-measure 1-D stochastic process

10.372 ql/processes/g2process.hpp File Reference

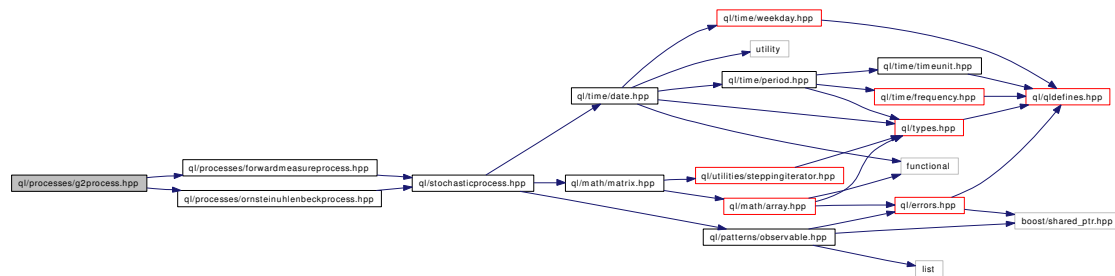
10.372.1 Detailed Description

G2 stochastic processes.

```
#include <ql/processes/forwardmeasureprocess.hpp>
```

```
#include <ql/processes/ornsteinuhlenbeckprocess.hpp>
```

Include dependency graph for g2process.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [G2Process](#)
G2 stochastic process
- class [G2ForwardProcess](#)
Forward G2 stochastic process

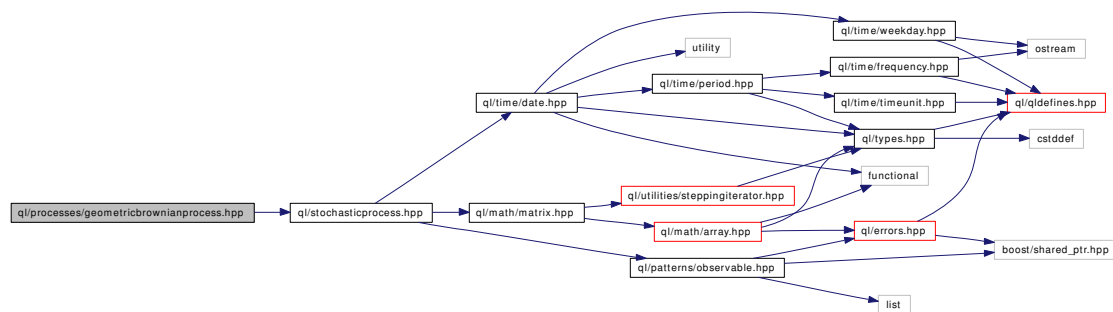
10.373 ql/processes/geometricbrownianprocess.hpp File Reference

10.373.1 Detailed Description

Geometric Brownian-motion process.

```
#include <ql/stochasticprocess.hpp>
```

Include dependency graph for geometricbrownianprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GeometricBrownianMotionProcess](#)
Geometric brownian-motion process.

10.374 ql/processes/hestonprocess.hpp File Reference

10.374.1 Detailed Description

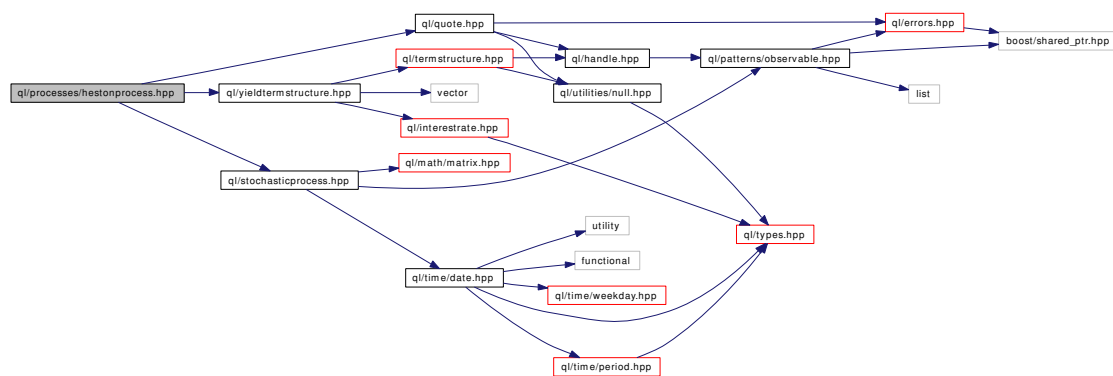
Heston stochastic process.

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/quote.hpp>
```

Include dependency graph for hestonprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [HestonProcess](#)
Square-root stochastic-volatility Heston process.

10.375 ql/processes/hullwhiteprocess.hpp File Reference

10.375.1 Detailed Description

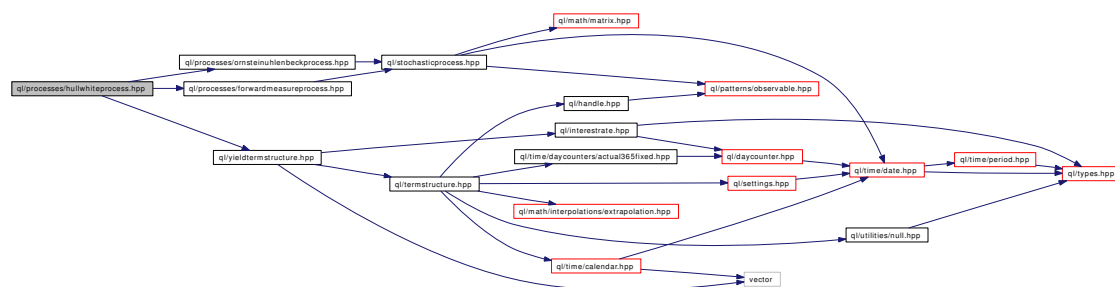
Hull-White stochastic processes.

```
#include <ql/processes/forwardmeasureprocess.hpp>
```

```
#include <ql/processes/ornsteinuhlenbeckprocess.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

Include dependency graph for hullwhiteprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [HullWhiteProcess](#)
Hull-White stochastic process.
- class [HullWhiteForwardProcess](#)
Forward Hull-White stochastic process

10.376 ql/processes/lfmcovarParams.hpp File Reference

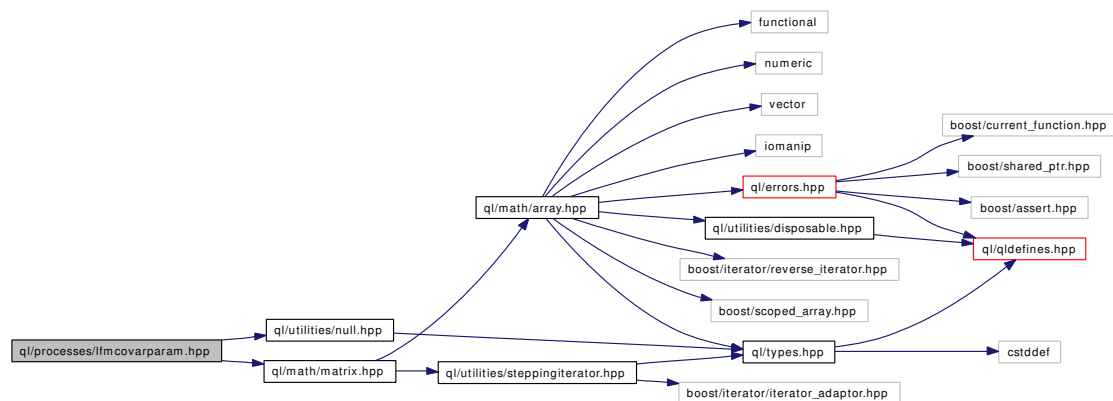
10.376.1 Detailed Description

volatility & correlation function for libor forward model process

```
#include <ql/math/matrix.hpp>
```

```
#include <ql/utilities/null.hpp>
```

Include dependency graph for lfmcovarParams.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LfmCovarianceParameterization](#)
Libor market model parameterization

10.377 ql/processes/lfmhullwhiteparam.hpp File Reference

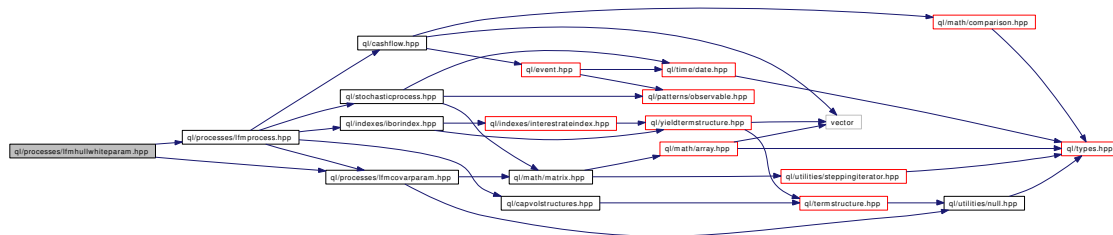
10.377.1 Detailed Description

libor market model parameterization based on Hull White

```
#include <ql/processes/lfmprocess.hpp>
```

```
#include <ql/processes/lfmcovarParams.hpp>
```

Include dependency graph for lfmhullwhiteparam.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LfmHullWhiteParameterization](#)
Libor market model parameterization based on Hull White paper

10.379 ql/processes/merton76process.hpp File Reference

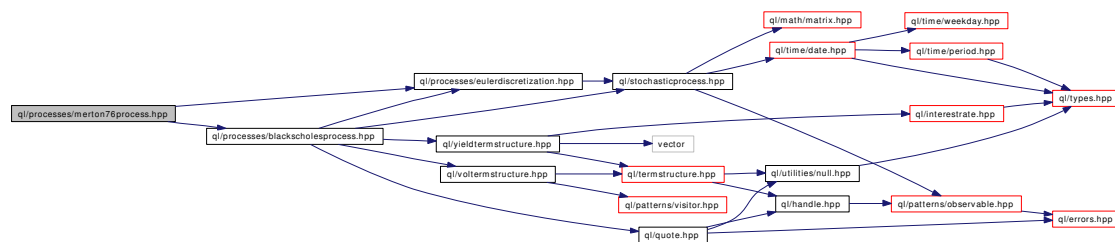
10.379.1 Detailed Description

Merton-76 process.

```
#include <ql/processes/blackscholesprocess.hpp>
```

```
#include <ql/processes/eulerdiscretization.hpp>
```

Include dependency graph for merton76process.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Merton76Process](#)
Merton-76 jump-diffusion process.

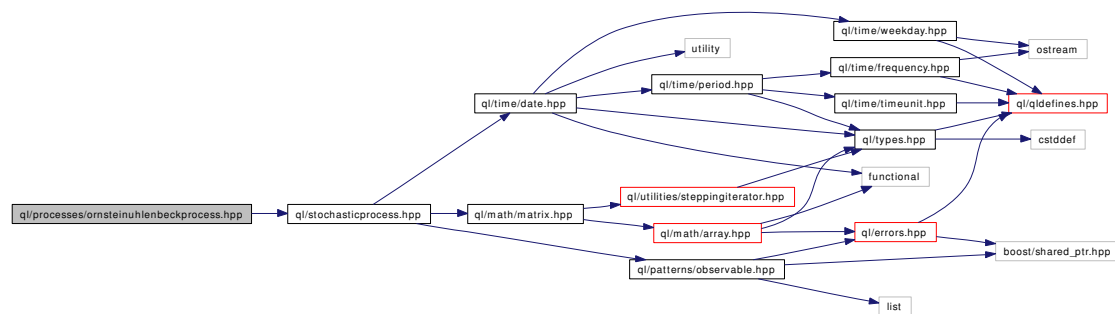
10.380 ql/processes/ornsteinuhlenbeckprocess.hpp File Reference

10.380.1 Detailed Description

Ornstein-Uhlenbeck process.

```
#include <ql/stochasticprocess.hpp>
```

Include dependency graph for ornsteinuhlenbeckprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [OrnsteinUhlenbeckProcess](#)
Ornstein-Uhlenbeck process class.

10.381 ql/processes/squarerootprocess.hpp File Reference

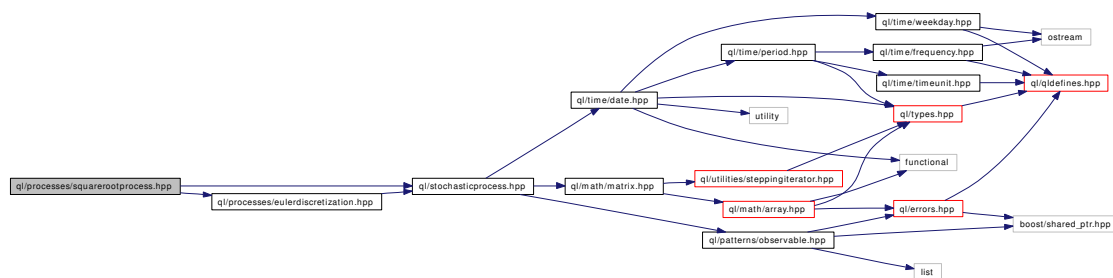
10.381.1 Detailed Description

square-root process

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/processes/eulerdiscretization.hpp>
```

Include dependency graph for squarerootprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SquareRootProcess](#)
Square-root process class.

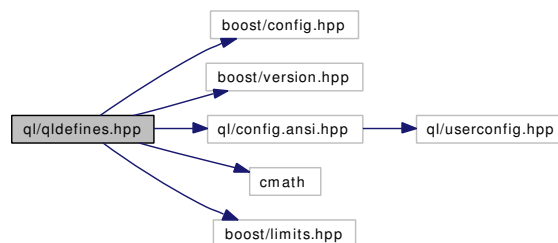
10.383 ql/qldefines.hpp File Reference

10.383.1 Detailed Description

Global definitions and compiler switches.

```
#include <boost/config.hpp>
#include <boost/version.hpp>
#include <ql/config.ansi.hpp>
#include <cmath>
#include <boost/limits.hpp>
```

Include dependency graph for qldefines.hpp:



Defines

- **#define BOOST_ENABLE_ASSERT_HANDLER**
- **#define QL_INTEGER** int
- **#define QL_BIG_INTEGER** long
- **#define QL_REAL** double
- **#define QL_VERSION** "0.8.1"
version string
- **#define QL_HEX_VERSION** 0x000801f0
version hexadecimal number
- **#define QL_LIB_VERSION** "0_8_1"
version string for output lib name
- **#define QL_DUMMY_RETURN(x)**
Is a dummy return statement required?
- **#define QL_MIN_INTEGER** ((std::numeric_limits<QL_INTEGER>::min)())
- **#define QL_MAX_INTEGER** ((std::numeric_limits<QL_INTEGER>::max)())
- **#define QL_MIN_REAL** -((std::numeric_limits<QL_REAL>::max)())
- **#define QL_MAX_REAL** ((std::numeric_limits<QL_REAL>::max)())
- **#define QL_MIN_POSITIVE_REAL** ((std::numeric_limits<QL_REAL>::min)())
- **#define QL_EPSILON** ((std::numeric_limits<QL_REAL>::epsilon)())
- **#define QL_NULL_INTEGER** ((std::numeric_limits<int>::max)())
- **#define QL_NULL_REAL** ((std::numeric_limits<float>::max)())

- #define [QL_TYPENAME](#) typename
- #define [QL_FULL_ITERATOR_SUPPORT](#)

10.384 ql/quote.hpp File Reference

10.384.1 Detailed Description

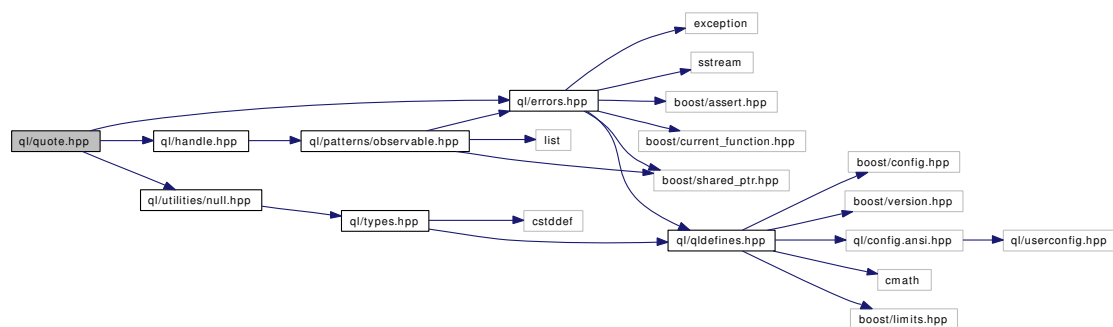
purely virtual base class for market observables

```
#include <ql/handle.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <ql/utilities/null.hpp>
```

Include dependency graph for quote.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Quote](#)
purely virtual base class for market observables

10.385 ql/quotes/compositequote.hpp File Reference

10.385.1 Detailed Description

purely virtual base class for market observables

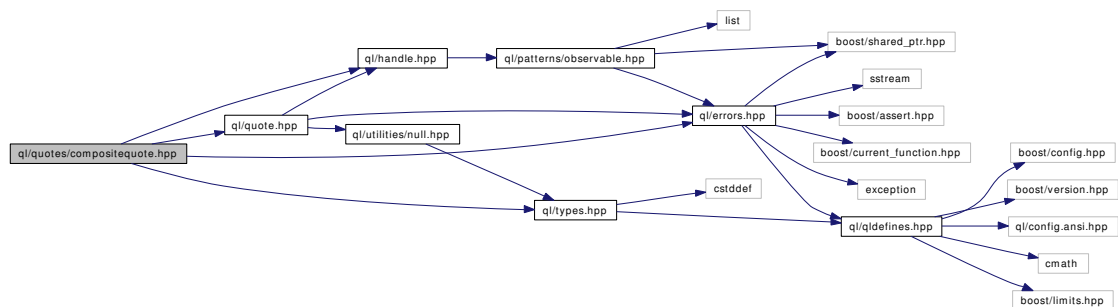
```
#include <ql/quote.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/handle.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for compositequote.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CompositeQuote**
market element whose value depends on two other market element

10.386 ql/quotes/derivedquote.hpp File Reference

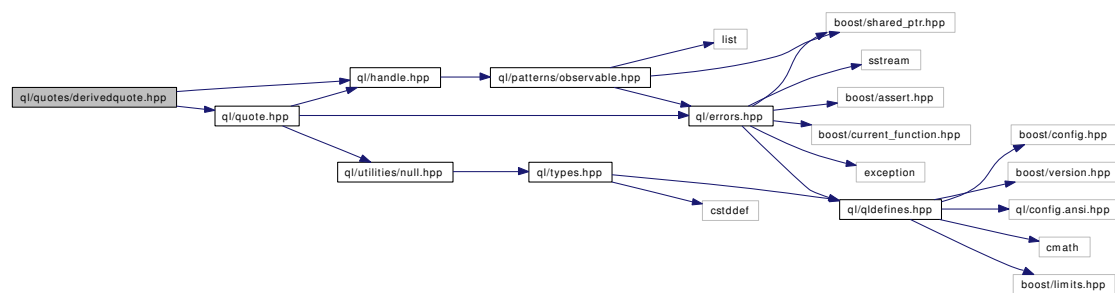
10.386.1 Detailed Description

market quote whose value depends on another quote

```
#include <ql/quote.hpp>
```

```
#include <ql/handle.hpp>
```

Include dependency graph for derivedquote.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DerivedQuote](#)
market quote whose value depends on another quote

10.387 ql/quotes/eurodollarfuturesquote.hpp File Reference

10.387.1 Detailed Description

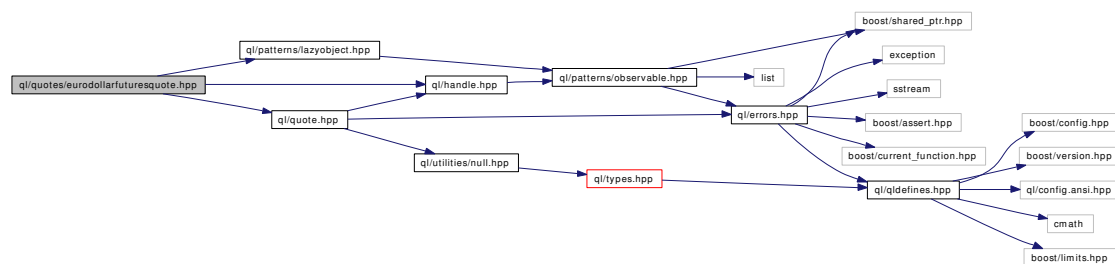
quote for the Eurodollar-future implied standard deviation

```
#include <ql/quote.hpp>
```

```
#include <ql/handle.hpp>
```

```
#include <ql/patterns/lazyobject.hpp>
```

Include dependency graph for eurodollarfuturesquote.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [EurodollarFuturesImpliedStdDevQuote](#)
quote for the Eurodollar-future implied standard deviation

10.388 ql/quotes/forwardvaluequote.hpp File Reference

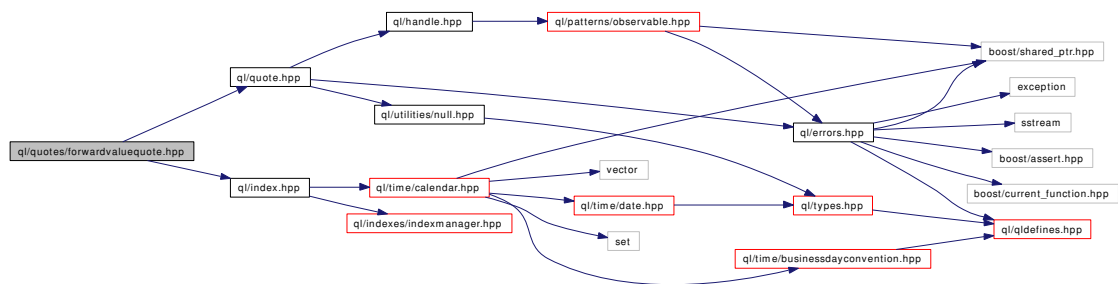
10.388.1 Detailed Description

quote for the forward value of an index

```
#include <ql/quote.hpp>
```

```
#include <ql/index.hpp>
```

Include dependency graph for forwardvaluequote.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ForwardValueQuote](#)
quote for the forward value of an index

10.389 ql/quotes/futuresconvadjustmentquote.hpp File Reference

10.389.1 Detailed Description

quote for the futures-convexity adjustment of an index

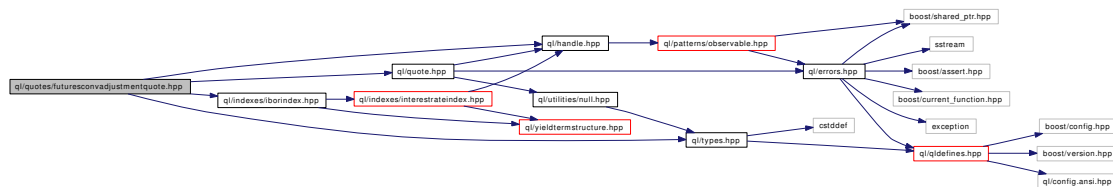
```
#include <ql/quote.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/handle.hpp>
```

```
#include <ql/indexes/iborindex.hpp>
```

Include dependency graph for futuresconvadjustmentquote.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FuturesConvAdjustmentQuote**
quote for the futures-convexity adjustment of an index

10.390 ql/quotes/IMPLIEDSTDDEVQUOTE.hpp File Reference

10.390.1 Detailed Description

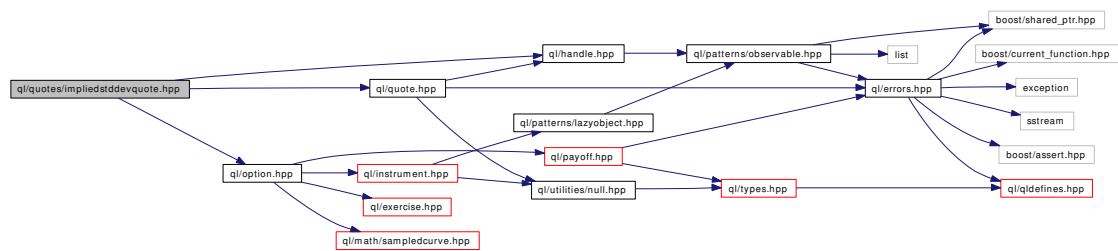
quote for the implied standard deviation of an underlying

```
#include <ql/quote.hpp>
```

```
#include <ql/handle.hpp>
```

```
#include <ql/option.hpp>
```

Include dependency graph for IMPLIEDSTDDEVQUOTE.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ImpliedStdDevQuote](#)
quote for the implied standard deviation of an underlying

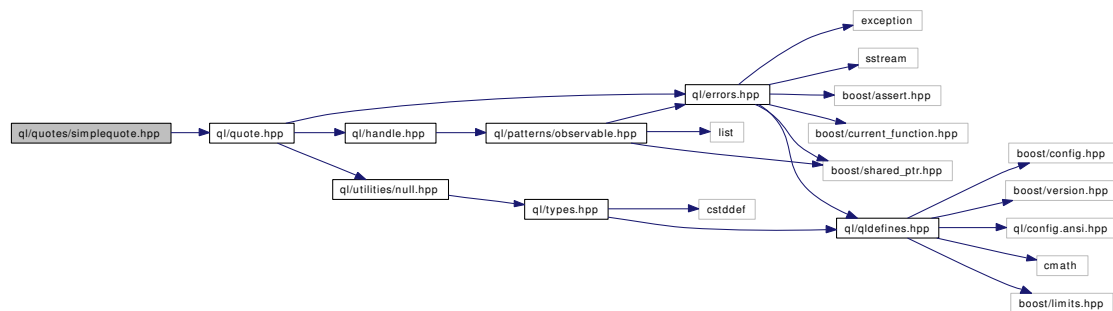
10.391 ql/quotes/simplequote.hpp File Reference

10.391.1 Detailed Description

simple quote class

```
#include <ql/quote.hpp>
```

Include dependency graph for simplequote.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SimpleQuote](#)
market element returning a stored value

10.392 ql/settings.hpp File Reference

10.392.1 Detailed Description

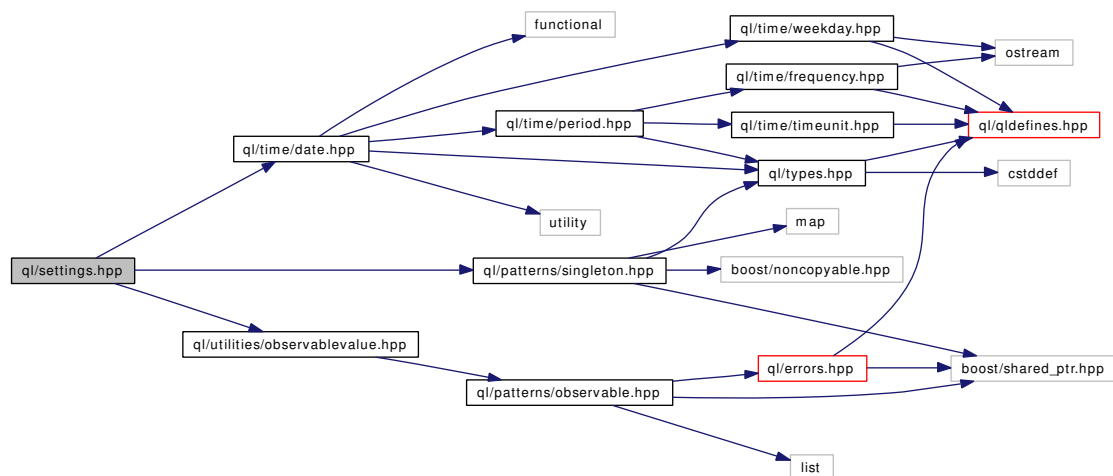
global repository for run-time library settings

```
#include <ql/patterns/singleton.hpp>
```

```
#include <ql/time/date.hpp>
```

```
#include <ql/utilities/observablevalue.hpp>
```

Include dependency graph for settings.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Settings](#)
global repository for run-time library settings

Functions

- `std::ostream & operator<< (std::ostream &out, const Settings::DateProxy &p)`

10.393 ql/stochasticprocess.hpp File Reference

10.393.1 Detailed Description

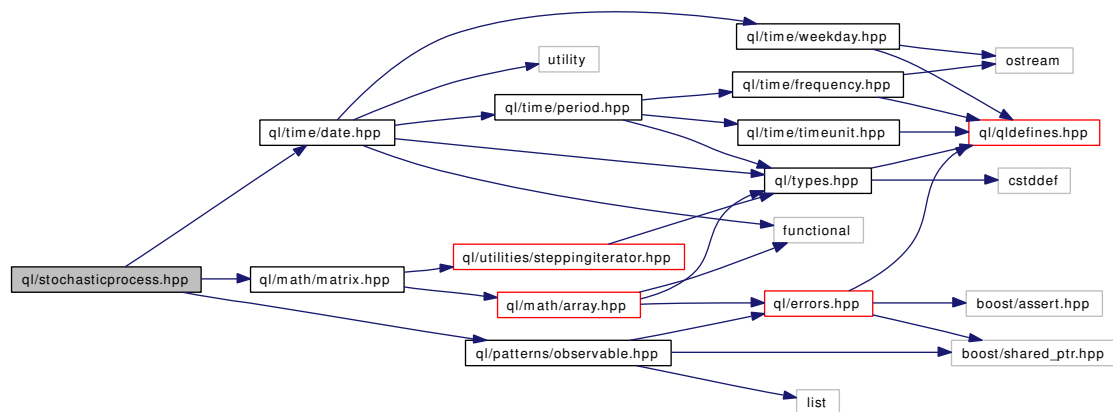
stochastic processes

```
#include <ql/time/date.hpp>
```

```
#include <ql/patterns/observable.hpp>
```

```
#include <ql/math/matrix.hpp>
```

Include dependency graph for stochasticprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [StochasticProcess](#)
multi-dimensional stochastic process class.
- class [StochasticProcess::discretization](#)
discretization of a stochastic process over a given time interval
- class [StochasticProcess1D](#)
1-dimensional stochastic process
- class [StochasticProcess1D::discretization](#)
discretization of a 1-D stochastic process

10.394 ql/swapvolstructure.hpp File Reference

10.394.1 Detailed Description

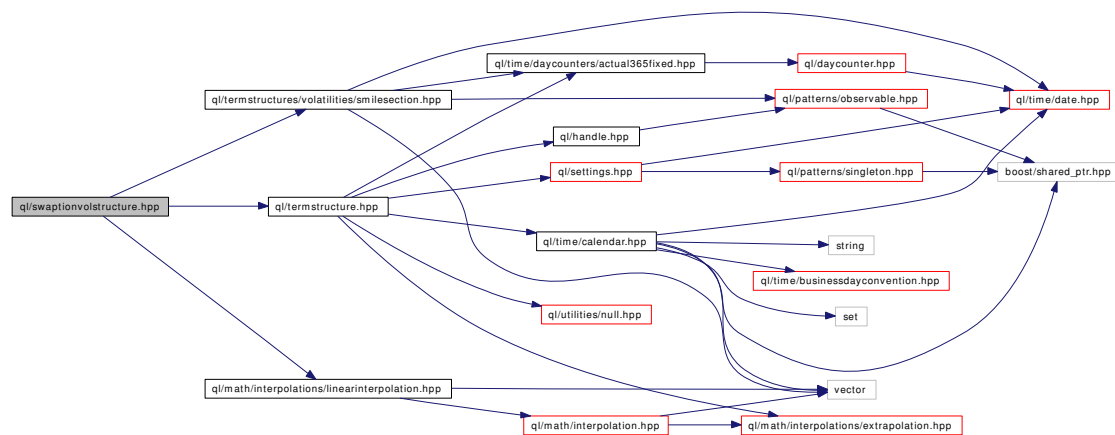
Swapion volatility structure.

```
#include <ql/termstructure.hpp>
```

```
#include <ql/math/interpolations/linearinterpolation.hpp>
```

```
#include <ql/termstructures/volatilities/smilesection.hpp>
```

Include dependency graph for swapvolstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **SwapionVolatilityStructure**
Swapion-volatility structure

10.395 ql/termstructure.hpp File Reference

10.395.1 Detailed Description

base class for term structures

```
#include <ql/time/calendar.hpp>
```

```
#include <ql/time/daycounters/actual365fixed.hpp>
```

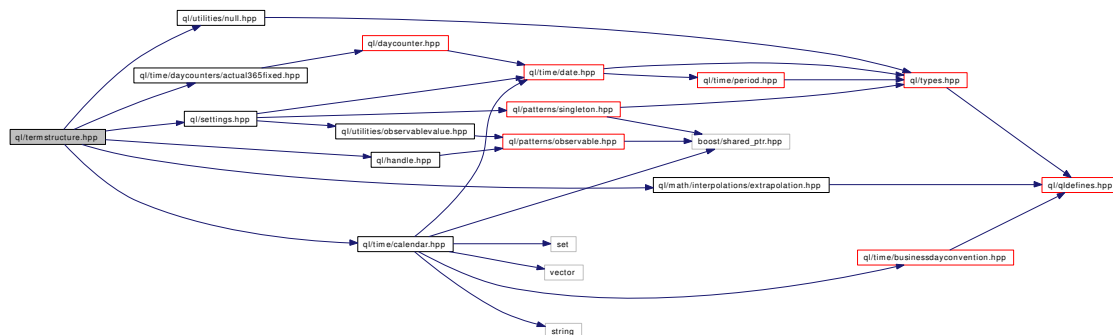
```
#include <ql/settings.hpp>
```

```
#include <ql/handle.hpp>
```

```
#include <ql/math/interpolations/extrapolation.hpp>
```

```
#include <ql/utilities/null.hpp>
```

Include dependency graph for termstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TermStructure**
Basic term-structure functionality.

10.397 ql/termstructures/volatilities/blackvariancecurve.hpp File Reference

10.397.1 Detailed Description

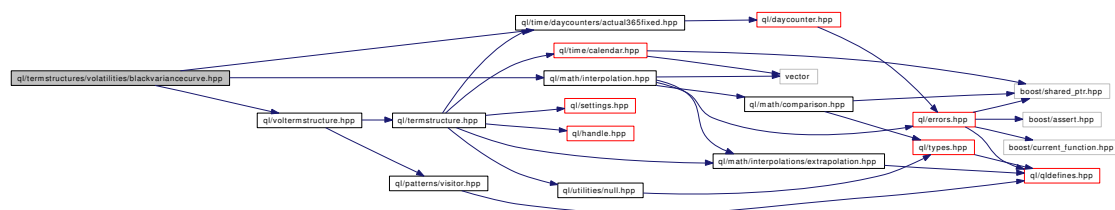
Black volatility curve modelled as variance curve.

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/math/interpolation.hpp>
```

```
#include <ql/time/daycounters/actual365fixed.hpp>
```

Include dependency graph for blackvariancecurve.hpp:



Namespaces

- namespace **QuantLib**

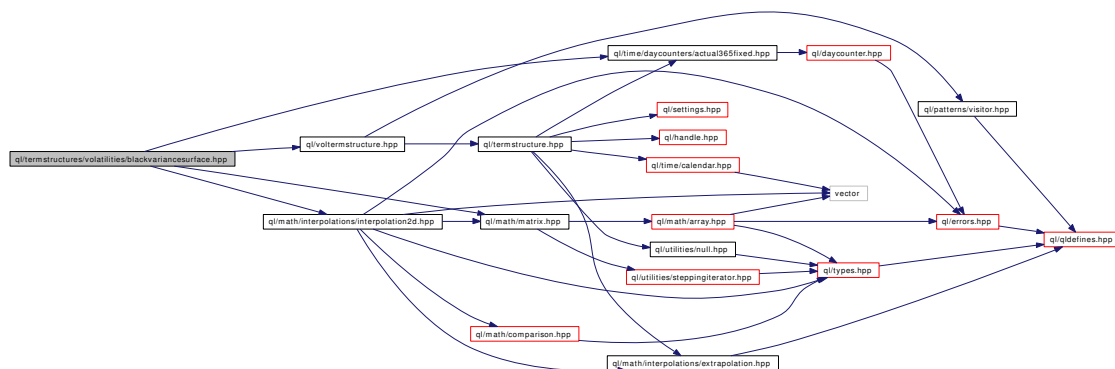
Classes

- class **BlackVarianceCurve**

Black volatility curve modelled as variance curve.

10.398.1 Detailed Description

```
#include <ql/voltermstructure.hpp>
#include <ql/math/matrix.hpp>
#include <ql/math/interpolations/interpolation2d.hpp>
#include <ql/time/daycounters/actual365fixed.hpp>
Include dependency graph for blackvariancesurface.hpp:
```



- namespace **QuantLib**

- class **BlackVarianceSurface**
Black volatility surface modelled as variance surface.

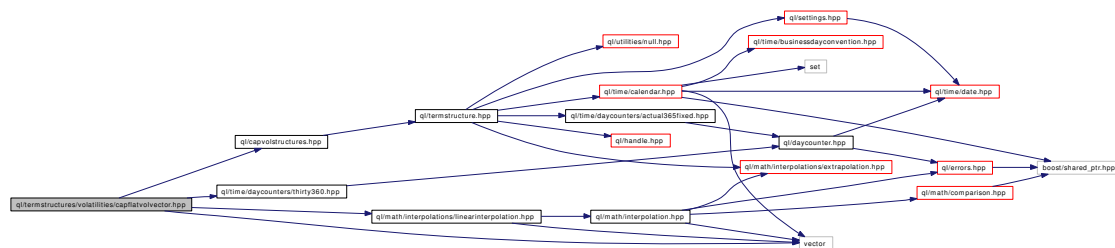
10.399 ql/termstructures/volatilities/capflatvolvector.hpp File Reference

10.399.1 Detailed Description

Cap/floor at-the-money flat volatility vector.

```
#include <ql/capvolstructures.hpp>
#include <ql/math/interpolations/linearinterpolation.hpp>
#include <ql/time/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.

10.400 ql/termstructures/volatilities/capletconstantvol.hpp File Reference

10.400.1 Detailed Description

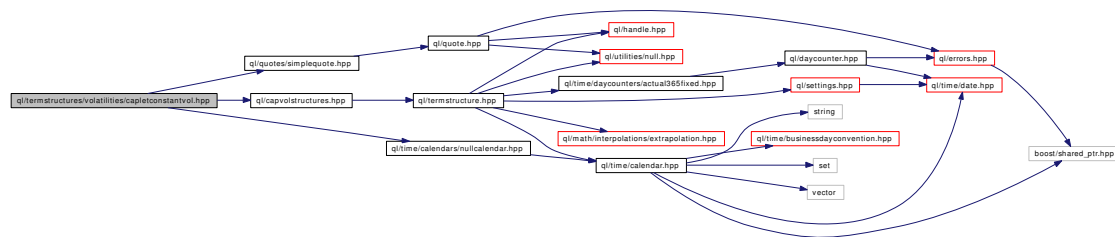
Constant caplet volatility.

```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/quotes/simplequote.hpp>
```

```
#include <ql/time/calendars/nullcalendar.hpp>
```

Include dependency graph for capletconstantvol.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CapletConstantVolatility**

Constant caplet volatility, no time-strike dependence.

10.402 **ql/termstructures/volatilities/capletvolatilitiesstructures.hpp**
File Reference

Caplet Volatilities Structures used during bootstrapping procedure.

Include dependency graph for capletvolatilitiesstructures.hpp:

Namespaces

Classes

Typedefs

Functions

10.403 ql/termstructures/volatilities/capstripper.hpp File Reference

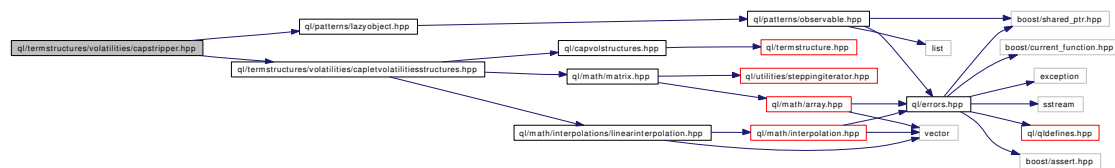
10.403.1 Detailed Description

caplet volatility stripper

```
#include <ql/patterns/lazyobject.hpp>
```

```
#include <ql/termstructures/volatilities/capletvolatilitiesstructures.hpp>
```

Include dependency graph for capstripper.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef std::vector< std::vector< boost::shared_ptr< CapFloor > > > **CapMatrix**

10.404 ql/termstructures/volatilities/cmsmarket.hpp File Reference

10.404.1 Detailed Description

set of CMS quotes

```
#include <ql/termstructures/volatilities/swaptionvolmatrix.hpp>
```

```
#include <ql/termstructures/volatilities/swaptionvolcube1.hpp>
```

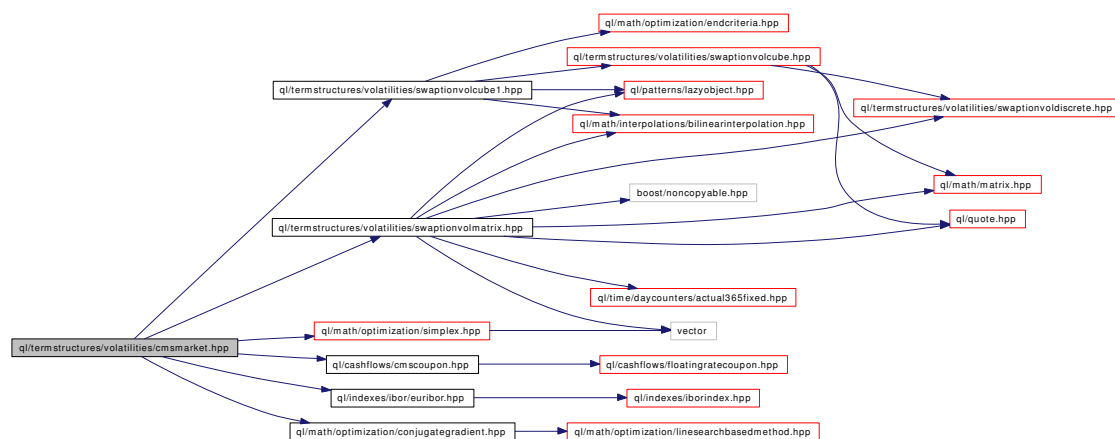
```
#include <ql/cashflows/cmscoupon.hpp>
```

```
#include <ql/indexes/ibor/euribor.hpp>
```

```
#include <ql/math/optimization/conjugategradient.hpp>
```

```
#include <ql/math/optimization/simplex.hpp>
```

Include dependency graph for cmsmarket.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CmsMarket**
set of CMS quotes

Typedefs

- typedef **Leg** **Leg**

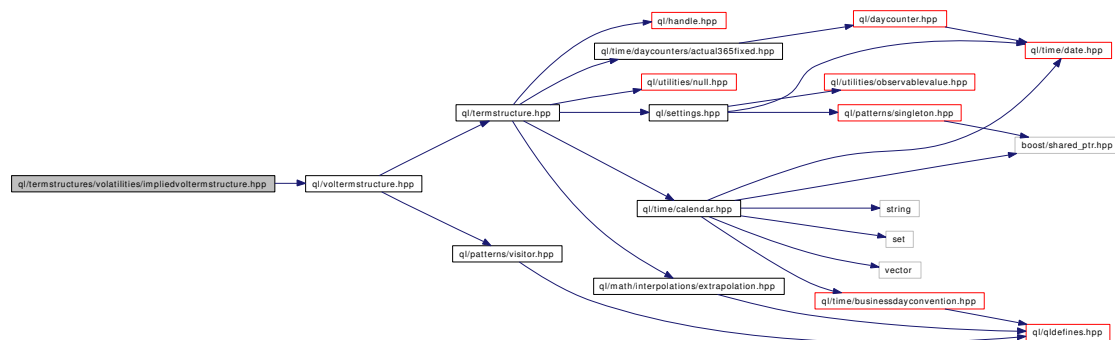
10.405 ql/termstructures/volatilities/impliedvoltermstructure.hpp File Reference

10.405.1 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.

10.406.1 Detailed Description

```
#include <ql/termstructure.hpp>
#include <ql/quotes/simplequote.hpp>
#include <ql/patterns/lazyobject.hpp>
#include <ql/math/interpolations/linearinterpolation.hpp>
#include <ql/termstructures/volatilities/smilesection.hpp>
```

- namespace **QuantLib**

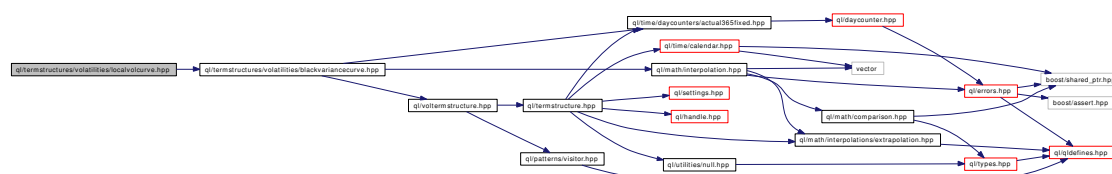
10.408 ql/termstructures/volatilities/localvolcurve.hpp File Reference

10.408.1 Detailed Description

Local volatility curve derived from a Black curve.

```
#include <ql/termstructures/volatilities/blackvariancecurve.hpp>
```

Include dependency graph for localvolcurve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **LocalVolCurve**
Local volatility curve derived from a Black curve.

10.409 ql/termstructures/volatilities/localvolsurface.hpp File Reference

10.409.1 Detailed Description

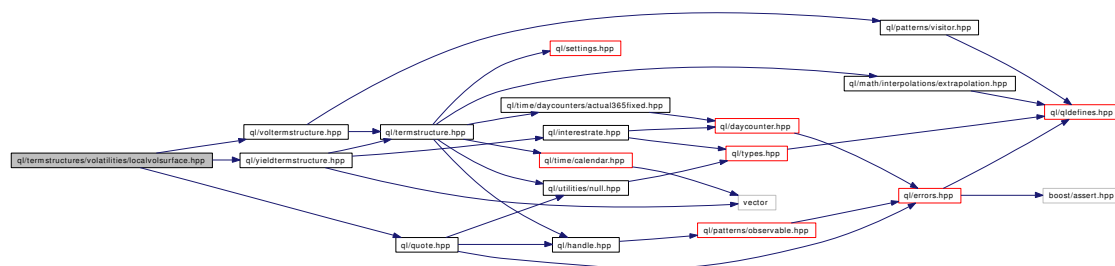
Local volatility surface derived from a Black vol surface.

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/quote.hpp>
```

Include dependency graph for localvolsurface.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **LocalVolSurface**

Local volatility surface derived from a Black vol surface.

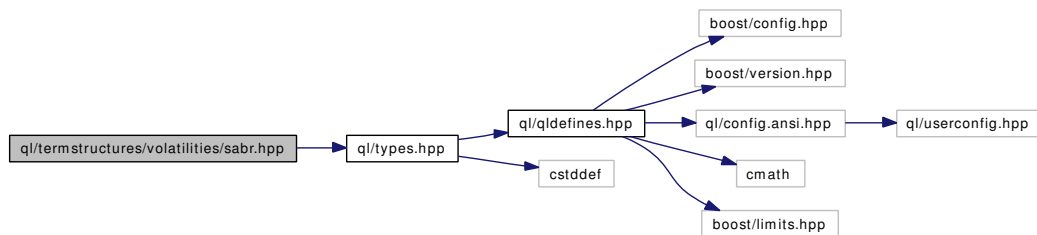
10.410 ql/termstructures/volatilities/sabr.hpp File Reference

10.410.1 Detailed Description

SABR functions.

```
#include <ql/types.hpp>
```

Include dependency graph for sabr.hpp:



Namespaces

- namespace **QuantLib**

Functions

- Real **unsafeSabrVolatility** (Rate strike, Rate forward, Time expiryTime, Real alpha, Real beta, Real nu, Real rho)
- Real **sabrVolatility** (Rate strike, Rate forward, Time expiryTime, Real alpha, Real beta, Real nu, Real rho)
- void **validateSabrParameters** (Real alpha, Real beta, Real nu, Real rho)

10.411 ql/termstructures/volatilities/sabrinterpolatedsmilesection.hpp File Reference

10.411.1 Detailed Description

Interpolated smile section class.

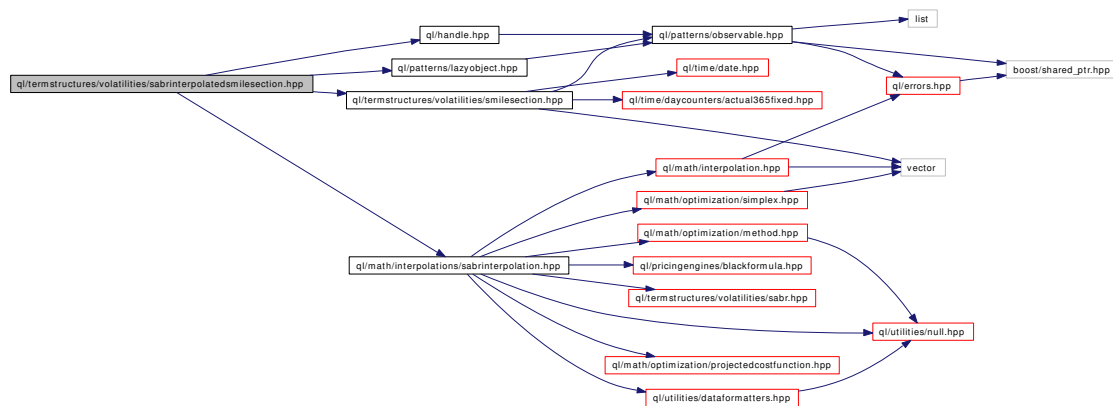
```
#include <ql/handle.hpp>
```

```
#include <ql/patterns/lazyobject.hpp>
```

```
#include <ql/termstructures/volatilities/smilesection.hpp>
```

```
#include <ql/math/interpolations/sabrinterpolation.hpp>
```

Include dependency graph for sabrinterpolatedsmilesection.hpp:



Namespaces

- namespace **QuantLib**

10.412 ql/termstructures/volatilities/smilesection.hpp File Reference

10.412.1 Detailed Description

Swaption volatility structure.

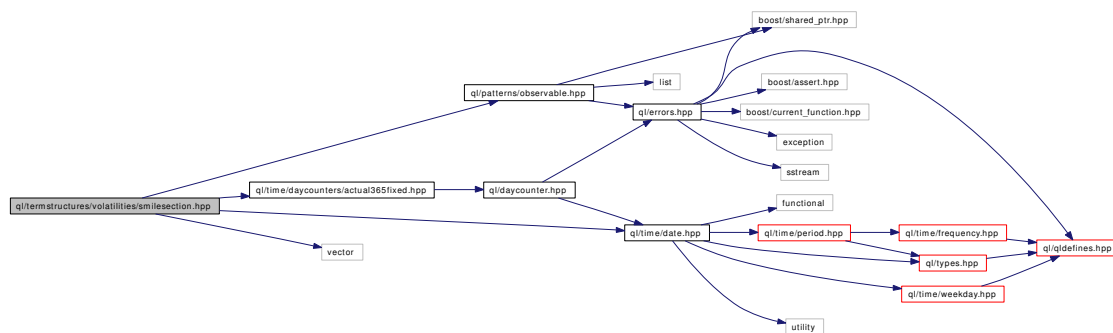
```
#include <ql/patterns/observable.hpp>
```

```
#include <ql/time/date.hpp>
```

```
#include <ql/time/daycounters/actual365fixed.hpp>
```

```
#include <vector>
```

Include dependency graph for smilesection.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SmileSection](#)
interest rate volatility smile section

10.414 ql/termstructures/volatilities/swaptionvolcube.hpp File Reference

10.414.1 Detailed Description

Swaption volatility cube.

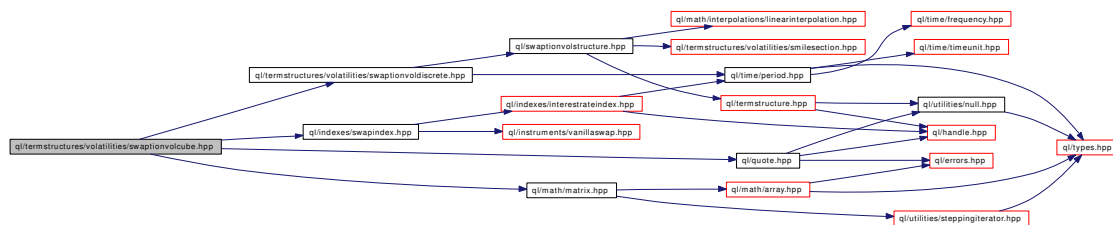
```
#include <ql/termstructures/volatilities/swaptionvoldiscrete.hpp>
```

```
#include <ql/indexes/swapindex.hpp>
```

```
#include <ql/math/matrix.hpp>
```

```
#include <ql/quote.hpp>
```

Include dependency graph for swaptionvolcube.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **SwaptionVolatilityCube**
swaption-volatility cube

10.415 ql/termstructures/volatilities/swaptionvolcube1.hpp File Reference

10.415.1 Detailed Description

Swaption volatility cube, fit-early-interpolate-later approach.

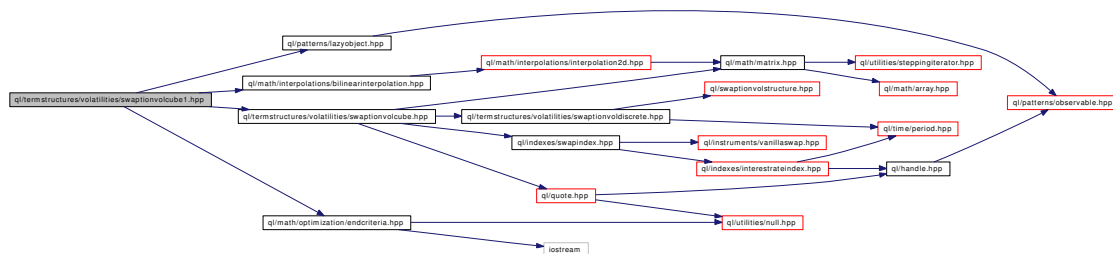
```
#include <ql/termstructures/volatilities/swaptionvolcube.hpp>
```

```
#include <ql/math/interpolations/bilinearinterpolation.hpp>
```

```
#include <ql/patterns/lazyobject.hpp>
```

```
#include <ql/math/optimization/endcriteria.hpp>
```

Include dependency graph for swaptionvolcube1.hpp:



Namespaces

- namespace **QuantLib**

10.416 ql/termstructures/volatilities/swaptionvolcube2.hpp File Reference

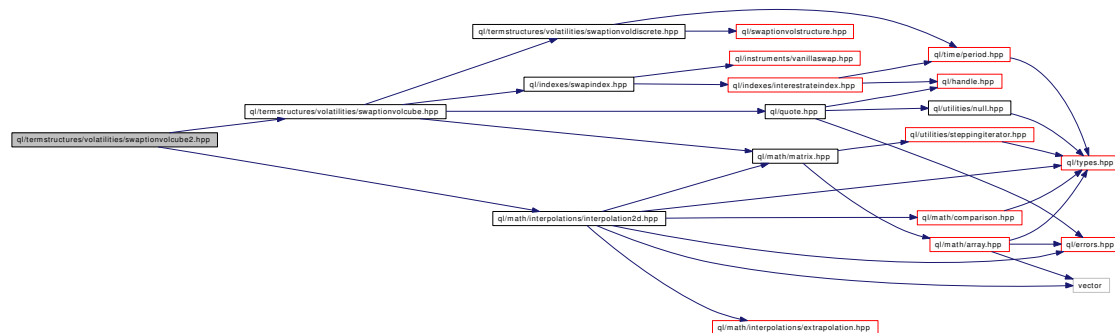
10.416.1 Detailed Description

Swaption volatility cube, fit-later-interpolate-early approach.

```
#include <ql/termstructures/volatilities/swaptionvolcube.hpp>
```

```
#include <ql/math/interpolations/interpolation2d.hpp>
```

Include dependency graph for swaptionvolcube2.hpp:



Namespaces

- namespace QuantLib

10.418 **ql/termstructures/volatilities/swaptionvolmatrix.hpp**
File Reference

10.418.1 Detailed Description

Swaption at-the-money volatility matrix.

```
#include <ql/termstructures/volatilities/swaptionvoldiscrete.hpp>
```

```
#include <ql/time/daycounters/actual365fixed.hpp>
```

```
#include <ql/math/matrix.hpp>
```

```
#include <gl/math/interpolations/bilinearinterpolation.hpp>
```

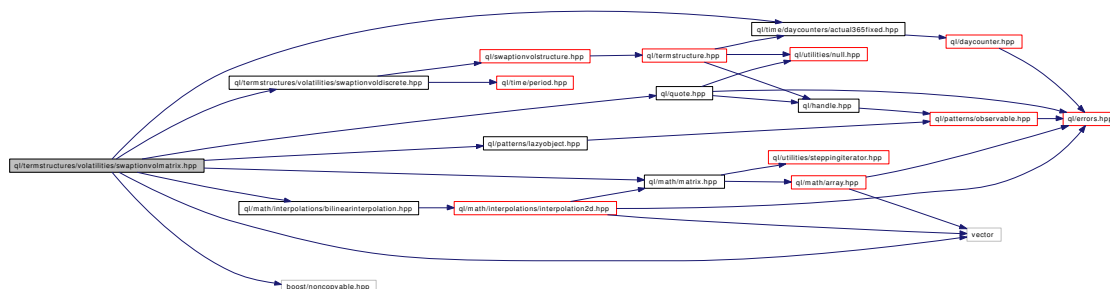
```
#include <ql/quote.hpp>
```

```
#include <ql/patterns/lazyobject.hpp>
```

```
#include <boost/noncopyable.hpp>
```

```
#include <vector>
```

Include dependency graph for swaptionvolmatrix.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class `SwaptionVolatilityMatrix`

At-the-money swaption-volatility matrix.

10.419 ql/termstructures/yieldcurves/bondhelpers.hpp File Reference

10.419.1 Detailed Description

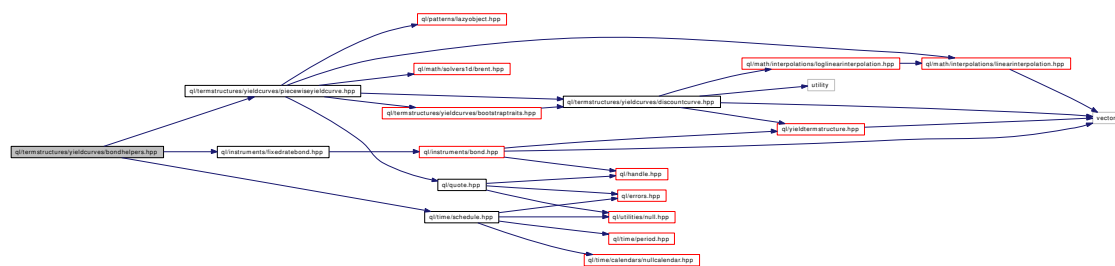
bond rate helpers

```
#include <ql/termstructures/yieldcurves/piecewiseyieldcurve.hpp>
```

```
#include <ql/instruments/fixedratebond.hpp>
```

```
#include <ql/time/schedule.hpp>
```

Include dependency graph for bondhelpers.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FixedCouponBondHelper**
fixed-coupon bond helper

10.420 ql/termstructures/yieldcurves/bootstraptraits.hpp File Reference

10.420.1 Detailed Description

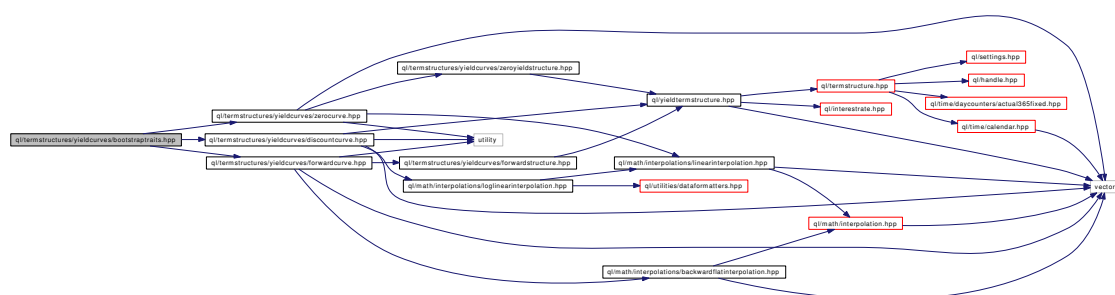
bootstrap traits

```
#include <ql/termstructures/yieldcurves/discountcurve.hpp>
```

```
#include <ql/termstructures/yieldcurves/zerocurve.hpp>
```

```
#include <ql/termstructures/yieldcurves/forwardcurve.hpp>
```

Include dependency graph for bootstraptraits.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct **Discount**
Discount-curve traits.
- struct **ZeroYield**
Zero-curve traits.
- struct **ForwardRate**
Forward-curve traits.

10.422 ql/termstructures/yieldcurves/discountcurve.hpp File Reference

10.422.1 Detailed Description

interpolated discount factor structure

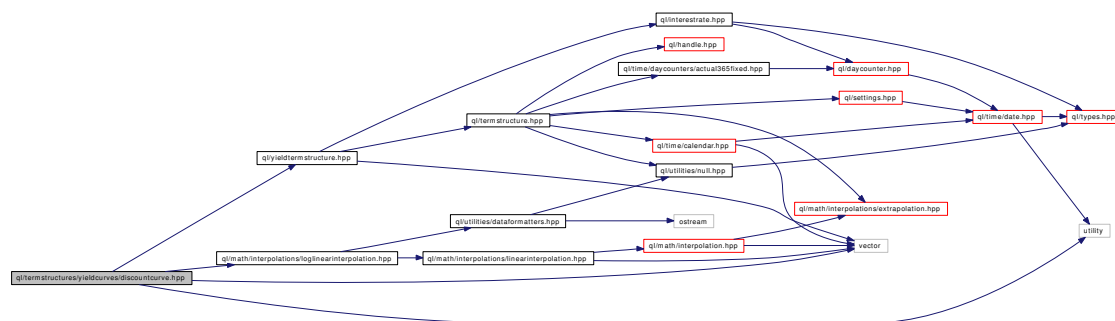
```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/math/interpolations/loglinearinterpolation.hpp>
```

```
#include <vector>
```

```
#include <utility>
```

Include dependency graph for discountcurve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InterpolatedDiscountCurve](#)
Term structure based on interpolation of discount factors.

Typedefs

- typedef [InterpolatedDiscountCurve](#)< [LogLinear](#) > [DiscountCurve](#)
Term structure based on log-linear interpolation of discount factors.

10.425 ql/termstructures/yieldcurves/flatforward.hpp File Reference

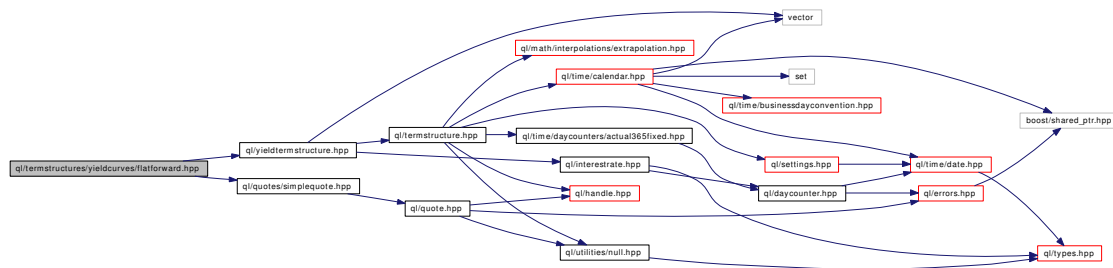
10.425.1 Detailed Description

flat forward rate term structure

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/quotes/simplequote.hpp>
```

Include dependency graph for flatforward.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FlatForward**
Flat interest-rate curve.

10.426.1 Detailed Description

```
#include <ql/termstructures/yieldcurves/forwardstructure.hpp>
#include <ql/math/interpolations/backwardflatinterpolation.hpp>
#include <vector>
#include <utility>
```

- namespace **QuantLib**

- class `InterpolatedForwardCurve`
Term structure based on interpolation of forward rates.

- `typedef InterpolatedForwardCurve< BackwardFlat > ForwardCurve`
Term structure based on flat interpolation of forward rates.

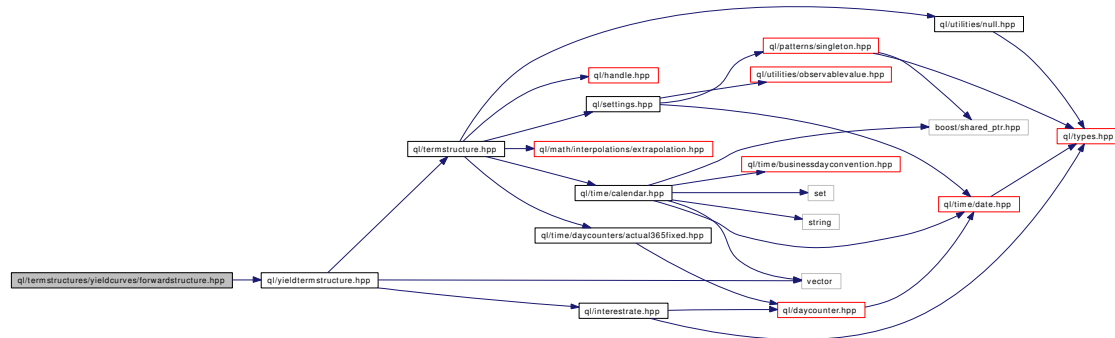
10.428 ql/termstructures/yieldcurves/forwardstructure.hpp File Reference

10.428.1 Detailed Description

Forward-based yield term structure.

```
#include <ql/yieldtermstructure.hpp>
```

Include dependency graph for forwardstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ForwardRateStructure](#)
Forward-rate term structure

10.432.1 Detailed Description

```
#include <ql/termstructures/yieldcurves/zeroyieldstructure.hpp>
#include <ql/voltermstructure.hpp>
```

[illegible]

- namespace **QuantLib**

- class `QuantoTermStructure`
Quanto term structure.

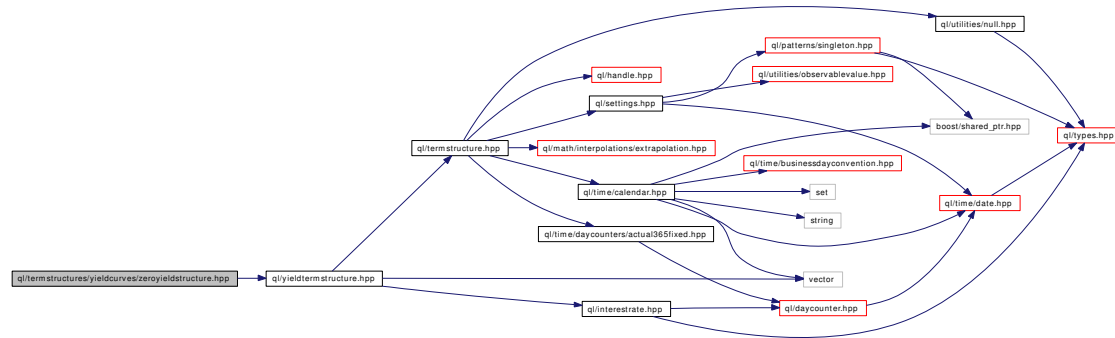
10.436 ql/termstructures/yieldcurves/zeroyieldstructure.hpp File Reference

10.436.1 Detailed Description

Zero-yield based term structure.

```
#include <ql/yieldtermstructure.hpp>
```

Include dependency graph for zeroyieldstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **ZeroYieldStructure**
Zero-yield term structure.

10.437 ql/time/businessdayconvention.hpp File Reference

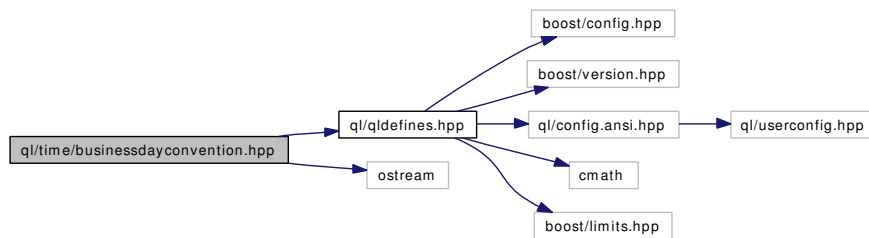
10.437.1 Detailed Description

BusinessDayConvention enumeration.

```
#include <ql/qldefines.hpp>
```

```
#include <ostream>
```

Include dependency graph for businessdayconvention.hpp:



Namespaces

- namespace **QuantLib**

Enumerations

- enum **BusinessDayConvention** {
 Following, ModifiedFollowing, Preceding, ModifiedPreceding,
 Unadjusted }
 Business Day conventions.

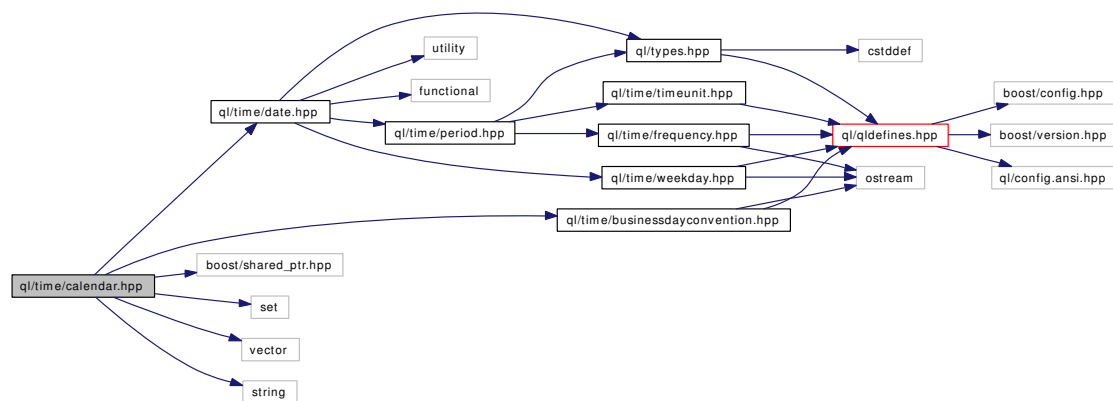
10.438 ql/time/calendar.hpp File Reference

10.438.1 Detailed Description

calendar class

```
#include <ql/time/date.hpp>
#include <ql/time/businessdayconvention.hpp>
#include <boost/shared_ptr.hpp>
#include <set>
#include <vector>
#include <string>
```

Include dependency graph for calendar.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Calendar**
calendar class
- class **Calendar::Impl**
abstract base class for calendar implementations
- class **Calendar::WesternImpl**
partial calendar implementation
- class **Calendar::OrthodoxImpl**
partial calendar implementation

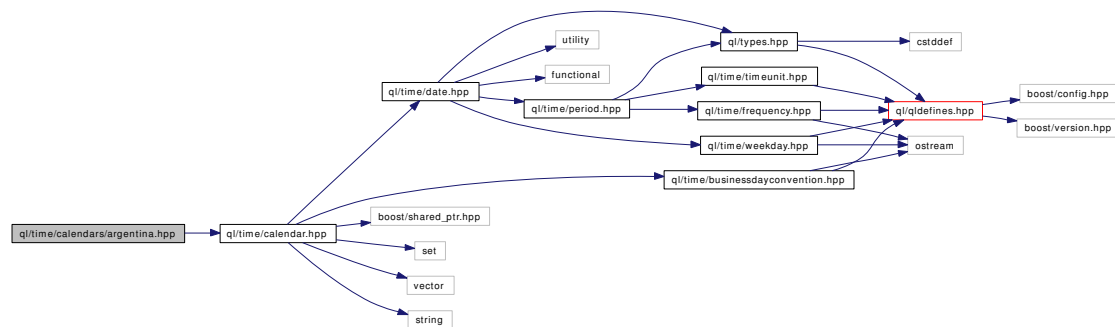
10.439 ql/time/calendars/argentina.hpp File Reference

10.439.1 Detailed Description

Argentinian calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for argentina.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Argentina](#)
Argentinian calendars.

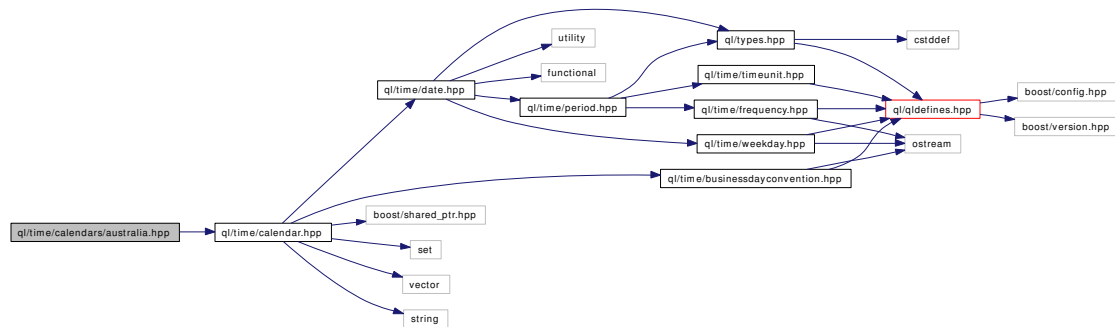
10.440 ql/time/calendars/australia.hpp File Reference

10.440.1 Detailed Description

Australian calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for australia.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Australia](#)
Australian calendar.

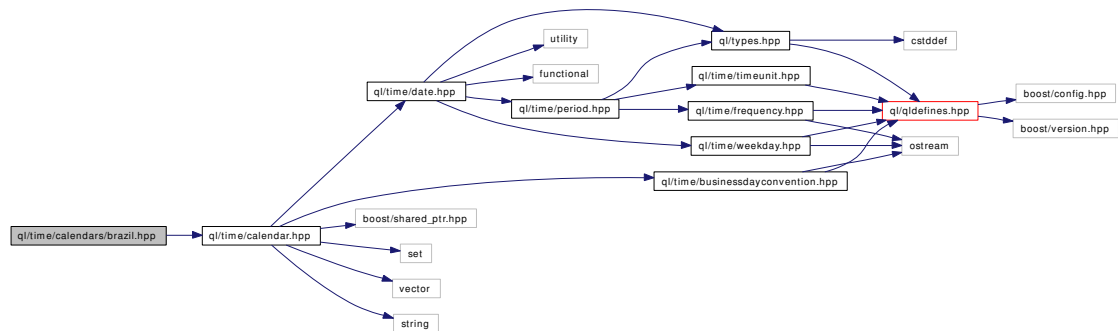
10.441 ql/time/calendars/brazil.hpp File Reference

10.441.1 Detailed Description

Brazilian calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for brazil.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Brazil**
Brazilian calendar.

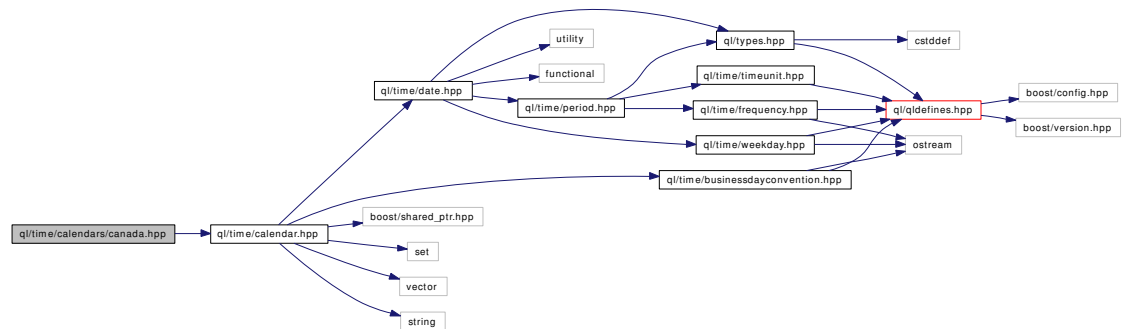
10.442 ql/time/calendars/canada.hpp File Reference

10.442.1 Detailed Description

Canadian calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for canada.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Canada**
Canadian calendar.

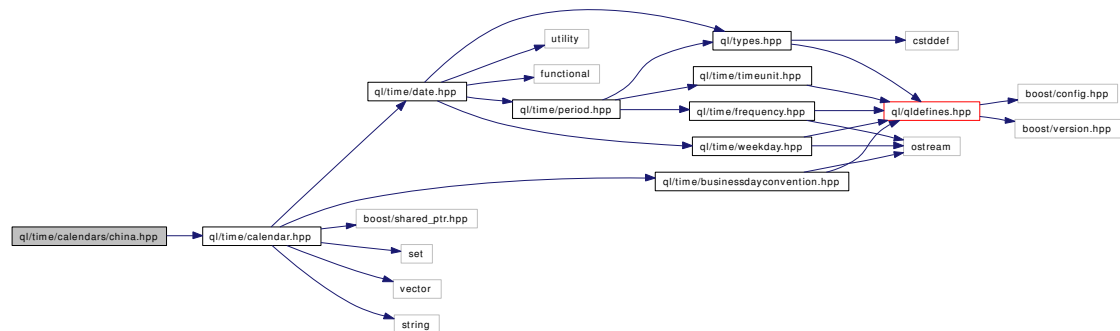
10.443 ql/time/calendars/china.hpp File Reference

10.443.1 Detailed Description

Chinese calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for china.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **China**
Chinese calendar.

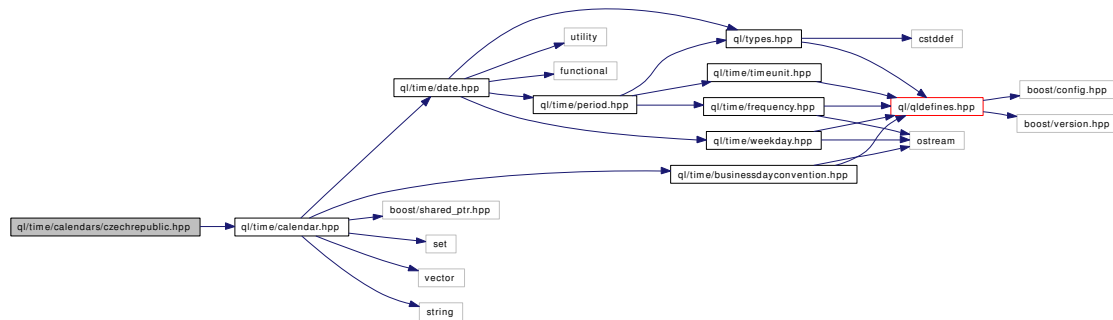
10.444 ql/time/calendars/czechrepublic.hpp File Reference

10.444.1 Detailed Description

Czech calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for czechrepublic.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CzechRepublic**
Czech calendars.

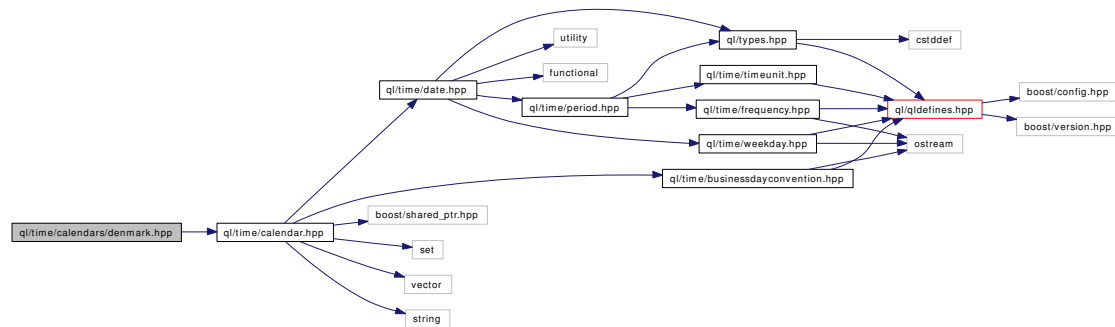
10.445 ql/time/calendars/denmark.hpp File Reference

10.445.1 Detailed Description

Danish calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for `denmark.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [Denmark](#)
Danish calendar.

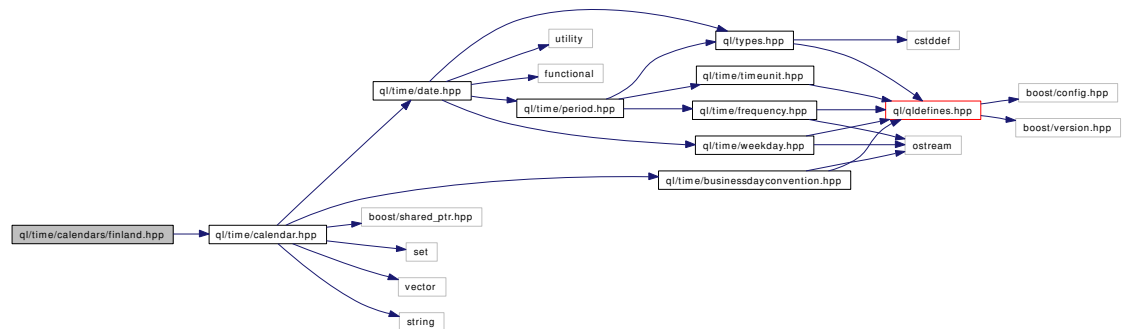
10.446 ql/time/calendars/finland.hpp File Reference

10.446.1 Detailed Description

Finnish calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for finland.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Finland](#)
Finnish calendar.

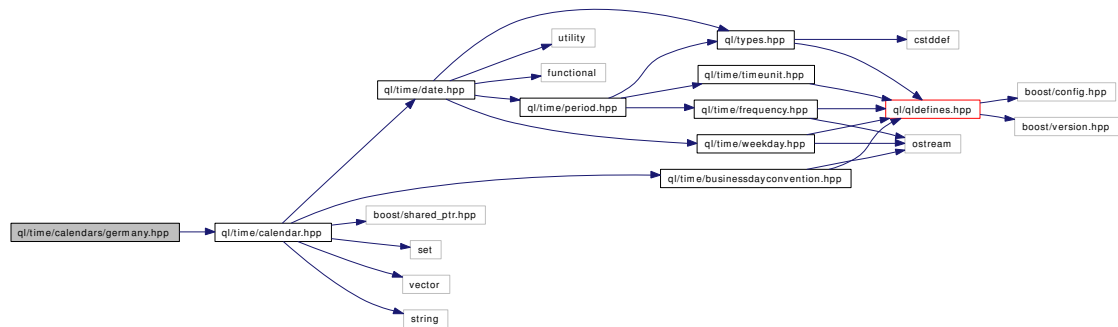
10.447 ql/time/calendars/germany.hpp File Reference

10.447.1 Detailed Description

German calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for germany.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Germany](#)
German calendars.

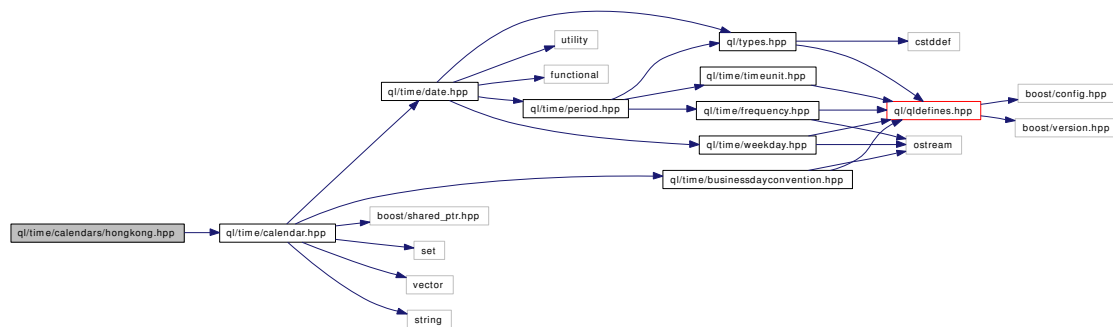
10.448 ql/time/calendars/hongkong.hpp File Reference

10.448.1 Detailed Description

Hong Kong calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for hongkong.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **HongKong**
Hong Kong calendars.

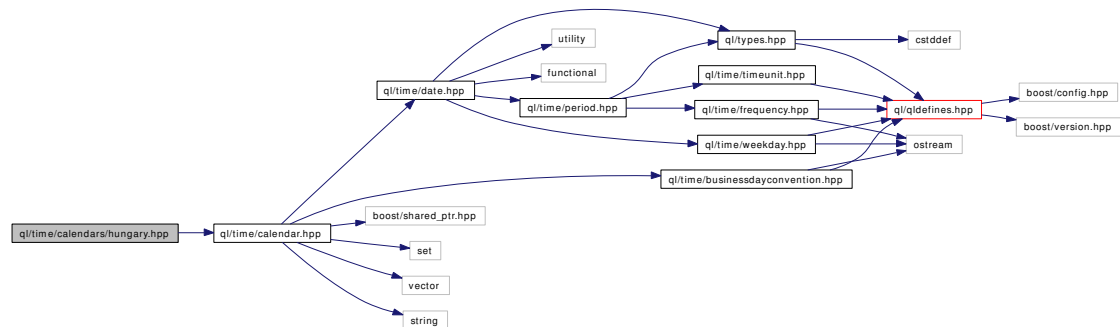
10.449 ql/time/calendars/hungary.hpp File Reference

10.449.1 Detailed Description

Hungarian calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for hungary.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Hungary**
Hungarian calendar.

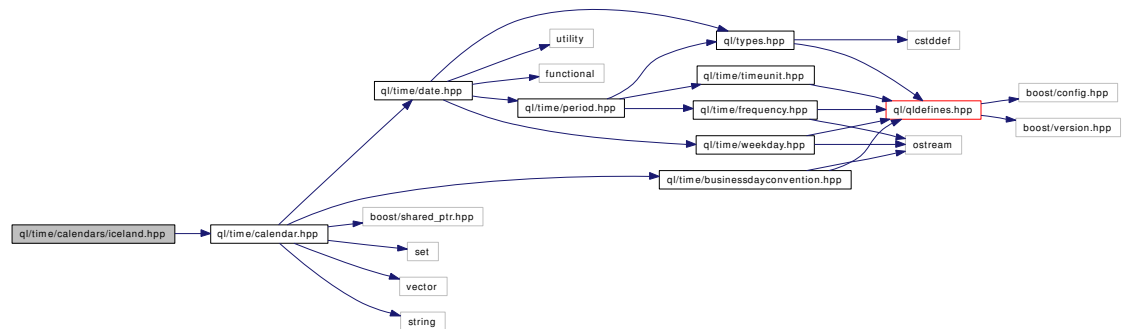
10.450 ql/time/calendars/iceland.hpp File Reference

10.450.1 Detailed Description

Icelandic calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for iceland.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Iceland](#)
Icelandic calendars.

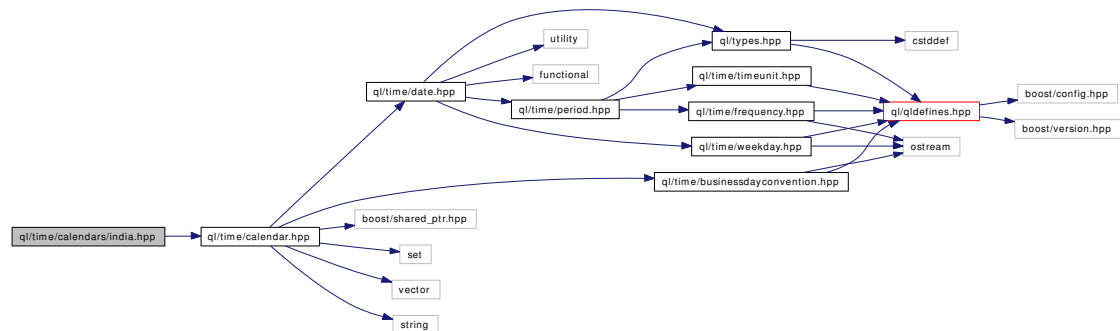
10.451 ql/time/calendars/india.hpp File Reference

10.451.1 Detailed Description

Indian calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for india.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [India](#)
Indian calendars.

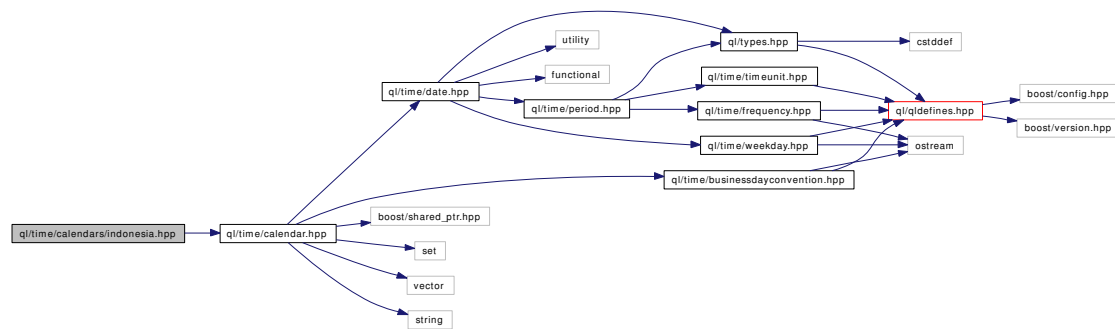
10.452 ql/time/calendars/indonesia.hpp File Reference

10.452.1 Detailed Description

Indonesian calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for indonesia.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Indonesia](#)
Indonesian calendars

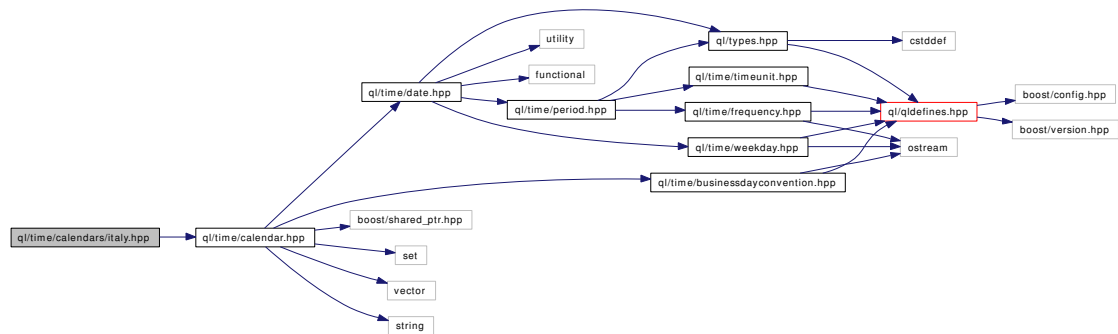
10.453 ql/time/calendars/italy.hpp File Reference

10.453.1 Detailed Description

Italian calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for italy.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Italy**
Italian calendars.

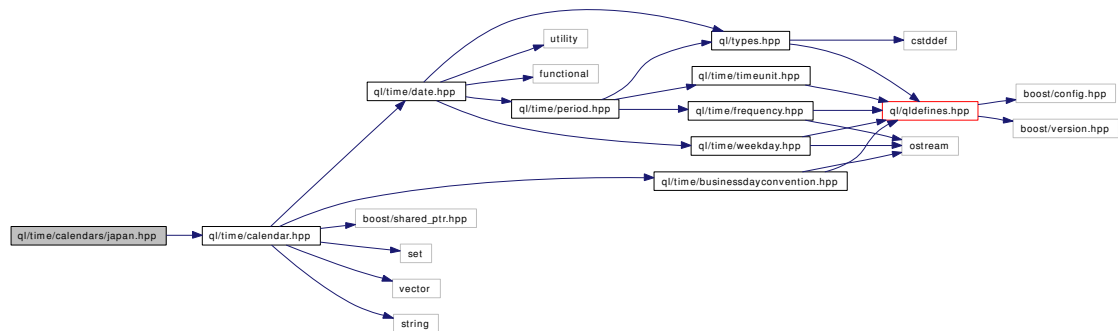
10.454 ql/time/calendars/japan.hpp File Reference

10.454.1 Detailed Description

Japanese calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for japan.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Japan](#)
Japanese calendar.

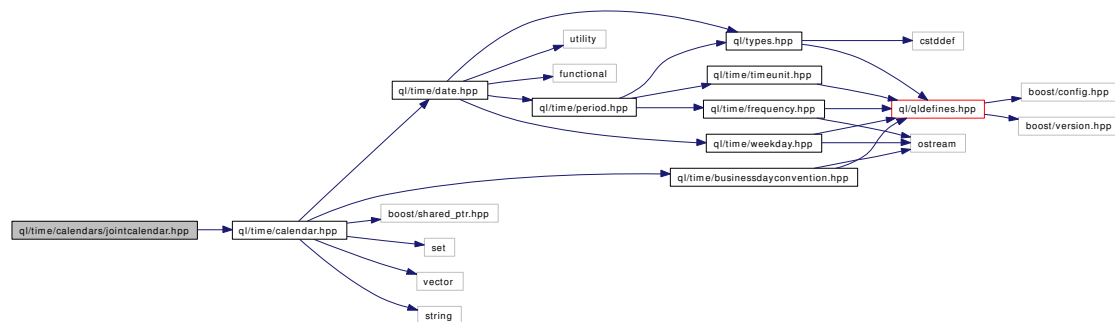
10.455 ql/time/calendars/jointcalendar.hpp File Reference

10.455.1 Detailed Description

Joint calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for jointcalendar.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [JointCalendar](#)
Joint calendar.

Enumerations

- enum **JointCalendarRule** { [JoinHolidays](#), [JoinBusinessDays](#) }
rules for joining calendars

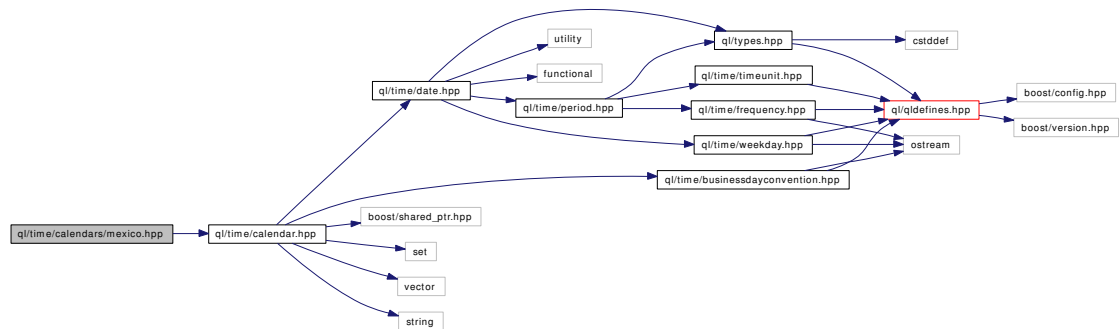
10.456 ql/time/calendars/mexico.hpp File Reference

10.456.1 Detailed Description

Mexican calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for mexico.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Mexico](#)
Mexican calendars

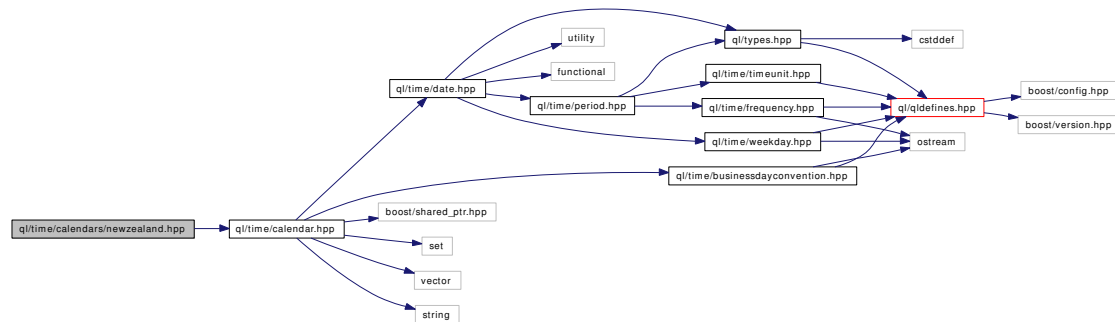
10.457 ql/time/calendars/newzealand.hpp File Reference

10.457.1 Detailed Description

New Zealand calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for newzealand.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [NewZealand](#)
New Zealand calendar.

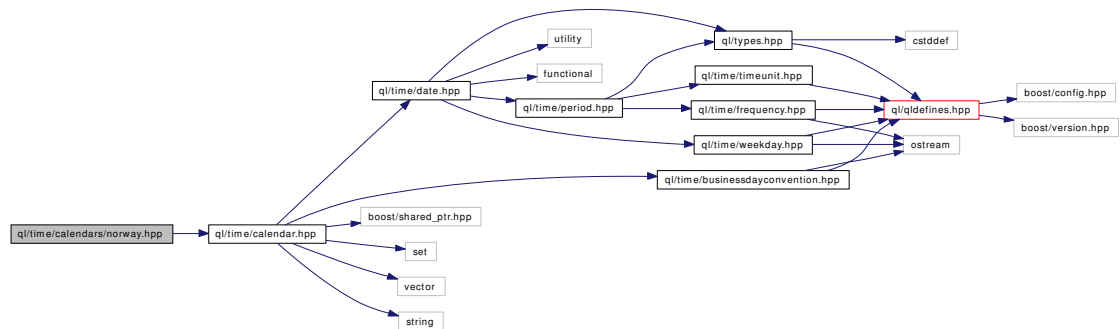
10.458 ql/time/calendars/norway.hpp File Reference

10.458.1 Detailed Description

Norwegian calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for norway.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Norway](#)
Norwegian calendar.

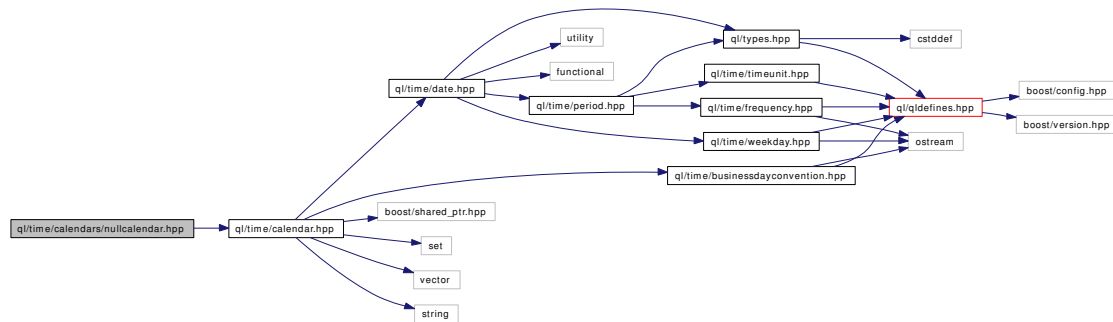
10.459 ql/time/calendars/nullcalendar.hpp File Reference

10.459.1 Detailed Description

Calendar for reproducing theoretical calculations.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for nullcalendar.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **NullCalendar**
Calendar for reproducing theoretical calculations.

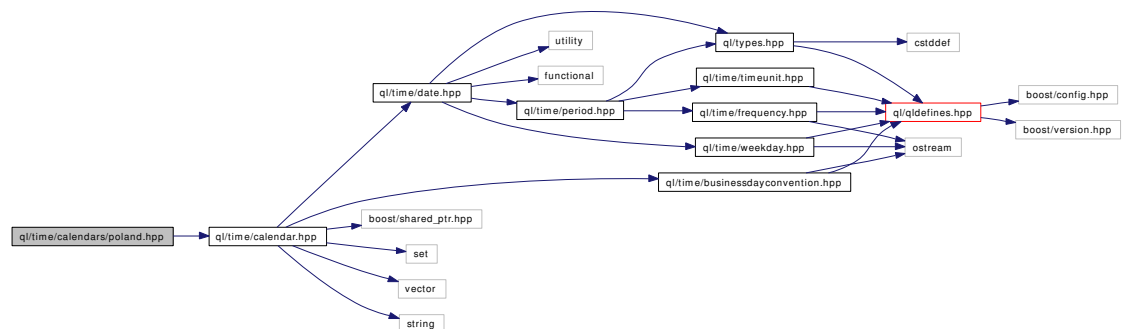
10.460 ql/time/calendars/poland.hpp File Reference

10.460.1 Detailed Description

Polish calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for poland.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Poland**
Polish calendar.

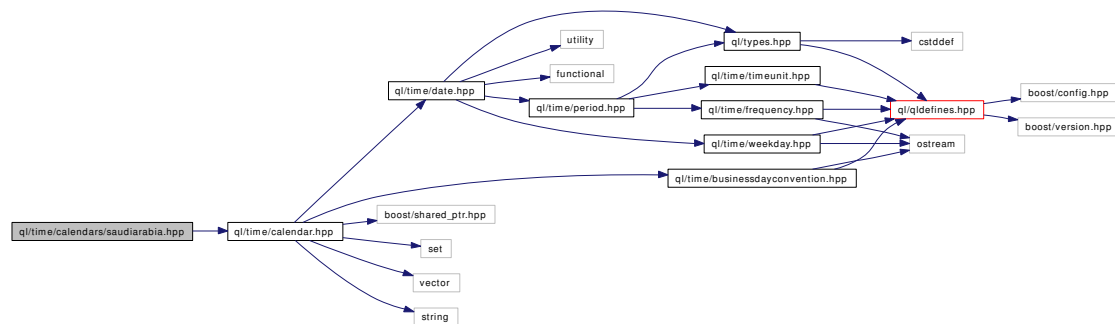
10.461 ql/time/calendars/saudiarabia.hpp File Reference

10.461.1 Detailed Description

Saudi Arabian calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for saudiarabia.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SaudiArabia](#)
Saudi Arabian calendar.

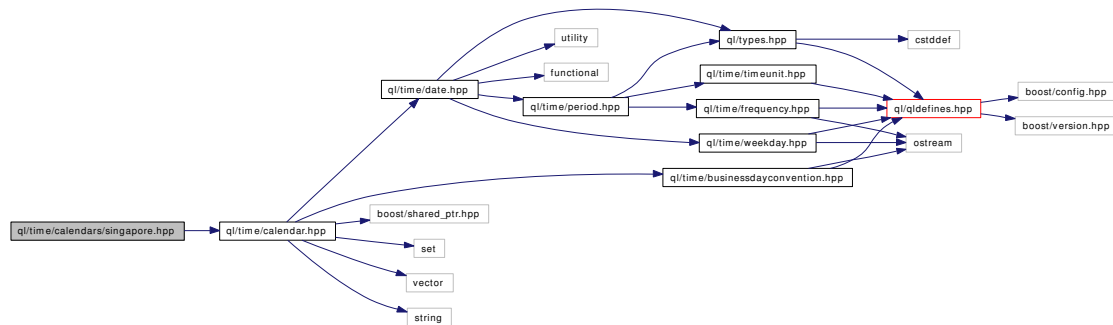
10.462 ql/time/calendars/singapore.hpp File Reference

10.462.1 Detailed Description

Singapore calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for singapore.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Singapore](#)
Singapore calendars

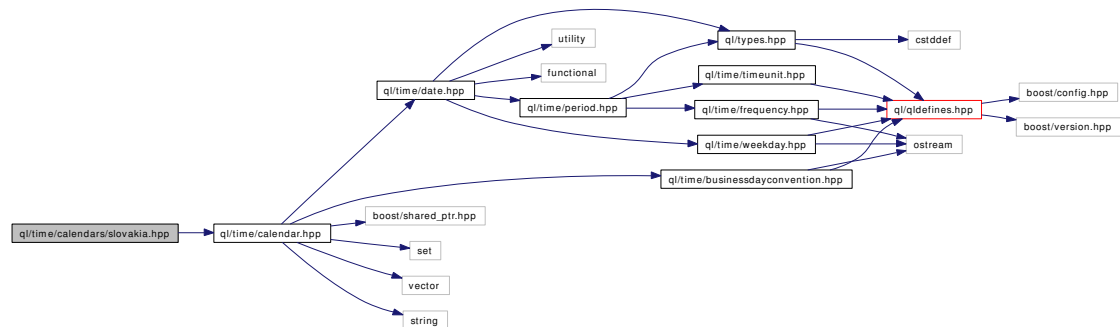
10.463 ql/time/calendars/slovakia.hpp File Reference

10.463.1 Detailed Description

Slovak calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for slovakia.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Slovakia**
Slovak calendars.

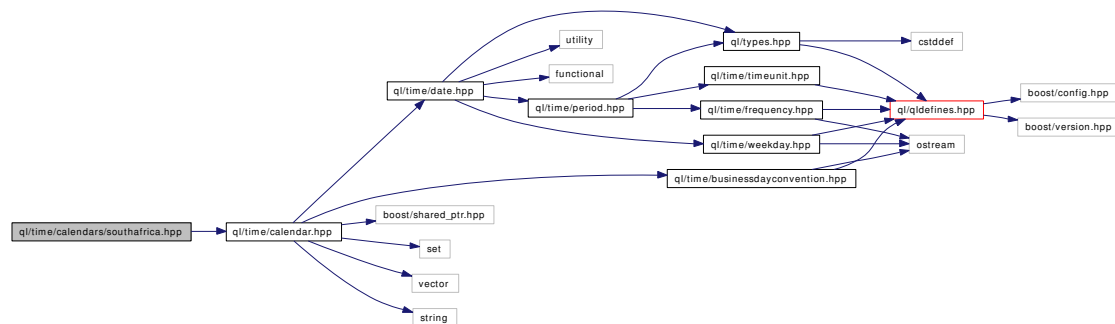
10.464 ql/time/calendars/southafrica.hpp File Reference

10.464.1 Detailed Description

South-African calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for southafrica.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SouthAfrica](#)
South-African calendar.

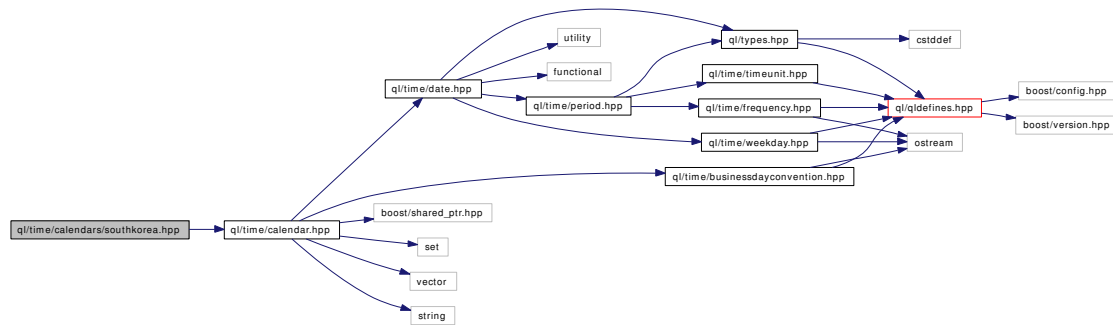
10.465 ql/time/calendars/southkorea.hpp File Reference

10.465.1 Detailed Description

South Korean calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for southkorea.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **SouthKorea**
South Korean calendars.

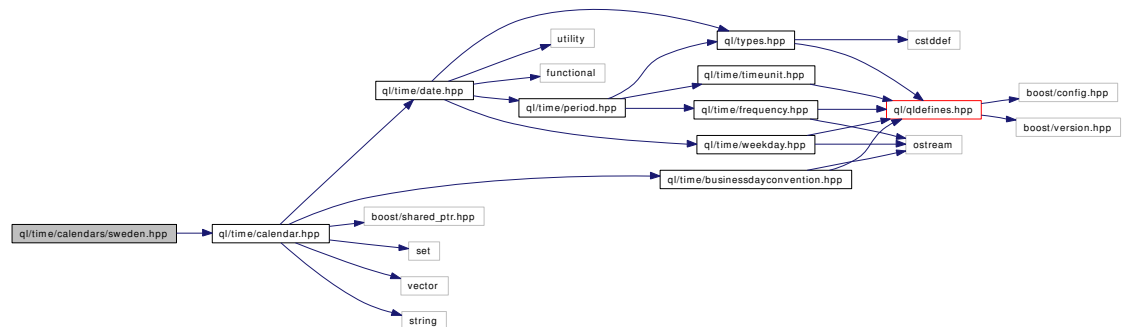
10.466 ql/time/calendars/sweden.hpp File Reference

10.466.1 Detailed Description

Swedish calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for sweden.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Sweden](#)
Swedish calendar.

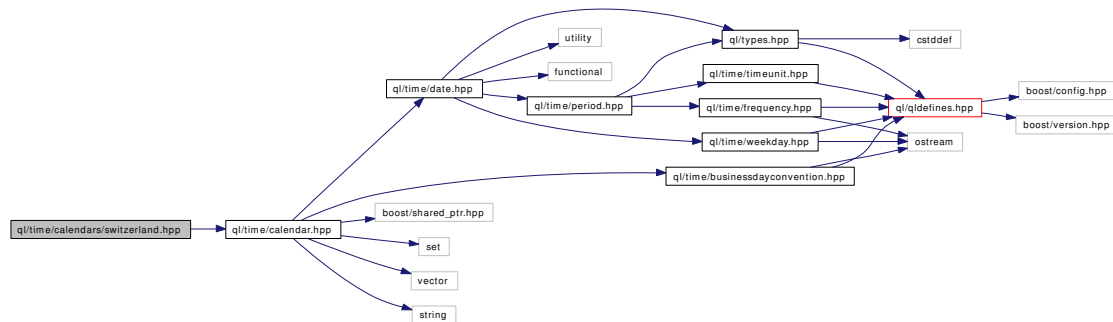
10.467 ql/time/calendars/switzerland.hpp File Reference

10.467.1 Detailed Description

Swiss calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for switzerland.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Switzerland**
Swiss calendar.

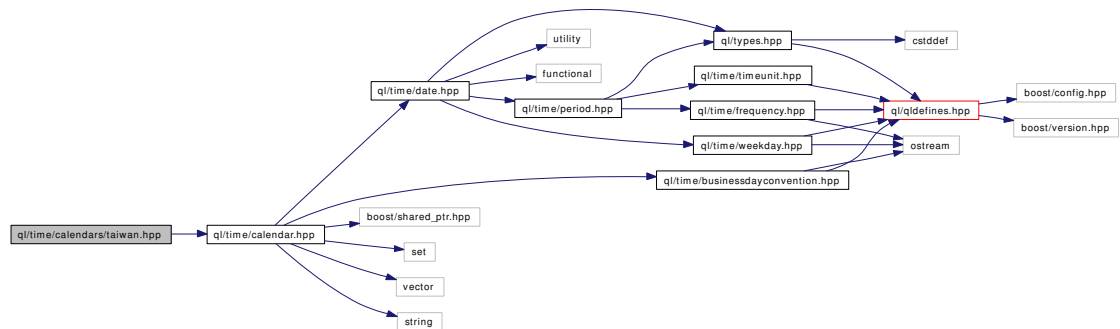
10.468 ql/time/calendars/taiwan.hpp File Reference

10.468.1 Detailed Description

Taiwanese calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for taiwan.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Taiwan**

Taiwanese calendars.

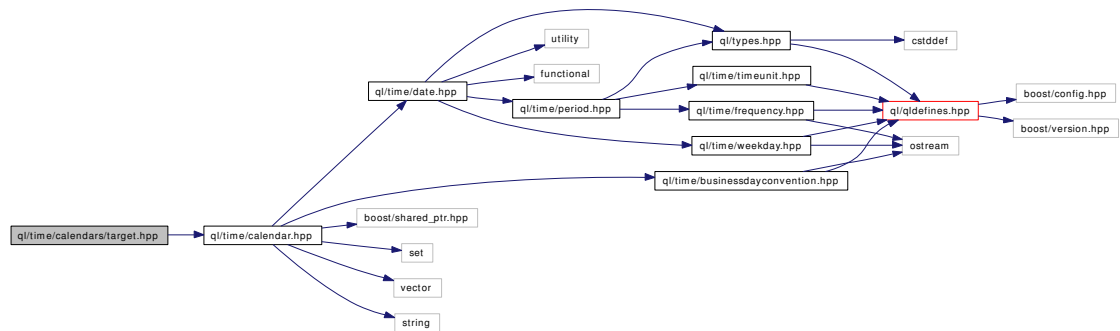
10.469 ql/time/calendars/target.hpp File Reference

10.469.1 Detailed Description

TARGET calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for target.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TARGET**
TARGET calendar

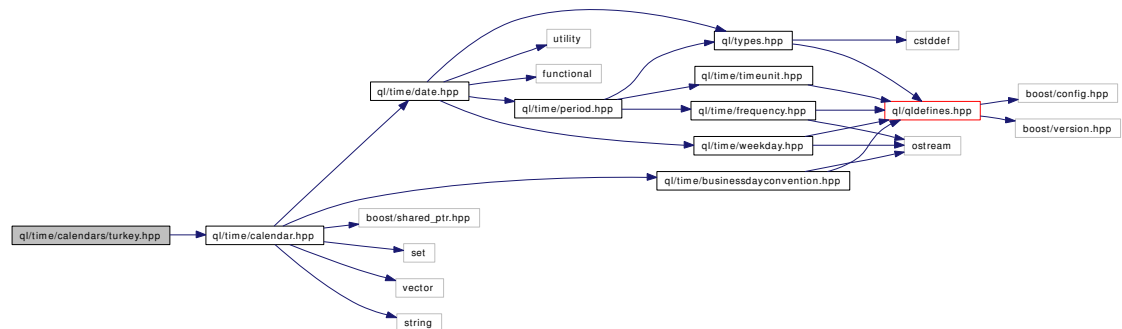
10.470 ql/time/calendars/turkey.hpp File Reference

10.470.1 Detailed Description

Turkish calendar.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for turkey.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Turkey**
Turkish calendar.

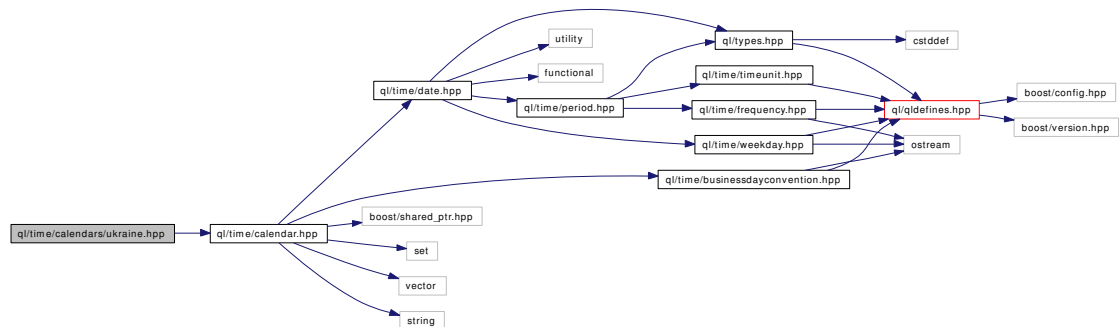
10.471 ql/time/calendars/ukraine.hpp File Reference

10.471.1 Detailed Description

Ukrainian calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for ukraine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Ukraine](#)
Ukrainian calendars.

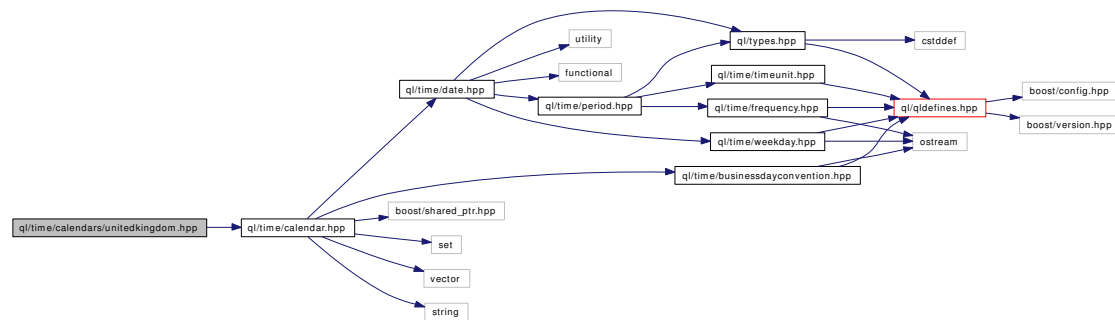
10.472 ql/time/calendars/unitedkingdom.hpp File Reference

10.472.1 Detailed Description

UK calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for unitedkingdom.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **UnitedKingdom**
United Kingdom calendars.

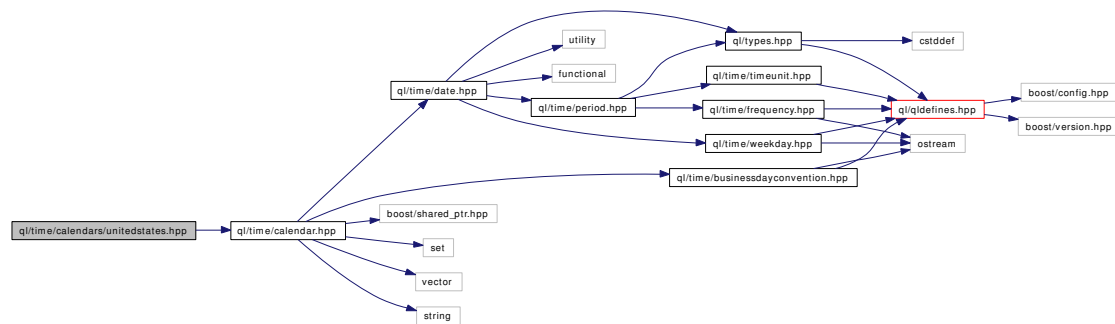
10.473 ql/time/calendars/unitedstates.hpp File Reference

10.473.1 Detailed Description

US calendars.

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for unitedstates.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **UnitedStates**
United States calendars.

10.474 ql/time/date.hpp File Reference

10.474.1 Detailed Description

date- and time-related classes, typedefs and enumerations

```
#include <ql/time/period.hpp>
```

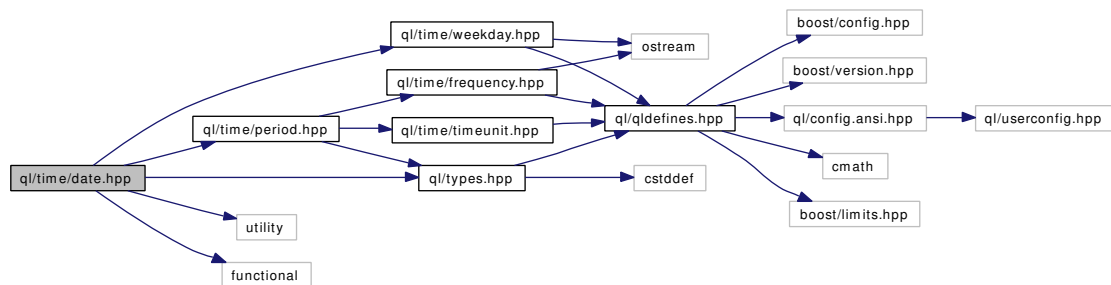
```
#include <ql/time/weekday.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <utility>
```

```
#include <functional>
```

Include dependency graph for date.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**
- namespace **QuantLib::io**

Classes

- class [Date](#)
Concrete date class.

Typedefs

- typedef Integer [Day](#)
Day number.
- typedef Integer [Year](#)
Year number.

Enumerations

- enum [Month](#) {
 January = 1, **February** = 2, **March** = 3, **April** = 4,
 May = 5, **June** = 6, **July** = 7, **August** = 8,
 September = 9, **October** = 10, **November** = 11, **December** = 12,
 Jan = 1, **Feb** = 2, **Mar** = 3, **Apr** = 4,
 Jun = 6, **Jul** = 7, **Aug** = 8, **Sep** = 9,
 Oct = 10, **Nov** = 11, **Dec** = 12 }
 Month names.

Functions

- `std::ostream & operator<< (std::ostream &, const short_date_holder &)`
- `std::ostream & operator<< (std::ostream &, const long_date_holder &)`
- `std::ostream & operator<< (std::ostream &, const iso_date_holder &)`
- `detail::short_date_holder short_date (const Date &)`
 output dates in short format (mm/dd/yyyy)
- `detail::long_date_holder long_date (const Date &)`
 output dates in long format (Month ddth, yyyy)
- `detail::iso_date_holder iso_date (const Date &)`
 output dates in ISO format (yyyy-mm-dd)
- `BigInteger operator- (const Date &d1, const Date &d2)`
- `bool operator== (const Date &d1, const Date &d2)`
- `bool operator!= (const Date &d1, const Date &d2)`
- `bool operator< (const Date &d1, const Date &d2)`
- `bool operator<= (const Date &d1, const Date &d2)`
- `bool operator> (const Date &d1, const Date &d2)`
- `bool operator>= (const Date &d1, const Date &d2)`

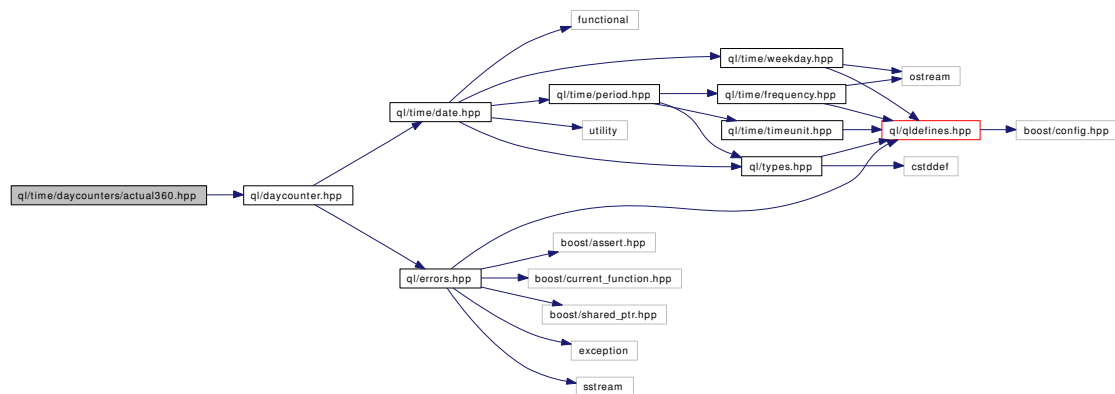
10.475 ql/time/daycounters/actual360.hpp File Reference

10.475.1 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.

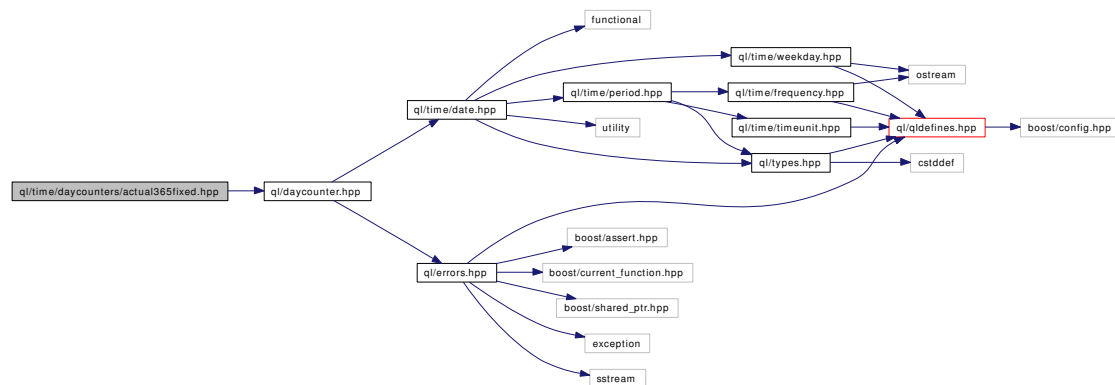
10.476 ql/time/daycounters/actual365fixed.hpp File Reference

10.476.1 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.

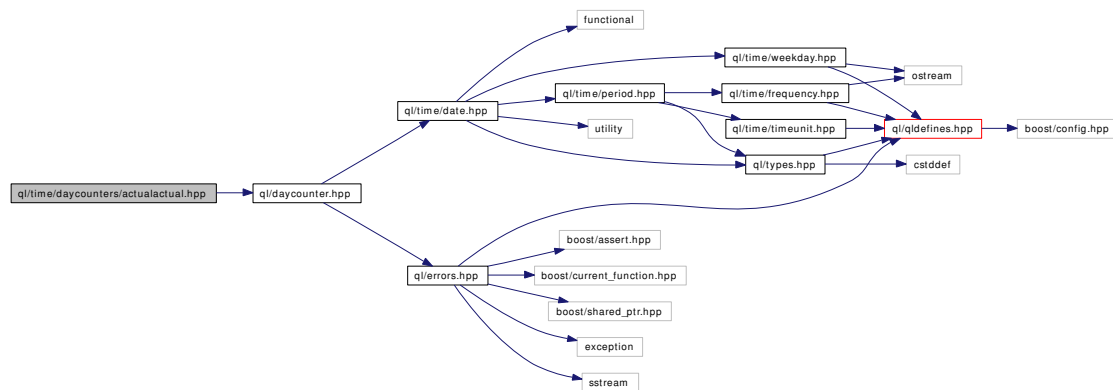
10.477 ql/time/daycounters/actualactual.hpp File Reference

10.477.1 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.

10.478 ql/time/daycounters/business252.hpp File Reference

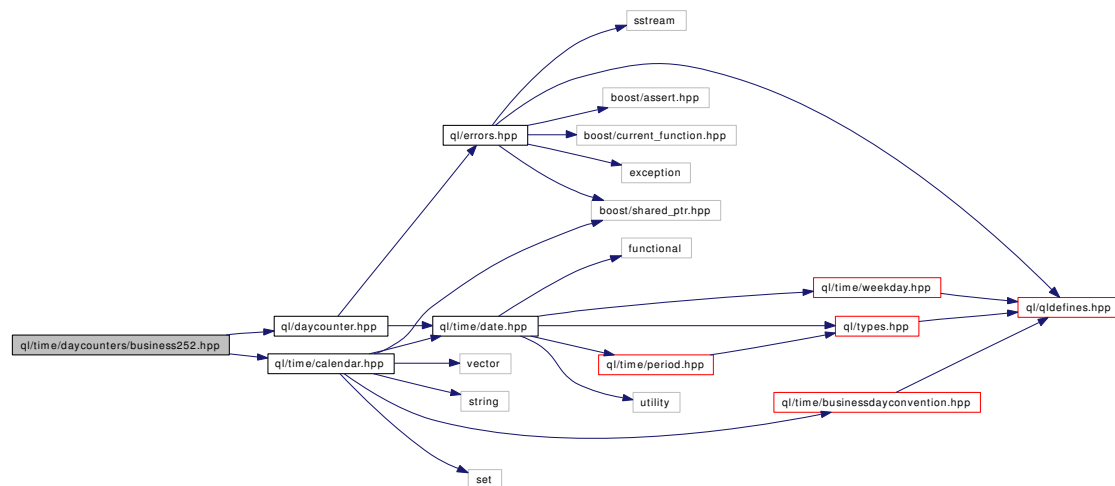
10.478.1 Detailed Description

business/252 day counter

```
#include <ql/daycounter.hpp>
```

```
#include <ql/time/calendar.hpp>
```

Include dependency graph for business252.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Business252](#)
Business/252 day count convention.

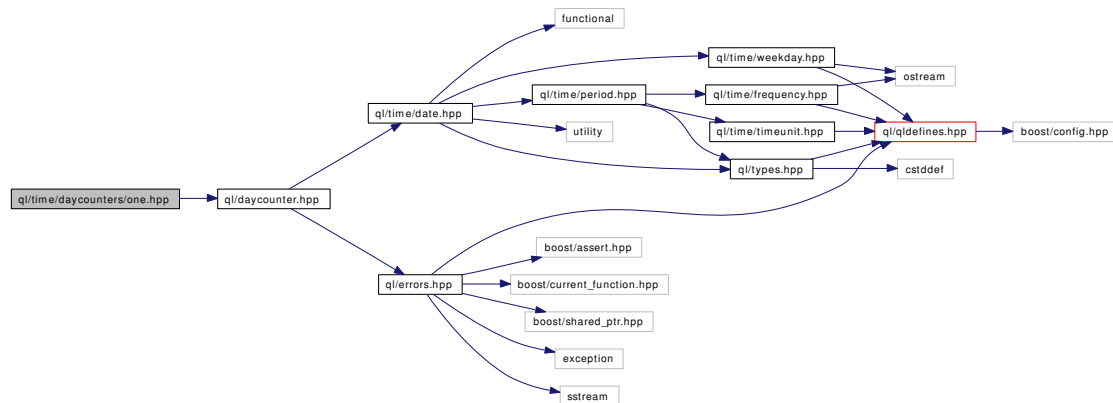
10.479 ql/time/daycounters/one.hpp File Reference

10.479.1 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

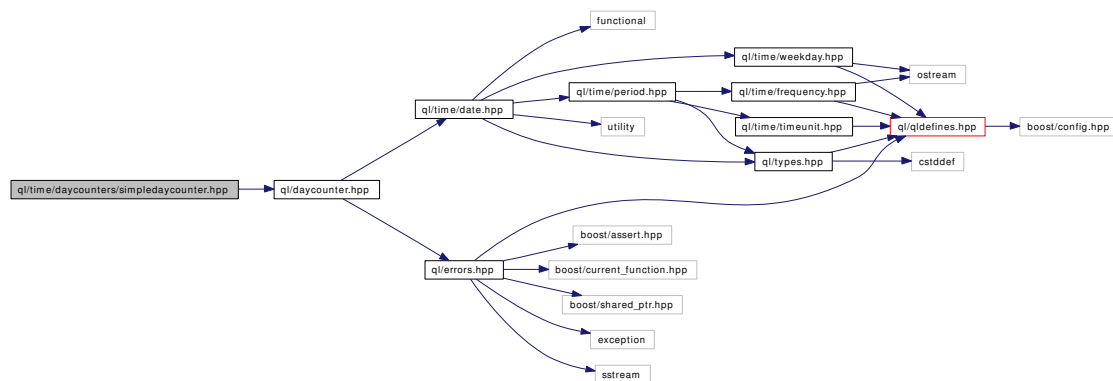
10.480 ql/time/daycounters/simpliedaycounter.hpp File Reference

10.480.1 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.

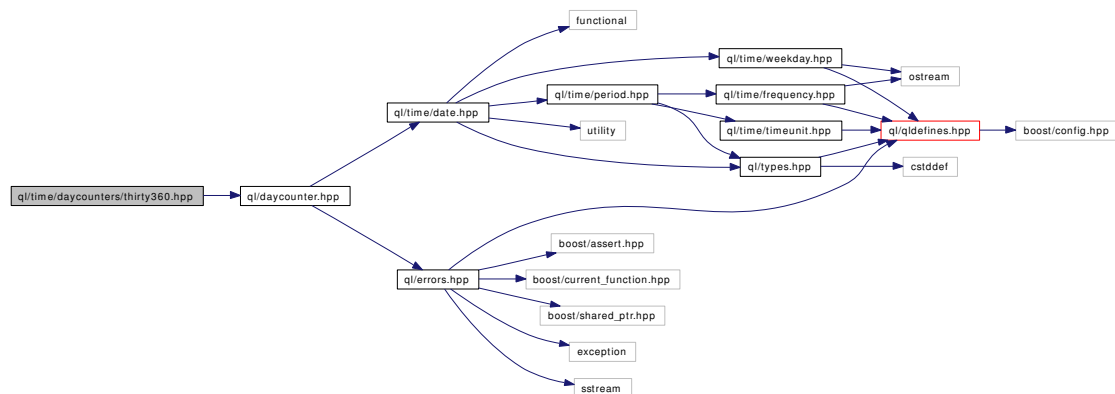
10.481 ql/time/daycounters/thirty360.hpp File Reference

10.481.1 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

10.482 ql/time/frequency.hpp File Reference

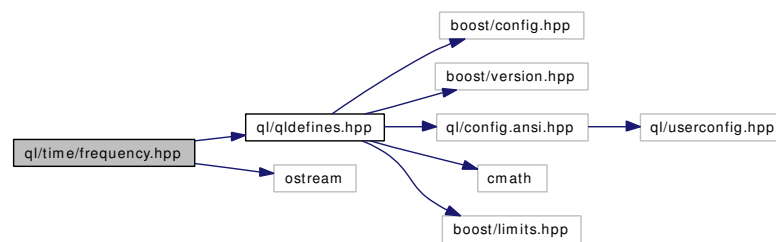
10.482.1 Detailed Description

Frequency enumeration.

```
#include <ql/qldefines.hpp>
```

```
#include <ostream>
```

Include dependency graph for frequency.hpp:



Namespaces

- namespace **QuantLib**

Enumerations

- enum **Frequency** {
 NoFrequency = -1, **Once** = 0, **Annual** = 1, **Semiannual** = 2,
 EveryFourthMonth = 3, **Quarterly** = 4, **Bimonthly** = 6, **Monthly** = 12,
 Biweekly = 26, **Weekly** = 52, **Daily** = 365 }
 Frequency of events.

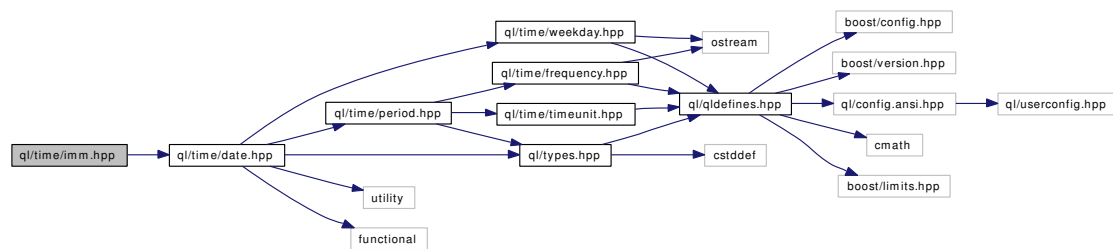
10.483 ql/time/imm.hpp File Reference

10.483.1 Detailed Description

IMM-related date functions.

```
#include <ql/time/date.hpp>
```

Include dependency graph for imm.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct [IMM](#)

Main cycle of the International Money Market (a.k.a. IMM) months.

10.484 ql/time/period.hpp File Reference

10.484.1 Detailed Description

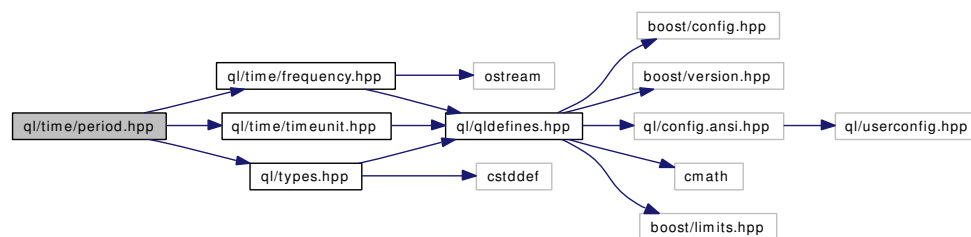
period- and frequency-related classes and enumerations

```
#include <ql/time/frequency.hpp>
```

```
#include <ql/time/timeunit.hpp>
```

```
#include <ql/types.hpp>
```

Include dependency graph for period.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**
- namespace **QuantLib::io**

Classes

- class **Period**
Time period described by a number of a given time unit.

Functions

- `std::ostream & operator<< (std::ostream &, const long_period_holder &)`
- `std::ostream & operator<< (std::ostream &, const short_period_holder &)`
- `detail::long_period_holder long_period (const Period &)`
output periods in long format (e.g. "2 weeks")
- `detail::short_period_holder short_period (const Period &)`
output periods in short format (e.g. "2w")
- `template<typename T>`
`Period operator * (T n, TimeUnit units)`
- `template<typename T>`
`Period operator * (TimeUnit units, T n)`
- `Period operator- (const Period &p)`
- `Period operator * (Integer n, const Period &p)`

- Period **operator** * (const Period &p, Integer n)
- bool **operator**== (const Period &p1, const Period &p2)
- bool **operator**!= (const Period &p1, const Period &p2)
- bool **operator**> (const Period &p1, const Period &p2)
- bool **operator**<= (const Period &p1, const Period &p2)
- bool **operator**>= (const Period &p1, const Period &p2)

10.485 ql/time/schedule.hpp File Reference

10.485.1 Detailed Description

date schedule

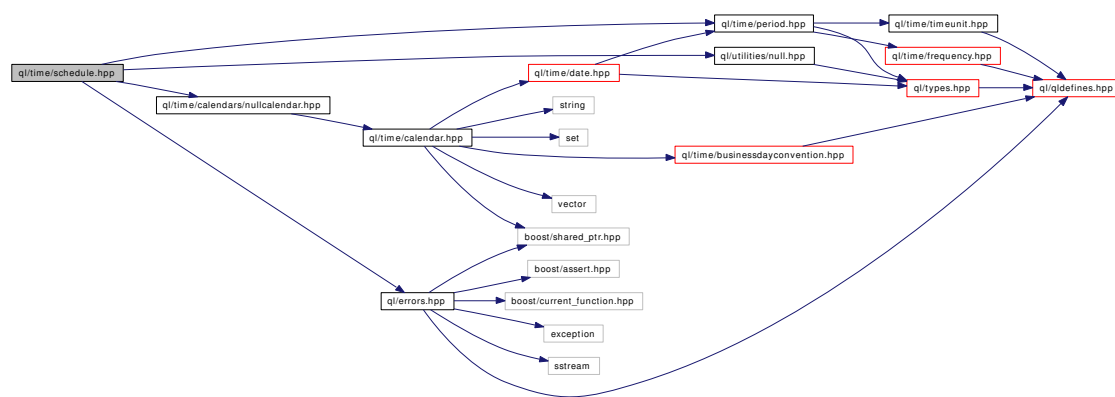
```
#include <ql/time/calendars/nullcalendar.hpp>
```

```
#include <ql/utilities/null.hpp>
```

```
#include <ql/time/period.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for schedule.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Schedule](#)
Payment schedule.
- class [MakeSchedule](#)
helper class

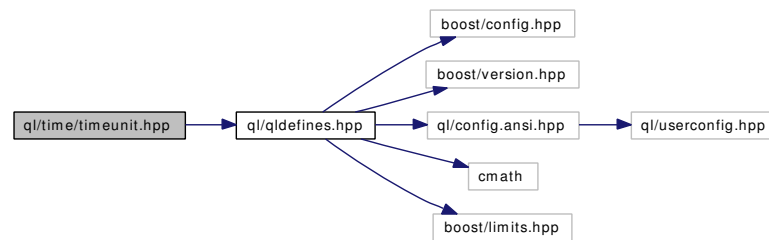
10.486 ql/time/timeunit.hpp File Reference

10.486.1 Detailed Description

TimeUnit enumeration.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for timeunit.hpp:



Namespaces

- namespace **QuantLib**

Enumerations

- enum **TimeUnit** { **Days**, **Weeks**, **Months**, **Years** }
Units used to describe time periods.

10.487 ql/time/weekday.hpp File Reference

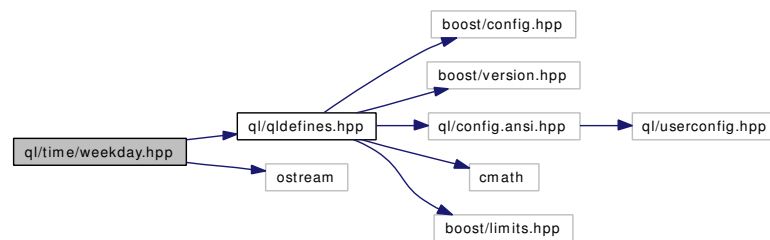
10.487.1 Detailed Description

Weekday enumeration.

```
#include <ql/qldefines.hpp>
```

```
#include <ostream>
```

Include dependency graph for weekday.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**
- namespace **QuantLib::io**

Enumerations

- enum [Weekday](#) {
Sunday = 1, **Monday** = 2, **Tuesday** = 3, **Wednesday** = 4,
Thursday = 5, **Friday** = 6, **Saturday** = 7, **Sun** = 1,
Mon = 2, **Tue** = 3, **Wed** = 4, **Thu** = 5,
Fri = 6, **Sat** = 7 }

Functions

- std::ostream & **operator<<** (std::ostream &, const long_weekday_holder &)
- std::ostream & **operator<<** (std::ostream &, const short_weekday_holder &)
- std::ostream & **operator<<** (std::ostream &, const shortest_weekday_holder &)
- detail::long_weekday_holder [long_weekday](#) (Weekday)
output weekdays in long format
- detail::short_weekday_holder [short_weekday](#) (Weekday)
output weekdays in short format (three letters)
- detail::shortest_weekday_holder [shortest_weekday](#) (Weekday)
output weekdays in shortest format (two letters)

10.488 ql/timegrid.hpp File Reference

10.488.1 Detailed Description

discrete time grid

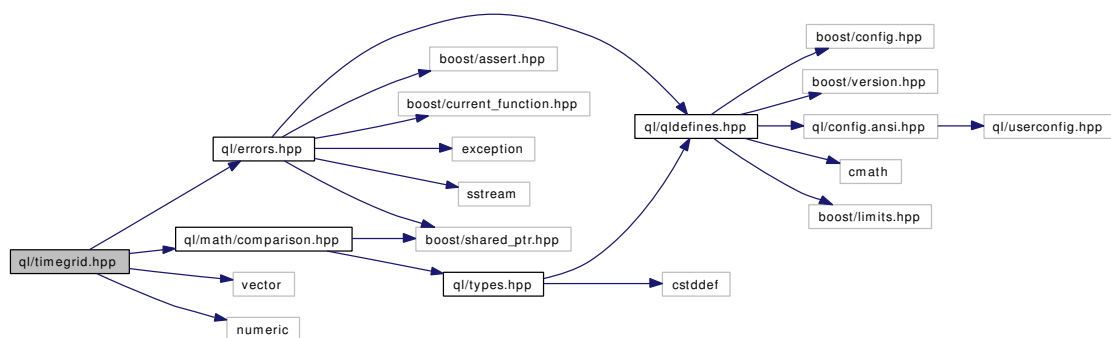
```
#include <ql/errors.hpp>
```

```
#include <ql/math/comparison.hpp>
```

```
#include <vector>
```

```
#include <numeric>
```

Include dependency graph for timegrid.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TimeGrid**
time grid class

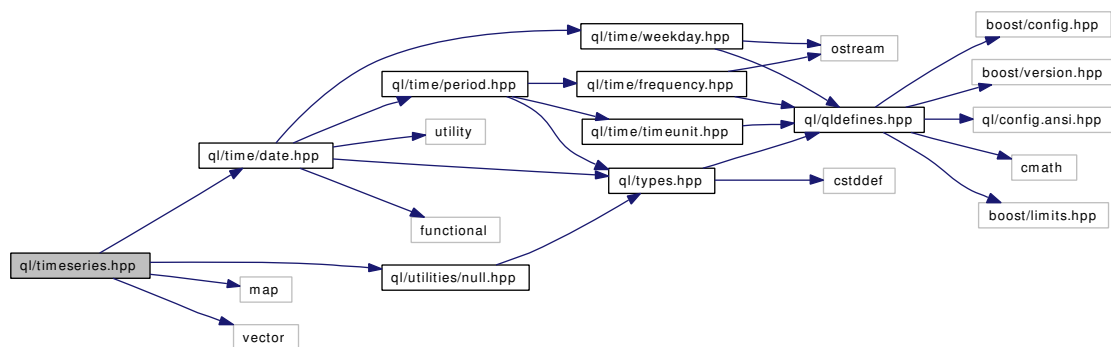
10.489 ql/timeseries.hpp File Reference

10.489.1 Detailed Description

Container for historical data.

```
#include <ql/time/date.hpp>
#include <ql/utilities/null.hpp>
#include <map>
#include <vector>
```

Include dependency graph for timeseries.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TimeSeries](#)
Container for historical data.

10.490 ql/types.hpp File Reference

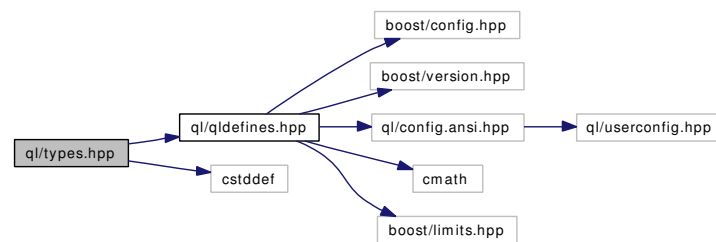
10.490.1 Detailed Description

Custom types.

```
#include <ql/qldefines.hpp>
```

```
#include <cstdint>
```

Include dependency graph for types.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef `QL_INTEGER` [Integer](#)
integer number
- typedef `QL_BIG_INTEGER` [BigInteger](#)
large integer number
- typedef `unsigned QL_INTEGER` [Natural](#)
positive integer
- typedef `unsigned QL_BIG_INTEGER` [BigNatural](#)
large positive integer
- typedef `QL_REAL` [Real](#)
real number
- typedef `Real` [Decimal](#)
decimal number
- typedef `std::size_t` [Size](#)
size of a container
- typedef `Real` [Time](#)
continuous quantity with 1-year units

- typedef Real [DiscountFactor](#)
discount factor between dates
- typedef Real [Rate](#)
interest rates
- typedef Real [Spread](#)
spreads on interest rates
- typedef Real [Volatility](#)
volatility

10.491 ql/utilities/clone.hpp File Reference

10.491.1 Detailed Description

cloning proxy to an underlying object

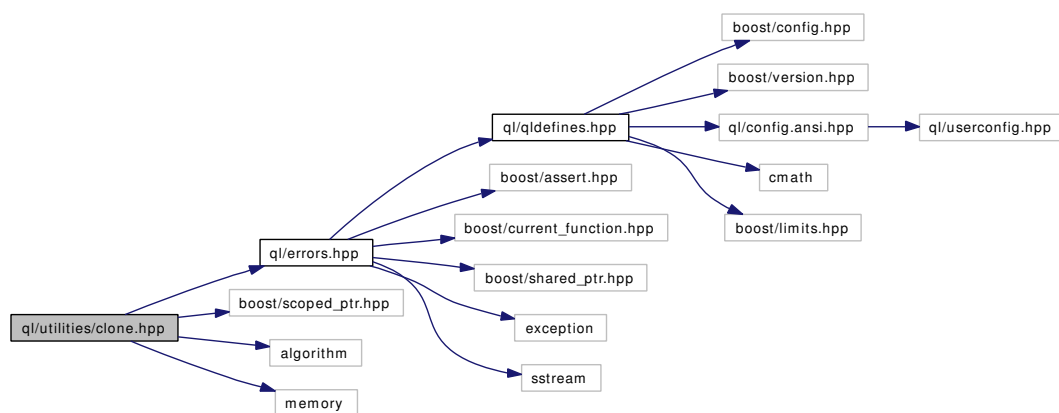
```
#include <ql/errors.hpp>
```

```
#include <boost/scoped_ptr.hpp>
```

```
#include <algorithm>
```

```
#include <memory>
```

Include dependency graph for clone.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Clone](#)
cloning proxy to an underlying object

10.492 ql/utilities/dataformatters.hpp File Reference

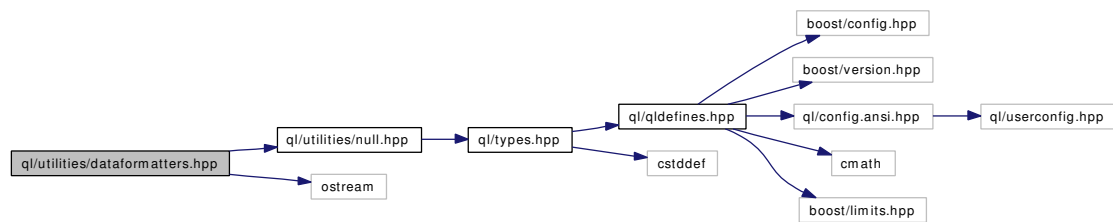
10.492.1 Detailed Description

output manipulators

```
#include <ql/utilities/null.hpp>
```

```
#include <ostream>
```

Include dependency graph for dataformatters.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**
- namespace **QuantLib::io**

Functions

- template<typename T>
std::ostream & **operator**<< (std::ostream &, const null_checker< T > &)
- std::ostream & **operator**<< (std::ostream &, const ordinal_holder &)
- template<typename T>
std::ostream & **operator**<< (std::ostream &, const power_of_two_holder< T > &)
- std::ostream & **operator**<< (std::ostream &, const percent_holder &)
- template<typename T>
detail::null_checker< T > [checknull](#) (T)
check for nulls before output
- detail::ordinal_holder [ordinal](#) (Size)
outputs naturals as 1st, 2nd, 3rd...
- template<typename T>
detail::power_of_two_holder< T > [power_of_two](#) (T)
output integers as powers of two
- detail::percent_holder [percent](#) (Real)
output reals as percentages
- detail::percent_holder [rate](#) (Rate)
output rates and spreads as percentages

- detail::percent_holder [volatility](#) (Volatility)
output volatilities as percentages

10.493 ql/utilities/dataparsers.hpp File Reference

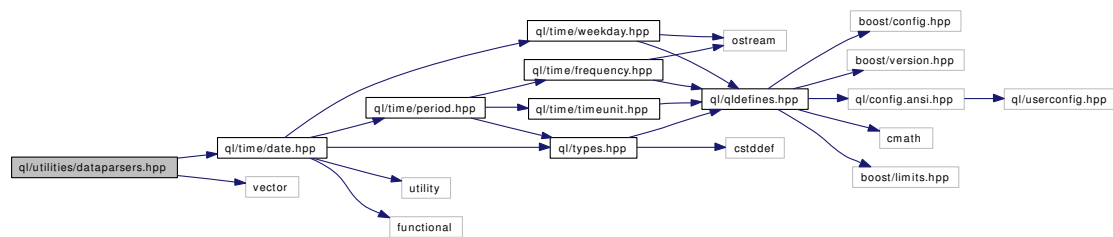
10.493.1 Detailed Description

Classes used to parse data for input.

```
#include <ql/time/date.hpp>
```

```
#include <vector>
```

Include dependency graph for dataparsers.hpp:



Namespaces

- namespace **QuantLib**

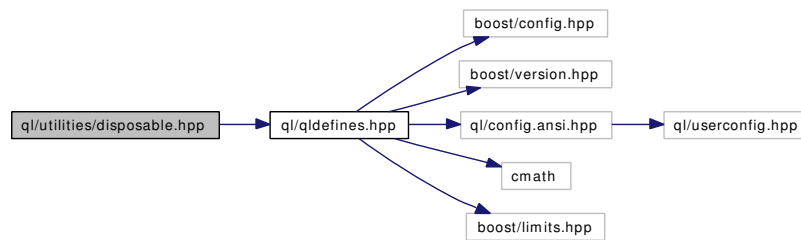
10.494 ql/utilities/disposable.hpp File Reference

10.494.1 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

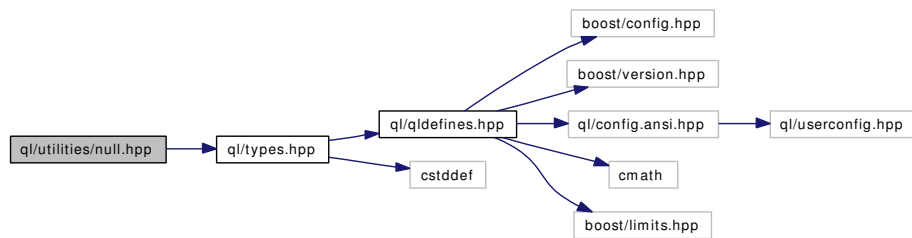
10.495 ql/utilities/null.hpp File Reference

10.495.1 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.

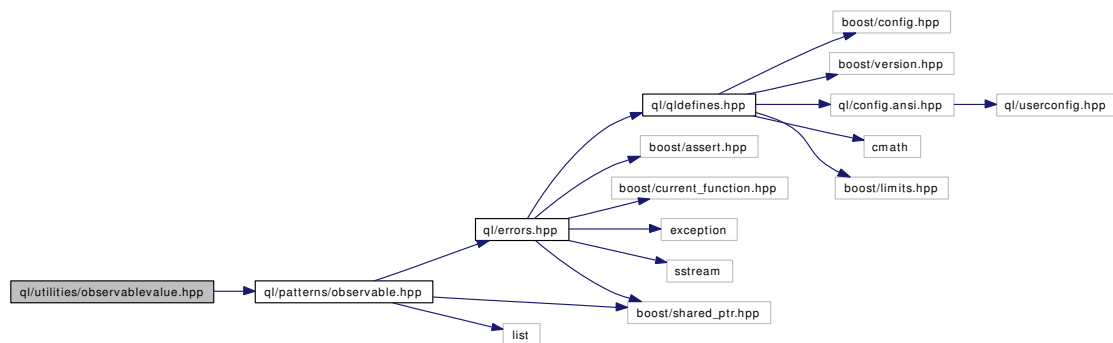
10.496 ql/utilities/observablevalue.hpp File Reference

10.496.1 Detailed Description

observable and assignable proxy to concrete value

```
#include <ql/patterns/observable.hpp>
```

Include dependency graph for observablevalue.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **ObservableValue**
observable and assignable proxy to concrete value

10.497 ql/utilities/steppingiterator.hpp File Reference

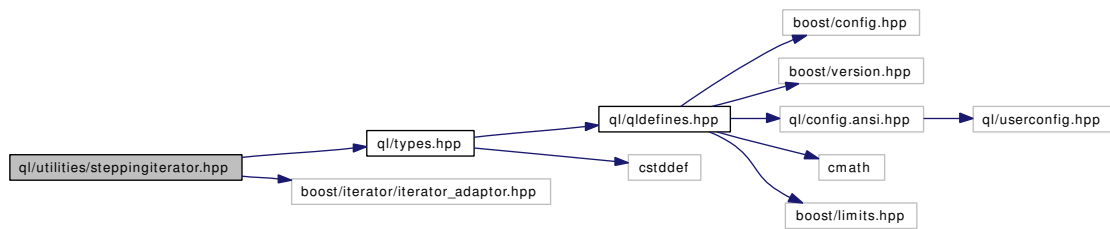
10.497.1 Detailed Description

Iterator advancing in constant steps.

```
#include <ql/types.hpp>
```

```
#include <boost/iterator/iterator_adaptor.hpp>
```

Include dependency graph for steppingiterator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [step_iterator](#)
Iterator advancing in constant steps.

10.498 ql/utilities/tracing.hpp File Reference

10.498.1 Detailed Description

tracing facilities

```
#include <ql/types.hpp>
```

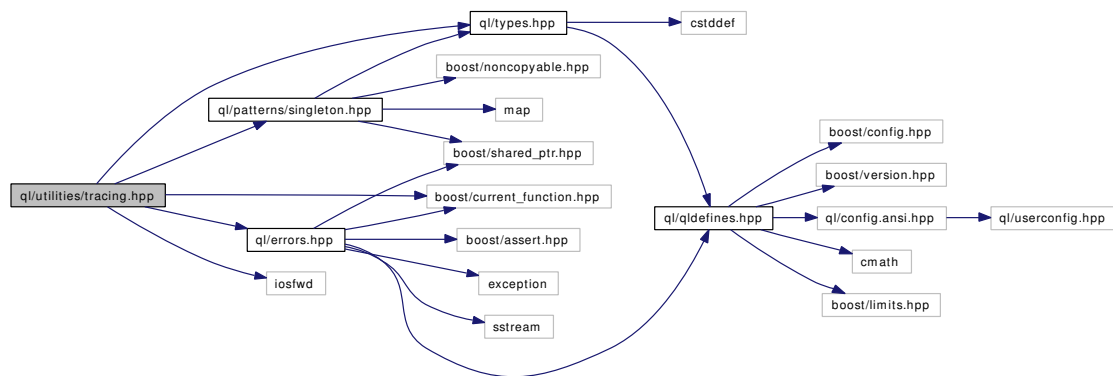
```
#include <ql/errors.hpp>
```

```
#include <ql/patterns/singleton.hpp>
```

```
#include <boost/current_function.hpp>
```

```
#include <iosfwd>
```

Include dependency graph for tracing.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Defines

- #define **QL_DEFAULT_TRACER**
- #define **QL_TRACE_ENABLE**
enable tracing
- #define **QL_TRACE_DISABLE**
disable tracing
- #define **QL_TRACE_ON**(out)
set tracing stream
- #define **QL_TRACE**(message)
output tracing information
- #define **QL_TRACE_ENTER_FUNCTION**

output tracing information

- #define [QL_TRACE_EXIT_FUNCTION](#)
output tracing information
- #define [QL_TRACE_LOCATION](#)
output tracing information
- #define [QL_TRACE_VARIABLE\(variable\)](#)
output tracing information

10.499 ql/volatilitymodel.hpp File Reference

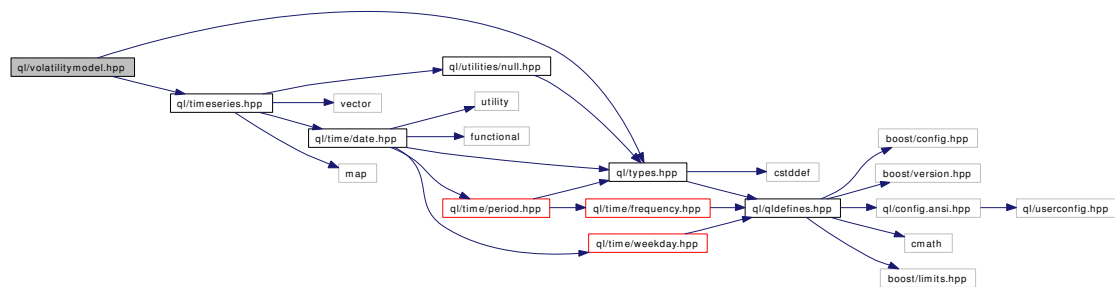
10.499.1 Detailed Description

Volatility term structures.

```
#include <ql/types.hpp>
```

```
#include <ql/timeseries.hpp>
```

Include dependency graph for volatilitymodel.hpp:



Namespaces

- namespace **QuantLib**

10.500 ql/voltermstructure.hpp File Reference

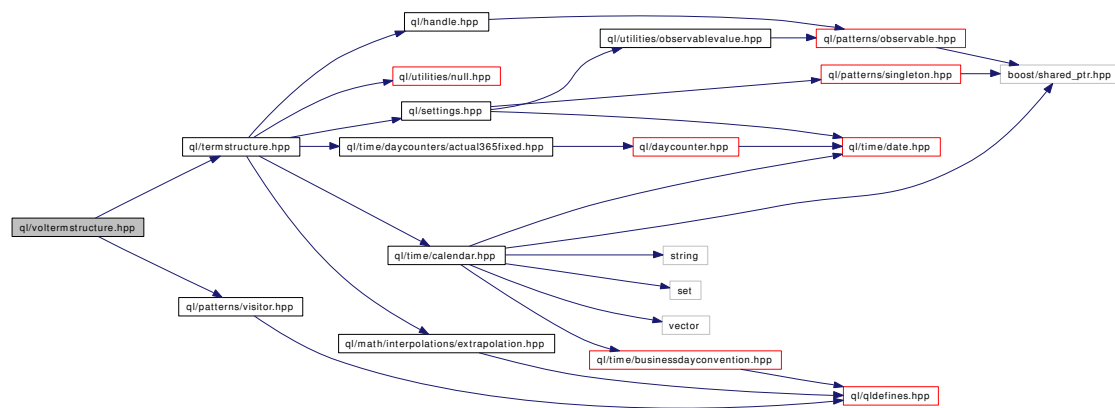
10.500.1 Detailed Description

Volatility term structures.

```
#include <ql/termstructure.hpp>
```

```
#include <ql/patterns/visitor.hpp>
```

Include dependency graph for voltermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackVolTermStructure**
Black-volatility term structure.
- class **BlackVolatilityTermStructure**
Black-volatility term structure.
- class **BlackVarianceTermStructure**
Black variance term structure.
- class **LocalVolTermStructure**
Local-volatility term structure.

10.501 ql/yieldtermstructure.hpp File Reference

10.501.1 Detailed Description

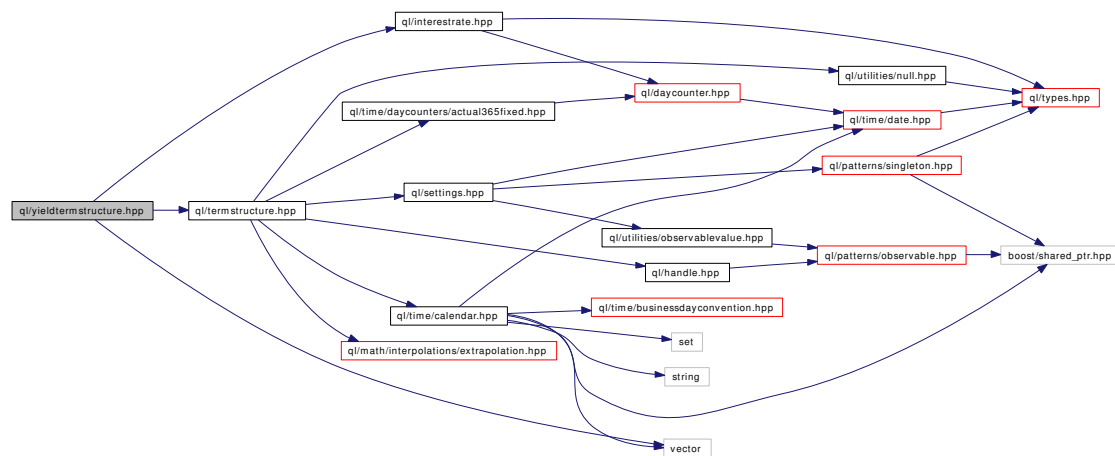
Interest-rate term structure.

```
#include <ql/termstructure.hpp>
```

```
#include <ql/interestrate.hpp>
```

```
#include <vector>
```

Include dependency graph for yieldtermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [YieldTermStructure](#)
Interest-rate term structure.

Chapter 11

QuantLib Example Documentation

11.1 BermudanSwaption.cpp

This is an example of using the QuantLib short rate models.

```
1 /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
22 #define BOOST_LIB_DIAGNOSTIC
23 # include <ql/quantlib.hpp>
24 #undef BOOST_LIB_DIAGNOSTIC
25
26 #ifdef BOOST_MSVC
27 /* Uncomment the following lines to unmask floating-point
28    exceptions. Warning: unpredictable results can arise...
29
30    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
31    Is there anyone with a definitive word about this?
32 */
33 // #include <float.h>
34 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
35 #endif
36
37 #include <boost/timer.hpp>
38 #include <iostream>
39 #include <iomanip>
40
41 using namespace QuantLib;
42
43 #if defined(QL_ENABLE_SESSIONS)
44 namespace QuantLib {
45     Integer sessionId() { return 0; }
46 }
47 #endif
48
49 #endif
50
51
52 //Number of swaptions to be calibrated to...
53
54 Size numRows = 5;
55 Size numCols = 5;
56
57 Integer swapLengths[] = {
58     1, 2, 3, 4, 5;
59
60 Volatility swaptionVols[] = {
61     0.1490, 0.1340, 0.1228, 0.1189, 0.1148,
```

```

61 0.1290, 0.1201, 0.1146, 0.1108, 0.1040,
62 0.1149, 0.1112, 0.1070, 0.1010, 0.0957,
63 0.1047, 0.1021, 0.0980, 0.0951, 0.1270,
64 0.1000, 0.0950, 0.0900, 0.1230, 0.1160};
65
66 void calibrateModel(
67     const boost::shared_ptr<ShortRateModel>& model,
68     const std::vector<boost::shared_ptr<CalibrationHelper> >& helpers) {
69
70     LevenbergMarquardt om;
71     model->calibrate(helpers, om,
72         EndCriteria(400, 100, 1.0e-8, 1.0e-8, 1.0e-8));
73
74     // Output the implied Black volatilities
75     for (Size i=0; i<numRows; i++) {
76         Size j = numCols - i - 1; // 1x5, 2x4, 3x3, 4x2, 5x1
77         Size k = i*numCols + j;
78         Real npv = helpers[i]->modelValue();
79         Volatility implied = helpers[i]->impliedVolatility(npv, 1e-4,
80             1000, 0.05, 0.50);
81         Volatility diff = implied - swaptionVols[k];
82
83         std::cout << i+1 << "x" << swapLengths[j]
84             << std::setprecision(5) << std::noshowpos
85             << ": model " << std::setw(7) << io::volatility(implied)
86             << ", market " << std::setw(7)
87             << io::volatility(swaptionVols[k])
88             << " (" << std::setw(7) << std::showpos
89             << io::volatility(diff) << std::noshowpos << ")\n";
90     }
91 }
92
93 int main(int, char* []) {
94
95     try {
96
97         boost::timer timer;
98         std::cout << std::endl;
99
100        Date todaysDate(15, February, 2002);
101        Calendar calendar = TARGET();
102        Date settlementDate(19, February, 2002);
103        Settings::instance().evaluationDate() = todaysDate;
104
105        // flat yield term structure impling 1x5 swap at 5%
106        boost::shared_ptr<Quote> flatRate(new SimpleQuote(0.04875825));
107        Handle<YieldTermStructure> rhTermStructure(
108            boost::shared_ptr<FlatForward>(
109                new FlatForward(settlementDate, Handle<Quote>(flatRate),
110                    Actual365Fixed())));
111
112        // Define the ATM/OTM/ITM swaps
113        Frequency fixedLegFrequency = Annual;
114        BusinessDayConvention fixedLegConvention = Unadjusted;
115        BusinessDayConvention floatingLegConvention = ModifiedFollowing;
116        DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
117        Frequency floatingLegFrequency = Semiannual;
118        VanillaSwap::Type type = VanillaSwap::Payer;
119        Rate dummyFixedRate = 0.03;
120        boost::shared_ptr<IborIndex> indexSixMonths(new
121            Euribor6M(rhTermStructure));
122
123        Date startDate = calendar.advance(settlementDate, 1, Years,
124            floatingLegConvention);
125        Date maturity = calendar.advance(startDate, 5, Years,
126            floatingLegConvention);
127        Schedule fixedSchedule(startDate, maturity, Period(fixedLegFrequency),

```



```

128             calendar,fixedLegConvention,fixedLegConvention,
129             false,false);
130     Schedule floatSchedule(startDate,maturity,Period(floatingLegFrequency),
131             calendar,floatingLegConvention,floatingLegConvention,
132             false,false);
133
134     boost::shared_ptr<VanillaSwap> swap(new VanillaSwap(
135         type, 1000.0,
136         fixedSchedule, dummyFixedRate, fixedLegDayCounter,
137         floatSchedule, indexSixMonths, 0.0,
138         indexSixMonths->dayCounter(), rhTermStructure));
139     Rate fixedATMRate = swap->fairRate();
140     Rate fixedOTMRate = fixedATMRate * 1.2;
141     Rate fixedITMRate = fixedATMRate * 0.8;
142
143     boost::shared_ptr<VanillaSwap> atmSwap(new VanillaSwap(
144         type, 1000.0,
145         fixedSchedule, fixedATMRate, fixedLegDayCounter,
146         floatSchedule, indexSixMonths, 0.0,
147         indexSixMonths->dayCounter(), rhTermStructure));
148     boost::shared_ptr<VanillaSwap> otmSwap(new VanillaSwap(
149         type, 1000.0,
150         fixedSchedule, fixedOTMRate, fixedLegDayCounter,
151         floatSchedule, indexSixMonths, 0.0,
152         indexSixMonths->dayCounter(), rhTermStructure));
153     boost::shared_ptr<VanillaSwap> itmSwap(new VanillaSwap(
154         type, 1000.0,
155         fixedSchedule, fixedITMRate, fixedLegDayCounter,
156         floatSchedule, indexSixMonths, 0.0,
157         indexSixMonths->dayCounter(), rhTermStructure));
158
159     // defining the swaptions to be used in model calibration
160     std::vector<Period> swaptionMaturities;
161     swaptionMaturities.push_back(Period(1, Years));
162     swaptionMaturities.push_back(Period(2, Years));
163     swaptionMaturities.push_back(Period(3, Years));
164     swaptionMaturities.push_back(Period(4, Years));
165     swaptionMaturities.push_back(Period(5, Years));
166
167     std::vector<boost::shared_ptr<CalibrationHelper> > swaptions;
168
169     // List of times that have to be included in the timegrid
170     std::list<Time> times;
171
172     Size i;
173     for (i=0; i<numRows; i++) {
174         Size j = numCols - i -1; // 1x5, 2x4, 3x3, 4x2, 5x1
175         Size k = i*numCols + j;
176         boost::shared_ptr<Quote> vol(new SimpleQuote(swaptionVols[k]));
177         swaptions.push_back(boost::shared_ptr<CalibrationHelper>(new
178             SwaptionHelper(swaptionMaturities[i],
179                 Period(swapLengths[j], Years),
180                 Handle<Quote>(vol),
181                 indexSixMonths,
182                 indexSixMonths->tenor(),
183                 indexSixMonths->dayCounter(),
184                 indexSixMonths->dayCounter(),
185                 rhTermStructure)));
186         swaptions.back()->addTimesTo(times);
187     }
188
189     // Building time-grid
190     TimeGrid grid(times.begin(), times.end(), 30);
191
192
193     // defining the models
194     boost::shared_ptr<G2> modelG2(new G2(rhTermStructure));

```

```

195 boost::shared_ptr<HullWhite> modelHW(new HullWhite(rhTermStructure));
196 boost::shared_ptr<HullWhite> modelHW2(new HullWhite(rhTermStructure));
197 boost::shared_ptr<BlackKarasinski> modelBK(
198     new BlackKarasinski(rhTermStructure));
199
200
201 // model calibrations
202
203 std::cout << "G2 (analytic formulae) calibration" << std::endl;
204 for (i=0; i<swaptions.size(); i++)
205     swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
206         new G2SwaptionEngine(modelG2, 6.0, 16)));
207
208 calibrateModel(modelG2, swaptions);
209 std::cout << "calibrated to:\n"
210     << "a      = " << modelG2->params()[0] << ", "
211     << "sigma = " << modelG2->params()[1] << "\n"
212     << "b      = " << modelG2->params()[2] << ", "
213     << "eta    = " << modelG2->params()[3] << "\n"
214     << "rho    = " << modelG2->params()[4]
215     << std::endl << std::endl;
216
217
218
219 std::cout << "Hull-White (analytic formulae) calibration" << std::endl;
220 for (i=0; i<swaptions.size(); i++)
221     swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
222         new JamshidianSwaptionEngine(modelHW)));
223
224 calibrateModel(modelHW, swaptions);
225 std::cout << "calibrated to:\n"
226     << "a = " << modelHW->params()[0] << ", "
227     << "sigma = " << modelHW->params()[1]
228     << std::endl << std::endl;
229
230 std::cout << "Hull-White (numerical) calibration" << std::endl;
231 for (i=0; i<swaptions.size(); i++)
232     swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
233         new TreeSwaptionEngine(modelHW2,grid)));
234
235 calibrateModel(modelHW2, swaptions);
236 std::cout << "calibrated to:\n"
237     << "a = " << modelHW2->params()[0] << ", "
238     << "sigma = " << modelHW2->params()[1]
239     << std::endl << std::endl;
240
241 std::cout << "Black-Karasinski (numerical) calibration" << std::endl;
242 for (i=0; i<swaptions.size(); i++)
243     swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
244         new TreeSwaptionEngine(modelBK,grid)));
245
246 calibrateModel(modelBK, swaptions);
247 std::cout << "calibrated to:\n"
248     << "a = " << modelBK->params()[0] << ", "
249     << "sigma = " << modelBK->params()[1]
250     << std::endl << std::endl;
251
252
253 // ATM Bermudan swaption pricing
254
255 std::cout << "Payer bermudan swaption "
256     << "struck at " << io::rate(fixedATMRate)
257     << " (ATM)" << std::endl;
258
259 std::vector<Date> bermudanDates;
260 const std::vector<boost::shared_ptr<CashFlow> >& leg =
261     swap->fixedLeg();

```

```

262     for (i=0; i<leg.size(); i++) {
263         boost::shared_ptr<Coupon> coupon =
264             boost::dynamic_pointer_cast<Coupon>(leg[i]);
265         bermudanDates.push_back(coupon->accrualStartDate());
266     }
267
268     boost::shared_ptr<Exercise> bermudanExercise(
269         new BermudanExercise(bermudanDates));
270
271     Swaption bermudanSwaption(atmSwap, bermudanExercise, rhTermStructure,
272         boost::shared_ptr<PricingEngine>());
273
274     // Do the pricing for each model
275
276     // G2 price the European swaption here, it should switch to bermudan
277     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
278         TreeSwaptionEngine(modelG2, 50)));
279     std::cout << "G2:      " << bermudanSwaption.NPV() << std::endl;
280
281     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
282         new TreeSwaptionEngine(modelHW, 50)));
283     std::cout << "HW:      " << bermudanSwaption.NPV() << std::endl;
284
285     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
286         TreeSwaptionEngine(modelHW2, 50)));
287     std::cout << "HW (num): " << bermudanSwaption.NPV() << std::endl;
288
289     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
290         TreeSwaptionEngine(modelBK, 50)));
291     std::cout << "BK:      " << bermudanSwaption.NPV() << std::endl;
292
293
294     // OTM Bermudan swaption pricing
295
296     std::cout << "Payer bermudan swaption "
297         << "struck at " << io::rate(fixedOTMRate)
298         << " (OTM)" << std::endl;
299
300     Swaption otmBermudanSwaption(otmSwap, bermudanExercise, rhTermStructure,
301         boost::shared_ptr<PricingEngine>());
302
303     // Do the pricing for each model
304     otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
305         new TreeSwaptionEngine(modelG2, 50)));
306     std::cout << "G2:      " << otmBermudanSwaption.NPV() << std::endl;
307
308     otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
309         new TreeSwaptionEngine(modelHW, 50)));
310     std::cout << "HW:      " << otmBermudanSwaption.NPV() << std::endl;
311
312     otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
313         new TreeSwaptionEngine(modelHW2, 50)));
314     std::cout << "HW (num): " << otmBermudanSwaption.NPV() << std::endl;
315
316     otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
317         new TreeSwaptionEngine(modelBK, 50)));
318     std::cout << "BK:      " << otmBermudanSwaption.NPV() << std::endl;
319
320
321     // ITM Bermudan swaption pricing
322
323     std::cout << "Payer bermudan swaption "
324         << "struck at " << io::rate(fixedITMRate)
325         << " (ITM)" << std::endl;
326
327     Swaption itmBermudanSwaption(itmSwap, bermudanExercise, rhTermStructure,
328         boost::shared_ptr<PricingEngine>());

```

```

329
330 // Do the pricing for each model
331 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
332     new TreeSwaptionEngine(modelG2, 50)));
333 std::cout << "G2:      " << itmBermudanSwaption.NPV() << std::endl;
334
335 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
336     new TreeSwaptionEngine(modelHW, 50)));
337 std::cout << "HW:      " << itmBermudanSwaption.NPV() << std::endl;
338
339 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
340     new TreeSwaptionEngine(modelHW2, 50)));
341 std::cout << "HW (num): " << itmBermudanSwaption.NPV() << std::endl;
342
343 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
344     new TreeSwaptionEngine(modelBK, 50)));
345 std::cout << "BK:      " << itmBermudanSwaption.NPV() << std::endl;
346
347 Real seconds = timer.elapsed();
348 Integer hours = int(seconds/3600);
349 seconds -= hours * 3600;
350 Integer minutes = int(seconds/60);
351 seconds -= minutes * 60;
352 std::cout << " \nRun completed in ";
353 if (hours > 0)
354     std::cout << hours << " h ";
355 if (hours > 0 || minutes > 0)
356     std::cout << minutes << " m ";
357 std::cout << std::fixed << std::setprecision(0)
358     << seconds << " s\n" << std::endl;
359
360 return 0;
361 } catch (std::exception& e) {
362     std::cout << e.what() << std::endl;
363     return 1;
364 } catch (...) {
365     std::cout << "unknown error" << std::endl;
366     return 1;
367 }
368 }
369

```

11.2 ConvertibleBonds.cpp

This example evaluates convertible bond prices.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
21 // the only header you need to use QuantLib
22 #define BOOST_LIB_DIAGNOSTIC
23 # include <ql/quantlib.hpp>
24 #undef BOOST_LIB_DIAGNOSTIC
25
26 #ifndef BOOST_MSVC
27 /* Uncomment the following lines to unmask floating-point
28    exceptions. Warning: unpredictable results can arise...
29
30    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
31    Is there anyone with a definitive word about this?
32 */
33 // #include <float.h>
34 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
35 #endif
36
37 #include <boost/timer.hpp>
38 #include <iostream>
39 #include <iomanip>
40
41 #define LENGTH(a) (sizeof(a)/sizeof(a[0]))
42
43 using namespace QuantLib;
44
45 #if defined(QL_ENABLE_SESSIONS)
46 namespace QuantLib {
47     Integer sessionId() { return 0; }
48 }
49 #endif
50
51 #endif
52
53
54 int main(int, char* []) {
55     try {
56         boost::timer timer;
57         std::cout << std::endl;
58
59         Option::Type type(Option::Put);
60         Real underlying = 36.0;
61         Real spreadRate = 0.005;
62
63         Spread dividendYield = 0.02;
64         Rate riskFreeRate = 0.06;
65         Volatility volatility = 0.20;
66
67         Integer settlementDays = 3;
68         Integer length = 5;
69         Real redemption = 100.0;
70         Real conversionRatio = redemption/underlying; // at the money
71
72         // set up dates/schedules
73         Calendar calendar = TARGET();
74         Date today = calendar.adjust(Date::todaysDate());
75
76         Settings::instance().evaluationDate() = today;
77         Date settlementDate = calendar.advance(today, settlementDays, Days);
78         Date exerciseDate = calendar.advance(settlementDate, length, Years);

```

```

81     Date issueDate = calendar.advance(exerciseDate, -length, Years);
82
83     BusinessDayConvention convention = ModifiedFollowing;
84
85     Frequency frequency = Annual;
86
87     Schedule schedule(issueDate, exerciseDate, Period(frequency), calendar,
88                       convention, convention, true, false);
89
90     DividendSchedule dividends;
91     CallabilitySchedule callability;
92
93     std::vector<Real> coupons(1, 0.05);
94
95     DayCounter bondDayCount = Thirty360();
96
97     Integer callLength[] = { 2, 4 }; // Call dates, years 2, 4.
98     Integer putLength[] = { 3 }; // Put dates year 3
99
100    Real callPrices[] = { 101.5, 100.85 };
101    Real putPrices[] = { 105.0 };
102
103    // Load call schedules
104    for (Size i=0; i<LENGTH(callLength); i++) {
105        callability.push_back(
106            boost::shared_ptr<Callability>(
107                new SoftCallability(Callability::Price(
108                    callPrices[i],
109                    Callability::Price::Clean),
110                    schedule.date(callLength[i]),
111                    1.20)));
112    }
113
114    for (Size j=0; j<LENGTH(putLength); j++) {
115        callability.push_back(
116            boost::shared_ptr<Callability>(
117                new Callability(Callability::Price(
118                    putPrices[j],
119                    Callability::Price::Clean),
120                    Callability::Put,
121                    schedule.date(putLength[j]))));
122    }
123
124    // Assume dividends are paid every 6 months.
125    for (Date d = today + 6*Months; d < exerciseDate; d += 6*Months) {
126        dividends.push_back(
127            boost::shared_ptr<Dividend>(new FixedDividend(1.0, d)));
128    }
129
130    DayCounter dayCounter = Actual365Fixed();
131    Time maturity = dayCounter.yearFraction(settlementDate,
132                                           exerciseDate);
133
134    std::cout << "option type = " << type << std::endl;
135    std::cout << "Time to maturity = " << maturity
136              << std::endl;
137    std::cout << "Underlying price = " << underlying
138              << std::endl;
139    std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
140              << std::endl;
141    std::cout << "Dividend yield = " << io::rate(dividendYield)
142              << std::endl;
143    std::cout << "Volatility = " << io::volatility(volatility)
144              << std::endl;
145    std::cout << std::endl;
146
147    std::string method;

```

```

148     std::cout << std::endl ;
149
150     // write column headings
151     Size widths[] = { 35, 14, 14 };
152     Size totalWidth = widths[0] + widths[1] + widths[2];
153     std::string rule(totalWidth, '-'), dblrule(totalWidth, '=');
154
155     std::cout << dblrule << std::endl;
156     std::cout << "Tsiveriotis-Fernandes method" << std::endl;
157     std::cout << dblrule << std::endl;
158     std::cout << std::setw(widths[0]) << std::left << "Tree type"
159             << std::setw(widths[1]) << std::left << "European"
160             << std::setw(widths[1]) << std::left << "American"
161             << std::endl;
162     std::cout << rule << std::endl;
163
164     boost::shared_ptr<Exercise> exercise(
165         new EuropeanExercise(exerciseDate));
166     boost::shared_ptr<Exercise> amExercise(
167         new AmericanExercise(settlementDate,
168                             exerciseDate));
169
170     Handle<Quote> underlyingH(
171         boost::shared_ptr<Quote>(new SimpleQuote(underlying)));
172
173     Handle<YieldTermStructure> flatTermStructure(
174         boost::shared_ptr<YieldTermStructure>(
175             new FlatForward(settlementDate, riskFreeRate, dayCounter)));
176
177     Handle<YieldTermStructure> flatDividendTS(
178         boost::shared_ptr<YieldTermStructure>(
179             new FlatForward(settlementDate, dividendYield, dayCounter)));
180
181     Handle<BlackVolTermStructure> flatVolTS(
182         boost::shared_ptr<BlackVolTermStructure>(
183             new BlackConstantVol(settlementDate, volatility, dayCounter)));
184
185
186     boost::shared_ptr<StochasticProcess> stochasticProcess(
187         new BlackScholesMertonProcess(underlyingH,
188                                       flatDividendTS,
189                                       flatTermStructure,
190                                       flatVolTS));
191
192     Size timeSteps = 801;
193
194     Handle<Quote> creditSpread(
195         boost::shared_ptr<Quote>(new SimpleQuote(spreadRate)));
196
197     boost::shared_ptr<Quote> rate(new SimpleQuote(riskFreeRate));
198
199     Handle<YieldTermStructure> discountCurve(
200         boost::shared_ptr<YieldTermStructure>(
201             new FlatForward(today, Handle<Quote>(rate), dayCounter)));
202
203     boost::shared_ptr<PricingEngine> engine(
204         new BinomialConvertibleEngine<JarrowRudd>(timeSteps));
205
206     ConvertibleFixedCouponBond europeanBond(
207         stochasticProcess, exercise, engine,
208         conversionRatio, dividends, callability,
209         creditSpread, issueDate, settlementDays,
210         coupons, bondDayCount, schedule, redemption);
211
212     ConvertibleFixedCouponBond americanBond(
213         stochasticProcess, amExercise, engine,
214         conversionRatio, dividends, callability,

```

```

215         creditSpread, issueDate, settlementDays,
216         coupons, bondDayCount, schedule, redemption));
217
218     method = "Jarrow-Rudd";
219     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
220         new BinomialConvertibleEngine<JarrowRudd>(timeSteps)));
221     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
222         new BinomialConvertibleEngine<JarrowRudd>(timeSteps)));
223     std::cout << std::setw(widths[0]) << std::left << method
224         << std::fixed
225         << std::setw(widths[1]) << std::left << europeanBond.NPV()
226         << std::setw(widths[2]) << std::left << americanBond.NPV()
227         << std::endl;
228
229     method = "Cox-Ross-Rubinstein";
230     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
231         new BinomialConvertibleEngine<CoxRossRubinstein>(timeSteps)));
232     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
233         new BinomialConvertibleEngine<CoxRossRubinstein>(timeSteps)));
234     std::cout << std::setw(widths[0]) << std::left << method
235         << std::fixed
236         << std::setw(widths[1]) << std::left << europeanBond.NPV()
237         << std::setw(widths[2]) << std::left << americanBond.NPV()
238         << std::endl;
239
240     method = "Additive equiprobabilities";
241     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
242         new BinomialConvertibleEngine<AdditiveEQPBinoialTree>(timeSteps)));
243     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
244         new BinomialConvertibleEngine<AdditiveEQPBinoialTree>(timeSteps)));
245     std::cout << std::setw(widths[0]) << std::left << method
246         << std::fixed
247         << std::setw(widths[1]) << std::left << europeanBond.NPV()
248         << std::setw(widths[2]) << std::left << americanBond.NPV()
249         << std::endl;
250
251     method = "Trigeorgis";
252     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
253         new BinomialConvertibleEngine<Trigeorgis>(timeSteps)));
254     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
255         new BinomialConvertibleEngine<Trigeorgis>(timeSteps)));
256     std::cout << std::setw(widths[0]) << std::left << method
257         << std::fixed
258         << std::setw(widths[1]) << std::left << europeanBond.NPV()
259         << std::setw(widths[2]) << std::left << americanBond.NPV()
260         << std::endl;
261
262     method = "Tian";
263     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
264         new BinomialConvertibleEngine<Tian>(timeSteps)));
265     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
266         new BinomialConvertibleEngine<Tian>(timeSteps)));
267     std::cout << std::setw(widths[0]) << std::left << method
268         << std::fixed
269         << std::setw(widths[1]) << std::left << europeanBond.NPV()
270         << std::setw(widths[2]) << std::left << americanBond.NPV()
271         << std::endl;
272
273     method = "Leisen-Reimer";
274     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
275         new BinomialConvertibleEngine<LeisenReimer>(timeSteps)));
276     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
277         new BinomialConvertibleEngine<LeisenReimer>(timeSteps)));
278     std::cout << std::setw(widths[0]) << std::left << method
279         << std::fixed
280         << std::setw(widths[1]) << std::left << europeanBond.NPV()
281         << std::setw(widths[2]) << std::left << americanBond.NPV()

```



```
282         << std::endl;
283
284     std::cout << dblrule << std::endl;
285
286     Real seconds = timer.elapsed();
287     Integer hours = int(seconds/3600);
288     seconds -= hours * 3600;
289     Integer minutes = int(seconds/60);
290     seconds -= minutes * 60;
291     std::cout << " \nRun completed in ";
292     if (hours > 0)
293         std::cout << hours << " h ";
294     if (hours > 0 || minutes > 0)
295         std::cout << minutes << " m ";
296     std::cout << std::fixed << std::setprecision(0)
297         << seconds << " s\n" << std::endl;
298
299     return 0;
300 } catch (std::exception& e) {
301     std::cout << e.what() << std::endl;
302     return 1;
303 } catch (...) {
304     std::cout << "unknown error" << std::endl;
305     return 1;
306 }
307
308 }
309
```

11.3 DiscreteHedging.cpp

This is an example of using the QuantLib Monte Carlo framework.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
21 /* This example computes profit and loss of a discrete interval hedging
22    strategy and compares with the results of Derman & Kamal's (Goldman Sachs
23    Equity Derivatives Research) Research Note: "When You Cannot Hedge
24    Continuously: The Corrections to Black-Scholes"
25    http://www.ederman.com/emanuelderman/GSQSpapers/when_you_cannot_hedge.pdf
26
27    Suppose an option hedger sells an European option and receives the
28    Black-Scholes value as the options premium.
29    Then he follows a Black-Scholes hedging strategy, rehedging at discrete,
30    evenly spaced time intervals as the underlying stock changes. At
31    expiration, the hedger delivers the option payoff to the option holder,
32    and unwinds the hedge. We are interested in understanding the final
33    profit or loss of this strategy.
34
35    If the hedger had followed the exact Black-Scholes replication strategy,
36    re-hedging continuously as the underlying stock evolved towards its final
37    value at expiration, then, no matter what path the stock took, the final
38    P&L would be exactly zero. When the replication strategy deviates from
39    the exact Black-Scholes method, the final P&L may deviate from zero. This
40    deviation is called the replication error. When the hedger rebalances at
41    discrete rather than continuous intervals, the hedge is imperfect and the
42    replication is inexact. The more often hedging occurs, the smaller the
43    replication error.
44
45    We examine the range of possibilities, computing the replication error.
46 */
47
48 // the only header you need to use QuantLib
49 #define BOOST_LIB_DIAGNOSTIC
50 # include <ql/quantlib.hpp>
51 #undef BOOST_LIB_DIAGNOSTIC
52
53 #ifdef BOOST_MSVC
54 /* Uncomment the following lines to unmask floating-point
55    exceptions. Warning: unpredictable results can arise...
56
57    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
58    Is there anyone with a definitive word about this?
59 */
60 // #include <float.h>
61 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
62 #endif
63
64 #include <boost/timer.hpp>
65 #include <iostream>
66 #include <iomanip>
67
68 using namespace QuantLib;
69
70 #if defined(QL_ENABLE_SESSIONS)
71 namespace QuantLib {
72     Integer sessionId() { return 0; }
73 }
74 #endif
75
76 /* The ReplicationError class carries out Monte Carlo simulations to evaluate
77    the outcome (the replication error) of the discrete hedging strategy over

```

```

81     different, randomly generated scenarios of future stock price evolution.
82 */
83 class ReplicationError
84 {
85 public:
86     ReplicationError(Option::Type type,
87                     Time maturity,
88                     Real strike,
89                     Real s0,
90                     Volatility sigma,
91                     Rate r)
92     : maturity_(maturity), payoff_(type, strike), s0_(s0),
93       sigma_(sigma), r_(r) {
94
95         // value of the option
96         DiscountFactor rDiscount = std::exp(-r_*maturity_);
97         DiscountFactor qDiscount = 1.0;
98         Real forward = s0_*qDiscount/rDiscount;
99         Real stdDev = std::sqrt(sigma_*sigma_*maturity_);
100         boost::shared_ptr<StrikedTypePayoff> payoff(
101             new PlainVanillaPayoff(payoff_));
102         BlackCalculator black(payoff,forward,stdDev,rDiscount);
103         std::cout << "Option value: " << black.value() << std::endl;
104
105         // store option's vega, since Derman and Kamal's formula needs it
106         vega_ = black.vega(maturity_);
107
108         std::cout << std::endl;
109         std::cout <<
110             "          |          | P&L \t| P&L          | Derman&Kamal | P&L"
111             "          \t| P&L" << std::endl;
112
113         std::cout <<
114             "samples | trades | Mean \t| Std Dev | Formula          |"
115             " skewness \t| kurt." << std::endl;
116
117         std::cout << "-----"
118             "-----" << std::endl;
119     }
120
121     // the actual replication error computation
122     void compute(Size nTimeSteps, Size nSamples);
123 private:
124     Time maturity_;
125     PlainVanillaPayoff payoff_;
126     Real s0_;
127     Volatility sigma_;
128     Rate r_;
129     Real vega_;
130 };
131
132 // The key for the MonteCarlo simulation is to have a PathPricer that
133 // implements a value(const Path& path) method.
134 // This method prices the portfolio for each Path of the random variable
135 class ReplicationPathPricer : public PathPricer<Path> {
136 public:
137     // real constructor
138     ReplicationPathPricer(Option::Type type,
139                          Real strike,
140                          Rate r,
141                          Time maturity,
142                          Volatility sigma)
143     : type_(type), strike_(strike),
144       r_(r), maturity_(maturity), sigma_(sigma) {
145         QL_REQUIRE(strike_ > 0.0, "strike must be positive");
146         QL_REQUIRE(r_ >= 0.0,
147             "risk free rate (r) must be positive or zero");

```

```

148     QL_REQUIRE(maturity_ > 0.0, "maturity must be positive");
149     QL_REQUIRE(sigma_ >= 0.0,
150               "volatility (sigma) must be positive or zero");
151
152 }
153 // The value() method encapsulates the pricing code
154 Real operator()(const Path& path) const;
155
156 private:
157     Option::Type type_;
158     Real strike_;
159     Rate r_;
160     Time maturity_;
161     Volatility sigma_;
162 };
163
164
165 // Compute Replication Error as in the Derman and Kamal's research note
166 int main(int, char* []) {
167
168     try {
169
170         boost::timer timer;
171         std::cout << std::endl;
172
173         Time maturity = 1.0/12.0;    // 1 month
174         Real strike = 100;
175         Real underlying = 100;
176         Volatility volatility = 0.20; // 20%
177         Rate riskFreeRate = 0.05;    // 5%
178         ReplicationError rp(Option::Call, maturity, strike, underlying,
179                           volatility, riskFreeRate);
180
181         Size scenarios = 50000;
182         Size hedgesNum;
183
184         hedgesNum = 21;
185         rp.compute(hedgesNum, scenarios);
186
187         hedgesNum = 84;
188         rp.compute(hedgesNum, scenarios);
189
190         Real seconds = timer.elapsed();
191         Integer hours = int(seconds/3600);
192         seconds -= hours * 3600;
193         Integer minutes = int(seconds/60);
194         seconds -= minutes * 60;
195         std::cout << " \nRun completed in ";
196         if (hours > 0)
197             std::cout << hours << " h ";
198         if (hours > 0 || minutes > 0)
199             std::cout << minutes << " m ";
200         std::cout << std::fixed << std::setprecision(0)
201             << seconds << " s\n" << std::endl;
202
203         return 0;
204     } catch (std::exception& e) {
205         std::cout << e.what() << std::endl;
206         return 1;
207     } catch (...) {
208         std::cout << "unknown error" << std::endl;
209         return 1;
210     }
211 }
212
213
214 /* The actual computation of the Profit&Loss for each single path.

```

```

215
216     In each scenario N reheding trades spaced evenly in time over
217     the life of the option are carried out, using the Black-Scholes
218     hedge ratio.
219 */
220 Real ReplicationPathPricer::operator()(const Path& path) const {
221
222     Size n = path.length()-1;
223     QL_REQUIRE(n>0, "the path cannot be empty");
224
225     // discrete hedging interval
226     Time dt = maturity_/n;
227
228     // For simplicity, we assume the stock pays no dividends.
229     Rate stockDividendYield = 0.0;
230
231     // let's start
232     Time t = 0;
233
234     // stock value at t=0
235     Real stock = path.front();
236
237     // money account at t=0
238     Real money_account = 0.0;
239
240     /***** the initial deal *****/
241     // option fair price (Black-Scholes) at t=0
242     DiscountFactor rDiscount = std::exp(-r_*maturity_);
243     DiscountFactor qDiscount = std::exp(-stockDividendYield*maturity_);
244     Real forward = stock*qDiscount/rDiscount;
245     Real stdDev = std::sqrt(sigma_*sigma_*maturity_);
246     boost::shared_ptr<StrikedTypePayoff> payoff(
247         new PlainVanillaPayoff(type_,strike_));
248     BlackCalculator black(payoff,forward,stdDev,rDiscount);
249     // sell the option, cash in its premium
250     money_account += black.value();
251     // compute delta
252     Real delta = black.delta(stock);
253     // delta-hedge the option buying stock
254     Real stockAmount = delta;
255     money_account -= stockAmount*stock;
256
257     /***** hedging during option life *****/
258     for (Size step = 0; step < n-1; step++){
259
260         // time flows
261         t += dt;
262
263         // accruing on the money account
264         money_account *= std::exp( r_*dt );
265
266         // stock growth:
267         stock = path[step+1];
268
269         // recalculate option value at the current stock value,
270         // and the current time to maturity
271         rDiscount = std::exp(-r_*(maturity_-t));
272         qDiscount = std::exp(-stockDividendYield*(maturity_-t));
273         forward = stock*qDiscount/rDiscount;
274         stdDev = std::sqrt(sigma_*sigma_*(maturity_-t));
275         BlackCalculator black(payoff,forward,stdDev,rDiscount);
276
277         // recalculate delta

```

```

282         delta = black.delta(stock);
283
284         // re-hedging
285         money_account -= (delta - stockAmount)*stock;
286         stockAmount = delta;
287     }
288
289     /*****
290     *** option expiration ***
291     *****/
292     // last accrual on my money account
293     money_account *= std::exp( r_*dt );
294     // last stock growth
295     stock = path[n];
296
297     // the hedger delivers the option payoff to the option holder
298     Real optionPayoff = PlainVanillaPayoff(type_, strike_)(stock);
299     money_account -= optionPayoff;
300
301     // and unwinds the hedge selling his stock position
302     money_account += stockAmount*stock;
303
304     // final Profit&Loss
305     return money_account;
306 }
307
308
309 // The computation over nSamples paths of the P&L distribution
310 void ReplicationError::compute(Size nTimeSteps, Size nSamples)
311 {
312     QL_REQUIRE(nTimeSteps>0, "the number of steps must be > 0");
313
314     // hedging interval
315     // Time tau = maturity_ / nTimeSteps;
316
317     /* Black-Scholes framework: the underlying stock price evolves
318        lognormally with a fixed known volatility that stays constant
319        throughout time.
320     */
321     Date today = Date::todaysDate();
322     DayCounter dayCount = Actual365Fixed();
323     Handle<Quote> stateVariable(
324         boost::shared_ptr<Quote>(new SimpleQuote(s0_)));
325     Handle<YieldTermStructure> riskFreeRate(
326         boost::shared_ptr<YieldTermStructure>(
327             new FlatForward(today, r_, dayCount)));
328     Handle<YieldTermStructure> dividendYield(
329         boost::shared_ptr<YieldTermStructure>(
330             new FlatForward(today, 0.0, dayCount)));
331     Handle<BlackVolTermStructure> volatility(
332         boost::shared_ptr<BlackVolTermStructure>(
333             new BlackConstantVol(today, sigma_, dayCount)));
334     boost::shared_ptr<StochasticProcess1D> diffusion(
335         new BlackScholesMertonProcess(stateVariable, dividendYield,
336             riskFreeRate, volatility));
337
338     // Black Scholes equation rules the path generator:
339     // at each step the log of the stock
340     // will have drift and sigma^2 variance
341     PseudoRandom::rsg_type rsg =
342         PseudoRandom::make_sequence_generator(nTimeSteps, 0);
343
344     bool brownianBridge = false;
345
346     typedef SingleVariate<PseudoRandom>::path_generator_type generator_type;
347     boost::shared_ptr<generator_type> myPathGenerator(new
348         generator_type(diffusion, maturity_, nTimeSteps,

```

```

349         rsg, brownianBridge));
350
351     // The replication strategy's Profit&Loss is computed for each path
352     // of the stock. The path pricer knows how to price a path using its
353     // value() method
354     boost::shared_ptr<PathPricer<Path> > myPathPricer(new
355         ReplicationPathPricer(payoff_.optionType(), payoff_.strike(),
356             r_, maturity_, sigma_));
357
358     // a statistics accumulator for the path-dependant Profit&Loss values
359     Statistics statisticsAccumulator;
360
361     // The Monte Carlo model generates paths using myPathGenerator
362     // each path is priced using myPathPricer
363     // prices will be accumulated into statisticsAccumulator
364     MonteCarloModel<SingleVariate,PseudoRandom>
365         MCSimulation(myPathGenerator,
366             myPathPricer,
367             statisticsAccumulator,
368             false);
369
370     // the model simulates nSamples paths
371     MCSimulation.addSamples(nSamples);
372
373     // the sampleAccumulator method
374     // gives access to all the methods of statisticsAccumulator
375     Real PLMean = MCSimulation.sampleAccumulator().mean();
376     Real PLStDev = MCSimulation.sampleAccumulator().standardDeviation();
377     Real PLSkew = MCSimulation.sampleAccumulator().skewness();
378     Real PLKurt = MCSimulation.sampleAccumulator().kurtosis();
379
380     // Derman and Kamal's formula
381     Real theorStd = std::sqrt(M_PI/4/nTimeSteps)*vega_*sigma_;
382
383
384     std::cout << std::fixed
385         << nSamples << "\t| "
386         << nTimeSteps << "\t | "
387         << std::setprecision(3) << PLMean << " \t| "
388         << std::setprecision(2) << PLStDev << " \t | "
389         << std::setprecision(2) << theorStd << " \t | "
390         << std::setprecision(2) << PLSkew << " \t| "
391         << std::setprecision(2) << PLKurt << std::endl;
392 }

```

11.4 EquityOption.cpp

This example evaluates European, American and Bermudan options using different methods

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
20 // the only header you need to use QuantLib
21 #define BOOST_LIB_DIAGNOSTIC
22 # include <ql/quantlib.hpp>
23 #undef BOOST_LIB_DIAGNOSTIC
24
25 #ifndef BOOST_MSVC
26 /* Uncomment the following lines to unmask floating-point
27    exceptions. Warning: unpredictable results can arise...
28
29    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
30    Is there anyone with a definitive word about this?
31 */
32 // #include <float.h>
33 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
34 #endif
35
36 #include <boost/timer.hpp>
37 #include <iostream>
38 #include <iomanip>
39
40 using namespace QuantLib;
41
42 #if defined(QL_ENABLE_SESSIONS)
43 namespace QuantLib {
44
45     Integer sessionId() { return 0; }
46
47 }
48 #endif
49
50
51 int main(int, char* []) {
52
53     try {
54
55         boost::timer timer;
56         std::cout << std::endl;
57
58         // our options
59         Option::Type type(Option::Put);
60         Real underlying = 36;
61         Real strike = 40;
62         Spread dividendYield = 0.00;
63         Rate riskFreeRate = 0.06;
64         Volatility volatility = 0.20;
65
66         Date todaysDate(15, May, 1998);
67         Date settlementDate(17, May, 1998);
68         Settings::instance().evaluationDate() = todaysDate;
69
70         Date maturity(17, May, 1999);
71         DayCounter dayCounter = Actual365Fixed();
72
73         std::cout << "Option type = " << type << std::endl;
74         std::cout << "Maturity = " << maturity << std::endl;
75         std::cout << "Underlying price = " << underlying << std::endl;
76         std::cout << "Strike = " << strike << std::endl;
77         std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
78             << std::endl;
79         std::cout << "Dividend yield = " << io::rate(dividendYield)

```



```

80         << std::endl;
81     std::cout << "Volatility = " << io::volatility(volatility)
82         << std::endl;
83     std::cout << std::endl;
84
85     std::string method;
86
87     std::cout << std::endl ;
88
89     // write column headings
90     Size widths[] = { 35, 14, 14, 14 };
91     std::cout << std::setw(widths[0]) << std::left << "Method"
92         << std::setw(widths[1]) << std::left << "European"
93         << std::setw(widths[2]) << std::left << "Bermudan"
94         << std::setw(widths[3]) << std::left << "American"
95         << std::endl;
96
97     std::vector<Date> exerciseDates;
98     for (Integer i=1; i<=4; i++)
99         exerciseDates.push_back(settlementDate + 3*i*Months);
100
101     boost::shared_ptr<Exercise> europeanExercise(
102         new EuropeanExercise(maturity));
103
104     boost::shared_ptr<Exercise> bermudanExercise(
105         new BermudanExercise(exerciseDates));
106
107     boost::shared_ptr<Exercise> americanExercise(
108         new AmericanExercise(settlementDate,
109             maturity));
110
111     Handle<Quote> underlyingH(
112         boost::shared_ptr<Quote>(new SimpleQuote(underlying)));
113
114     // bootstrap the yield/dividend/vol curves
115     Handle<YieldTermStructure> flatTermStructure(
116         boost::shared_ptr<YieldTermStructure>(
117             new FlatForward(settlementDate, riskFreeRate, dayCounter)));
118     Handle<YieldTermStructure> flatDividendTS(
119         boost::shared_ptr<YieldTermStructure>(
120             new FlatForward(settlementDate, dividendYield, dayCounter)));
121     Handle<BlackVolTermStructure> flatVolTS(
122         boost::shared_ptr<BlackVolTermStructure>(
123             new BlackConstantVol(settlementDate, volatility, dayCounter)));
124
125     boost::shared_ptr<StrikedTypePayoff> payoff(
126         new PlainVanillaPayoff(type, strike));
127
128     boost::shared_ptr<StochasticProcess> stochasticProcess(
129         new BlackScholesMertonProcess(underlyingH, flatDividendTS,
130             flatTermStructure, flatVolTS));
131
132     // options
133
134     VanillaOption europeanOption(stochasticProcess, payoff,
135         europeanExercise);
136
137     VanillaOption bermudanOption(stochasticProcess, payoff,
138         bermudanExercise);
139
140     VanillaOption americanOption(stochasticProcess, payoff,
141         americanExercise);
142
143     // Analytic formulas:
144
145     // Black-Scholes for European
146     method = "Black-Scholes";

```

```

147     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
148         new AnalyticEuropeanEngine));
149     std::cout << std::setw(widths[0]) << std::left << method
150         << std::fixed
151         << std::setw(widths[1]) << std::left << europeanOption.NPV()
152         << std::setw(widths[2]) << std::left << "N/A"
153         << std::setw(widths[3]) << std::left << "N/A"
154         << std::endl;
155
156     // Barone-Adesi and Whaley approximation for American
157     method = "Barone-Adesi/Whaley";
158     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
159         new BaroneAdesiWhaleyApproximationEngine));
160     std::cout << std::setw(widths[0]) << std::left << method
161         << std::fixed
162         << std::setw(widths[1]) << std::left << "N/A"
163         << std::setw(widths[2]) << std::left << "N/A"
164         << std::setw(widths[3]) << std::left << americanOption.NPV()
165         << std::endl;
166
167     // Bjerksund and Stensland approximation for American
168     method = "Bjerksund/Stensland";
169     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
170         new BjerksundStenslandApproximationEngine));
171     std::cout << std::setw(widths[0]) << std::left << method
172         << std::fixed
173         << std::setw(widths[1]) << std::left << "N/A"
174         << std::setw(widths[2]) << std::left << "N/A"
175         << std::setw(widths[3]) << std::left << americanOption.NPV()
176         << std::endl;
177
178     // Integral
179
180     method = "Integral";
181     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
182         new IntegralEngine));
183     std::cout << std::setw(widths[0]) << std::left << method
184         << std::fixed
185         << std::setw(widths[1]) << std::left << europeanOption.NPV()
186         << std::setw(widths[2]) << std::left << "N/A"
187         << std::setw(widths[3]) << std::left << "N/A"
188         << std::endl;
189
190     // Finite differences
191
192     Size timeSteps = 801;
193
194     method = "Finite differences";
195     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
196         new FDEuropeanEngine(timeSteps,timeSteps-1)));
197     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
198         new FDBermudanEngine(timeSteps,timeSteps-1)));
199     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
200         new FDAmericanEngine(timeSteps,timeSteps-1)));
201     std::cout << std::setw(widths[0]) << std::left << method
202         << std::fixed
203         << std::setw(widths[1]) << std::left << europeanOption.NPV()
204         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
205         << std::setw(widths[3]) << std::left << americanOption.NPV()
206         << std::endl;
207
208     // Binomial method
209
210     method = "Binomial Jarrow-Rudd";
211     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
212         new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
213     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(

```

```

214         new BinomialVanillaEngine<JarrowRudd>(timeSteps));
215     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
216         new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
217     std::cout << std::setw(widths[0]) << std::left << method
218         << std::fixed
219         << std::setw(widths[1]) << std::left << europeanOption.NPV()
220         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
221         << std::setw(widths[3]) << std::left << americanOption.NPV()
222         << std::endl;
223
224     method = "Binomial Cox-Ross-Rubinstein";
225     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
226         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
227     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
228         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
229     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
230         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
231     std::cout << std::setw(widths[0]) << std::left << method
232         << std::fixed
233         << std::setw(widths[1]) << std::left << europeanOption.NPV()
234         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
235         << std::setw(widths[3]) << std::left << americanOption.NPV()
236         << std::endl;
237
238     method = "Additive equiprobabilities";
239     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
240         new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
241     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
242         new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
243     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
244         new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
245     std::cout << std::setw(widths[0]) << std::left << method
246         << std::fixed
247         << std::setw(widths[1]) << std::left << europeanOption.NPV()
248         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
249         << std::setw(widths[3]) << std::left << americanOption.NPV()
250         << std::endl;
251
252     method = "Binomial Trigeorgis";
253     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
254         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
255     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
256         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
257     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
258         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
259     std::cout << std::setw(widths[0]) << std::left << method
260         << std::fixed
261         << std::setw(widths[1]) << std::left << europeanOption.NPV()
262         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
263         << std::setw(widths[3]) << std::left << americanOption.NPV()
264         << std::endl;
265
266     method = "Binomial Tian";
267     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
268         new BinomialVanillaEngine<Tian>(timeSteps)));
269     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
270         new BinomialVanillaEngine<Tian>(timeSteps)));
271     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
272         new BinomialVanillaEngine<Tian>(timeSteps)));
273     std::cout << std::setw(widths[0]) << std::left << method
274         << std::fixed
275         << std::setw(widths[1]) << std::left << europeanOption.NPV()
276         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
277         << std::setw(widths[3]) << std::left << americanOption.NPV()
278         << std::endl;
279
280     method = "Binomial Leisen-Reimer";

```

```

281     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
282         new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
283     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
284         new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
285     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
286         new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
287     std::cout << std::setw(widths[0]) << std::left << method
288         << std::fixed
289         << std::setw(widths[1]) << std::left << europeanOption.NPV()
290         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
291         << std::setw(widths[3]) << std::left << americanOption.NPV()
292         << std::endl;
293
294     method = "Binomial Joshi";
295     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
296         new BinomialVanillaEngine<Joshi4>(timeSteps)));
297     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
298         new BinomialVanillaEngine<Joshi4>(timeSteps)));
299     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
300         new BinomialVanillaEngine<Joshi4>(timeSteps)));
301     std::cout << std::setw(widths[0]) << std::left << method
302         << std::fixed
303         << std::setw(widths[1]) << std::left << europeanOption.NPV()
304         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
305         << std::setw(widths[3]) << std::left << americanOption.NPV()
306         << std::endl;
307
308     // Monte Carlo Method
309
310     timeSteps = 1;
311
312     method = "MC (crude)";
313     Size mcSeed = 42;
314
315     boost::shared_ptr<PricingEngine> mcengine1;
316     mcengine1 =
317         MakeMCEuropeanEngine<PseudoRandom>().withSteps(timeSteps)
318             .withTolerance(0.02)
319             .withSeed(mcSeed);
320     europeanOption.setPricingEngine(mcengine1);
321     // Real errorEstimate = europeanOption.errorEstimate();
322     std::cout << std::setw(widths[0]) << std::left << method
323         << std::fixed
324         << std::setw(widths[1]) << std::left << europeanOption.NPV()
325         << std::setw(widths[2]) << std::left << "N/A"
326         << std::setw(widths[3]) << std::left << "N/A"
327         << std::endl;
328
329     method = "MC (Sobol)";
330     Size nSamples = 32768; // 2^15
331
332     boost::shared_ptr<PricingEngine> mcengine2;
333     mcengine2 =
334         MakeMCEuropeanEngine<LowDiscrepancy>().withSteps(timeSteps)
335             .withSamples(nSamples);
336     europeanOption.setPricingEngine(mcengine2);
337     std::cout << std::setw(widths[0]) << std::left << method
338         << std::fixed
339         << std::setw(widths[1]) << std::left << europeanOption.NPV()
340         << std::setw(widths[2]) << std::left << "N/A"
341         << std::setw(widths[3]) << std::left << "N/A"
342         << std::endl;
343
344     method = "MC (Longstaff Schwartz)";
345     boost::shared_ptr<PricingEngine> mcengine3;
346     mcengine3 =
347         MakeMCAmericanEngine<PseudoRandom>().withSteps(100)

```

```

348                                     .withAntitheticVariate()
349                                     .withCalibrationSamples(4096)
350                                     .withTolerance(0.02)
351                                     .withSeed(mcSeed);
352 americanOption.setPricingEngine(mcengine3);
353 std::cout << std::setw(widths[0]) << std::left << method
354           << std::fixed
355           << std::setw(widths[1]) << std::left << "N/A"
356           << std::setw(widths[2]) << std::left << "N/A"
357           << std::setw(widths[3]) << std::left << americanOption.NPV()
358           << std::endl;
359
360 Real seconds = timer.elapsed();
361 Integer hours = int(seconds/3600);
362 seconds -= hours * 3600;
363 Integer minutes = int(seconds/60);
364 seconds -= minutes * 60;
365 std::cout << " \nRun completed in ";
366 if (hours > 0)
367     std::cout << hours << " h ";
368 if (hours > 0 || minutes > 0)
369     std::cout << minutes << " m ";
370 std::cout << std::fixed << std::setprecision(0)
371           << seconds << " s\n" << std::endl;
372
373 return 0;
374 } catch (std::exception& e) {
375     std::cout << e.what() << std::endl;
376     return 1;
377 } catch (...) {
378     std::cout << "unknown error" << std::endl;
379     return 1;
380 }
381 }

```

11.5 FRA.cpp

This example evaluates a forward-rate agreement.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
20 /* This example shows how to set up a term structure and price a simple
21    forward-rate agreement.
22 */
23
24 // the only header you need to use QuantLib
25 #define BOOST_LIB_DIAGNOSTIC
26 # include <ql/quantlib.hpp>
27 #undef BOOST_LIB_DIAGNOSTIC
28
29 #ifdef BOOST_MSVC
30 /* Uncomment the following lines to unmask floating-point
31    exceptions. Warning: unpredictable results can arise...
32
33    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
34    Is there anyone with a definitive word about this?
35 */
36 // #include <float.h>
37 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
38 #endif
39
40 #include <boost/timer.hpp>
41 #include <iostream>
42
43 #define LENGTH(a) (sizeof(a)/sizeof(a[0]))
44
45 using namespace std;
46 using namespace QuantLib;
47
48 #if defined(QL_ENABLE_SESSIONS)
49 namespace QuantLib {
50
51     Integer sessionId() { return 0; }
52
53 }
54 #endif
55
56 int main(int, char* []) {
57
58     try {
59
60         boost::timer timer;
61         std::cout << std::endl;
62
63         /*****
64          *** MARKET DATA ***
65          *****/
66
67         RelinkableHandle<YieldTermStructure> euriborTermStructure;
68         boost::shared_ptr<IborIndex> euribor3m(
69             new Euribor3M(euriborTermStructure));
70
71         Date todaysDate = Date(23, May, 2006);
72         Settings::instance().evaluationDate() = todaysDate;
73
74         Calendar calendar = euribor3m->fixingCalendar();
75         Integer fixingDays = euribor3m->fixingDays();
76         Date settlementDate = calendar.advance(todaysDate, fixingDays, Days);
77
78         std::cout << "Today: " << todaysDate.weekday()
79             << ", " << todaysDate << std::endl;

```

```

80
81     std::cout << "Settlement date: " << settlementDate.weekday()
82         << ", " << settlementDate << std::endl;
83
84
85     // 3 month term FRA quotes (index refers to monthsToStart)
86     Rate threeMonthFraQuote[10];
87
88     threeMonthFraQuote[1]=0.030;
89     threeMonthFraQuote[2]=0.031;
90     threeMonthFraQuote[3]=0.032;
91     threeMonthFraQuote[6]=0.033;
92     threeMonthFraQuote[9]=0.034;
93
94     /*****
95     ***   QUOTES   ***
96     *****/
97
98     // SimpleQuote stores a value which can be manually changed;
99     // other Quote subclasses could read the value from a database
100    // or some kind of data feed.
101
102
103    // FRAs
104    boost::shared_ptr<SimpleQuote> fra1x4Rate(
105        new SimpleQuote(threeMonthFraQuote[1]));
106    boost::shared_ptr<SimpleQuote> fra2x5Rate(
107        new SimpleQuote(threeMonthFraQuote[2]));
108    boost::shared_ptr<SimpleQuote> fra3x6Rate(
109        new SimpleQuote(threeMonthFraQuote[3]));
110    boost::shared_ptr<SimpleQuote> fra6x9Rate(
111        new SimpleQuote(threeMonthFraQuote[6]));
112    boost::shared_ptr<SimpleQuote> fra9x12Rate(
113        new SimpleQuote(threeMonthFraQuote[9]));
114
115    RelinkableHandle<Quote> h1x4; h1x4.linkTo(fra1x4Rate);
116    RelinkableHandle<Quote> h2x5; h2x5.linkTo(fra2x5Rate);
117    RelinkableHandle<Quote> h3x6; h3x6.linkTo(fra3x6Rate);
118    RelinkableHandle<Quote> h6x9; h6x9.linkTo(fra6x9Rate);
119    RelinkableHandle<Quote> h9x12; h9x12.linkTo(fra9x12Rate);
120
121    /*****
122    ***   RATE HELPERS   ***
123    *****/
124
125    // RateHelpers are built from the above quotes together with
126    // other instrument dependant infos. Quotes are passed in
127    // relinkable handles which could be relinked to some other
128    // data source later.
129
130    DayCounter fraDayCounter = euribor3m->dayCounter();
131    BusinessDayConvention convention = euribor3m->businessDayConvention();
132    bool endOfMonth = euribor3m->endOfMonth();
133
134    boost::shared_ptr<RateHelper> fra1x4(
135        new FraRateHelper(h1x4, 1, 4,
136            fixingDays, calendar, convention,
137            endOfMonth, fixingDays,
138            fraDayCounter));
139
140    boost::shared_ptr<RateHelper> fra2x5(
141        new FraRateHelper(h2x5, 2, 5,
142            fixingDays, calendar, convention,
143            endOfMonth, fixingDays,
144            fraDayCounter));
145
146    boost::shared_ptr<RateHelper> fra3x6(

```

```

147         new FraRateHelper(h3x6, 3, 6,
148                           fixingDays, calendar, convention,
149                           endOfMonth, fixingDays,
150                           fraDayCounter));
151
152     boost::shared_ptr<RateHelper> fra6x9(
153         new FraRateHelper(h6x9, 6, 9,
154                           fixingDays, calendar, convention,
155                           endOfMonth, fixingDays,
156                           fraDayCounter));
157
158     boost::shared_ptr<RateHelper> fra9x12(
159         new FraRateHelper(h9x12, 9, 12,
160                           fixingDays, calendar, convention,
161                           endOfMonth, fixingDays,
162                           fraDayCounter));
163
164
165     /*****
166     ** CURVE BUILDING **
167     *****/
168
169     // Any DayCounter would be fine.
170     // ActualActual::ISDA ensures that 30 years is 30.0
171     DayCounter termStructureDayCounter =
172         ActualActual(ActualActual::ISDA);
173
174     double tolerance = 1.0e-15;
175
176     // A FRA curve
177     std::vector<boost::shared_ptr<RateHelper> > fraInstruments;
178
179     fraInstruments.push_back(fra1x4);
180     fraInstruments.push_back(fra2x5);
181     fraInstruments.push_back(fra3x6);
182     fraInstruments.push_back(fra6x9);
183     fraInstruments.push_back(fra9x12);
184
185     boost::shared_ptr<YieldTermStructure> fraTermStructure(
186         new PiecewiseYieldCurve<Discount, LogLinear>(
187             settlementDate, fraInstruments,
188             termStructureDayCounter, tolerance));
189
190
191     // Term structures used for pricing/discounting
192
193     RelinkableHandle<YieldTermStructure> discountingTermStructure;
194     discountingTermStructure.linkTo(fraTermStructure);
195
196
197     /*****
198     *** construct FRA's ***
199     *****/
200
201     Calendar fraCalendar = euribor3m->fixingCalendar();
202     BusinessDayConvention fraBusinessDayConvention =
203         euribor3m->businessDayConvention();
204     Position::Type fraFwdType = Position::Long;
205     Real fraNotional = 100.0;
206     const Integer FraTermMonths = 3;
207     Integer monthsToStart[] = { 1, 2, 3, 6, 9 };
208
209     euriborTermStructure.linkTo(fraTermStructure);
210
211     cout << endl;
212     cout << "Test FRA construction, NPV calculation, and FRA purchase"
213         << endl

```



```

214         << endl;
215
216     Size i;
217     for (i=0; i<LENGTH(monthsToStart); i++) {
218
219         Date fraValueDate = fraCalendar.advance(
220             settlementDate, monthsToStart[i], Months,
221             fraBusinessDayConvention);
222
223         Date fraMaturityDate = fraCalendar.advance(
224             fraValueDate, FraTermMonths, Months,
225             fraBusinessDayConvention);
226
227         Rate fraStrikeRate = threeMonthFraQuote[monthsToStart[i]];
228
229         ForwardRateAgreement myFRA(fraValueDate, fraMaturityDate,
230             fraFwdType, fraStrikeRate,
231             fraNotional, euribor3m,
232             discountingTermStructure);
233
234         cout << "3m Term FRA, Months to Start: "
235             << monthsToStart[i]
236             << endl;
237         cout << "strike FRA rate: "
238             << io::rate(fraStrikeRate)
239             << endl;
240         cout << "FRA 3m forward rate: "
241             << myFRA.forwardRate()
242             << endl;
243         cout << "FRA market quote: "
244             << io::rate(threeMonthFraQuote[monthsToStart[i]])
245             << endl;
246         cout << "FRA spot value: "
247             << myFRA.spotValue()
248             << endl;
249         cout << "FRA forward value: "
250             << myFRA.forwardValue()
251             << endl;
252         cout << "FRA implied Yield: "
253             << myFRA.impliedYield(myFRA.spotValue(),
254                 myFRA.forwardValue(),
255                 settlementDate,
256                 Simple,
257                 fraDayCounter)
258             << endl;
259         cout << "market Zero Rate: "
260             << discountingTermStructure->zeroRate(fraMaturityDate,
261                 fraDayCounter,
262                 Simple)
263             << endl;
264         cout << "FRA NPV [should be zero]: "
265             << myFRA.NPV()
266             << endl;
267         << endl;
268     }
269
270
271
272
273
274     cout << endl << endl;
275     cout << "Now take a 100 basis-point upward shift in FRA quotes "
276         << "and examine NPV"
277         << endl
278         << endl;
279
280     const Real BpsShift = 0.01;

```

```

281
282     threeMonthFraQuote[1]=0.030+BpsShift;
283     threeMonthFraQuote[2]=0.031+BpsShift;
284     threeMonthFraQuote[3]=0.032+BpsShift;
285     threeMonthFraQuote[6]=0.033+BpsShift;
286     threeMonthFraQuote[9]=0.034+BpsShift;
287
288     fra1x4Rate->setValue(threeMonthFraQuote[1]);
289     fra2x5Rate->setValue(threeMonthFraQuote[2]);
290     fra3x6Rate->setValue(threeMonthFraQuote[3]);
291     fra6x9Rate->setValue(threeMonthFraQuote[6]);
292     fra9x12Rate->setValue(threeMonthFraQuote[9]);
293
294
295     for (i=0; i<LENGTH(monthsToStart); i++) {
296
297         Date fraValueDate = fraCalendar.advance(
298             settlementDate,monthsToStart[i],Months,
299             fraBusinessDayConvention);
300
301         Date fraMaturityDate = fraCalendar.advance(
302             fraValueDate,FraTermMonths,Months,
303             fraBusinessDayConvention);
304
305         Rate fraStrikeRate =
306             threeMonthFraQuote[monthsToStart[i]] - BpsShift;
307
308         ForwardRateAgreement myFRA(fraValueDate, fraMaturityDate,
309             fraFwdType, fraStrikeRate,
310             fraNotional, euribor3m,
311             discountingTermStructure);
312
313         cout << "3m Term FRA, 100 notional, Months to Start = "
314             << monthsToStart[i]
315             << endl;
316         cout << "strike FRA rate: "
317             << io::rate(fraStrikeRate)
318             << endl;
319         cout << "FRA 3m forward rate: "
320             << myFRA.forwardRate()
321             << endl;
322         cout << "FRA market quote: "
323             << io::rate(threeMonthFraQuote[monthsToStart[i]])
324             << endl;
325         cout << "FRA spot value: "
326             << myFRA.spotValue()
327             << endl;
328         cout << "FRA forward value: "
329             << myFRA.forwardValue()
330             << endl;
331         cout << "FRA implied Yield: "
332             << myFRA.impliedYield(myFRA.spotValue(),
333                 myFRA.forwardValue(),
334                 settlementDate,
335                 Simple,
336                 fraDayCounter)
337             << endl;
338         cout << "market Zero Rate: "
339             << discountingTermStructure->zeroRate(fraMaturityDate,
340                 fraDayCounter,
341                 Simple)
342             << endl;
343         cout << "FRA NPV [should be positive]: "
344             << myFRA.NPV()
345             << endl
346             << endl;
347     }

```

```
348
349     Real seconds = timer.elapsed();
350     Integer hours = int(seconds/3600);
351     seconds -= hours * 3600;
352     Integer minutes = int(seconds/60);
353     seconds -= minutes * 60;
354     cout << " \nRun completed in ";
355     if (hours > 0)
356         cout << hours << " h ";
357     if (hours > 0 || minutes > 0)
358         cout << minutes << " m ";
359     cout << fixed << setprecision(0)
360         << seconds << " s\n" << endl;
361
362     return 0;
363
364 } catch (exception& e) {
365     cout << e.what() << endl;
366     return 1;
367 } catch (...) {
368     cout << "unknown error" << endl;
369     return 1;
370 }
371 }
372
```

11.6 Replication.cpp

This example uses the CompositeInstrument class to perform static replication of a down-and-out barrier option.

```

1 /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
20 /* This example showcases the CompositeInstrument class. Such class
21    is used to build a static replication of a down-and-out barrier
22    option, as outlined in Section 10.2 of Mark Joshi's "The Concepts
23    and Practice of Mathematical Finance" to which we refer the
24    reader.
25 */
26
27 // the only header you need to use QuantLib
28 #define BOOST_LIB_DIAGNOSTIC
29 # include <ql/quantlib.hpp>
30 #undef BOOST_LIB_DIAGNOSTIC
31
32 #ifdef BOOST_MSVC
33 /* Uncomment the following lines to unmask floating-point
34    exceptions. Warning: unpredictable results can arise...
35
36    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
37    Is there anyone with a definitive word about this?
38 */
39 // #include <float.h>
40 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
41 #endif
42
43 #include <boost/timer.hpp>
44 #include <iostream>
45 #include <iomanip>
46
47 using namespace QuantLib;
48
49 #if defined(QL_ENABLE_SESSIONS)
50 namespace QuantLib {
51     Integer sessionId() { return 0; }
52 }
53 #endif
54
55 int main(int, char* []) {
56     try {
57         boost::timer timer;
58         std::cout << std::endl;
59
60         Date today(29, May, 2006);
61         Settings::instance().evaluationDate() = today;
62
63         // the option to replicate
64         Barrier::Type barrierType = Barrier::DownOut;
65         Real barrier = 70.0;
66         Real rebate = 0.0;
67         Option::Type type = Option::Put;
68         Real underlyingValue = 100.0;
69         boost::shared_ptr<SimpleQuote> underlying(
70             new SimpleQuote(underlyingValue));
71
72         Real strike = 100.0;
73         boost::shared_ptr<SimpleQuote> riskFreeRate(new SimpleQuote(0.04));
74         boost::shared_ptr<SimpleQuote> volatility(new SimpleQuote(0.20));
75         Date maturity = today + 1*Years;

```

```

79
80     std::cout << std::endl ;
81
82     // write column headings
83     Size widths[] = { 45, 15, 15 };
84     Size totalWidth = widths[0]+widths[1]+widths[2];
85     std::string rule(totalWidth, '-'), dblrule(totalWidth, '=');
86
87     std::cout << dblrule << std::endl;
88     std::cout << "Initial market conditions" << std::endl;
89     std::cout << dblrule << std::endl;
90     std::cout << std::setw(widths[0]) << std::left << "Option"
91               << std::setw(widths[1]) << std::left << "NPV"
92               << std::setw(widths[2]) << std::left << "Error"
93               << std::endl;
94     std::cout << rule << std::endl;
95
96     // bootstrap the yield/vol curves
97     DayCounter dayCounter = Actual365Fixed();
98     Handle<Quote> h1(riskFreeRate);
99     Handle<Quote> h2(volatility);
100    Handle<YieldTermStructure> flatRate(
101        boost::shared_ptr<YieldTermStructure>(
102            new FlatForward(0, NullCalendar(),
103                           h1, dayCounter)));
104    Handle<BlackVolTermStructure> flatVol(
105        boost::shared_ptr<BlackVolTermStructure>(
106            new BlackConstantVol(0, NullCalendar(),
107                                h2, dayCounter)));
108
109    // instantiate the option
110    boost::shared_ptr<Exercise> exercise(
111        new EuropeanExercise(maturity));
112    boost::shared_ptr<StrikedTypePayoff> payoff(
113        new PlainVanillaPayoff(type, strike));
114
115    boost::shared_ptr<StochasticProcess> stochasticProcess(
116        new BlackScholesProcess(Handle<Quote>(underlying),
117                                flatRate, flatVol));
118
119    BarrierOption referenceOption(barrierType, barrier, rebate,
120                                stochasticProcess, payoff, exercise);
121
122    Real referenceValue = referenceOption.NPV();
123
124    std::cout << std::setw(widths[0]) << std::left
125              << "Original barrier option"
126              << std::fixed
127              << std::setw(widths[1]) << std::left << referenceValue
128              << std::setw(widths[2]) << std::left << "N/A"
129              << std::endl;
130
131    // Replicating portfolios
132    CompositeInstrument portfolio1, portfolio2, portfolio3;
133
134    // Final payoff first (the same for all portfolios):
135    // as shown in Joshi, a put struck at K...
136    boost::shared_ptr<Instrument> put1(
137        new EuropeanOption(stochasticProcess, payoff, exercise));
138    portfolio1.add(put1);
139    portfolio2.add(put1);
140    portfolio3.add(put1);
141    // ...minus a digital put struck at B of notional K-B...
142    boost::shared_ptr<StrikedTypePayoff> digitalPayoff(
143        new CashOrNothingPayoff(Option::Put, barrier, 1.0));
144    boost::shared_ptr<Instrument> digitalPut(
145        new EuropeanOption(stochasticProcess, digitalPayoff, exercise));

```

```

146 portfolio1.subtract(digitalPut, strike-barrier);
147 portfolio2.subtract(digitalPut, strike-barrier);
148 portfolio3.subtract(digitalPut, strike-barrier);
149 // ...minus a put option struck at B.
150 boost::shared_ptr<StrikedTypePayoff> lowerPayoff(
151     new PlainVanillaPayoff(Option::Put, barrier));
152 boost::shared_ptr<Instrument> put2(
153     new EuropeanOption(stochasticProcess, lowerPayoff, exercise));
154 portfolio1.subtract(put2);
155 portfolio2.subtract(put2);
156 portfolio3.subtract(put2);
157
158 // Now we use puts struck at B to kill the value of the
159 // portfolio on a number of points (B,t). For the first
160 // portfolio, we'll use 12 dates at one-month's distance.
161 Integer i;
162 for (i=12; i>=1; i--) {
163     // First, we instantiate the option...
164     Date innerMaturity = today + i*Months;
165     boost::shared_ptr<Exercise> innerExercise(
166         new EuropeanExercise(innerMaturity));
167     boost::shared_ptr<StrikedTypePayoff> innerPayoff(
168         new PlainVanillaPayoff(Option::Put, barrier));
169     boost::shared_ptr<Instrument> putn(
170         new EuropeanOption(stochasticProcess,
171             innerPayoff, innerExercise));
172     // ...second, we evaluate the current portfolio and the
173     // latest put at (B,t)...
174     Date killDate = today + (i-1)*Months;
175     Settings::instance().evaluationDate() = killDate;
176     underlying->setValue(barrier);
177     Real portfolioValue = portfolio1.NPV();
178     Real putValue = putn->NPV();
179     // ...finally, we estimate the notional that kills the
180     // portfolio value at that point...
181     Real notional = portfolioValue/putValue;
182     // ...and we subtract from the portfolio a put with such
183     // notional.
184     portfolio1.subtract(putn, notional);
185 }
186 // The portfolio being complete, we return to today's market...
187 Settings::instance().evaluationDate() = today;
188 underlying->setValue(underlyingValue);
189 // ...and output the value.
190 Real portfolioValue = portfolio1.NPV();
191 Real error = portfolioValue - referenceValue;
192 std::cout << std::setw(widths[0]) << std::left
193     << "Replicating portfolio (12 dates)"
194     << std::fixed
195     << std::setw(widths[1]) << std::left << portfolioValue
196     << std::setw(widths[2]) << std::left << error
197     << std::endl;
198
199 // For the second portfolio, we'll use 26 dates at two-weeks'
200 // distance.
201 for (i=52; i>=2; i-=2) {
202     // Same as above.
203     Date innerMaturity = today + i*Weeks;
204     boost::shared_ptr<Exercise> innerExercise(
205         new EuropeanExercise(innerMaturity));
206     boost::shared_ptr<StrikedTypePayoff> innerPayoff(
207         new PlainVanillaPayoff(Option::Put, barrier));
208     boost::shared_ptr<Instrument> putn(
209         new EuropeanOption(stochasticProcess,
210             innerPayoff, innerExercise));
211     Date killDate = today + (i-2)*Weeks;
212     Settings::instance().evaluationDate() = killDate;

```

```

213     underlying->setValue(barrier);
214     Real portfolioValue = portfolio2.NPV();
215     Real putValue = putn->NPV();
216     Real notional = portfolioValue/putValue;
217     portfolio2.subtract(putn, notional);
218 }
219 Settings::instance().evaluationDate() = today;
220 underlying->setValue(underlyingValue);
221 portfolioValue = portfolio2.NPV();
222 error = portfolioValue - referenceValue;
223 std::cout << std::setw(widths[0]) << std::left
224           << "Replicating portfolio (26 dates)"
225           << std::fixed
226           << std::setw(widths[1]) << std::left << portfolioValue
227           << std::setw(widths[2]) << std::left << error
228           << std::endl;
229
230 // For the third portfolio, we'll use 52 dates at one-week's
231 // distance.
232 for (i=52; i>=1; i--) {
233     // Same as above.
234     Date innerMaturity = today + i*Weeks;
235     boost::shared_ptr<Exercise> innerExercise(
236         new EuropeanExercise(innerMaturity));
237     boost::shared_ptr<StrikedTypePayoff> innerPayoff(
238         new PlainVanillaPayoff(Option::Put, barrier));
239     boost::shared_ptr<Instrument> putn(
240         new EuropeanOption(stochasticProcess,
241                             innerPayoff, innerExercise));
242     Date killDate = today + (i-1)*Weeks;
243     Settings::instance().evaluationDate() = killDate;
244     underlying->setValue(barrier);
245     Real portfolioValue = portfolio3.NPV();
246     Real putValue = putn->NPV();
247     Real notional = portfolioValue/putValue;
248     portfolio3.subtract(putn, notional);
249 }
250 Settings::instance().evaluationDate() = today;
251 underlying->setValue(underlyingValue);
252 portfolioValue = portfolio3.NPV();
253 error = portfolioValue - referenceValue;
254 std::cout << std::setw(widths[0]) << std::left
255           << "Replicating portfolio (52 dates)"
256           << std::fixed
257           << std::setw(widths[1]) << std::left << portfolioValue
258           << std::setw(widths[2]) << std::left << error
259           << std::endl;
260
261 // Now we modify the market condition to see whether the
262 // replication holds. First, we change the underlying value so
263 // that the option is out of the money.
264 std::cout << dblrule << std::endl;
265 std::cout << "Modified market conditions: out of the money"
266           << std::endl;
267 std::cout << dblrule << std::endl;
268 std::cout << std::setw(widths[0]) << std::left << "Option"
269           << std::setw(widths[1]) << std::left << "NPV"
270           << std::setw(widths[2]) << std::left << "Error"
271           << std::endl;
272 std::cout << rule << std::endl;
273
274 underlying->setValue(110.0);
275
276 referenceValue = referenceOption.NPV();
277 std::cout << std::setw(widths[0]) << std::left
278           << "Original barrier option"
279           << std::fixed

```

```

280         << std::setw(widths[1]) << std::left << referenceValue
281         << std::setw(widths[2]) << std::left << "N/A"
282         << std::endl;
283     portfolioValue = portfolio1.NPV();
284     error = portfolioValue - referenceValue;
285     std::cout << std::setw(widths[0]) << std::left
286         << "Replicating portfolio (12 dates)"
287         << std::fixed
288         << std::setw(widths[1]) << std::left << portfolioValue
289         << std::setw(widths[2]) << std::left << error
290         << std::endl;
291     portfolioValue = portfolio2.NPV();
292     error = portfolioValue - referenceValue;
293     std::cout << std::setw(widths[0]) << std::left
294         << "Replicating portfolio (26 dates)"
295         << std::fixed
296         << std::setw(widths[1]) << std::left << portfolioValue
297         << std::setw(widths[2]) << std::left << error
298         << std::endl;
299     portfolioValue = portfolio3.NPV();
300     error = portfolioValue - referenceValue;
301     std::cout << std::setw(widths[0]) << std::left
302         << "Replicating portfolio (52 dates)"
303         << std::fixed
304         << std::setw(widths[1]) << std::left << portfolioValue
305         << std::setw(widths[2]) << std::left << error
306         << std::endl;
307
308     // Next, we change the underlying value so that the option is
309     // in the money.
310     std::cout << dblrule << std::endl;
311     std::cout << "Modified market conditions: in the money" << std::endl;
312     std::cout << dblrule << std::endl;
313     std::cout << std::setw(widths[0]) << std::left << "Option"
314         << std::setw(widths[1]) << std::left << "NPV"
315         << std::setw(widths[2]) << std::left << "Error"
316         << std::endl;
317     std::cout << rule << std::endl;
318
319     underlying->setValue(90.0);
320
321     referenceValue = referenceOption.NPV();
322     std::cout << std::setw(widths[0]) << std::left
323         << "Original barrier option"
324         << std::fixed
325         << std::setw(widths[1]) << std::left << referenceValue
326         << std::setw(widths[2]) << std::left << "N/A"
327         << std::endl;
328     portfolioValue = portfolio1.NPV();
329     error = portfolioValue - referenceValue;
330     std::cout << std::setw(widths[0]) << std::left
331         << "Replicating portfolio (12 dates)"
332         << std::fixed
333         << std::setw(widths[1]) << std::left << portfolioValue
334         << std::setw(widths[2]) << std::left << error
335         << std::endl;
336     portfolioValue = portfolio2.NPV();
337     error = portfolioValue - referenceValue;
338     std::cout << std::setw(widths[0]) << std::left
339         << "Replicating portfolio (26 dates)"
340         << std::fixed
341         << std::setw(widths[1]) << std::left << portfolioValue
342         << std::setw(widths[2]) << std::left << error
343         << std::endl;
344     portfolioValue = portfolio3.NPV();
345     error = portfolioValue - referenceValue;
346     std::cout << std::setw(widths[0]) << std::left

```



```
347         << "Replicating portfolio (52 dates)"
348         << std::fixed
349         << std::setw(widths[1]) << std::left << portfolioValue
350         << std::setw(widths[2]) << std::left << error
351         << std::endl;
352
353     // Finally, a word of warning for those (shame on them) who
354     // run the example but do not read the code.
355     std::cout << dblrule << std::endl;
356     std::cout
357         << std::endl
358         << "The replication seems to be less robust when volatility and \n"
359         << "risk-free rate are changed. Feel free to experiment with \n"
360         << "the example and contribute a patch if you spot any errors."
361         << std::endl;
362
363     Real seconds = timer.elapsed();
364     Integer hours = int(seconds/3600);
365     seconds -= hours * 3600;
366     Integer minutes = int(seconds/60);
367     seconds -= minutes * 60;
368     std::cout << " \nRun completed in ";
369     if (hours > 0)
370         std::cout << hours << " h ";
371     if (hours > 0 || minutes > 0)
372         std::cout << minutes << " m ";
373     std::cout << std::fixed << std::setprecision(0)
374         << seconds << " s\n" << std::endl;
375
376     return 0;
377 } catch (std::exception& e) {
378     std::cout << e.what() << std::endl;
379     return 1;
380 } catch (...) {
381     std::cout << "unknown error" << std::endl;
382     return 1;
383 }
384 }
```

11.7 Repo.cpp

This example evaluates a repo on a fixed-coupon bond.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
20 /* a Repo calculation done using the FixedRateBondForward class
21    cf. aaBondFwd() repo example at
22    http://www.fincad.com/support/developerFunc/mathref/BFWD.htm
23
24    This repo is set up to use the repo rate to do all discounting
25    (including the underlying bond income). Forward delivery price is
26    also obtained using this repo rate. All this is done by supplying
27    the FixedRateBondForward constructor with a flat repo
28    YieldTermStructure.
29 */
30
31 // the only header you need to use QuantLib
32 #define BOOST_LIB_DIAGNOSTIC
33 # include <ql/quantlib.hpp>
34 #undef BOOST_LIB_DIAGNOSTIC
35
36 #ifdef BOOST_MSVC
37 /* Uncomment the following lines to unmask floating-point
38    exceptions. Warning: unpredictable results can arise...
39
40    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
41    Is there anyone with a definitive word about this?
42 */
43 // #include <float.h>
44 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
45 #endif
46
47 #include <boost/timer.hpp>
48 #include <iostream>
49
50 using namespace std;
51 using namespace QuantLib;
52
53 #if defined(QL_ENABLE_SESSIONS)
54 namespace QuantLib {
55     Integer sessionId() { return 0; }
56 }
57 #endif
58
59 int main(int, char* []) {
60     try {
61         boost::timer timer;
62         std::cout << std::endl;
63
64         Date repoSettlementDate(14, February, 2000);
65         Date repoDeliveryDate(15, August, 2000);
66         Rate repoRate = 0.05;
67         DayCounter repoDayCountConvention = Actual360();
68         Integer repoSettlementDays = 0;
69         Compounding repoCompounding = Simple;
70         Frequency repoCompoundFreq = Annual;
71
72         // assume a ten year bond- this is irrelevant
73         Date bondIssueDate(15, September, 1995);
74         Date bondDatedDate(15, September, 1995);
75         Date bondMaturityDate(15, September, 2005);

```

```

80     Real bondCoupon = 0.08;
81     Frequency bondCouponFrequency = Semiannual;
82     // unknown what calendar fincad is using
83     Calendar bondCalendar = NullCalendar();
84     DayCounter bondDayCountConvention = Thirty360(Thirty360::BondBasis);
85     // unknown what fincad is using. this may affect accrued calculation
86     Integer bondSettlementDays = 0;
87     BusinessDayConvention bondBusinessDayConvention = Unadjusted;
88     Real bondCleanPrice = 89.97693786;
89     Real bondRedemption = 100.0;
90     Real faceAmount = 100.0;
91
92
93     Settings::instance().evaluationDate() = repoSettlementDate;
94
95     RelinkableHandle<YieldTermStructure> bondCurve;
96     bondCurve.linkTo(boost::shared_ptr<YieldTermStructure>(
97         new FlatForward(repoSettlementDate,
98             .01, // dummy rate
99             bondDayCountConvention,
100             Compounded,
101             bondCouponFrequency)));
102
103     /*
104     boost::shared_ptr<FixedRateBond> bond(
105         new FixedRateBond(faceAmount,
106             bondIssueDate,
107             bondDatedDate,
108             bondMaturityDate,
109             bondSettlementDays,
110             std::vector<Rate>(1,bondCoupon),
111             bondCouponFrequency,
112             bondCalendar,
113             bondDayCountConvention,
114             bondBusinessDayConvention,
115             bondBusinessDayConvention,
116             bondRedemption,
117             bondCurve));
118     */
119
120     Schedule bondSchedule(bondDatedDate, bondMaturityDate,
121         Period(bondCouponFrequency),
122         bondCalendar, bondBusinessDayConvention,
123         bondBusinessDayConvention, true, false);
124     boost::shared_ptr<FixedRateBond> bond(
125         new FixedRateBond(bondSettlementDays,
126             faceAmount,
127             bondSchedule,
128             std::vector<Rate>(1,bondCoupon),
129             bondDayCountConvention,
130             bondBusinessDayConvention,
131             bondRedemption,
132             bondIssueDate,
133             bondCurve));
134
135     bondCurve.linkTo(boost::shared_ptr<YieldTermStructure> (
136         new FlatForward(repoSettlementDate,
137             bond->yield(bondCleanPrice, Compounded),
138             bondDayCountConvention,
139             Compounded,
140             bondCouponFrequency)));
141
142     Position::Type fwdType = Position::Long;
143     double dummyStrike = 91.5745;
144
145     RelinkableHandle<YieldTermStructure> repoCurve;
146     repoCurve.linkTo(boost::shared_ptr<YieldTermStructure> (

```

```

147         new FlatForward(repoSettlementDate,
148                         repoRate,
149                         repoDayCountConvention,
150                         repoCompounding,
151                         repoCompoundFreq));
152
153
154     FixedRateBondForward bondFwd(repoSettlementDate,
155                                 repoDeliveryDate,
156                                 fwdType,
157                                 dummyStrike,
158                                 repoSettlementDays,
159                                 repoDayCountConvention,
160                                 bondCalendar,
161                                 bondBusinessDayConvention,
162                                 bond,
163                                 repoCurve,
164                                 repoCurve);
165
166
167     cout << "Underlying bond clean price: "
168           << bond->cleanPrice()
169           << endl;
170     cout << "Underlying bond dirty price: "
171           << bond->dirtyPrice()
172           << endl;
173     cout << "Underlying bond accrued at settlement: "
174           << bond->accruedAmount(repoSettlementDate)
175           << endl;
176     cout << "Underlying bond accrued at delivery: "
177           << bond->accruedAmount(repoDeliveryDate)
178           << endl;
179     cout << "Underlying bond spot income: "
180           << bondFwd.spotIncome(repoCurve)
181           << endl;
182     cout << "Underlying bond fwd income: "
183           << bondFwd.spotIncome(repoCurve)/
184           << repoCurve->discount(repoDeliveryDate)
185           << endl;
186     cout << "Repo strike: "
187           << dummyStrike
188           << endl;
189     cout << "Repo NPV: "
190           << bondFwd.NPV()
191           << endl;
192     cout << "Repo clean forward price: "
193           << bondFwd.cleanForwardPrice()
194           << endl;
195     cout << "Repo dirty forward price: "
196           << bondFwd.forwardPrice()
197           << endl;
198     cout << "Repo implied yield: "
199           << bondFwd.impliedYield(bond->dirtyPrice(),
200                                 dummyStrike,
201                                 repoSettlementDate,
202                                 repoCompounding,
203                                 repoDayCountConvention)
204           << endl;
205     cout << "Market repo rate: "
206           << repoCurve->zeroRate(repoDeliveryDate,
207                                 repoDayCountConvention,
208                                 repoCompounding,
209                                 repoCompoundFreq)
210           << endl;
211           << endl;
212
213     cout << "Compare with example given at \n"

```

```
214         << "http://www.fincad.com/support/developerFunc/mathref/BFWD.htm"
215         << endl;
216     cout << "Clean forward price = 88.2408"
217         << endl
218         << endl;
219     cout << "In that example, it is unknown what bond calendar they are\n"
220         << "using, as well as settlement Days. For that reason, I have\n"
221         << "made the simplest possible assumptions here: NullCalendar\n"
222         << "and 0 settlement days."
223         << endl;
224
225
226     Real seconds = timer.elapsed();
227     Integer hours = int(seconds/3600);
228     seconds -= hours * 3600;
229     Integer minutes = int(seconds/60);
230     seconds -= minutes * 60;
231     cout << " \nRun completed in ";
232     if (hours > 0)
233         cout << hours << " h ";
234     if (hours > 0 || minutes > 0)
235         cout << minutes << " m ";
236     cout << fixed << setprecision(0)
237         << seconds << " s\n" << endl;
238
239     return 0;
240
241 } catch (exception& e) {
242     cout << e.what() << endl;
243     return 1;
244 } catch (...) {
245     cout << "unknown error" << endl;
246     return 1;
247 }
248 }
249
```

11.8 swapvaluation.cpp

This is an example of using the QuantLib Term Structure for pricing a simple swap.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
22 /* This example shows how to set up a Term Structure and then price a simple
23    swap.
24 */
25
26 // the only header you need to use QuantLib
27 #define BOOST_LIB_DIAGNOSTIC
28 # include <ql/quantlib.hpp>
29 #undef BOOST_LIB_DIAGNOSTIC
30
31 #ifdef BOOST_MSVC
32 /* Uncomment the following lines to unmask floating-point
33    exceptions. Warning: unpredictable results can arise...
34
35    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
36    Is there anyone with a definitive word about this?
37 */
38 // #include <float.h>
39 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
40 #endif
41
42 #include <boost/timer.hpp>
43 #include <iostream>
44 #include <iomanip>
45
46 using namespace QuantLib;
47
48 #if defined(QL_ENABLE_SESSIONS)
49 namespace QuantLib {
50
51     Integer sessionId() { return 0; }
52
53 }
54 #endif
55
56
57 int main(int, char* []) {
58     try {
59
60         boost::timer timer;
61         std::cout << std::endl;
62
63         /*****
64          *** MARKET DATA ***
65          *****/
66
67         Calendar calendar = TARGET();
68         // uncommenting the following line generates an error
69         // calendar = Tokyo();
70         Date settlementDate(22, September, 2004);
71         // must be a business day
72         settlementDate = calendar.adjust(settlementDate);
73
74         Integer fixingDays = 2;
75         Date todaysDate = calendar.advance(settlementDate, -fixingDays, Days);
76         // nothing to do with Date::todaysDate
77         Settings::instance().evaluationDate() = todaysDate;
78
79
80         todaysDate = Settings::instance().evaluationDate();

```

```

82     std::cout << "Today: " << todaysDate.weekday()
83         << ", " << todaysDate << std::endl;
84
85     std::cout << "Settlement date: " << settlementDate.weekday()
86         << ", " << settlementDate << std::endl;
87
88     // deposits
89     Rate d1wQuote=0.0382;
90     Rate d1mQuote=0.0372;
91     Rate d3mQuote=0.0363;
92     Rate d6mQuote=0.0353;
93     Rate d9mQuote=0.0348;
94     Rate d1yQuote=0.0345;
95     // FRAs
96     Rate fra3x6Quote=0.037125;
97     Rate fra6x9Quote=0.037125;
98     Rate fra6x12Quote=0.037125;
99     // futures
100    Real fut1Quote=96.2875;
101    Real fut2Quote=96.7875;
102    Real fut3Quote=96.9875;
103    Real fut4Quote=96.6875;
104    Real fut5Quote=96.4875;
105    Real fut6Quote=96.3875;
106    Real fut7Quote=96.2875;
107    Real fut8Quote=96.0875;
108    // swaps
109    Rate s2yQuote=0.037125;
110    Rate s3yQuote=0.0398;
111    Rate s5yQuote=0.0443;
112    Rate s10yQuote=0.05165;
113    Rate s15yQuote=0.055175;
114
115
116    /*****
117    ***   QUOTES   ***
118    *****/
119
120    // SimpleQuote stores a value which can be manually changed;
121    // other Quote subclasses could read the value from a database
122    // or some kind of data feed.
123
124    // deposits
125    boost::shared_ptr<Quote> d1wRate(new SimpleQuote(d1wQuote));
126    boost::shared_ptr<Quote> d1mRate(new SimpleQuote(d1mQuote));
127    boost::shared_ptr<Quote> d3mRate(new SimpleQuote(d3mQuote));
128    boost::shared_ptr<Quote> d6mRate(new SimpleQuote(d6mQuote));
129    boost::shared_ptr<Quote> d9mRate(new SimpleQuote(d9mQuote));
130    boost::shared_ptr<Quote> d1yRate(new SimpleQuote(d1yQuote));
131    // FRAs
132    boost::shared_ptr<Quote> fra3x6Rate(new SimpleQuote(fra3x6Quote));
133    boost::shared_ptr<Quote> fra6x9Rate(new SimpleQuote(fra6x9Quote));
134    boost::shared_ptr<Quote> fra6x12Rate(new SimpleQuote(fra6x12Quote));
135    // futures
136    boost::shared_ptr<Quote> fut1Price(new SimpleQuote(fut1Quote));
137    boost::shared_ptr<Quote> fut2Price(new SimpleQuote(fut2Quote));
138    boost::shared_ptr<Quote> fut3Price(new SimpleQuote(fut3Quote));
139    boost::shared_ptr<Quote> fut4Price(new SimpleQuote(fut4Quote));
140    boost::shared_ptr<Quote> fut5Price(new SimpleQuote(fut5Quote));
141    boost::shared_ptr<Quote> fut6Price(new SimpleQuote(fut6Quote));
142    boost::shared_ptr<Quote> fut7Price(new SimpleQuote(fut7Quote));
143    boost::shared_ptr<Quote> fut8Price(new SimpleQuote(fut8Quote));
144    // swaps
145    boost::shared_ptr<Quote> s2yRate(new SimpleQuote(s2yQuote));
146    boost::shared_ptr<Quote> s3yRate(new SimpleQuote(s3yQuote));
147    boost::shared_ptr<Quote> s5yRate(new SimpleQuote(s5yQuote));
148    boost::shared_ptr<Quote> s10yRate(new SimpleQuote(s10yQuote));

```

```

149     boost::shared_ptr<Quote> s15yRate(new SimpleQuote(s15yQuote));
150
151
152     /*****
153     ***   RATE HELPERS   ***
154     *****/
155
156     // RateHelpers are built from the above quotes together with
157     // other instrument dependant infos. Quotes are passed in
158     // relinkable handles which could be relinked to some other
159     // data source later.
160
161     // deposits
162     DayCounter depositDayCounter = Actual360();
163
164     boost::shared_ptr<RateHelper> dlw(new DepositRateHelper(
165         Handle<Quote>(dlwRate),
166         1*Weeks, fixingDays,
167         calendar, ModifiedFollowing,
168         true, fixingDays, depositDayCounter));
169     boost::shared_ptr<RateHelper> d1m(new DepositRateHelper(
170         Handle<Quote>(d1mRate),
171         1*Months, fixingDays,
172         calendar, ModifiedFollowing,
173         true, fixingDays, depositDayCounter));
174     boost::shared_ptr<RateHelper> d3m(new DepositRateHelper(
175         Handle<Quote>(d3mRate),
176         3*Months, fixingDays,
177         calendar, ModifiedFollowing,
178         true, fixingDays, depositDayCounter));
179     boost::shared_ptr<RateHelper> d6m(new DepositRateHelper(
180         Handle<Quote>(d6mRate),
181         6*Months, fixingDays,
182         calendar, ModifiedFollowing,
183         true, fixingDays, depositDayCounter));
184     boost::shared_ptr<RateHelper> d9m(new DepositRateHelper(
185         Handle<Quote>(d9mRate),
186         9*Months, fixingDays,
187         calendar, ModifiedFollowing,
188         true, fixingDays, depositDayCounter));
189     boost::shared_ptr<RateHelper> d1y(new DepositRateHelper(
190         Handle<Quote>(d1yRate),
191         1*Years, fixingDays,
192         calendar, ModifiedFollowing,
193         true, fixingDays, depositDayCounter));
194
195
196     // setup FRAs
197     boost::shared_ptr<RateHelper> fra3x6(new FraRateHelper(
198         Handle<Quote>(fra3x6Rate),
199         3, 6, fixingDays, calendar, ModifiedFollowing,
200         true, fixingDays, depositDayCounter));
201     boost::shared_ptr<RateHelper> fra6x9(new FraRateHelper(
202         Handle<Quote>(fra6x9Rate),
203         6, 9, fixingDays, calendar, ModifiedFollowing,
204         true, fixingDays, depositDayCounter));
205     boost::shared_ptr<RateHelper> fra6x12(new FraRateHelper(
206         Handle<Quote>(fra6x12Rate),
207         6, 12, fixingDays, calendar, ModifiedFollowing,
208         true, fixingDays, depositDayCounter));
209
210
211     // setup futures
212     Rate convexityAdjustment = 0.0;
213     Integer futMonths = 3;
214     Date imm = IMM::nextDate(settlementDate);
215     boost::shared_ptr<RateHelper> fut1(new FuturesRateHelper(

```



```

216         Handle<Quote>(fut1Price),
217         imm,
218         futMonths, calendar, ModifiedFollowing,
219         depositDayCounter, convexityAdjustment));
220     imm = IMM::nextDate(imm+1);
221     boost::shared_ptr<RateHelper> fut2(new FuturesRateHelper(
222         Handle<Quote>(fut1Price),
223         imm,
224         futMonths, calendar, ModifiedFollowing,
225         depositDayCounter, convexityAdjustment));
226     imm = IMM::nextDate(imm+1);
227     boost::shared_ptr<RateHelper> fut3(new FuturesRateHelper(
228         Handle<Quote>(fut1Price),
229         imm,
230         futMonths, calendar, ModifiedFollowing,
231         depositDayCounter, convexityAdjustment));
232     imm = IMM::nextDate(imm+1);
233     boost::shared_ptr<RateHelper> fut4(new FuturesRateHelper(
234         Handle<Quote>(fut1Price),
235         imm,
236         futMonths, calendar, ModifiedFollowing,
237         depositDayCounter, convexityAdjustment));
238     imm = IMM::nextDate(imm+1);
239     boost::shared_ptr<RateHelper> fut5(new FuturesRateHelper(
240         Handle<Quote>(fut1Price),
241         imm,
242         futMonths, calendar, ModifiedFollowing,
243         depositDayCounter, convexityAdjustment));
244     imm = IMM::nextDate(imm+1);
245     boost::shared_ptr<RateHelper> fut6(new FuturesRateHelper(
246         Handle<Quote>(fut1Price),
247         imm,
248         futMonths, calendar, ModifiedFollowing,
249         depositDayCounter, convexityAdjustment));
250     imm = IMM::nextDate(imm+1);
251     boost::shared_ptr<RateHelper> fut7(new FuturesRateHelper(
252         Handle<Quote>(fut1Price),
253         imm,
254         futMonths, calendar, ModifiedFollowing,
255         depositDayCounter, convexityAdjustment));
256     imm = IMM::nextDate(imm+1);
257     boost::shared_ptr<RateHelper> fut8(new FuturesRateHelper(
258         Handle<Quote>(fut1Price),
259         imm,
260         futMonths, calendar, ModifiedFollowing,
261         depositDayCounter, convexityAdjustment));
262
263
264     // setup swaps
265     Frequency swFixedLegFrequency = Annual;
266     BusinessDayConvention swFixedLegConvention = Unadjusted;
267     DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
268     boost::shared_ptr<IborIndex> swFloatingLegIndex(new Euribor6M);
269
270     boost::shared_ptr<RateHelper> s2y(new SwapRateHelper(
271         Handle<Quote>(s2yRate),
272         2*Years, fixingDays,
273         calendar, swFixedLegFrequency,
274         swFixedLegConvention, swFixedLegDayCounter,
275         swFloatingLegIndex));
276     boost::shared_ptr<RateHelper> s3y(new SwapRateHelper(
277         Handle<Quote>(s3yRate),
278         3*Years, fixingDays,
279         calendar, swFixedLegFrequency,
280         swFixedLegConvention, swFixedLegDayCounter,
281         swFloatingLegIndex));
282     boost::shared_ptr<RateHelper> s5y(new SwapRateHelper(

```

```

283         Handle<Quote>(s5yRate),
284         5*Years, fixingDays,
285         calendar, swFixedLegFrequency,
286         swFixedLegConvention, swFixedLegDayCounter,
287         swFloatingLegIndex));
288     boost::shared_ptr<RateHelper> s10y(new SwapRateHelper(
289         Handle<Quote>(s10yRate),
290         10*Years, fixingDays,
291         calendar, swFixedLegFrequency,
292         swFixedLegConvention, swFixedLegDayCounter,
293         swFloatingLegIndex));
294     boost::shared_ptr<RateHelper> s15y(new SwapRateHelper(
295         Handle<Quote>(s15yRate),
296         15*Years, fixingDays,
297         calendar, swFixedLegFrequency,
298         swFixedLegConvention, swFixedLegDayCounter,
299         swFloatingLegIndex));
300
301
302     /*****
303     **  CURVE BUILDING **
304     *****/
305
306     // Any DayCounter would be fine.
307     // ActualActual::ISDA ensures that 30 years is 30.0
308     DayCounter termStructureDayCounter =
309         ActualActual(ActualActual::ISDA);
310
311
312     double tolerance = 1.0e-15;
313
314     // A depo-swap curve
315     std::vector<boost::shared_ptr<RateHelper> > depoSwapInstruments;
316     depoSwapInstruments.push_back(d1w);
317     depoSwapInstruments.push_back(d1m);
318     depoSwapInstruments.push_back(d3m);
319     depoSwapInstruments.push_back(d6m);
320     depoSwapInstruments.push_back(d9m);
321     depoSwapInstruments.push_back(d1y);
322     depoSwapInstruments.push_back(s2y);
323     depoSwapInstruments.push_back(s3y);
324     depoSwapInstruments.push_back(s5y);
325     depoSwapInstruments.push_back(s10y);
326     depoSwapInstruments.push_back(s15y);
327     boost::shared_ptr<YieldTermStructure> depoSwapTermStructure(
328         new PiecewiseYieldCurve<Discount,LogLinear>(
329             settlementDate, depoSwapInstruments,
330             termStructureDayCounter, tolerance));
331
332
333     // A depo-futures-swap curve
334     std::vector<boost::shared_ptr<RateHelper> > depoFutSwapInstruments;
335     depoFutSwapInstruments.push_back(d1w);
336     depoFutSwapInstruments.push_back(d1m);
337     depoFutSwapInstruments.push_back(fut1);
338     depoFutSwapInstruments.push_back(fut2);
339     depoFutSwapInstruments.push_back(fut3);
340     depoFutSwapInstruments.push_back(fut4);
341     depoFutSwapInstruments.push_back(fut5);
342     depoFutSwapInstruments.push_back(fut6);
343     depoFutSwapInstruments.push_back(fut7);
344     depoFutSwapInstruments.push_back(fut8);
345     depoFutSwapInstruments.push_back(s3y);
346     depoFutSwapInstruments.push_back(s5y);
347     depoFutSwapInstruments.push_back(s10y);
348     depoFutSwapInstruments.push_back(s15y);
349     boost::shared_ptr<YieldTermStructure> depoFutSwapTermStructure(

```

```

350         new PiecewiseYieldCurve<Discount,LogLinear>(
351             settlementDate, depoFutSwapInstruments,
352             termStructureDayCounter, tolerance));
353
354
355     // A depo-FRA-swap curve
356     std::vector<boost::shared_ptr<RateHelper> > depoFRASwapInstruments;
357     depoFRASwapInstruments.push_back(d1w);
358     depoFRASwapInstruments.push_back(d1m);
359     depoFRASwapInstruments.push_back(d3m);
360     depoFRASwapInstruments.push_back(fra3x6);
361     depoFRASwapInstruments.push_back(fra6x9);
362     depoFRASwapInstruments.push_back(fra6x12);
363     depoFRASwapInstruments.push_back(s2y);
364     depoFRASwapInstruments.push_back(s3y);
365     depoFRASwapInstruments.push_back(s5y);
366     depoFRASwapInstruments.push_back(s10y);
367     depoFRASwapInstruments.push_back(s15y);
368     boost::shared_ptr<YieldTermStructure> depoFRASwapTermStructure(
369         new PiecewiseYieldCurve<Discount,LogLinear>(
370             settlementDate, depoFRASwapInstruments,
371             termStructureDayCounter, tolerance));
372
373
374     // Term structures that will be used for pricing:
375     // the one used for discounting cash flows
376     RelinkableHandle<YieldTermStructure> discountingTermStructure;
377     // the one used for forward rate forecasting
378     RelinkableHandle<YieldTermStructure> forecastingTermStructure;
379
380
381     /*****
382     * SWAPS TO BE PRICED *
383     *****/
384
385     // constant nominal 1,000,000 Euro
386     Real nominal = 1000000.0;
387     // fixed leg
388     Frequency fixedLegFrequency = Annual;
389     BusinessDayConvention fixedLegConvention = Unadjusted;
390     BusinessDayConvention floatingLegConvention = ModifiedFollowing;
391     DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
392     Rate fixedRate = 0.04;
393     DayCounter floatingLegDayCounter = Actual360();
394
395     // floating leg
396     Frequency floatingLegFrequency = Semiannual;
397     boost::shared_ptr<IborIndex> euriborIndex(
398         new Euribor6M(forecastingTermStructure));
399     Spread spread = 0.0;
400
401     Integer lenghtInYears = 5;
402     VanillaSwap::Type swapType = VanillaSwap::Payer;
403
404     Date maturity = settlementDate + lenghtInYears*Years;
405     Schedule fixedSchedule(settlementDate, maturity,
406         Period(fixedLegFrequency),
407         calendar, fixedLegConvention,
408         fixedLegConvention,
409         false, false);
410     Schedule floatSchedule(settlementDate, maturity,
411         Period(floatingLegFrequency),
412         calendar, floatingLegConvention,
413         floatingLegConvention,
414         false, false);
415     VanillaSwap spot5YearSwap(swapType, nominal,
416         fixedSchedule, fixedRate, fixedLegDayCounter,

```

```

417         floatSchedule, euriborIndex, spread,
418         floatingLegDayCounter, discountingTermStructure);
419
420     Date fwdStart = calendar.advance(settlementDate, 1, Years);
421     Date fwdMaturity = fwdStart + lenghtInYears*Years;
422     Schedule fwdFixedSchedule(fwdStart, fwdMaturity,
423                               Period(fixedLegFrequency),
424                               calendar, fixedLegConvention,
425                               fixedLegConvention,
426                               false, false);
427     Schedule fwdFloatSchedule(fwdStart, fwdMaturity,
428                               Period(floatingLegFrequency),
429                               calendar, floatingLegConvention,
430                               floatingLegConvention,
431                               false, false);
432     VanillaSwap oneYearForward5YearSwap(swapType, nominal,
433     fwdFixedSchedule, fixedRate, fixedLegDayCounter,
434     fwdFloatSchedule, euriborIndex, spread,
435     floatingLegDayCounter, discountingTermStructure);
436
437
438     /*****
439     * SWAP PRICING *
440     *****/
441
442     // utilities for reporting
443     std::vector<std::string> headers(4);
444     headers[0] = "term structure";
445     headers[1] = "net present value";
446     headers[2] = "fair spread";
447     headers[3] = "fair fixed rate";
448     std::string separator = " | ";
449     Size width = headers[0].size() + separator.size()
450                 + headers[1].size() + separator.size()
451                 + headers[2].size() + separator.size()
452                 + headers[3].size() + separator.size() - 1;
453     std::string rule(width, '-'), dblrule(width, '=');
454     std::string tab(8, ' ');
455
456     // calculations
457
458     std::cout << dblrule << std::endl;
459     std::cout << "5-year market swap-rate = "
460               << std::setprecision(2) << io::rate(s5yRate->value())
461               << std::endl;
462     std::cout << dblrule << std::endl;
463
464     std::cout << tab << "5-years swap paying "
465               << io::rate(fixedRate) << std::endl;
466     std::cout << headers[0] << separator
467               << headers[1] << separator
468               << headers[2] << separator
469               << headers[3] << separator << std::endl;
470     std::cout << rule << std::endl;
471
472     Real NPV;
473     Rate fairRate;
474     Spread fairSpread;
475
476     // Of course, you're not forced to really use different curves
477     forecastingTermStructure.linkTo(depoSwapTermStructure);
478     discountingTermStructure.linkTo(depoSwapTermStructure);
479
480     NPV = spot5YearSwap.NPV();
481     fairSpread = spot5YearSwap.fairSpread();
482     fairRate = spot5YearSwap.fairRate();
483

```

```

484     std::cout << std::setw(headers[0].size())
485     << "depo-swap" << separator;
486     std::cout << std::setw(headers[1].size())
487     << std::fixed << std::setprecision(2) << NPV << separator;
488     std::cout << std::setw(headers[2].size())
489     << io::rate(fairSpread) << separator;
490     std::cout << std::setw(headers[3].size())
491     << io::rate(fairRate) << separator;
492     std::cout << std::endl;
493
494
495     // let's check that the 5 years swap has been correctly re-priced
496     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
497     "5-years swap mispriced by "
498     << io::rate(std::fabs(fairRate-s5yQuote)));
499
500
501     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
502     discountingTermStructure.linkTo(depoFutSwapTermStructure);
503
504     NPV = spot5YearSwap.NPV();
505     fairSpread = spot5YearSwap.fairSpread();
506     fairRate = spot5YearSwap.fairRate();
507
508     std::cout << std::setw(headers[0].size())
509     << "depo-fut-swap" << separator;
510     std::cout << std::setw(headers[1].size())
511     << std::fixed << std::setprecision(2) << NPV << separator;
512     std::cout << std::setw(headers[2].size())
513     << io::rate(fairSpread) << separator;
514     std::cout << std::setw(headers[3].size())
515     << io::rate(fairRate) << separator;
516     std::cout << std::endl;
517
518     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
519     "5-years swap mispriced!");
520
521
522     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
523     discountingTermStructure.linkTo(depoFRASwapTermStructure);
524
525     NPV = spot5YearSwap.NPV();
526     fairSpread = spot5YearSwap.fairSpread();
527     fairRate = spot5YearSwap.fairRate();
528
529     std::cout << std::setw(headers[0].size())
530     << "depo-FRA-swap" << separator;
531     std::cout << std::setw(headers[1].size())
532     << std::fixed << std::setprecision(2) << NPV << separator;
533     std::cout << std::setw(headers[2].size())
534     << io::rate(fairSpread) << separator;
535     std::cout << std::setw(headers[3].size())
536     << io::rate(fairRate) << separator;
537     std::cout << std::endl;
538
539     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
540     "5-years swap mispriced!");
541
542
543     std::cout << rule << std::endl;
544
545     // now let's price the 1Y forward 5Y swap
546
547     std::cout << tab << "5-years, 1-year forward swap paying "
548     << io::rate(fixedRate) << std::endl;
549     std::cout << headers[0] << separator
550     << headers[1] << separator

```

```

551         << headers[2] << separator
552         << headers[3] << separator << std::endl;
553     std::cout << rule << std::endl;
554
555
556     forecastingTermStructure.linkTo(depoSwapTermStructure);
557     discountingTermStructure.linkTo(depoSwapTermStructure);
558
559     NPV = oneYearForward5YearSwap.NPV();
560     fairSpread = oneYearForward5YearSwap.fairSpread();
561     fairRate = oneYearForward5YearSwap.fairRate();
562
563     std::cout << std::setw(headers[0].size())
564               << "depo-swap" << separator;
565     std::cout << std::setw(headers[1].size())
566               << std::fixed << std::setprecision(2) << NPV << separator;
567     std::cout << std::setw(headers[2].size())
568               << io::rate(fairSpread) << separator;
569     std::cout << std::setw(headers[3].size())
570               << io::rate(fairRate) << separator;
571     std::cout << std::endl;
572
573
574     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
575     discountingTermStructure.linkTo(depoFutSwapTermStructure);
576
577     NPV = oneYearForward5YearSwap.NPV();
578     fairSpread = oneYearForward5YearSwap.fairSpread();
579     fairRate = oneYearForward5YearSwap.fairRate();
580
581     std::cout << std::setw(headers[0].size())
582               << "depo-fut-swap" << separator;
583     std::cout << std::setw(headers[1].size())
584               << std::fixed << std::setprecision(2) << NPV << separator;
585     std::cout << std::setw(headers[2].size())
586               << io::rate(fairSpread) << separator;
587     std::cout << std::setw(headers[3].size())
588               << io::rate(fairRate) << separator;
589     std::cout << std::endl;
590
591
592     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
593     discountingTermStructure.linkTo(depoFRASwapTermStructure);
594
595     NPV = oneYearForward5YearSwap.NPV();
596     fairSpread = oneYearForward5YearSwap.fairSpread();
597     fairRate = oneYearForward5YearSwap.fairRate();
598
599     std::cout << std::setw(headers[0].size())
600               << "depo-FRA-swap" << separator;
601     std::cout << std::setw(headers[1].size())
602               << std::fixed << std::setprecision(2) << NPV << separator;
603     std::cout << std::setw(headers[2].size())
604               << io::rate(fairSpread) << separator;
605     std::cout << std::setw(headers[3].size())
606               << io::rate(fairRate) << separator;
607     std::cout << std::endl;
608
609
610     // now let's say that the 5-years swap rate goes up to 4.60%.
611     // A smarter market element--say, connected to a data source-- would
612     // notice the change itself. Since we're using SimpleQuotes,
613     // we'll have to change the value manually--which forces us to
614     // downcast the handle and use the SimpleQuote
615     // interface. In any case, the point here is that a change in the
616     // value contained in the Quote triggers a new bootstrapping
617     // of the curve and a repricing of the swap.

```

```

618
619     boost::shared_ptr<SimpleQuote> fiveYearsRate =
620         boost::dynamic_pointer_cast<SimpleQuote>(s5yRate);
621     fiveYearsRate->setValue(0.0460);
622
623     std::cout << dblrule << std::endl;
624     std::cout << "5-year market swap-rate = "
625         << io::rate(s5yRate->value()) << std::endl;
626     std::cout << dblrule << std::endl;
627
628     std::cout << tab << "5-years swap paying "
629         << io::rate(fixedRate) << std::endl;
630     std::cout << headers[0] << separator
631         << headers[1] << separator
632         << headers[2] << separator
633         << headers[3] << separator << std::endl;
634     std::cout << rule << std::endl;
635
636     // now get the updated results
637     forecastingTermStructure.linkTo(depoSwapTermStructure);
638     discountingTermStructure.linkTo(depoSwapTermStructure);
639
640     NPV = spot5YearSwap.NPV();
641     fairSpread = spot5YearSwap.fairSpread();
642     fairRate = spot5YearSwap.fairRate();
643
644     std::cout << std::setw(headers[0].size())
645         << "depo-swap" << separator;
646     std::cout << std::setw(headers[1].size())
647         << std::fixed << std::setprecision(2) << NPV << separator;
648     std::cout << std::setw(headers[2].size())
649         << io::rate(fairSpread) << separator;
650     std::cout << std::setw(headers[3].size())
651         << io::rate(fairRate) << separator;
652     std::cout << std::endl;
653
654     QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
655         "5-years swap mispriced!");
656
657
658     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
659     discountingTermStructure.linkTo(depoFutSwapTermStructure);
660
661     NPV = spot5YearSwap.NPV();
662     fairSpread = spot5YearSwap.fairSpread();
663     fairRate = spot5YearSwap.fairRate();
664
665     std::cout << std::setw(headers[0].size())
666         << "depo-fut-swap" << separator;
667     std::cout << std::setw(headers[1].size())
668         << std::fixed << std::setprecision(2) << NPV << separator;
669     std::cout << std::setw(headers[2].size())
670         << io::rate(fairSpread) << separator;
671     std::cout << std::setw(headers[3].size())
672         << io::rate(fairRate) << separator;
673     std::cout << std::endl;
674
675     QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
676         "5-years swap mispriced!");
677
678
679     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
680     discountingTermStructure.linkTo(depoFRASwapTermStructure);
681
682     NPV = spot5YearSwap.NPV();
683     fairSpread = spot5YearSwap.fairSpread();
684     fairRate = spot5YearSwap.fairRate();

```

```

685
686     std::cout << std::setw(headers[0].size())
687               << "depo-FRA-swap" << separator;
688     std::cout << std::setw(headers[1].size())
689               << std::fixed << std::setprecision(2) << NPV << separator;
690     std::cout << std::setw(headers[2].size())
691               << io::rate(fairSpread) << separator;
692     std::cout << std::setw(headers[3].size())
693               << io::rate(fairRate) << separator;
694     std::cout << std::endl;
695
696     QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
697               "5-years swap mispriced!");
698
699     std::cout << rule << std::endl;
700
701     // the 1Y forward 5Y swap changes as well
702
703     std::cout << tab << "5-years, 1-year forward swap paying "
704               << io::rate(fixedRate) << std::endl;
705     std::cout << headers[0] << separator
706               << headers[1] << separator
707               << headers[2] << separator
708               << headers[3] << separator << std::endl;
709     std::cout << rule << std::endl;
710
711
712     forecastingTermStructure.linkTo(depoSwapTermStructure);
713     discountingTermStructure.linkTo(depoSwapTermStructure);
714
715     NPV = oneYearForward5YearSwap.NPV();
716     fairSpread = oneYearForward5YearSwap.fairSpread();
717     fairRate = oneYearForward5YearSwap.fairRate();
718
719     std::cout << std::setw(headers[0].size())
720               << "depo-swap" << separator;
721     std::cout << std::setw(headers[1].size())
722               << std::fixed << std::setprecision(2) << NPV << separator;
723     std::cout << std::setw(headers[2].size())
724               << io::rate(fairSpread) << separator;
725     std::cout << std::setw(headers[3].size())
726               << io::rate(fairRate) << separator;
727     std::cout << std::endl;
728
729
730     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
731     discountingTermStructure.linkTo(depoFutSwapTermStructure);
732
733     NPV = oneYearForward5YearSwap.NPV();
734     fairSpread = oneYearForward5YearSwap.fairSpread();
735     fairRate = oneYearForward5YearSwap.fairRate();
736
737     std::cout << std::setw(headers[0].size())
738               << "depo-fut-swap" << separator;
739     std::cout << std::setw(headers[1].size())
740               << std::fixed << std::setprecision(2) << NPV << separator;
741     std::cout << std::setw(headers[2].size())
742               << io::rate(fairSpread) << separator;
743     std::cout << std::setw(headers[3].size())
744               << io::rate(fairRate) << separator;
745     std::cout << std::endl;
746
747
748     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
749     discountingTermStructure.linkTo(depoFRASwapTermStructure);
750
751     NPV = oneYearForward5YearSwap.NPV();

```



```
752     fairSpread = oneYearForward5YearSwap.fairSpread();
753     fairRate = oneYearForward5YearSwap.fairRate();
754
755     std::cout << std::setw(headers[0].size())
756               << "depo-FRA-swap" << separator;
757     std::cout << std::setw(headers[1].size())
758               << std::fixed << std::setprecision(2) << NPV << separator;
759     std::cout << std::setw(headers[2].size())
760               << io::rate(fairSpread) << separator;
761     std::cout << std::setw(headers[3].size())
762               << io::rate(fairRate) << separator;
763     std::cout << std::endl;
764
765     Real seconds = timer.elapsed();
766     Integer hours = int(seconds/3600);
767     seconds -= hours * 3600;
768     Integer minutes = int(seconds/60);
769     seconds -= minutes * 60;
770     std::cout << " \nRun completed in ";
771     if (hours > 0)
772         std::cout << hours << " h ";
773     if (hours > 0 || minutes > 0)
774         std::cout << minutes << " m ";
775     std::cout << std::fixed << std::setprecision(0)
776               << seconds << " s\n" << std::endl;
777
778     return 0;
779
780 } catch (std::exception& e) {
781     std::cout << e.what() << std::endl;
782     return 1;
783 } catch (...) {
784     std::cout << "unknown error" << std::endl;
785     return 1;
786 }
787 }
788
```

11.9 tracing_example.cpp

This code exemplifies how to insert trace statements to follow the flow of program execution. When compiler under gcc 3.3 and run, the following program will output the following trace:

```
1      trace[1]: Entering int main()
2      trace[2]: Entering int foo(int)
3      trace[3]: Entering int Foo::bar(int)
4      trace[3]: i = 21
5      trace[3]: At line 16 in tracing_example.cpp
6      trace[3]: Wrong answer
7      trace[3]: i = 42
8      trace[3]: Exiting int Foo::bar(int)
9      trace[3]: Entering int Foo::bar(int)
10     trace[3]: i = 42
11     trace[3]: At line 13 in tracing_example.cpp
12     trace[3]: Right answer, but no question
13     trace[3]: i = 42
14     trace[3]: Exiting int Foo::bar(int)
15     trace[2]: Exiting int foo(int)
16     trace[1]: Exiting int main()
```

Of course, a word of warning must be added: adding so much tracing to your code might degrade its readability, at least until we devise an Emacs macro to hide trace statements with a couple of keystrokes.

```
1
2 #include <ql/quantlib.hpp>
3
4 using namespace QuantLib;
5
6 namespace Foo {
7
8     int bar(int i) {
9         QL_TRACE_ENTER_FUNCTION;
10        QL_TRACE_VARIABLE(i);
11
12        if (i == 42) {
13            QL_TRACE_LOCATION;
14            QL_TRACE("Right answer, but no question");
15        } else {
16            QL_TRACE_LOCATION;
17            QL_TRACE("Wrong answer");
18            i *= 2;
19        }
20
21        QL_TRACE_VARIABLE(i);
22        QL_TRACE_EXIT_FUNCTION;
23        return i;
24    }
25
26 }
27
28 int foo(int i) {
29     using namespace Foo;
30     QL_TRACE_ENTER_FUNCTION;
31
32     int j = bar(i);
33     int k = bar(j);
34
35     QL_TRACE_EXIT_FUNCTION;
36     return k;
37 }
38
```

```
39 int main() {  
40  
41     QL_TRACE_ENABLE;  
42  
43     QL_TRACE_ENTER_FUNCTION;  
44  
45     int i = foo(21);  
46  
47     QL_TRACE_EXIT_FUNCTION;  
48     return 0;  
49 }  
50
```


Chapter 12

Caveats

Class `Actual365Fixed` According to ISDA, "Actual/365" (without "Fixed") is an alias for "Actual/Actual (ISDA)" (see `ActualActual`.) If Actual/365 is not explicitly specified as fixed in an instrument specification, you might want to double-check its meaning.

Class `BlackSwaptionEngine` The engine assumes that the exercise date equals the start date of the passed swap.

Member `BlackVarianceTermStructure::BlackVarianceTermStructure(const DayCounter &dc=Actual365Fixed())` term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

Member `BlackVolatilityTermStructure::BlackVolatilityTermStructure(const DayCounter &dc=Actual365Fixed())` term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

Member `BlackVolTermStructure::BlackVolTermStructure(const DayCounter &dc=Actual365Fixed())` term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

Class `Bond` Most methods assume that the cashflows are stored sorted by date, the redemption being the last one.

Member `Bond::cashflows() const` the returned vector includes the redemption as the last cash flow.

Member `Bond::cleanPrice() const` the theoretical price calculated from a flat term structure might differ slightly from the price calculated from the corresponding yield by means of the other overload of this function. If the price from a constant yield is desired, it is advisable to use such other overload.

Member `Bond::dirtyPrice()` const the theoretical price calculated from a flat term structure might differ slightly from the price calculated from the corresponding yield by means of the other overload of this function. If the price from a constant yield is desired, it is advisable to use such other overload.

Class `CADLibor` This is the rate fixed in London by BBA. Use CDOR if you're interested in the Canadian fixing by IDA.

Member `Calendar::name()` const This method is used for output and comparison between calendars. It is **not** meant to be used for writing switch-on-type code.

Member `CapletVolatilityStructure::CapletVolatilityStructure(const DayCounter &dc=Actual365Fixed())` term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

Member `CapVolatilityStructure::CapVolatilityStructure(const DayCounter &dc=Actual365Fixed())` term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

Class `Cdor` This is the rate fixed in Canada by IDA. Use CADLibor if you're interested in the London fixing by BBA.

Class `CHFLibor` This is the rate fixed in London by BBA. Use ZIBOR if you're interested in the Zurich fixing.

Class `CmsCoupon` This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Class `CompositeInstrument` Methods that drive the calculation directly (such as `recalculate()`, `freeze()` and others) might not work correctly.

Class `ConvertibleFixedCouponBond` Most methods inherited from `Bond` (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

Class `ConvertibleFloatingRateBond` Most methods inherited from `Bond` (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

Class `ConvertibleZeroCouponBond` Most methods inherited from `Bond` (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

Member `Coupon::Coupon`(Real nominal, const Date &paymentDate, const Date &accrualStartDate, const Date &...
the coupon does not adjust the payment date which must already be a business day.

Class `CrankNicolson` The differential operator must be linear for this evolver to work.

Member `DayCounter::name()` const This method is used for output and comparison between day counters. It is **not** meant to be used for writing switch-on-type code.

Class `DiscretizedOption` it is advised that derived classes take care of creating and initializing themselves an instance of the underlying.

Class `Disposable` In order to avoid copies in code such as shown above, the conversion from T to Disposable<T> is destructive, i.e., it does **not** preserve the state of the original object. Therefore, it is necessary for the developer to avoid code such as

Class `Euribor` This is the rate fixed by the ECB. Use EurLibor if you're interested in the London fixing by BBA.

Class `EURLibor` This is the rate fixed in London by BBA. Use Euribor if you're interested in the fixing by the ECB.

Class `EURLibor` This is not a valid base class for the O/N index

Member `ExchangeRateManager::lookup`(const Currency &source, const Currency &target, Date date=Date(), Ex...
if two or more exchange-rate chains are possible which allow to specify a requested rate, it is unspecified which one is returned.

Member `FiniteDifferenceModel::rollback`(array_type &a, Time from, Time to, Size steps)
being this a rollback, from must be a later time than to.

Member `FiniteDifferenceModel::rollback`(array_type &a, Time from, Time to, Size steps, const condition_type &...
being this a rollback, from must be a later time than to.

Class `FixedCouponBondHelper` This class assumes that the reference date does not change between calls of setTermStructure().

Class `FixedRateBondForward` This class still needs to be rigorously tested

Class `Forward` This class still needs to be rigorously tested

Class `ForwardRateAgreement` This class still needs to be rigorously tested

Member `ForwardRateStructure::zeroYieldImpl(Time) const` This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own `zeroYield` method.

Class `G2SwaptionEngine` The engine assumes that the exercise date equals the start date of the passed swap.

Class `GapPayoff` this payoff can be negative depending on the strikes

Member `Handle::Handle(const boost::shared_ptr< T > &h=boostshared_ptr< T >(), bool registerAsObserver=true)` `registerAsObserver` is left as a backdoor in case the programmer cannot guarantee that the object pointed to will remain alive for the whole lifetime of the handle—namely, it should be set to `false` when the passed shared pointer does not own the pointee (this should only happen in a controlled environment, so that the programmer is aware of it). Failure to do so can very likely result in a program crash. If the programmer does want the handle to register as observer of such a shared pointer, it is his responsibility to ensure that the handle gets destroyed before the pointed object does.

Member `IMM::code(const Date &immDate)` It raises an exception if the input date is not an IMM date

Member `IMM::date(const std::string &immCode, const Date &referenceDate=Date())` It raises an exception if the input string is not an IMM code

Class `ImpliedVolTermStructure` It doesn't make financial sense to have an asset-dependant implied Vol Term Structure. This class should be used with term structures that are time dependant only.

Class `IncrementalStatistics` high moments are numerically unstable for high average/standardDeviation ratios.

Member `Index::name() const=0` This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Member `Instrument::setPricingEngine(const boost::shared_ptr< PricingEngine > &)` calling this method will have no effects in case the `performCalculation` method was overridden in a derived class.

Member `InterestRate::discountFactor(Time t) const` Time must be measured using `InterestRate`'s own day counter.

Member `InterestRate::compoundFactor(Time t) const` Time must be measured using `InterestRate`'s own day counter.

Member `InterestRate::equivalentRate(Time t, Compounding comp, Frequency freq=Annual) const` Time must be measured using the `InterestRate`'s own day counter.

Member `InterestRate::impliedRate(Real compound, Time t, const DayCounter &resultDC, Compounding comp)` Time must be measured using the day-counter provided as input.

Class `JamshidianSwaptionEngine` The engine assumes that the exercise date equals the start date of the passed swap.

Class `JPYLibor` This is the rate fixed in London by BBA. Use TIBOR if you're interested in the Tokio fixing.

Class `JuQuadraticApproximationEngine` Barone-Adesi-Whaley critical commodity price calculation is used, it has not been modified to see whether the method of Ju is faster. Ju does not say how he solves the equation for the critical stock price, e.g. Newton method. He just gives the solution. The method of BAW gives answers to the same accuracy as in Ju (1999).

Member `Lattice::partialRollback(DiscretizedAsset &, Time to) const=0` In version 0.3.7 and earlier, this method was called `rollAlmostBack` method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

```
method->partialRollback(asset,t);
asset->preAdjustValues();
```

Member `LazyObject::calculate() const` Objects cache the results of the previous calculation. Such results will be returned upon later invocations of `calculate`. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.

Member `LazyObject::calculate() const` Should this method be redefined in derived classes, `LazyObject::calculate()` should be called in the overriding method.

Class `Libor` This is not a valid base class for the O/N, S/N index

Class `LiborForwardModelProcess` this class does not work correctly with Visual C++ 6.

Member `LocalVolTermStructure::LocalVolTermStructure(const DayCounter &dc=Actual365Fixed())` term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

Member `pseudoSqrt` Higham algorithm only works for correlation matrices.

Class `MCAmericanBasketEngine` This method is intrinsically weak for out-of-the-money options.

Class `MCDiscreteAveragingAsianEngine` control-variate calculation is disabled under VC++6.

Class `MixedScheme` The differential operator must be linear for this evolver to work.

Class `NeumannBC` The value passed must not be the value of the derivative. Instead, it must be comprehensive of the grid step between the first two points—i.e., it must be the difference between $f[0]$ and $f[1]$.

Member `Observable::operator=(const Observable &)` notification is sent before the copy constructor has a chance of actually change the data members. Therefore, observers whose `update()` method tries to use their observables will not see the updated values. It is suggested that the `update()` method just raise a flag in order to trigger a later recalculation.

Member `OneAssetOption::impliedVolatility(Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility target=0.01)` currently, this method returns the Black-Scholes implied volatility. It will give inconsistent results if the pricing was performed with any other methods (such as jump-diffusion models.)

Member `OneAssetOption::impliedVolatility(Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility target=0.01)` 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.

Member `Payoff::name() const=0` This method is used for output and comparison between pay-offs. It is **not** meant to be used for writing switch-on-type code.

Class `PiecewiseYieldCurve` The bootstrapping algorithm will raise an exception if any two instruments have the same maturity date.

Member `Problem::reset()` it does not reset the current minimum to any initial value

Class **QuantoEngine** for the time being, this engine will only work with simple Black-Scholes processes (i.e., no Merton.)

Class **RandomizedLDS** Inverting LDS and PRS is possible, but it doesn't make sense.

Class **RandomSequenceGenerator** do not use with low-discrepancy sequence generator.

Member **RateHelper::setTermStructure(YieldTermStructure *)** Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Member **RelinkableHandle::RelinkableHandle(const boost::shared_ptr< T > &h=boostshared_ptr< T >(), bool** see the Handle documentation for issues relatives to registerAsObserver.

Member **RelinkableHandle::linkTo(const boost::shared_ptr< T > &, bool registerAsObserver=true)** see the Handle documentation for issues relatives to registerAsObserver.

Member **Rounding::Type** the names of the Floor and Ceiling methods might be misleading. Check the provided reference.

Member **Settings::evaluationDate()** a notification is not sent when the evaluation date changes for natural causes—i.e., a date was not explicitly set (which results in today's date being used for pricing) and the current date changes as the clock strikes midnight.

Class **SimpleDayCounter** this day counter should be used together with NullCalendar, which ensures that dates at whole-month distances share the same day of month. It is **not** guaranteed to work with any other calendar.

Member **SingleAssetOption::impliedVolatility(Real targetValue, Real accuracy=1e-4, Size maxEvaluations=100,** Options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g binary options. In these cases impliedVolatility can fail and in any case is meaningless. Another possible source of failure is to have a targetValue that is not attainable with any volatility, e.g. a targetValue lower than the intrinsic value in the case of American options.

Member **SwapIndex::underlyingSwap(const Date &fixingDate) const** Relinking the term structure underlying the index will not have effect on the returned swap.

Class **SwaptionVolatilityCube** this class is not finalized and its interface might change in subsequent releases.

Member `SwaptionVolatilityStructure::SwaptionVolatilityStructure(const DayCounter &dc=Actual365Fixed(), B`
term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

Member `TermStructure::TermStructure(const DayCounter &dc=Actual365Fixed())` term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

Class `Tibor` This is the rate fixed in Tokio by JBA. Use JPYLibor if you're interested in the London fixing by BBA.

Class `TreeSwaptionEngine` This engine is not guaranteed to work if the underlying swap has a start date in the past, i.e., before today's date. When using this engine, prune the initial part of the swap so that it starts at $t \geq 0$.

Class `TridiagonalOperator` to use real time-dependant algebra, you must overload the corresponding operators in the inheriting time-dependent class.

Class `TrinomialTree` The diffusion term of the SDE must be independent of the underlying process.

Class `VarianceSwap` This class does not manage seasoned variance swaps.

Member `YieldTermStructure::YieldTermStructure(const DayCounter &dc=Actual365Fixed())`
term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

Member `YieldTermStructure::forwardRate(const Date &d, const Period &p, const DayCounter &resultDayCoun`
dates are not adjusted for holidays

Class `Zibor` This is the rate fixed in Zurich by BBA. Use CHFLibor if you're interested in the London fixing by BBA.

Chapter 13

Test Suite

Class **ActualActual** the correctness of the results is checked against known good values.

Class **AnalyticBarrierEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **AnalyticCliquetEngine** • the correctness of the returned value is tested by reproducing results available in literature.
• the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **AnalyticContinuousFixedLookbackEngine** returned values are verified against results from literature

Class **AnalyticContinuousFloatingLookbackEngine** returned values verified against results from literature

Class **AnalyticContinuousGeometricAveragePriceAsianEngine** • the correctness of the returned value is tested by reproducing results available in literature, and results obtained using a discrete average approximation.
• the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **AnalyticDigitalAmericanEngine** • the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
• the correctness of the returned value in case of asset-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
• the correctness of the returned value in case of cash-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.
• the correctness of the returned value in case of asset-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.

- the correctness of the returned greeks in case of cash-or-nothing at-hit digital payoff is tested by reproducing numerical derivatives.

Class **AnalyticDiscreteGeometricAveragePriceAsianEngine** • the correctness of the returned value is tested by reproducing results available in literature.

- the correctness of the available greeks is tested against numerical calculations.

Class **AnalyticDividendEuropeanEngine** the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **AnalyticEuropeanEngine** • the correctness of the returned value is tested by reproducing results available in literature.

- the correctness of the returned greeks is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the implied-volatility calculation is tested by checking that it does not modify the option.
- the correctness of the returned value in case of cash-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of gap digital payoff is tested by reproducing results available in literature.
- the correctness of the returned greeks in case of cash-or-nothing digital payoff is tested by reproducing numerical derivatives.

Class **AnalyticHestonEngine** the correctness of the returned value is tested by reproducing results available in web/literature and comparison with Black pricing.

Class **AnalyticPerformanceEngine** the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **Array** construction of arrays is checked in a number of cases

Class **BaroneAdesiWhaleyApproximationEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **BatesEngine** the correctness of the returned value is tested by reproducing results available in web/literature, testing against QuantLib's jump diffusion engine and comparison with Black pricing.

Class **BatesModel** calibration is tested against known values.

Class **BinomialVanillaEngine** the correctness of the returned values is tested by checking it against analytic results.

Class **Bisection** the correctness of the returned values is tested by checking them against known good results.

Class **BivariateCumulativeNormalDistributionDr78** the correctness of the returned value is tested by checking it against known good results.

Class **BivariateCumulativeNormalDistributionWe04DP** the correctness of the returned value is tested by checking it against known good results.

Class **BjerksundStenslandApproximationEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **Bond**

- price/yield calculations are cross-checked for consistency.
- price/yield calculations are checked against known good values.

Class **Brazil** the correctness of the returned results is tested against a list of known holidays.

Class **Brent** the correctness of the returned values is tested by checking them against known good results.

Class **Calendar** the methods for adding and removing holidays are tested by inspecting the calendar before and after their invocation.

Class **CapFloor**

- the correctness of the returned value is tested by checking that the price of a cap (resp. floor) decreases (resp. increases) with the strike rate.
- the relationship between the values of caps, floors and the resulting collars is checked.
- the put-call parity between the values of caps, floors and swaps is checked.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the correctness of the returned value is tested by checking it against a known good value.

Class **CmsRateBond** calculations are tested by checking results against cached values.

Class **CompositeQuote** the correctness of the returned values is tested by checking them against numerical calculations.

Class **CompoundForward**

- the correctness of the curve is tested by reproducing the input data.
- the correctness of the curve is tested by checking the consistency between returned rates and swaps priced on the curve.

Class **ConvergenceStatistics** results are tested against known good values.

Class **CovarianceDecomposition** cross checked with getCovariance

Class **CubicSpline** the correctness of the returned values is tested by reproducing results available in literature.

Class **CumulativePoissonDistribution** the correctness of the returned value is tested by checking it against known good results.

Class **Date** self-consistency of dates, serial numbers, days of month, months, and weekdays is checked over the whole date range.

Class **DerivedQuote** the correctness of the returned values is tested by checking them against numerical calculations.

Class **DPlusDMinus** the correctness of the returned values is tested by checking them against numerical calculations.

Class **DZero** the correctness of the returned values is tested by checking them against numerical calculations.

Class **ExchangeRate** application of direct and derived exchange rate is tested against calculations.

Class **ExchangeRateManager** lookup of direct, triangulated, and derived exchange rates is tested.

Class **Factorial** the correctness of the returned value is tested by checking it against numerical calculations.

Class **FalsePosition** the correctness of the returned values is tested by checking them against known good results.

Class **FaureRsg** the correctness of the returned values is tested by reproducing known good values.

Class **FDEuropeanEngine** the correctness of the returned value is tested by checking it against analytic results.

Class **FixedRateBond** calculations are tested by checking results against cached values.

Class **FloatingRateBond** calculations are tested by checking results against cached values.

Class **ForwardEngine** • the correctness of the returned value is tested by reproducing results available in literature.
• the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **ForwardPerformanceEngine** • the correctness of the returned value is tested by reproducing results available in literature.
• the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **ForwardSpreadedTermStructure** • the correctness of the returned values is tested by checking them against numerical calculations.
• observability against changes in the underlying term structure and in the added spread is checked.

Class **GammaFunction** the correctness of the returned value is tested by checking it against known good results.

Class **GaussianQuadrature** the correctness of the result is tested by checking it against known good values.

Class **GaussKronrodAdaptive** the correctness of the result is tested by checking it against known good values.

Class **GenericSequenceStatistics** the correctness of the returned values is tested by checking them against numerical calculations.

Class **Germany** the correctness of the returned results is tested against a list of known holidays.

Class **HaltonRsg** • the correctness of the returned values is tested by reproducing known good values.

- the correctness of the returned values is tested by checking their discrepancy against known good values.

Class **HestonModel** calibration is tested against known good values.

Class **HullWhite** calibration results are tested against cached values

- Class **ImpliedTermStructure**
- the correctness of the returned values is tested by checking them against numerical calculations.
 - observability against changes in the underlying term structure is checked.

Class **Instrument** observability of class instances is checked.

Class **InterestRate** Converted rates are checked against known good results

Class **InverseCumulativePoisson** the correctness of the returned value is tested by checking it against known good results.

Class **Italy** the correctness of the returned results is tested against a list of known holidays.

Class **JointCalendar** the correctness of the returned results is tested by reproducing the calculations.

- Class **JumpDiffusionEngine**
- the correctness of the returned value is tested by reproducing results available in literature.
 - the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **JuQuadraticApproximationEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **LfmHullWhiteParameterization** the correctness is tested by Monte-Carlo reproduction of caplet & ratchet npvs and comparison with Black pricing.

Class **LiborForwardModel** the correctness is tested using Monte-Carlo Simulation to reproduce swaption npvs, model calibration and exact cap pricing

Class **LiborForwardModelProcess** the correctness is tested by Monte-Carlo reproduction of caplet & ratchet NPVs and comparison with Black pricing.

Class **LinearLeastSquaresRegression** the correctness of the returned values is tested by checking their properties.

Class **LongstaffSchwartzPathPricer** the correctness of the returned value is tested by reproducing results available in web/literature

Member **pseudoSqrt**

- the correctness of the results is tested by reproducing known good data.
- the correctness of the results is tested by checking returned values against numerical calculations.

Class **MCAmericanEngine** the correctness of the returned value is tested by reproducing results available in web/literature

Class **MCBarrierEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **MCBasketEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **MCDigitalEngine** the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing known good results.

Class **MCDiscreteArithmeticAPEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **MCDiscreteGeometricAPEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **MCEuropeanEngine** the correctness of the returned value is tested by checking it against analytic results.

Class **MCEuropeanHestonEngine** the correctness of the returned value is tested by reproducing results available in web/literature

Class **MCLongstaffSchwartzEngine** the correctness of the returned value is tested by reproducing results available in web/literature

Class **MCVarianceSwapEngine** returned fair variances checked for consistency with implied volatility curve.

Class **MersenneTwisterUniformRng** the correctness of the returned values is tested by checking them against known good results.

Class **Money** money arithmetic is tested with and without currency conversions.

Class **MultiCubicSpline** interpolated values are checked against the original function.

Class **MultiPathGenerator** the generated paths are checked against cached results

Class **Newton** the correctness of the returned values is tested by checking them against known good results.

Class **NewtonSafe** the correctness of the returned values is tested by checking them against known good results.

Class **NormalDistribution** the correctness of the returned value is tested by checking it against numerical calculations. Cross-checks are also performed against the CumulativeNormalDistribution and InverseCumulativeNormal classes.

Class **OperatorFactory** coefficients are tested against constant BSM operator

Class **PathGenerator** the generated paths are checked against cached results

Class **PiecewiseYieldCurve**

- the correctness of the returned values is tested by checking them against the original inputs.
- the observability of the term structure is tested.

Class **PoissonDistribution** the correctness of the returned value is tested by checking it against known good results.

Class **QuantoEngine**

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **Quote** the observability of class instances is tested.

Class **RandomizedLDS** correct initialization is tested.

Class **ReplicatingVarianceSwapEngine** returned fair variances verified against results from literature

Class **Ridder** the correctness of the returned values is tested by checking them against known good results.

Class **Rounding** the correctness of the returned values is tested by checking them against known good results.

Class **Secant** the correctness of the returned values is tested by checking them against known good results.

Class **SeedGenerator** correct initialization of the single instance is tested.

Class **SegmentIntegral** the correctness of the result is tested by checking it against known good values.

Class **SimpleDayCounter** the correctness of the results is checked against known good values.

Class **SimpsonIntegral** the correctness of the result is tested by checking it against known good values.

Class **SobolRsg**

- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

Class **StulzEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **SVD** the correctness of the returned values is tested by checking their properties.

Class **Swaption**

- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption decreases (resp. increases) with the strike.
- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption increases (resp. decreases) with the spread.
- the correctness of the returned value is tested by checking it against that of a swaption on a swap with no spread and a correspondingly adjusted fixed rate.
- the correctness of the returned value is tested by checking it against a known good value.

- the correctness of the returned value of cash settled swaptions is tested by checking the modified annuity against a value calculated without using the Swaption class.

Class **SymmetricSchurDecomposition** the correctness of the returned values is tested by checking their properties.

Class **TARGET** the correctness of the returned results is tested against a list of known holidays.

Class **TqrEigenDecomposition** the correctness of the result is tested by checking it against known good values.

Class **TrapezoidIntegral** the correctness of the result is tested by checking it against known good values.

Class **TreeSwaptionEngine** calculations are checked against cached results

Class **TreeVanillaSwapEngine** calculations are checked against known good results

Class **UnitedKingdom** the correctness of the returned results is tested against a list of known holidays.

Class **UnitedStates** the correctness of the returned results is tested against a list of known holidays.

Class **YieldTermStructure** observability against evaluation date changes is checked.

Class **ZeroCouponBond** calculations are tested by checking results against cached values.

Class **ZeroSpreadedTermStructure**

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

Member **FDAmericanEngine**

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Member **FDDividendAmericanEngine**

- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the invariance of the results upon addition of null dividends is tested.

Member [FDDividendEuropeanEngine](#) • the correctness of the returned greeks is tested by reproducing numerical derivatives.

- the invariance of the results upon addition of null dividends is tested.

Member [FDShoutEngine](#) the correctness of the returned greeks is tested by reproducing numerical derivatives.

Member [BSMTermOperator](#) coefficients are tested against constant BSM operator

Chapter 14

Todo List

Class [AmericanCondition](#) unify the intrinsicValues/Payoff thing

Class [AmericanExercise](#) check that everywhere the American condition is applied from earliestDate and not earlier

Class [AmericanPayoffAtExpiry](#) calculate greeks

Class [AmericanPayoffAtHit](#) calculate greeks

Class [AnalyticBarrierEngine](#) rework to avoid repeated casts inside utility methods

Class [AnalyticContinuousGeometricAveragePriceAsianEngine](#) handle seasoned options

Class [AnalyticDigitalAmericanEngine](#) add more greeks (as of now only delta and rho available)

Class [AnalyticDiscreteGeometricAveragePriceAsianEngine](#) implement correct theta, rho, and dividend-rho calculation

Class [BasketOption](#) Replace with STL algorithms

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 **BinomialVanillaEngine** Greeks are not overly accurate. They could be improved by building a tree so that it has three points at the current time. The value would be fetched from the middle one, while the two side points would be used for estimating partial derivatives.

Class **BivariateCumulativeNormalDistributionDr78** check accuracy of this algorithm and compare with: 1) Drezner, Z, (1978), Computation of the bivariate normal integral, Mathematics of Computation 32, pp. 277-279. 2) Drezner, Z. and Wesolowsky, G. O. (1990) 'On the Computation of the Bivariate Normal Integral', Journal of Statistical Computation and Simulation 35, pp. 101-107. 3) Drezner, Z (1992) Computation of the Multivariate Normal Integral, ACM Transactions on Mathematics Software 18, pp. 450-460. 4) Drezner, Z (1994) Computation of the Trivariate Normal Integral, Mathematics of Computation 62, pp. 289-294. 5) Genz, A. (1992) 'Numerical Computation of the Multivariate Normal Probabilities', J. Comput. Graph. Stat. 1, pp. 141-150.

Class **BlackVarianceCurve** check time extrapolation

Class **BlackVarianceSurface** check time extrapolation

Member **BoundaryCondition::Side** Generalize for n-dimensional conditions

Class **CapVolatilityVector** either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the length vector but an interpolation pointing to the original ones.

Class **CashFlows** add tests

Class **Cdor** check settlement days, end-of-month adjustment, and day-count convention.

Class **CliquetOption** • add local/global caps/floors
 • add accrued coupon and last fixing

Class **ContinuousAveragingAsianOption** add running average

Class **DirichletBC** generalize to time-dependent conditions.

Class **DiscreteGeometricASO** add analytical greeks

Member **Event::hasOccurred(const Date &d, bool includeToday=false) const** make QL_-
 TODAYSPAYMENT dynamically configurable?

Class **ExplicitEuler** add Richardson extrapolation

Class **FixedRateBondForward** Add preconditions and tests

Class **FixedRateBondForward** Create switch- if coupon goes to seller is toggled on, don't consider income in the $P_{DirtyFwd}(t)$ calculation.

Class **FixedRateBondForward** Verify this works when the underlying is paper (in which case ignore all AI.)

Class **Forward** Add preconditions and tests

Class **ForwardRateAgreement** Add preconditions and tests

Class **ForwardRateAgreement** Should put an instance of ForwardRateAgreement in the FraRateHelper to ensure consistency with the piecewise yield curve.

Class **ForwardRateAgreement** Differentiate between BBA (British)/AFB (French) [assumed here] and ABA (Australian) banker conventions in the calculations.

Class **FuturesRateHelper** implement/refactor constructors with: Index instead of (nMonths, calendar, convention, dayCounter), IMM code

Member **GeneralizedBlackScholesProcess::drift**(Time t, Real x) const revise extrapolation

Member **GeneralizedBlackScholesProcess::diffusion**(Time t, Real x) const revise extrapolation

Class **GenericRiskStatistics** add historical annualized volatility

Class **IborIndex** add methods returning InterestRate

Class **IntegralEngine** define tolerance for calculate()

Class **InterestRateIndex** add methods returning InterestRate

Class **Jibar** check settlement days and day-count convention.

Class **LogLinearInterpolation** implement primitive, derivative, and secondDerivative functions.

Class **MCVarianceSwapEngine** define tolerance of numerical integral and incorporate it in errorEstimate

Class **MixedScheme** • derive variable theta schemes
 • introduce multi time-level schemes.

Class **MultiCubicSpline** • allow extrapolation as for the other interpolations
 • investigate if and how to implement Hyman filters and different boundary conditions

Class **NeumannBC** generalize to time-dependent conditions.

Class **Option::arguments** • remove `std::vector<Time> stoppingTimes`
 • how to handle strike-less option (asian average strike, forward, etc.)?

Class **RandomizedLDS** implement the other randomization algorithms

Member **SampledCurve::valueAtCenter() const** replace or complement with a more general function `valueAt(spot)`

Member **SampledCurve::firstDerivativeAtCenter() const** replace or complement with a more general function `firstDerivativeAt(spot)`

Member **SampledCurve::secondDerivativeAtCenter() const** replace or complement with a more general function `secondDerivativeAt(spot)`

Class **Solver1D** • clean up the interface so that it is clear whether the accuracy is specified for x or $f(x)$.
 • add target value (now the target value is 0.0)

Class **Swaption** add greeks and explicit exercise lag

Class **Tibor** check settlement days and end-of-month adjustment.

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 **TRLibor** check end-of-month adjustment.

Class **UnitedKingdom** add LIFFE

Class **YieldTermStructure** add derived class `ParSwapTermStructure` similar to `ZeroYieldTermStructure`, `DiscountStructure`, `ForwardRateStructure`

Class **Zibor** check settlement days, end-of-month adjustment, and day-count convention.

Chapter 15

Known Bugs

Class **AssetSwap** fair prices are not calculated correctly when using indexed coupons.

Class **BlackCalculator** When the variance is null, division by zero occur during the calculation of delta, delta forward, gamma, gamma forward, rho, dividend rho, vega, and strike sensitivity.

Class **CompoundForward** swap rates are not reproduced exactly when using indexed coupons. Apparently, some assumption about the swap fixings is hard-coded into the bootstrapping algorithm.

Class **CoxIngersollRoss** this class was not tested enough to guarantee its functionality.

Class **ExtendedCoxIngersollRoss** this class was not tested enough to guarantee its functionality.

Class **G2** This class was not tested enough to guarantee its functionality.

Class **HullWhite** When the term structure is relinked, the r_0 parameter of the underlying Vasicek model is not updated.

Class **LocalVolSurface** this class is untested, probably unreliable.

Class **MarketModelCapFloorEngine** This engine is not yet working correctly (results are off the expected ones.)

Class **MultiCubicSpline** cannot interpolate at the grid points on the boundary surface of the N-dimensional region

Member **FDDividendAmericanEngine** results are not overly reliable.

Member **FDDividendAmericanEngine** method `impliedVolatility()` utterly fails

Member **FDDividendShoutEngine** results are not overly reliable.

Chapter 16

Deprecated List

Member [SwaptionVolatilityMatrix::SwaptionVolatilityMatrix](#)(const Date &referenceDate, const std::vector< Da
alternative constructors instead

Index

- ~Error
 - QuantLib::Error, [457](#)
- Abs
 - QuantLib::Array, [198](#)
- accruedAmount
 - QuantLib::Bond, [270](#)
- add
 - QuantLib::ExchangeRateManager, [620](#)
 - QuantLib::GeneralStatistics, [733](#)
 - QuantLib::IncrementalStatistics, [788](#)
- addFixing
 - QuantLib::Index, [790](#)
- addFixings
 - QuantLib::Index, [790](#)
- addHoliday
 - QuantLib::Calendar, [287](#)
- addSequence
 - QuantLib::IncrementalStatistics, [788](#)
- adjust
 - QuantLib::Calendar, [287](#)
- adjustValues
 - QuantLib::DiscretizedAsset, [425](#)
- advance
 - QuantLib::Calendar, [287](#), [288](#)
- amount
 - QuantLib::CashFlow, [315](#)
 - QuantLib::Dividend, [432](#)
 - QuantLib::FixedDividend, [645](#)
 - QuantLib::FixedRateCoupon, [652](#)
 - QuantLib::FloatingRateCoupon, [658](#)
 - QuantLib::FractionalDividend, [686](#)
 - QuantLib::SimpleCashFlow, [1092](#)
- Annual
 - datetime, [110](#)
- apply
 - QuantLib::GeneralizedBlackScholesProcess, [730](#)
 - QuantLib::HestonProcess, [757](#)
 - QuantLib::LiborForwardModelProcess, [863](#)
 - QuantLib::Merton76Process, [953](#)
 - QuantLib::StochasticProcess, [1131](#)
 - QuantLib::StochasticProcess1D, [1133](#)
 - QuantLib::StochasticProcessArray, [1137](#)
- applyAfterApplying
 - QuantLib::BoundaryCondition, [271](#)
 - QuantLib::DirichletBC, [415](#)
 - QuantLib::NeumannBC, [980](#)
- applyAfterSolving
 - QuantLib::BoundaryCondition, [272](#)
 - QuantLib::DirichletBC, [415](#)
 - QuantLib::NeumannBC, [981](#)
- applyBeforeApplying
 - QuantLib::BoundaryCondition, [271](#)
 - QuantLib::DirichletBC, [415](#)
 - QuantLib::NeumannBC, [980](#)
- applyBeforeSolving
 - QuantLib::BoundaryCondition, [271](#)
 - QuantLib::DirichletBC, [415](#)
 - QuantLib::NeumannBC, [980](#)
- Asian option engines, [116](#)
- atmRate
 - QuantLib::CashFlows, [318](#)
- AutomatedConversion
 - QuantLib::Money, [958](#)
- averageShortfall
 - QuantLib::GenericRiskStatistics, [742](#)
- BackwardFlatInterpolation
 - QuantLib::BackwardFlatInterpolation, [211](#)
- Barrier option engines, [117](#)
- BaseCurrencyConversion
 - QuantLib::Money, [958](#)
- Basket option engines, [118](#)
- BEJ
 - QuantLib::Indonesia, [795](#)
- BicubicSpline
 - QuantLib::BicubicSpline, [229](#)
- BigInteger
 - types, [102](#)
- BilinearInterpolation
 - QuantLib::BilinearInterpolation, [231](#)
- Bimonthly
 - datetime, [110](#)
- Biweekly
 - datetime, [110](#)
- blackVarianceImpl
 - QuantLib::BlackVolatilityTermStructure, [263](#)

- BlackVarianceTermStructure
 - QuantLib::BlackVarianceTermStructure, 261
- BlackVolatilityTermStructure
 - QuantLib::BlackVolatilityTermStructure, 263
- blackVolImpl
 - QuantLib::BlackVarianceTermStructure, 261
- BlackVolTermStructure
 - QuantLib::BlackVolTermStructure, 266
- BMV
 - QuantLib::Mexico, 954
- BoundaryCondition
 - QuantLib::CubicSpline, 389
- bps
 - QuantLib::CashFlows, 318
- BrownianBridge
 - QuantLib::BrownianBridge, 279
- BSMTermOperator
 - findiff, 129
- BSSE
 - QuantLib::Slovakia, 1108
- BusinessDayConvention
 - datetime, 110
- businessDaysBetween
 - QuantLib::Calendar, 288
- calculate
 - QuantLib::Instrument, 798
 - QuantLib::LazyObject, 847
- Calendar
 - QuantLib::Calendar, 286
- Calendars, 111
- calibrate
 - QuantLib::CalibratedModel, 293
- Cap/floor engines, 119
- CapletVolatilityStructure
 - QuantLib::CapletVolatilityStructure, 308
- CapVolatilityStructure
 - QuantLib::CapVolatilityStructure, 312
- cashflows
 - QuantLib::Bond, 269
- Ceiling
 - QuantLib::Rounding, 1072
- checkMaxIterations
 - QuantLib::EndCriteria, 454
- checkStationaryFunctionAccuracy
 - QuantLib::EndCriteria, 454
- checkStationaryFunctionValue
 - QuantLib::EndCriteria, 454
- checkStationaryPoint
 - QuantLib::EndCriteria, 454
- checkZeroGradientNorm
 - QuantLib::EndCriteria, 454
- CholeskyDecomposition
 - QuantLib::Matrix, 918
- cleanPrice
 - QuantLib::Bond, 269
- Cliquet option engines, 120
- close
 - QuantLib::Money, 959
- close_enough
 - QuantLib::Money, 959
- Closest
 - QuantLib::Rounding, 1072
- code
 - QuantLib::IMM, 778
- compoundFactor
 - QuantLib::InterestRate, 802
- compoundForwardImpl
 - QuantLib::ExtendedDiscountCurve, 630
- computePlain
 - QuantLib::LMMDriftCalculator, 876
 - QuantLib::LMMNormalDriftCalculator, 877
- computeReduced
 - QuantLib::LMMDriftCalculator, 876
 - QuantLib::LMMNormalDriftCalculator, 877
- ConversionType
 - QuantLib::Money, 958
- convexity
 - QuantLib::CashFlows, 319
- convexityBias
 - QuantLib::HullWhite, 763
- correlationMatrix
 - QuantLib::CovarianceDecomposition, 380
- Coupon
 - QuantLib::Coupon, 379
- covariance
 - QuantLib::Abcd, 165
 - QuantLib::AbcdFunction, 167
 - QuantLib::EulerDiscretization, 461
 - QuantLib::G2ForwardProcess, 697
 - QuantLib::G2Process, 699
 - QuantLib::LiborForwardModelProcess, 863
 - QuantLib::StochasticProcess, 1130
 - QuantLib::StochasticProcessArray, 1137
- CovarianceDecomposition
 - QuantLib::CovarianceDecomposition, 380
- CubicSpline
 - QuantLib::CubicSpline, 389
- Currencies and FX rates, 104
- Currency
 - QuantLib::Currency, 395

- Daily
 - datetime, 110
- date
 - QuantLib::IMM, 778
- Date and time calculations, 108
- datetime
 - Annual, 110
 - Bimonthly, 110
 - Biweekly, 110
 - BusinessDayConvention, 110
 - Daily, 110
 - Day, 109
 - EveryFourthMonth, 110
 - Following, 110
 - Frequency, 110
 - ModifiedFollowing, 110
 - ModifiedPreceding, 110
 - Month, 110
 - Monthly, 110
 - NoFrequency, 110
 - Once, 110
 - Preceding, 110
 - Quarterly, 110
 - Semiannual, 110
 - TimeUnit, 110
 - Unadjusted, 110
 - Weekday, 110
 - Weekly, 110
 - Year, 109
- Day
 - datetime, 109
- Day counters, 114
- DayCounter
 - QuantLib::DayCounter, 407
- Debugging macros, 157
- debugMacros
 - QL_TRACE, 158
 - QL_TRACE_DISABLE, 157
 - QL_TRACE_ENABLE, 157
 - QL_TRACE_ENTER_FUNCTION, 158
 - QL_TRACE_EXIT_FUNCTION, 158
 - QL_TRACE_LOCATION, 159
 - QL_TRACE_ON, 158
 - QL_TRACE_VARIABLE, 159
- Decimal
 - types, 102
- DEFINE_SEQUENCE_STAT_CONST_-METHOD_DOUBLE
 - sequencestatistics.hpp, 1489
- DEFINE_SEQUENCE_STAT_CONST_-METHOD_VOID
 - sequencestatistics.hpp, 1489
- delta
 - QuantLib::BlackCalculator, 241
 - QuantLib::BlackScholesCalculator, 250
- deltaForward
 - QuantLib::BlackCalculator, 241
- Derived
 - QuantLib::ExchangeRate, 619
- Design patterns, 143
- Diagonal
 - QuantLib::SobolBrownianGenerator, 1111
- diffusion
 - QuantLib::EulerDiscretization, 460
 - QuantLib::GeneralizedBlackScholesProcess, 730
- Direct
 - QuantLib::ExchangeRate, 619
- dirtyPrice
 - QuantLib::Bond, 269
- discount
 - QuantLib::YieldTermStructure, 1243
- DiscountCurve
 - yieldtermstructures, 147
- DiscountFactor
 - types, 103
- discountFactor
 - QuantLib::InterestRate, 802
- discountImpl
 - QuantLib::CompoundForward, 348
 - QuantLib::ForwardRateStructure, 678
 - QuantLib::ZeroYieldStructure, 1252
- dividendRho
 - QuantLib::BlackCalculator, 242
- DotProduct
 - QuantLib::Array, 198
- Down
 - QuantLib::Rounding, 1072
- downsideDeviation
 - QuantLib::GenericRiskStatistics, 741
 - QuantLib::IncrementalStatistics, 788
- downsideVariance
 - QuantLib::GenericRiskStatistics, 741
 - QuantLib::IncrementalStatistics, 788
- drift
 - QuantLib::EulerDiscretization, 460
 - QuantLib::GeneralizedBlackScholesProcess, 730
- duration
 - QuantLib::CashFlows, 319
- earliestDate
 - QuantLib::RateHelper, 1064
- elasticity
 - QuantLib::BlackCalculator, 241
 - QuantLib::BlackScholesCalculator, 250
- elasticityForward
 - QuantLib::BlackCalculator, 241

- equivalentRate
 - QuantLib::InterestRate, 803
- Error
 - QuantLib::Error, 457
- errorEstimate
 - QuantLib::GeneralStatistics, 732
 - QuantLib::IncrementalStatistics, 787
- errors.hpp
 - QL_ASSERT, 1287
 - QL_ENSURE, 1287
 - QL_FAIL, 1287
 - QL_REQUIRE, 1287
- Eurex
 - QuantLib::Germany, 748
- evaluationDate
 - QuantLib::Settings, 1086
- EveryFourthMonth
 - datetime, 110
- evolve
 - QuantLib::HestonProcess, 757
 - QuantLib::LiborForwardModelProcess, 863
 - QuantLib::StochasticProcess, 1130
 - QuantLib::StochasticProcess1D, 1133
 - QuantLib::StochasticProcessArray, 1137
- Exchange
 - QuantLib::Italy, 829
 - QuantLib::UnitedKingdom, 1219
- ExchangeRate
 - QuantLib::ExchangeRate, 619
- Exp
 - QuantLib::Array, 198
- expectation
 - QuantLib::G2ForwardProcess, 697
 - QuantLib::G2Process, 699
 - QuantLib::HullWhiteForwardProcess, 767
 - QuantLib::HullWhiteProcess, 769
 - QuantLib::OrnsteinUhlenbeckProcess, 1019
 - QuantLib::StochasticProcess, 1130
 - QuantLib::StochasticProcess1D, 1133
 - QuantLib::StochasticProcessArray, 1137
- expectationValue
 - QuantLib::GeneralStatistics, 733
- expectedShortfall
 - QuantLib::GenericRiskStatistics, 741
- Factors
 - QuantLib::SobolBrownianGenerator, 1111
- FDAmericanEngine
 - vanillaengines, 125
- FDDividendAmericanEngine
 - vanillaengines, 126
- FDDividendEuropeanEngine
 - vanillaengines, 126
- FDDividendShoutEngine
 - vanillaengines, 126
- FDShoutEngine
 - vanillaengines, 126
- fetchResults
 - QuantLib::AssetSwap, 202
 - QuantLib::ForwardVanillaOption, 684
 - QuantLib::Instrument, 798
 - QuantLib::MultiAssetOption, 967
 - QuantLib::OneAssetOption, 1003
 - QuantLib::OneAssetStrikedOption, 1008
 - QuantLib::QuantoVanillaOption, 1055
 - QuantLib::VarianceSwap, 1232
- Financial instruments, 133
- findiff
 - BSMTermOperator, 129
 - OneFactorOperator, 129
- Finite-differences framework, 128
- FirstDerivative
 - QuantLib::CubicSpline, 389
- firstDerivativeAtCenter
 - QuantLib::SampledCurve, 1078
- FixedRateBondForward
 - QuantLib::FixedRateBondForward, 649
- fixing
 - QuantLib::Index, 790
 - QuantLib::InterestRateIndex, 805
- Floor
 - QuantLib::Rounding, 1072
- Following
 - datetime, 110
- format
 - QuantLib::Currency, 395
- Forward option engines, 121
- ForwardCurve
 - yieldtermstructures, 147
- ForwardFlatInterpolation
 - QuantLib::ForwardFlatInterpolation, 668
- forwardRate
 - QuantLib::YieldTermStructure, 1243
- forwardValue
 - QuantLib::Forward, 665
- FrankfurtStockExchange
 - QuantLib::Germany, 748
- freeze
 - QuantLib::LazyObject, 847
- Frequency
 - datetime, 110
- gamma
 - QuantLib::BlackCalculator, 241
 - QuantLib::BlackScholesCalculator, 250
- gammaForward

- QuantLib::BlackCalculator, [241](#)
- gaussianDownsideDeviation
 - QuantLib::GenericGaussianStatistics, [737](#)
- gaussianDownsideVariance
 - QuantLib::GenericGaussianStatistics, [736](#)
- gaussianExpectedShortfall
 - QuantLib::GenericGaussianStatistics, [737](#)
- gaussianPercentile
 - QuantLib::GenericGaussianStatistics, [737](#)
- gaussianPotentialUpside
 - QuantLib::GenericGaussianStatistics, [737](#)
- gaussianRegret
 - QuantLib::GenericGaussianStatistics, [737](#)
- gaussianTopPercentile
 - QuantLib::GenericGaussianStatistics, [737](#)
- gaussianValueAtRisk
 - QuantLib::GenericGaussianStatistics, [737](#)
- Generic macros, [151](#)
- GovernmentBond
 - QuantLib::UnitedStates, [1222](#)
- Handle
 - QuantLib::Handle, [753](#)
- hasOccurred
 - QuantLib::Event, [615](#)
- HKEx
 - QuantLib::HongKong, [760](#)
- ICEX
 - QuantLib::Iceland, [775](#)
- impliedRate
 - QuantLib::InterestRate, [803](#)
- impliedVolatility
 - QuantLib::OneAssetOption, [1003](#)
 - QuantLib::SingleAssetOption, [1101](#)
- impliedYield
 - QuantLib::Forward, [665](#)
- incomeDiscountCurve_
 - QuantLib::Forward, [665](#)
- instantaneousCovariance
 - QuantLib::Abcd, [164](#)
- instantaneousVariance
 - QuantLib::Abcd, [164](#)
- instantaneousVolatility
 - QuantLib::Abcd, [164](#)
- Integer
 - types, [102](#)
- inverse
 - QuantLib::Matrix, [918](#)
- irr
 - QuantLib::CashFlows, [318](#)
- isBusinessDay
 - QuantLib::Calendar, [287](#)
- isEndOfMonth
 - QuantLib::Calendar, [287](#)
- isExpired
 - QuantLib::ForwardRateAgreement, [675](#)
- isHoliday
 - QuantLib::Calendar, [287](#)
- iso_date
 - manips, [156](#)
- isOnTime
 - QuantLib::DiscretizedAsset, [425](#)
- isWeekend
 - QuantLib::Calendar, [287](#)
- Iterator support, [154](#)
- iteratorMacros
 - QL_FULL_ITERATOR_SUPPORT, [154](#)
- itmAssetProbability
 - QuantLib::BlackCalculator, [242](#)
- itmCashProbability
 - QuantLib::BlackCalculator, [242](#)
- JSX
 - QuantLib::Indonesia, [795](#)
- KnuthUniformRng
 - QuantLib::KnuthUniformRng, [840](#)
- KRX
 - QuantLib::SouthKorea, [1119](#)
- kurtosis
 - QuantLib::GeneralStatistics, [732](#)
 - QuantLib::IncrementalStatistics, [787](#)
- Lagrange
 - QuantLib::CubicSpline, [389](#)
- latestDate
 - QuantLib::RateHelper, [1064](#)
- Lattice methods, [137](#)
- LecuyerUniformRng
 - QuantLib::LecuyerUniformRng, [850](#)
- limitMacros
 - QL_EPSILON, [152](#)
 - QL_MAX_INTEGER, [152](#)
 - QL_MAX_REAL, [152](#)
 - QL_MIN_INTEGER, [152](#)
 - QL_MIN_POSITIVE_REAL, [152](#)
 - QL_MIN_REAL, [152](#)
- LinearInterpolation
 - QuantLib::LinearInterpolation, [865](#)
- linkTo
 - QuantLib::RelinkableHandle, [1066](#)
- localVolImpl
 - QuantLib::LocalVolCurve, [881](#)
- LocalVolTermStructure
 - QuantLib::LocalVolTermStructure, [885](#)
- Log
 - QuantLib::Array, [198](#)

- LogLinearInterpolation
 - QuantLib::LogLinearInterpolation, [887](#)
- long_date
 - manips, [156](#)
- long_period
 - manips, [156](#)
- long_weekday
 - manips, [156](#)
- lookup
 - QuantLib::ExchangeRateManager, [620](#)
- make_step_iterator
 - QuantLib::step_iterator, [1123](#)
- mandatoryTimes
 - QuantLib::DiscretizedAsset, [425](#)
 - QuantLib::DiscretizedDiscountBond, [427](#)
 - QuantLib::DiscretizedOption, [428](#)
- manips
 - iso_date, [156](#)
 - long_date, [156](#)
 - long_period, [156](#)
 - long_weekday, [156](#)
 - short_date, [156](#)
 - short_period, [156](#)
 - short_weekday, [156](#)
 - shortest_weekday, [156](#)
- Market
 - QuantLib::Argentina, [193](#)
 - QuantLib::Brazil, [276](#)
 - QuantLib::CzechRepublic, [400](#)
 - QuantLib::Germany, [748](#)
 - QuantLib::HongKong, [760](#)
 - QuantLib::Iceland, [775](#)
 - QuantLib::India, [793](#)
 - QuantLib::Indonesia, [795](#)
 - QuantLib::Italy, [829](#)
 - QuantLib::Mexico, [954](#)
 - QuantLib::SaudiArabia, [1080](#)
 - QuantLib::Singapore, [1099](#)
 - QuantLib::Slovakia, [1108](#)
 - QuantLib::SouthKorea, [1119](#)
 - QuantLib::Taiwan, [1173](#)
 - QuantLib::Ukraine, [1217](#)
 - QuantLib::UnitedKingdom, [1219](#)
 - QuantLib::UnitedStates, [1222](#)
- Math tools, [140](#)
- max
 - QuantLib::GeneralStatistics, [733](#)
 - QuantLib::IncrementalStatistics, [788](#)
- mean
 - QuantLib::GeneralStatistics, [732](#)
 - QuantLib::IncrementalStatistics, [787](#)
- MersenneTwisterUniformRng
 - QuantLib::MersenneTwisterUniformRng, [951](#)
- Merval
 - QuantLib::Argentina, [193](#)
- min
 - QuantLib::GeneralStatistics, [733](#)
 - QuantLib::IncrementalStatistics, [788](#)
- miscMacros
 - QL_DUMMY_RETURN, [151](#)
- ModifiedFollowing
 - datetime, [110](#)
- ModifiedPreceding
 - datetime, [110](#)
- MonotonicCubicSpline
 - QuantLib::MonotonicCubicSpline, [960](#)
- Monte Carlo framework, [142](#)
- Month
 - datetime, [110](#)
- Monthly
 - datetime, [110](#)
- name
 - QuantLib::AssetOrNothingPayoff, [200](#)
 - QuantLib::Calendar, [287](#)
 - QuantLib::CashOrNothingPayoff, [320](#)
 - QuantLib::DayCounter, [407](#)
 - QuantLib::DoubleStickyRatchetPayoff, [442](#)
 - QuantLib::FloatingTypePayoff, [660](#)
 - QuantLib::ForwardTypePayoff, [681](#)
 - QuantLib::GapPayoff, [703](#)
 - QuantLib::Index, [790](#)
 - QuantLib::InterestRateIndex, [805](#)
 - QuantLib::Payoff, [1026](#)
 - QuantLib::PercentageStrikePayoff, [1027](#)
 - QuantLib::PlainVanillaPayoff, [1036](#)
 - QuantLib::RatchetMaxPayoff, [1060](#)
 - QuantLib::RatchetMinPayoff, [1061](#)
 - QuantLib::RatchetPayoff, [1062](#)
 - QuantLib::StickyMaxPayoff, [1126](#)
 - QuantLib::StickyMinPayoff, [1127](#)
 - QuantLib::StickyPayoff, [1128](#)
 - QuantLib::SuperFundPayoff, [1142](#)
 - QuantLib::SuperSharePayoff, [1143](#)
- Natural
 - types, [102](#)
- NaturalCubicSpline
 - QuantLib::NaturalCubicSpline, [978](#)
- NaturalMonotonicCubicSpline
 - QuantLib::NaturalMonotonicCubicSpline, [979](#)
- NERC
 - QuantLib::UnitedStates, [1222](#)
- next

- QuantLib::KnuthUniformRng, 840
- QuantLib::LecuyerUniformRng, 850
- QuantLib::MersenneTwisterUniformRng, 951
- nextCode
 - QuantLib::IMM, 779
- nextDate
 - QuantLib::IMM, 779
- nextRandomizer
 - QuantLib::RandomizedLDS, 1058
- nextWeekday
 - QuantLib::Date, 404
- NoConversion
 - QuantLib::Money, 958
- NoFrequency
 - datetime, 110
- None
 - QuantLib::Rounding, 1072
- NotAKnot
 - QuantLib::CubicSpline, 389
- notifyObservers
 - QuantLib::Observable, 999
- npv
 - QuantLib::CashFlows, 318
- NSE
 - QuantLib::India, 793
- nthWeekday
 - QuantLib::Date, 404
- Numeric limits, 152
- Numeric types, 101
- NYSE
 - QuantLib::UnitedStates, 1222
- Once
 - datetime, 110
- OneFactorOperator
 - findiff, 129
- operator *
 - QuantLib::Array, 198
 - QuantLib::Matrix, 918
 - QuantLib::Money, 959
 - QuantLib::Period, 1029
- operator!=
 - QuantLib::Calendar, 288
 - QuantLib::Currency, 395
 - QuantLib::Date, 405
 - QuantLib::DayCounter, 407
 - QuantLib::Money, 959
 - QuantLib::Period, 1029
- operator()
 - QuantLib::EndCriteria, 454
- operator+
 - QuantLib::Array, 198
 - QuantLib::Matrix, 918
 - QuantLib::Money, 959
- operator+=
 - QuantLib::Matrix, 917
- operator-
 - QuantLib::Array, 198
 - QuantLib::Matrix, 918
 - QuantLib::Money, 959
 - QuantLib::Period, 1029
- operator/
 - QuantLib::Array, 198
 - QuantLib::Matrix, 918
 - QuantLib::Money, 959
- operator<
 - QuantLib::Date, 405
 - QuantLib::Money, 959
 - QuantLib::Period, 1029
- operator<<
 - QuantLib::Array, 198
 - QuantLib::Calendar, 288
 - QuantLib::Currency, 395
 - QuantLib::Date, 405
 - QuantLib::DayCounter, 407
 - QuantLib::InterestRate, 803
 - QuantLib::Matrix, 918
 - QuantLib::Money, 959
 - QuantLib::Option, 1017
 - QuantLib::Period, 1029
- operator<=
 - QuantLib::Date, 405
 - QuantLib::Money, 959
 - QuantLib::Period, 1029
- operator=
 - QuantLib::Observable, 999
- operator==
 - QuantLib::Calendar, 288
 - QuantLib::Currency, 395
 - QuantLib::Date, 405
 - QuantLib::DayCounter, 407
 - QuantLib::Money, 959
 - QuantLib::Period, 1029
- operator>
 - QuantLib::Date, 405
 - QuantLib::Money, 959
 - QuantLib::Period, 1029
- operator>=
 - QuantLib::Date, 405
 - QuantLib::Money, 959
 - QuantLib::Period, 1029
- Ordering
 - QuantLib::SobolBrownianGenerator, 1111
- outerProduct
 - QuantLib::Matrix, 918
- Output manipulators, 155

- parRate
 - QuantLib::YieldTermStructure, 1243, 1244
- partialRollback
 - QuantLib::Lattice, 844
 - QuantLib::TreeLattice, 1194
 - QuantLib::TsiveriotisFernandesLattice, 1207
- percentile
 - QuantLib::GeneralStatistics, 733
- performCalculations
 - QuantLib::Bond, 270
 - QuantLib::CompositeInstrument, 345
 - QuantLib::ConvertibleBond, 371
 - QuantLib::EurodollarFuturesImpliedStdDevQuote, 612
 - QuantLib::FixedRateBondForward, 650
 - QuantLib::Forward, 665
 - QuantLib::ForwardRateAgreement, 676
 - QuantLib::ImpliedStdDevQuote, 781
 - QuantLib::Instrument, 799
 - QuantLib::LazyObject, 847
 - QuantLib::Stock, 1139
 - QuantLib::Swap, 1148
 - QuantLib::SwaptionVolatilityMatrix, 1164
 - QuantLib::VarianceSwap, 1232
- Periodic
 - QuantLib::CubicSpline, 389
- postAdjustValues
 - QuantLib::DiscretizedAsset, 425
- postAdjustValuesImpl
 - QuantLib::DiscretizedAsset, 426
 - QuantLib::DiscretizedOption, 429
- potentialUpside
 - QuantLib::GenericRiskStatistics, 741
- preAdjustValues
 - QuantLib::DiscretizedAsset, 425
- preAdjustValuesImpl
 - QuantLib::DiscretizedAsset, 426
- Preceding
 - datetime, 110
- Pricing engines, 115
- primitive
 - QuantLib::AbcdFunction, 167
- PSE
 - QuantLib::CzechRepublic, 400
- pseudoSqrt
 - QuantLib::Matrix, 918
- ql/capvolstructures.hpp, 1255
- ql/cashflow.hpp, 1256
- ql/cashflows/analysis.hpp, 1257
- ql/cashflows/capflooredcoupon.hpp, 1258
- ql/cashflows/cashflowvectors.hpp, 1259
- ql/cashflows/cmscoupon.hpp, 1261
- ql/cashflows/conundrumpricer.hpp, 1262
- ql/cashflows/coupon.hpp, 1263
- ql/cashflows/couponpricer.hpp, 1264
- ql/cashflows/dividend.hpp, 1266
- ql/cashflows/fixedratecoupon.hpp, 1267
- ql/cashflows/floatingratecoupon.hpp, 1268
- ql/cashflows/iborcoupon.hpp, 1269
- ql/cashflows/rangeaccrual.hpp, 1270
- ql/cashflows/simplecashflow.hpp, 1271
- ql/cashflows/timebasket.hpp, 1272
- ql/currencies/africa.hpp, 1273
- ql/currencies/america.hpp, 1274
- ql/currencies/asia.hpp, 1276
- ql/currencies/europe.hpp, 1278
- ql/currencies/exchangeratemanager.hpp, 1281
- ql/currencies/oceania.hpp, 1282
- ql/currency.hpp, 1283
- ql/daycounter.hpp, 1284
- ql/discretizedasset.hpp, 1285
- ql/errors.hpp, 1286
- ql/event.hpp, 1289
- ql/exchangerate.hpp, 1290
- ql/exercise.hpp, 1291
- ql/grid.hpp, 1292
- ql/handle.hpp, 1293
- ql/index.hpp, 1294
- ql/indexes/ibor/audlibor.hpp, 1295
- ql/indexes/ibor/cadlibor.hpp, 1296
- ql/indexes/ibor/cdor.hpp, 1297
- ql/indexes/ibor/chflibor.hpp, 1298
- ql/indexes/ibor/dkklbtor.hpp, 1299
- ql/indexes/ibor/euribor.hpp, 1300
- ql/indexes/ibor/eurlibor.hpp, 1303
- ql/indexes/ibor/gbplbtor.hpp, 1305
- ql/indexes/ibor/jibar.hpp, 1306
- ql/indexes/ibor/jpylibor.hpp, 1307
- ql/indexes/ibor/libor.hpp, 1308
- ql/indexes/ibor/nzdlbtor.hpp, 1309
- ql/indexes/ibor/tlibor.hpp, 1310
- ql/indexes/ibor/trlibor.hpp, 1311
- ql/indexes/ibor/usdlbtor.hpp, 1312
- ql/indexes/ibor/zibor.hpp, 1313
- ql/indexes/iborindex.hpp, 1314
- ql/indexes/indexmanager.hpp, 1315
- ql/indexes/interestrateindex.hpp, 1316
- ql/indexes/swap/euriborswapfixa.hpp, 1317
- ql/indexes/swap/euriborswapfixb.hpp, 1319
- ql/indexes/swap/euriborswapfixifr.hpp, 1321
- ql/indexes/swap/eurliborswapfixa.hpp, 1323
- ql/indexes/swap/eurliborswapfixb.hpp, 1325
- ql/indexes/swap/eurliborswapfixifr.hpp, 1327
- ql/indexes/swapindex.hpp, 1329
- ql/instrument.hpp, 1330
- ql/instruments/asianoption.hpp, 1331

- ql/instruments/assetswap.hpp, 1332
- ql/instruments/barrieroption.hpp, 1333
- ql/instruments/basketoption.hpp, 1334
- ql/instruments/bond.hpp, 1335
- ql/instruments/callabilityschedule.hpp, 1337
- ql/instruments/capfloor.hpp, 1338
- ql/instruments/cliquestoption.hpp, 1340
- ql/instruments/cmsratebond.hpp, 1341
- ql/instruments/compositeinstrument.hpp, 1342
- ql/instruments/convertiblebond.hpp, 1343
- ql/instruments/dividendschedule.hpp, 1345
- ql/instruments/dividendvanillaoption.hpp, 1346
- ql/instruments/europeanoption.hpp, 1347
- ql/instruments/fixedratebond.hpp, 1348
- ql/instruments/fixedratebondforward.hpp, 1349
- ql/instruments/floatingratebond.hpp, 1350
- ql/instruments/forward.hpp, 1351
- ql/instruments/forwardrateagreement.hpp, 1352
- ql/instruments/forwardvanillaoption.hpp, 1353
- ql/instruments/lookbackoption.hpp, 1354
- ql/instruments/makecapfloor.hpp, 1355
- ql/instruments/makecms.hpp, 1356
- ql/instruments/makevanillaswap.hpp, 1357
- ql/instruments/multiassetoption.hpp, 1358
- ql/instruments/oneassetoption.hpp, 1359
- ql/instruments/oneassetstrikedoption.hpp, 1360
- ql/instruments/payoffs.hpp, 1361
- ql/instruments/quantoforwardvanillaoption.hpp, 1363
- ql/instruments/quantovanillaoption.hpp, 1364
- ql/instruments/stickyratech.hpp, 1365
- ql/instruments/stock.hpp, 1366
- ql/instruments/swap.hpp, 1367
- ql/instruments/swaption.hpp, 1368
- ql/instruments/vanillaoption.hpp, 1369
- ql/instruments/vanillaswap.hpp, 1370
- ql/instruments/varianceswap.hpp, 1371
- ql/instruments/zerocouponbond.hpp, 1372
- ql/interestrate.hpp, 1373
- ql/legacy/libormarketmodels/lmfcovarproxy.hpp, 1374
- ql/legacy/libormarketmodels/liborforwardmodel.hpp, 1375
- ql/legacy/libormarketmodels/lmconstwrappercorrelationmodel.hpp, 1376
- ql/legacy/libormarketmodels/lmconstwrappervolatilitymodel.hpp, 1377
- ql/legacy/libormarketmodels/lmcorrmodel.hpp, 1378
- ql/legacy/libormarketmodels/lmexpcorrmodel.hpp, 1379
- ql/legacy/libormarketmodels/lmextlinexpvolmodel.hpp, 1380
- ql/legacy/libormarketmodels/lmfixedvolmodel.hpp, 1381
- ql/legacy/libormarketmodels/lmlinexpcorrmodel.hpp, 1382
- ql/legacy/libormarketmodels/lmlinexpvolmodel.hpp, 1383
- ql/legacy/libormarketmodels/lmvolmodel.hpp, 1384
- ql/legacy/pricers/discretegeometricaso.hpp, 1385
- ql/legacy/pricers/mccliquestoption.hpp, 1386
- ql/legacy/pricers/mcdiscretearithmeticsaso.hpp, 1387
- ql/legacy/pricers/mceverest.hpp, 1388
- ql/legacy/pricers/mchimalaya.hpp, 1389
- ql/legacy/pricers/mcmaxbasket.hpp, 1390
- ql/legacy/pricers/mcpagoda.hpp, 1391
- ql/legacy/pricers/mcperformanceoption.hpp, 1392
- ql/legacy/pricers/mcpricer.hpp, 1393
- ql/legacy/pricers/singleassetoption.hpp, 1394
- ql/math/array.hpp, 1395
- ql/math/beta.hpp, 1396
- ql/math/comparison.hpp, 1397
- ql/math/curve.hpp, 1398
- ql/math/distributions/binomialdistribution.hpp, 1399
- ql/math/distributions/bivariatenormaldistribution.hpp, 1400
- ql/math/distributions/chisquaredistribution.hpp, 1401
- ql/math/distributions/gammadistribution.hpp, 1402
- ql/math/distributions/normaldistribution.hpp, 1403
- ql/math/distributions/poissondistribution.hpp, 1404
- ql/math/domain.hpp, 1405
- ql/math/errorfunction.hpp, 1406
- ql/math/factorial.hpp, 1407
- ql/math/functional.hpp, 1408
- ql/math/incompleategamma.hpp, 1409
- ql/math/integrals/gaussianorthogonalpolynomial.hpp, 1410
- ql/math/integrals/gaussianquadratures.hpp, 1412
- ql/math/integrals/integral.hpp, 1414
- ql/math/integrals/kronrodintegral.hpp, 1415
- ql/math/integrals/segmentintegral.hpp, 1416
- ql/math/integrals/simpsonintegral.hpp, 1417

- ql/math/integrals/trapezoidintegral.hpp, 1418
- ql/math/interpolation.hpp, 1419
- ql/math/interpolations/backwardflatinterpolation.hpp, 1420
- ql/math/interpolations/bicubicsplineinterpolation.hpp, 1421
- ql/math/interpolations/bilinearinterpolation.hpp, 1422
- ql/math/interpolations/cubicspline.hpp, 1423
- ql/math/interpolations/extrapolation.hpp, 1424
- ql/math/interpolations/flatextrapolation2d.hpp, 1425
- ql/math/interpolations/forwardflatinterpolation.hpp, 1426
- ql/math/interpolations/interpolation2d.hpp, 1427
- ql/math/interpolations/linearinterpolation.hpp, 1428
- ql/math/interpolations/loglinearinterpolation.hpp, 1429
- ql/math/interpolations/multicubicspline.hpp, 1430
- ql/math/interpolations/sabrinterpolation.hpp, 1432
- ql/math/lexicographicalview.hpp, 1433
- ql/math/linearleastquaresregression.hpp, 1434
- ql/math/matrix.hpp, 1435
- ql/math/matrixutilities/choleskydecomposition.hpp, 1436
- ql/math/matrixutilities/getcovariance.hpp, 1437
- ql/math/matrixutilities/pseudosqrt.hpp, 1438
- ql/math/matrixutilities/svd.hpp, 1439
- ql/math/matrixutilities/symmetricsschurdecomposition.hpp, 1440
- ql/math/matrixutilities/tqreigendecomposition.hpp, 1441
- ql/math/optimization/armijo.hpp, 1442
- ql/math/optimization/conjugategradient.hpp, 1443
- ql/math/optimization/constraint.hpp, 1444
- ql/math/optimization/costfunction.hpp, 1445
- ql/math/optimization/endcriteria.hpp, 1446
- ql/math/optimization/leastsquare.hpp, 1447
- ql/math/optimization/levenbergmarquardt.hpp, 1448
- ql/math/optimization/linearssearch.hpp, 1449
- ql/math/optimization/linearssearchbasedmethod.hpp, 1450
- ql/math/optimization/lmdif.hpp, 1451
- ql/math/optimization/method.hpp, 1452
- ql/math/optimization/problem.hpp, 1453
- ql/math/optimization/projectedcostfunction.hpp, 1454
- ql/math/optimization/simplex.hpp, 1455
- ql/math/optimization/steepestdescent.hpp, 1456
- ql/math/primenumbers.hpp, 1457
- ql/math/randomnumbers/boxmullergaussianrng.hpp, 1458
- ql/math/randomnumbers/centrallimitgaussianrng.hpp, 1459
- ql/math/randomnumbers/faurersg.hpp, 1460
- ql/math/randomnumbers/haltonrsg.hpp, 1461
- ql/math/randomnumbers/inversecumulativerng.hpp, 1462
- ql/math/randomnumbers/inversecumulativersg.hpp, 1463
- ql/math/randomnumbers/knuthuniformrng.hpp, 1464
- ql/math/randomnumbers/lecuyeruniformrng.hpp, 1465
- ql/math/randomnumbers/mt19937uniformrng.hpp, 1466
- ql/math/randomnumbers/randomizedlds.hpp, 1467
- ql/math/randomnumbers/randomsequencegenerator.hpp, 1468
- ql/math/randomnumbers/rngtraits.hpp, 1469
- ql/math/randomnumbers/seedgenerator.hpp, 1470
- ql/math/randomnumbers/sobolrsg.hpp, 1471
- ql/math/rounding.hpp, 1472
- ql/math/sampledcurve.hpp, 1473
- ql/math/solver1d.hpp, 1474
- ql/math/solvers1d/bisection.hpp, 1475
- ql/math/solvers1d/brent.hpp, 1476
- ql/math/solvers1d/falseposition.hpp, 1477
- ql/math/solvers1d/newton.hpp, 1478
- ql/math/solvers1d/newtonsafe.hpp, 1479
- ql/math/solvers1d/ridder.hpp, 1480
- ql/math/solvers1d/secant.hpp, 1481
- ql/math/statistics/convergencestatistics.hpp, 1482
- ql/math/statistics/discrepancystatistics.hpp, 1483
- ql/math/statistics/gaussianstatistics.hpp, 1484
- ql/math/statistics/generalstatistics.hpp, 1485
- ql/math/statistics/incrementalstatistics.hpp, 1486
- ql/math/statistics/riskstatistics.hpp, 1487
- ql/math/statistics/sequencestatistics.hpp, 1488
- ql/math/statistics/statistics.hpp, 1490
- ql/math/surface.hpp, 1491
- ql/math/transformedgrid.hpp, 1492
- ql/methods/finitedifferences/americancondition.hpp, 1493

- [ql/methods/finitedifferences/boundarycondition.hpp](#), 1494
[ql/methods/finitedifferences/bsmoperator.hpp](#), 1495
[ql/methods/finitedifferences/bsmtermoperator.hpp](#), 1496
[ql/methods/finitedifferences/cranknicolson.hpp](#), 1497
[ql/methods/finitedifferences/dminus.hpp](#), 1498
[ql/methods/finitedifferences/dplus.hpp](#), 1499
[ql/methods/finitedifferences/dplusdminus.hpp](#), 1500
[ql/methods/finitedifferences/dzero.hpp](#), 1501
[ql/methods/finitedifferences/expliciteuler.hpp](#), 1502
[ql/methods/finitedifferences/fdtypedefs.hpp](#), 1503
[ql/methods/finitedifferences/finitedifferencemodel.hpp](#), 1504
[ql/methods/finitedifferences/impliciteuler.hpp](#), 1505
[ql/methods/finitedifferences/mixedscheme.hpp](#), 1506
[ql/methods/finitedifferences/onefactoroperator.hpp](#), 1507
[ql/methods/finitedifferences/operatorfactory.hpp](#), 1508
[ql/methods/finitedifferences/operatortraits.hpp](#), 1509
[ql/methods/finitedifferences/parallelevolver.hpp](#), 1510
[ql/methods/finitedifferences/pde.hpp](#), 1511
[ql/methods/finitedifferences/pdebsm.hpp](#), 1512
[ql/methods/finitedifferences/pdeshortrate.hpp](#), 1513
[ql/methods/finitedifferences/shoutcondition.hpp](#), 1514
[ql/methods/finitedifferences/stepcondition.hpp](#), 1515
[ql/methods/finitedifferences/tridiagonaloperator.hpp](#), 1516
[ql/methods/finitedifferences/zerocondition.hpp](#), 1518
[ql/methods/lattices/binomialtree.hpp](#), 1519
[ql/methods/lattices/bsmlattice.hpp](#), 1521
[ql/methods/lattices/lattice.hpp](#), 1522
[ql/methods/lattices/lattice1d.hpp](#), 1523
[ql/methods/lattices/lattice2d.hpp](#), 1524
[ql/methods/lattices/tflattice.hpp](#), 1525
[ql/methods/lattices/tree.hpp](#), 1526
[ql/methods/lattices/trinomialtree.hpp](#), 1527
[ql/methods/montecarlo/brownianbridge.hpp](#), 1528
[ql/methods/montecarlo/earlyexercisepathpricer.hpp](#), 1529
[ql/methods/montecarlo/longstaffschwartzpathpricer.hpp](#), 1530
[ql/methods/montecarlo/lsmbasisystem.hpp](#), 1531
[ql/methods/montecarlo/mctraits.hpp](#), 1532
[ql/methods/montecarlo/montecarlomodel.hpp](#), 1533
[ql/methods/montecarlo/multipath.hpp](#), 1534
[ql/methods/montecarlo/multipathgenerator.hpp](#), 1535
[ql/methods/montecarlo/path.hpp](#), 1536
[ql/methods/montecarlo/pathgenerator.hpp](#), 1537
[ql/methods/montecarlo/pathpricer.hpp](#), 1538
[ql/methods/montecarlo/sample.hpp](#), 1539
[ql/models/calibrationhelper.hpp](#), 1540
[ql/models/equity/batesmodel.hpp](#), 1541
[ql/models/equity/hestonmodel.hpp](#), 1542
[ql/models/equity/hestonmodelhelper.hpp](#), 1543
[ql/models/marketmodels/driftcomputation/cmsmmdriftcalculator.hpp](#), 1544
[ql/models/marketmodels/driftcomputation/lmmdriftcalculator.hpp](#), 1545
[ql/models/marketmodels/driftcomputation/lmmnormaldriftcalculator.hpp](#), 1546
[ql/models/marketmodels/driftcomputation/smmdriftcalculator.hpp](#), 1547
[ql/models/marketmodels/swapforwardmappings.hpp](#), 1548
[ql/models/model.hpp](#), 1549
[ql/models/parameter.hpp](#), 1550
[ql/models/shortrate/calibrationhelpers/caphelper.hpp](#), 1551
[ql/models/shortrate/calibrationhelpers/swaptionhelper.hpp](#), 1552
[ql/models/shortrate/onefactormodel.hpp](#), 1553
[ql/models/shortrate/onefactormodels/blackkarasinski.hpp](#), 1554
[ql/models/shortrate/onefactormodels/coxingersollross.hpp](#), 1555
[ql/models/shortrate/onefactormodels/extendedcoxingersollross.hpp](#), 1556
[ql/models/shortrate/onefactormodels/hullwhite.hpp](#), 1557
[ql/models/shortrate/onefactormodels/vasicek.hpp](#), 1558
[ql/models/shortrate/twofactormodel.hpp](#), 1559
[ql/models/shortrate/twofactormodels/g2.hpp](#), 1560
[ql/models/volatility/constantestimator.hpp](#), 1561
[ql/models/volatility/garch.hpp](#), 1562

- ql/models/volatility/garmanklass.hpp, 1563
- ql/models/volatility/simplelocalestimator.hpp, 1564
- ql/money.hpp, 1565
- ql/numericalmethod.hpp, 1566
- ql/option.hpp, 1567
- ql/patterns/composite.hpp, 1568
- ql/patterns/curiouslyrecurring.hpp, 1569
- ql/patterns/lazyobject.hpp, 1570
- ql/patterns/observable.hpp, 1571
- ql/patterns/singleton.hpp, 1572
- ql/patterns/visitor.hpp, 1573
- ql/payoff.hpp, 1574
- ql/position.hpp, 1575
- ql/prices.hpp, 1576
- ql/pricingengine.hpp, 1577
- ql/pricingengines/americanpayoffatexpiry.hpp, 1578
- ql/pricingengines/americanpayoffathit.hpp, 1579
- ql/pricingengines/asian/analytic_cont_geom_av_price.hpp, 1580
- ql/pricingengines/asian/analytic_discr_geom_av_price.hpp, 1581
- ql/pricingengines/asian/mc_discr_arith_av_price.hpp, 1582
- ql/pricingengines/asian/mc_discr_geom_av_price.hpp, 1583
- ql/pricingengines/asian/mcdiscreteasianengine.hpp, 1584
- ql/pricingengines/barrier/analyticbarrierengine.hpp, 1585
- ql/pricingengines/barrier/mcbarrierengine.hpp, 1586
- ql/pricingengines/basket/mcamericanbasketengine.hpp, 1587
- ql/pricingengines/basket/mcbasketengine.hpp, 1588
- ql/pricingengines/basket/stulzengine.hpp, 1589
- ql/pricingengines/blackcalculator.hpp, 1590
- ql/pricingengines/blackformula.hpp, 1591
- ql/pricingengines/blackscholescalculator.hpp, 1593
- ql/pricingengines/capfloor/analyticcapfloorengine.hpp, 1594
- ql/pricingengines/capfloor/blackcapfloorengine.hpp, 1595
- ql/pricingengines/capfloor/discretizedcapfloor.hpp, 1596
- ql/pricingengines/capfloor/marketmodelcapfloorengine.hpp, 1597
- ql/pricingengines/capfloor/mchullwhiteengine.hpp, 1598
- ql/pricingengines/capfloor/treecapfloorengine.hpp, 1599
- ql/pricingengines/cliquet/analyticcliquetengine.hpp, 1600
- ql/pricingengines/cliquet/analyticperformanceengine.hpp, 1601
- ql/pricingengines/forward/forwardengine.hpp, 1602
- ql/pricingengines/forward/forwardperformanceengine.hpp, 1603
- ql/pricingengines/forward/mcvarianceswapengine.hpp, 1604
- ql/pricingengines/forward/replicatingvarianceswapengine.hpp, 1605
- ql/pricingengines/genericmodelengine.hpp, 1606
- ql/pricingengines/greeks.hpp, 1607
- ql/pricingengines/hybrid/binomialconvertibleengine.hpp, 1608
- ql/pricingengines/hybrid/discretizedconvertible.hpp, 1609
- ql/pricingengines/latticeshortratemodelengine.hpp, 1610
- ql/pricingengines/lookback/analyticcontinuousfixedlookback.hpp, 1611
- ql/pricingengines/lookback/analyticcontinuousfloatinglookback.hpp, 1612
- ql/pricingengines/mclongstaffschwartzengine.hpp, 1613
- ql/pricingengines/mcsimulation.hpp, 1614
- ql/pricingengines/quanto/quantoengine.hpp, 1615
- ql/pricingengines/swaption/blackswaptionengine.hpp, 1616
- ql/pricingengines/swaption/discretizedswaption.hpp, 1617
- ql/pricingengines/swaption/g2swaptionengine.hpp, 1618
- ql/pricingengines/swaption/jamshidianswaptionengine.hpp, 1619
- ql/pricingengines/swaption/lfmswaptionengine.hpp, 1620
- ql/pricingengines/swaption/treeswaptionengine.hpp, 1621
- ql/pricingengines/vanilla/analyticdigitalamericanengine.hpp, 1622
- ql/pricingengines/vanilla/analyticdividendeuropeanengine.hpp, 1623
- ql/pricingengines/vanilla/analyticeuropeanengine.hpp, 1624
- ql/pricingengines/vanilla/analytichestonengine.hpp, 1625
- ql/pricingengines/vanilla/baroneadesiwhaleyengine.hpp, 1626

- ql/pricingengines/vanilla/batesengine.hpp, 1627
- ql/pricingengines/vanilla/binomialengine.hpp, 1628
- ql/pricingengines/vanilla/bjerkhundstenglandengine.hpp, 1629
- ql/pricingengines/vanilla/discretizedvanillaoption.hpp, 1630
- ql/pricingengines/vanilla/fdamericanengine.hpp, 1631
- ql/pricingengines/vanilla/fdbermudanengine.hpp, 1632
- ql/pricingengines/vanilla/fdconditions.hpp, 1633
- ql/pricingengines/vanilla/fddividendamericanengine.hpp, 1634
- ql/pricingengines/vanilla/fddividendengine.hpp, 1635
- ql/pricingengines/vanilla/fddividendeuropeanengine.hpp, 1636
- ql/pricingengines/vanilla/fddividendshoutengine.hpp, 1637
- ql/pricingengines/vanilla/fdeuropeanengine.hpp, 1638
- ql/pricingengines/vanilla/fdmultiengine.hpp, 1639
- ql/pricingengines/vanilla/fdshoutengine.hpp, 1640
- ql/pricingengines/vanilla/fdstepconditionengine.hpp, 1641
- ql/pricingengines/vanilla/fdvanillaengine.hpp, 1642
- ql/pricingengines/vanilla/integralengine.hpp, 1643
- ql/pricingengines/vanilla/jumpdiffusionengine.hpp, 1644
- ql/pricingengines/vanilla/juquadraticengine.hpp, 1645
- ql/pricingengines/vanilla/mcamericanengine.hpp, 1646
- ql/pricingengines/vanilla/mcdigitalengine.hpp, 1647
- ql/pricingengines/vanilla/mceuropeanengine.hpp, 1648
- ql/pricingengines/vanilla/mceuropeanhestonengine.hpp, 1649
- ql/pricingengines/vanilla/mcvanillaengine.hpp, 1650
- ql/processes/blackscholesprocess.hpp, 1651
- ql/processes/eulerdiscretization.hpp, 1652
- ql/processes/forwardmeasureprocess.hpp, 1653
- ql/processes/g2process.hpp, 1654
- ql/processes/geometricbrownianprocess.hpp, 1655
- ql/processes/hestonprocess.hpp, 1656
- ql/processes/hullwhiteprocess.hpp, 1657
- ql/processes/lfmcovarParams.hpp, 1658
- ql/processes/lfmhullwhiteparam.hpp, 1659
- ql/processes/lfmprocess.hpp, 1660
- ql/processes/merton76process.hpp, 1661
- ql/processes/ornsteinuhlenbeckprocess.hpp, 1662
- ql/processes/squarerootprocess.hpp, 1663
- ql/processes/stochasticprocessarray.hpp, 1664
- ql/qldefines.hpp, 1665
- ql/quote.hpp, 1667
- ql/quotes/compositequote.hpp, 1668
- ql/quotes/derivedquote.hpp, 1669
- ql/quotes/eurodollarfuturesquote.hpp, 1670
- ql/quotes/forwardvaluequote.hpp, 1671
- ql/quotes/futuresconvadjustmentquote.hpp, 1672
- ql/quotes/impliedstddevquote.hpp, 1673
- ql/quotes/simplequote.hpp, 1674
- ql/settings.hpp, 1675
- ql/stochasticprocess.hpp, 1676
- ql/swaptionvolstructure.hpp, 1677
- ql/termstructure.hpp, 1678
- ql/termstructures/volatilities/blackconstantvol.hpp, 1679
- ql/termstructures/volatilities/blackvariancecurve.hpp, 1680
- ql/termstructures/volatilities/blackvariancesurface.hpp, 1681
- ql/termstructures/volatilities/capflatvolvector.hpp, 1682
- ql/termstructures/volatilities/capletconstantvol.hpp, 1683
- ql/termstructures/volatilities/capletvariancecurve.hpp, 1684
- ql/termstructures/volatilities/capletvolatilitiesstructures.hpp, 1685
- ql/termstructures/volatilities/capstripper.hpp, 1686
- ql/termstructures/volatilities/cmsmarket.hpp, 1687
- ql/termstructures/volatilities/impliedvoltermstructure.hpp, 1688
- ql/termstructures/volatilities/interpolatedsmilesection.hpp, 1689
- ql/termstructures/volatilities/localconstantvol.hpp, 1690
- ql/termstructures/volatilities/localvolcurve.hpp, 1691
- ql/termstructures/volatilities/localvolsurface.hpp, 1692
- ql/termstructures/volatilities/sabr.hpp, 1693

- ql/termstructures/volatilities/sabrinterpolatedsmilesection.hpp, 1722
- 1694
- ql/termstructures/volatilities/smilesection.hpp, 1723
- 1695
- ql/termstructures/volatilities/swaptionconstantvol.hpp, 1724
- 1696
- ql/termstructures/volatilities/swaptionvolcube.hpp, 1725
- 1697
- ql/termstructures/volatilities/swaptionvolcube1.hpp, 1726
- 1698
- ql/termstructures/volatilities/swaptionvolcube2.hpp, 1727
- 1699
- ql/termstructures/volatilities/swaptionvoldiscreted.hpp, 1728
- 1700
- ql/termstructures/volatilities/swaptionvolmatrix.hpp, 1729
- 1701
- ql/termstructures/yieldcurves/bondhelpers.hpp, 1730
- 1702
- ql/termstructures/yieldcurves/bootstraptraits.hpp, 1731
- 1703
- ql/termstructures/yieldcurves/compoundforwardcurve.hpp, 1732
- 1704
- ql/termstructures/yieldcurves/discountcurve.hpp, 1733
- 1705
- ql/termstructures/yieldcurves/drifttermstructure.hpp, 1734
- 1706
- ql/termstructures/yieldcurves/extendeddiscountcurve.hpp, 1735
- 1707
- ql/termstructures/yieldcurves/flatforward.hpp, 1736
- 1708
- ql/termstructures/yieldcurves/forwardcurve.hpp, 1737
- 1709
- ql/termstructures/yieldcurves/forwardspreadedtermstructure.hpp, 1738
- 1710
- ql/termstructures/yieldcurves/forwardstructure.hpp, 1739
- 1711
- ql/termstructures/yieldcurves/implicittermstructure.hpp, 1740
- 1712
- ql/termstructures/yieldcurves/piecewiseyieldcurve.hpp, 1741
- 1713
- ql/termstructures/yieldcurves/piecewisezerospreadedtermstructure.hpp, 1742
- 1714
- ql/termstructures/yieldcurves/quantotermstructure.hpp, 1743
- 1715
- ql/termstructures/yieldcurves/ratehelpers.hpp, 1744
- 1716
- ql/termstructures/yieldcurves/zerocurve.hpp, 1745
- 1717
- ql/termstructures/yieldcurves/zerospreadedtermstructure.hpp, 1746
- 1718
- ql/termstructures/yieldcurves/zeroyieldstructure.hpp, 1747
- 1719
- ql/time/businessdayconvention.hpp, 1748
- ql/time/calendar.hpp, 1721
- ql/time/calendars/argentina.hpp, 1722
- ql/time/calendars/australia.hpp, 1723
- ql/time/calendars/brazil.hpp, 1724
- ql/time/calendars/canada.hpp, 1725
- ql/time/calendars/china.hpp, 1726
- ql/time/calendars/czechrepublic.hpp, 1727
- ql/time/calendars/denmark.hpp, 1728
- ql/time/calendars/finland.hpp, 1729
- ql/time/calendars/germany.hpp, 1730
- ql/time/calendars/hongkong.hpp, 1731
- ql/time/calendars/hungary.hpp, 1732
- ql/time/calendars/iceland.hpp, 1733
- ql/time/calendars/india.hpp, 1734
- ql/time/calendars/indonesia.hpp, 1735
- ql/time/calendars/italy.hpp, 1736
- ql/time/calendars/japan.hpp, 1737
- ql/time/calendars/jointcalendar.hpp, 1738
- ql/time/calendars/mexico.hpp, 1739
- ql/time/calendars/newzealand.hpp, 1740
- ql/time/calendars/norway.hpp, 1741
- ql/time/calendars/nullcalendar.hpp, 1742
- ql/time/calendars/poland.hpp, 1743
- ql/time/calendars/saudiArabia.hpp, 1744
- ql/time/calendars/singapore.hpp, 1745
- ql/time/calendars/slovakia.hpp, 1746
- ql/time/calendars/southAfrica.hpp, 1747
- ql/time/calendars/southKorea.hpp, 1748
- ql/time/calendars/sweden.hpp, 1749
- ql/time/calendars/switzerland.hpp, 1750
- ql/time/calendars/taiwan.hpp, 1751
- ql/time/calendars/target.hpp, 1752
- ql/time/calendars/turkey.hpp, 1753
- ql/time/calendars/ukraine.hpp, 1754
- ql/time/calendars/unitedkingdom.hpp, 1755
- ql/time/calendars/unitedstates.hpp, 1756
- ql/time/date.hpp, 1757
- ql/time/daycounters/actual360.hpp, 1759
- ql/time/daycounters/actual365fixed.hpp, 1760
- ql/time/daycounters/actualactual.hpp, 1761
- ql/time/daycounters/business252.hpp, 1762
- ql/time/daycounters/isle.hpp, 1763
- ql/time/daycounters/simpledaycounter.hpp, 1764
- ql/time/daycounters/thirty360.hpp, 1765
- ql/time/frequency.hpp, 1766
- ql/time/imm.hpp, 1767
- ql/time/period.hpp, 1768
- ql/time/schedule.hpp, 1770
- ql/time/timeunit.hpp, 1771
- ql/time/weekday.hpp, 1772
- ql/timegrid.hpp, 1773
- ql/timeseries.hpp, 1774
- ql/types.hpp, 1775
- ql/utilities/clone.hpp, 1777

ql/utilities/dataformatters.hpp, 1778
 ql/utilities/dataparsers.hpp, 1780
 ql/utilities/disposable.hpp, 1781
 ql/utilities/null.hpp, 1782
 ql/utilities/observablevalue.hpp, 1783
 ql/utilities/steppingiterator.hpp, 1784
 ql/utilities/tracing.hpp, 1785
 ql/volatilitymodel.hpp, 1787
 ql/voltermstructure.hpp, 1788
 ql/yieldtermstructure.hpp, 1789
 QL_ASSERT
 errors.hpp, 1287
 QL_DUMMY_RETURN
 miscMacros, 151
 QL_ENSURE
 errors.hpp, 1287
 QL_EPSILON
 limitMacros, 152
 QL_FAIL
 errors.hpp, 1287
 QL_FULL_ITERATOR_SUPPORT
 iteratorMacros, 154
 QL_MAX_INTEGER
 limitMacros, 152
 QL_MAX_REAL
 limitMacros, 152
 QL_MIN_INTEGER
 limitMacros, 152
 QL_MIN_POSITIVE_REAL
 limitMacros, 152
 QL_MIN_REAL
 limitMacros, 152
 QL_REQUIRE
 errors.hpp, 1287
 QL_TRACE
 debugMacros, 158
 QL_TRACE_DISABLE
 debugMacros, 157
 QL_TRACE_ENABLE
 debugMacros, 157
 QL_TRACE_ENTER_FUNCTION
 debugMacros, 158
 QL_TRACE_EXIT_FUNCTION
 debugMacros, 158
 QL_TRACE_LOCATION
 debugMacros, 159
 QL_TRACE_ON
 debugMacros, 158
 QL_TRACE_VARIABLE
 debugMacros, 159
 QL_TYPENAME
 templateMacros, 153
 QuantLib macros, 150
 QuantLib::Abcd, 163

covariance, 165
 instantaneousCovariance, 164
 instantaneousVariance, 164
 instantaneousVolatility, 164
 variance, 165
 volatility, 165
 QuantLib::AbcdFunction, 166
 covariance, 167
 primitive, 167
 variance, 167
 volatility, 167
 QuantLib::AbcdVol, 168
 QuantLib::AccountingEngine, 169
 QuantLib::Actual360, 170
 QuantLib::Actual365Fixed, 171
 QuantLib::ActualActual, 172
 QuantLib::AcyclicVisitor, 173
 QuantLib::AdditiveEQPBinoomialTree, 174
 QuantLib::AffineModel, 175
 QuantLib::AmericanCondition, 176
 QuantLib::AmericanExercise, 177
 QuantLib::AmericanPayoffAtExpiry, 178
 QuantLib::AmericanPayoffAtHit, 179
 QuantLib::AnalyticBarrierEngine, 180
 QuantLib::AnalyticCapFloorEngine, 181
 QuantLib::AnalyticCliquetEngine, 182
 QuantLib::AnalyticContinuousFixedLookbackEngine,
 183
 QuantLib::AnalyticContinuousFloatingLookbackEngine,
 184
 QuantLib::AnalyticContinuousGeometricAveragePriceAsianEngine,
 185
 QuantLib::AnalyticDigitalAmericanEngine,
 186
 QuantLib::AnalyticDiscreteGeometricAveragePriceAsianEngine,
 187
 QuantLib::AnalyticDividendEuropeanEngine,
 188
 QuantLib::AnalyticEuropeanEngine, 189
 QuantLib::AnalyticHestonEngine, 190
 QuantLib::AnalyticPerformanceEngine, 191
 QuantLib::Argentina, 192
 Market, 193
 Merval, 193
 QuantLib::ArmijoLineSearch, 194
 QuantLib::Array, 195
 Abs, 198
 DotProduct, 198
 Exp, 198
 Log, 198
 operator *, 198
 operator +, 198
 operator -, 198
 operator /, 198

- operator<<, 198
- Sqrt, 198
- swap, 198
- QuantLib::ARSCurrency, 199
- QuantLib::AssetOrNothingPayoff, 200
 - name, 200
- QuantLib::AssetSwap, 201
 - fetchResults, 202
- QuantLib::AssetSwap::arguments, 203
- QuantLib::AssetSwap::results, 204
- QuantLib::ATSCurrency, 205
- QuantLib::AUDCurrency, 206
- QuantLib::AUDLibor, 207
- QuantLib::Australia, 208
- QuantLib::Average, 209
- QuantLib::BackwardFlat, 210
- QuantLib::BackwardFlatInterpolation, 211
 - BackwardFlatInterpolation, 211
- QuantLib::BaroneAdesiWhaleyApproximationEngine, 212
- QuantLib::Barrier, 213
- QuantLib::BarrierOption, 214
 - QuantLib::BarrierOption::arguments, 216
 - QuantLib::BarrierOption::engine, 217
- QuantLib::BasketOption, 218
 - QuantLib::BasketOption::arguments, 219
 - QuantLib::BasketOption::engine, 220
- QuantLib::BatesEngine, 221
- QuantLib::BatesModel, 223
- QuantLib::BDTCurrency, 224
- QuantLib::BEFCurrency, 225
- QuantLib::BermudanExercise, 226
- QuantLib::BGLCurrency, 227
- QuantLib::Bicubic, 228
- QuantLib::BicubicSpline, 229
 - BicubicSpline, 229
- QuantLib::Bilinear, 230
- QuantLib::BilinearInterpolation, 231
 - BilinearInterpolation, 231
- QuantLib::BinomialConvertibleEngine, 232
- QuantLib::BinomialDistribution, 233
- QuantLib::BinomialTree, 234
- QuantLib::BinomialVanillaEngine, 235
- QuantLib::Bisection, 236
- QuantLib::BivariateCumulativeNormalDistribution, 237
 - QuantLib::BivariateCumulativeNormalDistribution, 238
- QuantLib::Bjerk SundStenslandApproximationEngine, 239
- QuantLib::BlackCalculator, 240
 - delta, 241
 - deltaForward, 241
 - dividendRho, 242
 - elasticity, 241
 - elasticityForward, 241
 - gamma, 241
 - gammaForward, 241
 - itmAssetProbability, 242
 - itmCashProbability, 242
 - rho, 242
 - strikeSensitivity, 242
 - theta, 242
 - thetaPerDay, 242
 - vega, 242
- QuantLib::BlackCapFloorEngine, 243
 - update, 243
- QuantLib::BlackConstantVol, 244
- QuantLib::BlackIborCouponPricer, 246
- QuantLib::BlackKarasinski, 247
- QuantLib::BlackKarasinski::Dynamics, 248
- QuantLib::BlackProcess, 249
- QuantLib::BlackScholesCalculator, 250
 - delta, 250
 - elasticity, 250
 - gamma, 250
 - theta, 250
 - thetaPerDay, 250
- QuantLib::BlackScholesLattice, 252
- QuantLib::BlackScholesMertonProcess, 253
- QuantLib::BlackScholesProcess, 254
- QuantLib::BlackSwaptionEngine, 255
 - update, 255
- QuantLib::BlackVarianceCurve, 256
- QuantLib::BlackVarianceSurface, 258
- QuantLib::BlackVarianceTermStructure, 260
 - BlackVarianceTermStructure, 261
 - blackVollImpl, 261
- QuantLib::BlackVolatilityTermStructure, 262
 - blackVarianceImpl, 263
 - BlackVolatilityTermStructure, 263
- QuantLib::BlackVolTermStructure, 264
 - BlackVolTermStructure, 266
- QuantLib::Bond, 267
 - accruedAmount, 270
 - cashflows, 269
 - cleanPrice, 269
 - dirtyPrice, 269
 - formCalculations, 270
 - yield, 269
- QuantLib::BoundaryCondition, 271
 - applyAfterApplying, 271
 - applyAfterSolving, 272
 - applyBeforeApplying, 271
 - applyBeforeSolving, 271
 - setTime, 272
 - Side, 271
- QuantLib::BoundaryConstraint, 273

- QuantLib::BoxMullerGaussianRng, 274
- QuantLib::Brazil, 275
 - Market, 276
 - Settlement, 276
- QuantLib::Brent, 277
- QuantLib::BRLCurrency, 278
- QuantLib::BrownianBridge, 279
 - BrownianBridge, 279
- QuantLib::BSMOperator, 280
- QuantLib::Business252, 281
- QuantLib::BYRCurrency, 282
- QuantLib::CADCurrency, 283
- QuantLib::CADLibor, 284
- QuantLib::Calendar, 285
 - addHoliday, 287
 - adjust, 287
 - advance, 287, 288
 - businessDaysBetween, 288
 - Calendar, 286
 - isBusinessDay, 287
 - isEndOfMonth, 287
 - isHoliday, 287
 - isWeekend, 287
 - name, 287
 - operator!=, 288
 - operator<<, 288
 - operator==, 288
 - removeHoliday, 287
- QuantLib::Calendar::Impl, 289
- QuantLib::Calendar::OrthodoxImpl, 290
- QuantLib::Calendar::WesternImpl, 291
- QuantLib::CalibratedModel, 292
 - calibrate, 293
 - update, 293
- QuantLib::CalibrationHelper, 294
 - update, 295
- QuantLib::Callability, 296
- QuantLib::Callability::Price, 297
- QuantLib::Canada, 298
- QuantLib::Cap, 299
- QuantLib::CapFloor, 300
- QuantLib::CapFloor::arguments, 302
- QuantLib::CapFloor::engine, 303
- QuantLib::CapHelper, 304
- QuantLib::CapletConstantVolatility, 305
- QuantLib::CapletVolatilityStructure, 307
 - CapletVolatilityStructure, 308
- QuantLib::CappedFlooredCoupon, 309
 - update, 310
- QuantLib::CapVolatilityStructure, 311
 - CapVolatilityStructure, 312
- QuantLib::CapVolatilityVector, 313
 - update, 314
- QuantLib::CashFlow, 315
 - amount, 315
- QuantLib::CashFlows, 317
 - atmRate, 318
 - bps, 318
 - convexity, 319
 - duration, 319
 - irr, 318
 - npv, 318
- QuantLib::CashOrNothingPayoff, 320
 - name, 320
- QuantLib::Cdor, 321
- QuantLib::CeilingTruncation, 322
- QuantLib::CHFCurrency, 323
- QuantLib::CHFLibor, 324
- QuantLib::China, 325
- QuantLib::CLGaussianRng, 326
- QuantLib::CliquetOption, 327
- QuantLib::CliquetOption::arguments, 329
- QuantLib::CliquetOption::engine, 330
- QuantLib::Clone, 331
 - swap, 331
- QuantLib::ClosestRounding, 332
- QuantLib::CLPCurrency, 333
- QuantLib::CmsCoupon, 334
- QuantLib::CmsCouponPricer, 335
- QuantLib::CmsMarket, 336
 - update, 336
- QuantLib::CMSMMDriftCalculator, 337
- QuantLib::CmsRateBond, 338
- QuantLib::CMSwapCurveState, 339
- QuantLib::CNYCurrency, 340
- QuantLib::Collar, 341
- QuantLib::Composite, 342
- QuantLib::CompositeConstraint, 343
- QuantLib::CompositeInstrument, 344
 - performCalculations, 345
- QuantLib::CompositeQuote, 346
 - update, 346
- QuantLib::CompoundForward, 347
 - discountImpl, 348
 - zeroYieldImpl, 348
- QuantLib::ConjugateGradient, 349
- QuantLib::ConstantEstimator, 350
- QuantLib::ConstantParameter, 351
- QuantLib::ConstrainedEvolver, 352
- QuantLib::Constraint, 353
- QuantLib::Constraint::Impl, 354
- QuantLib::ContinuousAveragingAsianOption, 355
 - QuantLib::ContinuousAveragingAsianOption::arguments, 357
 - QuantLib::ContinuousAveragingAsianOption::engine, 358

- QuantLib::ContinuousFixedLookbackOption,
 - 359
- QuantLib::ContinuousFixedLookbackOption::arguments,
 - 360
- QuantLib::ContinuousFixedLookbackOption::engine,
 - 361
- QuantLib::ContinuousFloatingLookbackOption,
 - 362
- QuantLib::ContinuousFloatingLookbackOption::arguments,
 - 363
- QuantLib::ContinuousFloatingLookbackOption::engine,
 - 364
- QuantLib::ConundrumPricer, 365
- QuantLib::ConundrumPricerByBlack, 367
- QuantLib::ConundrumPricerByNumericalIntegration,
 - 368
- QuantLib::ConvergenceStatistics, 369
- QuantLib::ConvertibleBond, 370
 - performCalculations, 371
- QuantLib::ConvertibleFixedCouponBond, 372
- QuantLib::ConvertibleFloatingRateBond, 373
- QuantLib::ConvertibleZeroCouponBond, 374
- QuantLib::COPCurrency, 375
- QuantLib::CostFunction, 376
- QuantLib::CoterminalSwapCurveState, 377
- QuantLib::Coupon, 378
 - Coupon, 379
- QuantLib::CovarianceDecomposition, 380
 - correlationMatrix, 380
 - CovarianceDecomposition, 380
 - standardDeviations, 380
 - variances, 380
- QuantLib::CoxIngersollRoss, 381
- QuantLib::CoxIngersollRoss::Dynamics, 383
- QuantLib::CoxRossRubinstein, 384
- QuantLib::CrankNicolson, 385
- QuantLib::Cubic, 387
- QuantLib::CubicSpline, 388
 - BoundaryCondition, 389
 - CubicSpline, 389
 - FirstDerivative, 389
 - Lagrange, 389
 - NotAKnot, 389
 - Periodic, 389
 - SecondDerivative, 389
- QuantLib::CumulativeBinomialDistribution,
 - 390
- QuantLib::CumulativeNormalDistribution,
 - 391
- QuantLib::CumulativePoissonDistribution,
 - 392
- QuantLib::CuriouslyRecurringTemplate, 393
- QuantLib::Currency, 394
 - Currency, 395
 - format, 395
 - operator!=, 395
 - operator<<, 395
 - operator==, 395
- QuantLib::Curve, 396
 - QuantLib::CurveState, 397
- QuantLib::CYPCurrency, 398
- QuantLib::CzechRepublic, 399
- Market, 400
 - PSE, 400
- QuantLib::CZKCurrency, 401
- QuantLib::Date, 402
 - nextWeekday, 404
 - nthWeekday, 404
- operator!=, 405
- operator<, 405
- operator<<, 405
- operator<=, 405
- operator==, 405
- operator>, 405
- operator>=, 405
- QuantLib::DayCounter, 406
 - DayCounter, 407
 - name, 407
 - operator!=, 407
 - operator<<, 407
 - operator==, 407
- QuantLib::DayCounter::Impl, 408
- QuantLib::DecInterpCapletVolStructure, 409
 - update, 409
- QuantLib::DEMCurrency, 410
- QuantLib::Denmark, 411
- QuantLib::DepositRateHelper, 412
 - setTermStructure, 412
- QuantLib::DerivedQuote, 414
 - update, 414
- QuantLib::DirichletBC, 415
 - applyAfterApplying, 415
 - applyAfterSolving, 415
 - applyBeforeApplying, 415
 - applyBeforeSolving, 415
 - setTime, 416
- QuantLib::Discount, 417
- QuantLib::DiscrepancyStatistics, 418
- QuantLib::DiscreteAveragingAsianOption, 419
- QuantLib::DiscreteAveragingAsianOption::arguments,
 - 421
- QuantLib::DiscreteAveragingAsianOption::engine,
 - 422
- QuantLib::DiscreteGeometricASO, 423
- QuantLib::DiscretizedAsset, 424
 - adjustValues, 425
 - isOnTime, 425
 - mandatoryTimes, 425

- postAdjustValues, [425](#)
- postAdjustValuesImpl, [426](#)
- preAdjustValues, [425](#)
- preAdjustValuesImpl, [426](#)
- reset, [425](#)
- QuantLib::DiscretizedDiscountBond, [427](#)
 - mandatoryTimes, [427](#)
 - reset, [427](#)
- QuantLib::DiscretizedOption, [428](#)
 - mandatoryTimes, [428](#)
 - postAdjustValuesImpl, [429](#)
 - reset, [428](#)
- QuantLib::Disposable, [430](#)
- QuantLib::Dividend, [431](#)
 - amount, [432](#)
- QuantLib::DividendVanillaOption, [433](#)
- QuantLib::DividendVanillaOption::arguments, [435](#)
- QuantLib::DividendVanillaOption::engine, [436](#)
- QuantLib::DKKCurrency, [437](#)
- QuantLib::DKKLibor, [438](#)
- QuantLib::DMinus, [439](#)
- QuantLib::Domain, [440](#)
- QuantLib::DoubleStickyRatchetPayoff, [441](#)
 - name, [442](#)
- QuantLib::DownRounding, [443](#)
- QuantLib::DPlus, [444](#)
- QuantLib::DPlusDMinus, [445](#)
- QuantLib::DriftTermStructure, [446](#)
- QuantLib::Duration, [448](#)
- QuantLib::DZero, [449](#)
- QuantLib::EarlyExercise, [450](#)
- QuantLib::EarlyExercisePathPricer, [451](#)
- QuantLib::EEKCurrency, [452](#)
- QuantLib::EndCriteria, [453](#)
 - checkMaxIterations, [454](#)
 - checkStationaryFunctionAccuracy, [454](#)
 - checkStationaryFunctionValue, [454](#)
 - checkStationaryPoint, [454](#)
 - checkZeroGradientNorm, [454](#)
 - operator(), [454](#)
- QuantLib::EqualJumpsBinomialTree, [455](#)
- QuantLib::EqualProbabilitiesBinomialTree, [456](#)
- QuantLib::Error, [457](#)
 - ~Error, [457](#)
 - Error, [457](#)
- QuantLib::ErrorFunction, [458](#)
- QuantLib::ESPCurrency, [459](#)
- QuantLib::EulerDiscretization, [460](#)
 - covariance, [461](#)
 - diffusion, [460](#)
 - drift, [460](#)
 - variance, [461](#)
- QuantLib::EURCurrency, [462](#)
- QuantLib::Euribor, [463](#)
- QuantLib::Euribor10M, [464](#)
- QuantLib::Euribor11M, [465](#)
- QuantLib::Euribor1M, [466](#)
- QuantLib::Euribor1Y, [467](#)
- QuantLib::Euribor2M, [468](#)
- QuantLib::Euribor2W, [469](#)
- QuantLib::Euribor365, [470](#)
- QuantLib::Euribor365_10M, [471](#)
- QuantLib::Euribor365_11M, [472](#)
- QuantLib::Euribor365_1M, [473](#)
- QuantLib::Euribor365_1Y, [474](#)
- QuantLib::Euribor365_2M, [475](#)
- QuantLib::Euribor365_2W, [476](#)
- QuantLib::Euribor365_3M, [477](#)
- QuantLib::Euribor365_3W, [478](#)
- QuantLib::Euribor365_4M, [479](#)
- QuantLib::Euribor365_5M, [480](#)
- QuantLib::Euribor365_6M, [481](#)
- QuantLib::Euribor365_7M, [482](#)
- QuantLib::Euribor365_8M, [483](#)
- QuantLib::Euribor365_9M, [484](#)
- QuantLib::Euribor365_SW, [485](#)
- QuantLib::Euribor3M, [486](#)
- QuantLib::Euribor3W, [487](#)
- QuantLib::Euribor4M, [488](#)
- QuantLib::Euribor5M, [489](#)
- QuantLib::Euribor6M, [490](#)
- QuantLib::Euribor7M, [491](#)
- QuantLib::Euribor8M, [492](#)
- QuantLib::Euribor9M, [493](#)
- QuantLib::EuriborSW, [494](#)
- QuantLib::EuriborSwapFixA10Y, [495](#)
- QuantLib::EuriborSwapFixA12Y, [496](#)
- QuantLib::EuriborSwapFixA15Y, [497](#)
- QuantLib::EuriborSwapFixA1Y, [498](#)
- QuantLib::EuriborSwapFixA20Y, [499](#)
- QuantLib::EuriborSwapFixA25Y, [500](#)
- QuantLib::EuriborSwapFixA2Y, [501](#)
- QuantLib::EuriborSwapFixA30Y, [502](#)
- QuantLib::EuriborSwapFixA3Y, [503](#)
- QuantLib::EuriborSwapFixA4Y, [504](#)
- QuantLib::EuriborSwapFixA5Y, [505](#)
- QuantLib::EuriborSwapFixA6Y, [506](#)
- QuantLib::EuriborSwapFixA7Y, [507](#)
- QuantLib::EuriborSwapFixA8Y, [508](#)
- QuantLib::EuriborSwapFixA9Y, [509](#)
- QuantLib::EuriborSwapFixAvs3M, [510](#)
- QuantLib::EuriborSwapFixAvs6M, [511](#)
- QuantLib::EuriborSwapFixB10Y, [512](#)
- QuantLib::EuriborSwapFixB12Y, [513](#)
- QuantLib::EuriborSwapFixB15Y, [514](#)
- QuantLib::EuriborSwapFixB1Y, [515](#)
- QuantLib::EuriborSwapFixB20Y, [516](#)

- QuantLib::EuriborSwapFixB25Y, [517](#)
- QuantLib::EuriborSwapFixB2Y, [518](#)
- QuantLib::EuriborSwapFixB30Y, [519](#)
- QuantLib::EuriborSwapFixB3Y, [520](#)
- QuantLib::EuriborSwapFixB4Y, [521](#)
- QuantLib::EuriborSwapFixB5Y, [522](#)
- QuantLib::EuriborSwapFixB6Y, [523](#)
- QuantLib::EuriborSwapFixB7Y, [524](#)
- QuantLib::EuriborSwapFixB8Y, [525](#)
- QuantLib::EuriborSwapFixB9Y, [526](#)
- QuantLib::EuriborSwapFixBvs3M, [527](#)
- QuantLib::EuriborSwapFixBvs6M, [528](#)
- QuantLib::EuriborSwapFixIFR10Y, [529](#)
- QuantLib::EuriborSwapFixIFR12Y, [530](#)
- QuantLib::EuriborSwapFixIFR15Y, [531](#)
- QuantLib::EuriborSwapFixIFR1Y, [532](#)
- QuantLib::EuriborSwapFixIFR20Y, [533](#)
- QuantLib::EuriborSwapFixIFR25Y, [534](#)
- QuantLib::EuriborSwapFixIFR2Y, [535](#)
- QuantLib::EuriborSwapFixIFR30Y, [536](#)
- QuantLib::EuriborSwapFixIFR3Y, [537](#)
- QuantLib::EuriborSwapFixIFR4Y, [538](#)
- QuantLib::EuriborSwapFixIFR5Y, [539](#)
- QuantLib::EuriborSwapFixIFR6Y, [540](#)
- QuantLib::EuriborSwapFixIFR7Y, [541](#)
- QuantLib::EuriborSwapFixIFR8Y, [542](#)
- QuantLib::EuriborSwapFixIFR9Y, [543](#)
- QuantLib::EuriborSwapFixIFRvs3M, [544](#)
- QuantLib::EuriborSwapFixIFRvs6M, [545](#)
- QuantLib::EURLibor, [546](#)
- QuantLib::EURLibor10M, [547](#)
- QuantLib::EURLibor11M, [548](#)
- QuantLib::EURLibor1M, [549](#)
- QuantLib::EURLibor1Y, [550](#)
- QuantLib::EURLibor2M, [551](#)
- QuantLib::EURLibor2W, [552](#)
- QuantLib::EURLibor3M, [553](#)
- QuantLib::EURLibor4M, [554](#)
- QuantLib::EURLibor5M, [555](#)
- QuantLib::EURLibor6M, [556](#)
- QuantLib::EURLibor7M, [557](#)
- QuantLib::EURLibor8M, [558](#)
- QuantLib::EURLibor9M, [559](#)
- QuantLib::EURLiborSW, [560](#)
- QuantLib::EurliborSwapFixA10Y, [561](#)
- QuantLib::EurliborSwapFixA12Y, [562](#)
- QuantLib::EurliborSwapFixA15Y, [563](#)
- QuantLib::EurliborSwapFixA1Y, [564](#)
- QuantLib::EurliborSwapFixA20Y, [565](#)
- QuantLib::EurliborSwapFixA25Y, [566](#)
- QuantLib::EurliborSwapFixA2Y, [567](#)
- QuantLib::EurliborSwapFixA30Y, [568](#)
- QuantLib::EurliborSwapFixA3Y, [569](#)
- QuantLib::EurliborSwapFixA4Y, [570](#)
- QuantLib::EurliborSwapFixA5Y, [571](#)
- QuantLib::EurliborSwapFixA6Y, [572](#)
- QuantLib::EurliborSwapFixA7Y, [573](#)
- QuantLib::EurliborSwapFixA8Y, [574](#)
- QuantLib::EurliborSwapFixA9Y, [575](#)
- QuantLib::EurliborSwapFixAvs3M, [576](#)
- QuantLib::EurliborSwapFixAvs6M, [577](#)
- QuantLib::EurliborSwapFixB10Y, [578](#)
- QuantLib::EurliborSwapFixB12Y, [579](#)
- QuantLib::EurliborSwapFixB15Y, [580](#)
- QuantLib::EurliborSwapFixB1Y, [581](#)
- QuantLib::EurliborSwapFixB20Y, [582](#)
- QuantLib::EurliborSwapFixB25Y, [583](#)
- QuantLib::EurliborSwapFixB2Y, [584](#)
- QuantLib::EurliborSwapFixB30Y, [585](#)
- QuantLib::EurliborSwapFixB3Y, [586](#)
- QuantLib::EurliborSwapFixB4Y, [587](#)
- QuantLib::EurliborSwapFixB5Y, [588](#)
- QuantLib::EurliborSwapFixB6Y, [589](#)
- QuantLib::EurliborSwapFixB7Y, [590](#)
- QuantLib::EurliborSwapFixB8Y, [591](#)
- QuantLib::EurliborSwapFixB9Y, [592](#)
- QuantLib::EurliborSwapFixBvs3M, [593](#)
- QuantLib::EurliborSwapFixBvs6M, [594](#)
- QuantLib::EurliborSwapFixIFR10Y, [595](#)
- QuantLib::EurliborSwapFixIFR12Y, [596](#)
- QuantLib::EurliborSwapFixIFR15Y, [597](#)
- QuantLib::EurliborSwapFixIFR1Y, [598](#)
- QuantLib::EurliborSwapFixIFR20Y, [599](#)
- QuantLib::EurliborSwapFixIFR25Y, [600](#)
- QuantLib::EurliborSwapFixIFR2Y, [601](#)
- QuantLib::EurliborSwapFixIFR30Y, [602](#)
- QuantLib::EurliborSwapFixIFR3Y, [603](#)
- QuantLib::EurliborSwapFixIFR4Y, [604](#)
- QuantLib::EurliborSwapFixIFR5Y, [605](#)
- QuantLib::EurliborSwapFixIFR6Y, [606](#)
- QuantLib::EurliborSwapFixIFR7Y, [607](#)
- QuantLib::EurliborSwapFixIFR8Y, [608](#)
- QuantLib::EurliborSwapFixIFR9Y, [609](#)
- QuantLib::EurliborSwapFixIFRvs3M, [610](#)
- QuantLib::EurliborSwapFixIFRvs6M, [611](#)
- QuantLib::EurodollarFuturesImpliedStdDevQuote, [612](#)
- performCalculations, [612](#)
- QuantLib::EuropeanExercise, [613](#)
- QuantLib::EuropeanOption, [614](#)
- QuantLib::Event, [615](#)
- hasOccurred, [615](#)
- QuantLib::EvolutionDescription, [617](#)
- QuantLib::ExchangeRate, [618](#)
- Derived, [619](#)
- Direct, [619](#)
- ExchangeRate, [619](#)
- Type, [619](#)

- QuantLib::ExchangeRateManager, 620
 - add, 620
 - lookup, 620
- QuantLib::Exercise, 622
- QuantLib::ExplicitEuler, 623
- QuantLib::ExtendedCoxIngersollRoss, 625
- QuantLib::ExtendedCoxIngersollRoss::Dynamics, 627
- QuantLib::ExtendedCoxIngersollRoss::FittingParameters, 628
- QuantLib::ExtendedDiscountCurve, 629
 - compoundForwardImpl, 630
 - update, 629
 - zeroYieldImpl, 630
- QuantLib::Extrapolator, 631
- QuantLib::Factorial, 632
- QuantLib::FalsePosition, 633
- QuantLib::FaureRsg, 634
- QuantLib::FDBermudanEngine, 635
- QuantLib::FDDividendEngineMerton73, 636
- QuantLib::FDDividendEngineShiftScale, 637
- QuantLib::FDEuropeanEngine, 638
- QuantLib::FDStepConditionEngine, 639
- QuantLib::FIMCurrency, 640
- QuantLib::FiniteDifferenceModel, 641
 - rollback, 641
- QuantLib::Finland, 642
- QuantLib::FixedCouponBondHelper, 643
 - setTermStructure, 644
- QuantLib::FixedDividend, 645
 - amount, 645
- QuantLib::FixedRateBond, 647
- QuantLib::FixedRateBondForward, 648
 - FixedRateBondForward, 649
 - performCalculations, 650
 - spotIncome, 650
- QuantLib::FixedRateCoupon, 651
 - amount, 652
- QuantLib::FlatForward, 653
 - update, 654
- QuantLib::FloatingRateBond, 655
- QuantLib::FloatingRateCoupon, 656
 - amount, 658
 - update, 658
- QuantLib::FloatingRateCouponPricer, 659
 - update, 659
- QuantLib::FloatingTypePayoff, 660
 - name, 660
- QuantLib::Floor, 661
- QuantLib::FloorTruncation, 662
- QuantLib::Forward, 663
 - forwardValue, 665
 - impliedYield, 665
 - incomeDiscountCurve_, 665
 - performCalculations, 665
 - underlyingIncome_, 665
 - underlyingSpotValue_, 665
 - valueDate_, 665
- QuantLib::ForwardEngine, 666
- QuantLib::ForwardFlat, 667
- QuantLib::ForwardFlatInterpolation, 668
 - ForwardFlatInterpolation, 668
- QuantLib::ForwardMeasureProcess, 669
- QuantLib::ForwardMeasureProcess1D, 670
- QuantLib::ForwardOptionArguments, 671
- QuantLib::ForwardPerformanceEngine, 672
- QuantLib::ForwardRate, 673
- QuantLib::ForwardRateAgreement, 674
 - isExpired, 675
 - performCalculations, 676
 - settlementDate, 675
 - spotIncome, 676
 - spotValue, 676
- QuantLib::ForwardRateStructure, 677
 - discountImpl, 678
 - zeroYieldImpl, 678
- QuantLib::ForwardSpreadedTermStructure, 679
- QuantLib::ForwardTypePayoff, 681
 - name, 681
- QuantLib::ForwardValueQuote, 682
 - update, 682
- QuantLib::ForwardVanillaOption, 683
 - fetchResults, 684
- QuantLib::FractionalDividend, 685
 - amount, 686
- QuantLib::FraRateHelper, 687
 - setTermStructure, 687
- QuantLib::FRFCurrency, 689
- QuantLib::FuturesConvAdjustmentQuote, 690
 - update, 691
- QuantLib::FuturesRateHelper, 692
- QuantLib::G2, 693
- QuantLib::G2::FittingParameter, 695
- QuantLib::G2ForwardProcess, 696
 - covariance, 697
 - expectation, 697
 - stdDeviation, 697
- QuantLib::G2Process, 698
 - covariance, 699
 - expectation, 699
 - stdDeviation, 699
- QuantLib::G2SwaptionEngine, 700
- QuantLib::GammaFunction, 701
- QuantLib::GapPayoff, 702
 - name, 703
- QuantLib::Garch11, 704
- QuantLib::GarmanKlassAbstract, 705

- QuantLib::GarmanKohlagenProcess, [706](#)
- QuantLib::GaussChebyshev2thIntegration, [707](#)
- QuantLib::GaussChebyshev2thPolynomial, [708](#)
- QuantLib::GaussChebyshevIntegration, [709](#)
- QuantLib::GaussChebyshevPolynomial, [710](#)
- QuantLib::GaussGegenbauerIntegration, [711](#)
- QuantLib::GaussGegenbauerPolynomial, [712](#)
- QuantLib::GaussHermiteIntegration, [713](#)
- QuantLib::GaussHermitePolynomial, [714](#)
- QuantLib::GaussHyperbolicIntegration, [715](#)
- QuantLib::GaussHyperbolicPolynomial, [716](#)
- QuantLib::GaussianOrthogonalPolynomial, [717](#)
- QuantLib::GaussianQuadrature, [718](#)
- QuantLib::GaussJacobiIntegration, [719](#)
- QuantLib::GaussJacobiPolynomial, [720](#)
- QuantLib::GaussKronrodAdaptive, [721](#)
- QuantLib::GaussKronrodNonAdaptive, [722](#)
- QuantLib::GaussLaguerreIntegration, [723](#)
- QuantLib::GaussLaguerrePolynomial, [724](#)
- QuantLib::GaussLegendreIntegration, [725](#)
- QuantLib::GaussLegendrePolynomial, [726](#)
- QuantLib::GBPCurrency, [727](#)
- QuantLib::GBPLibor, [728](#)
- QuantLib::GeneralizedBlackScholesProcess, [729](#)
 - apply, [730](#)
 - diffusion, [730](#)
 - drift, [730](#)
 - time, [730](#)
 - update, [730](#)
- QuantLib::GeneralStatistics, [731](#)
 - add, [733](#)
 - errorEstimate, [732](#)
 - expectationValue, [733](#)
 - kurtosis, [732](#)
 - max, [733](#)
 - mean, [732](#)
 - min, [733](#)
 - percentile, [733](#)
 - skewness, [732](#)
 - standardDeviation, [732](#)
 - topPercentile, [733](#)
 - variance, [732](#)
- QuantLib::GenericEngine, [735](#)
- QuantLib::GenericGaussianStatistics, [736](#)
 - gaussianDownsideDeviation, [737](#)
 - gaussianDownsideVariance, [736](#)
 - gaussianExpectedShortfall, [737](#)
 - gaussianPercentile, [737](#)
 - gaussianPotentialUpside, [737](#)
 - gaussianRegret, [737](#)
 - gaussianTopPercentile, [737](#)
 - gaussianValueAtRisk, [737](#)
- QuantLib::GenericModelEngine, [739](#)
 - update, [739](#)
- QuantLib::GenericRiskStatistics, [740](#)
 - averageShortfall, [742](#)
 - downsideDeviation, [741](#)
 - downsideVariance, [741](#)
 - expectedShortfall, [741](#)
 - potentialUpside, [741](#)
 - regret, [741](#)
 - semiDeviation, [741](#)
 - semiVariance, [741](#)
 - shortfall, [742](#)
 - valueAtRisk, [741](#)
- QuantLib::GenericSequenceStatistics, [743](#)
- QuantLib::GeometricBrownianMotionProcess, [745](#)
- QuantLib::Germany, [746](#)
 - Eurex, [748](#)
 - FrankfurtStockExchange, [748](#)
 - Market, [748](#)
 - Settlement, [748](#)
 - Xetra, [748](#)
- QuantLib::GRDCurrency, [749](#)
- QuantLib::Greeks, [750](#)
- QuantLib::HaltonRsg, [751](#)
- QuantLib::Handle, [752](#)
 - Handle, [753](#)
- QuantLib::HestonModel, [754](#)
- QuantLib::HestonModelHelper, [755](#)
- QuantLib::HestonProcess, [756](#)
 - apply, [757](#)
 - evolve, [757](#)
 - time, [757](#)
 - update, [757](#)
- QuantLib::HKDCurrency, [758](#)
- QuantLib::HongKong, [759](#)
 - HKEEx, [760](#)
 - Market, [760](#)
- QuantLib::HUFCurrency, [761](#)
- QuantLib::HullWhite, [762](#)
 - convexityBias, [763](#)
- QuantLib::HullWhite::Dynamics, [764](#)
- QuantLib::HullWhite::FittingParameter, [765](#)
- QuantLib::HullWhiteForwardProcess, [766](#)
 - expectation, [767](#)
 - stdDeviation, [767](#)
 - variance, [767](#)
- QuantLib::HullWhiteProcess, [768](#)
 - expectation, [769](#)
 - stdDeviation, [769](#)
 - variance, [769](#)
- QuantLib::Hungary, [770](#)
- QuantLib::IborCoupon, [771](#)

- QuantLib::IborCouponPricer, 772
- QuantLib::IborIndex, 773
- QuantLib::Iceland, 774
 - ICEX, 775
 - Market, 775
- QuantLib::IEPCurrency, 776
- QuantLib::ILSCurrency, 777
- QuantLib::IMM, 778
 - code, 778
 - date, 778
 - nextCode, 779
 - nextDate, 779
- QuantLib::ImplicitEuler, 780
- QuantLib::ImpliedStdDevQuote, 781
 - performCalculations, 781
- QuantLib::ImpliedTermStructure, 782
- QuantLib::ImpliedVolTermStructure, 784
- QuantLib::IncrementalStatistics, 786
 - add, 788
 - addSequence, 788
 - downsideDeviation, 788
 - downsideVariance, 788
 - errorEstimate, 787
 - kurtosis, 787
 - max, 788
 - mean, 787
 - min, 788
 - skewness, 787
 - standardDeviation, 787
 - variance, 787
- QuantLib::Index, 789
 - addFixing, 790
 - addFixings, 790
 - fixing, 790
 - name, 790
- QuantLib::IndexManager, 791
- QuantLib::India, 792
 - Market, 793
 - NSE, 793
- QuantLib::Indonesia, 794
 - BEJ, 795
 - JSX, 795
 - Market, 795
- QuantLib::INRCurrency, 796
- QuantLib::Instrument, 797
 - calculate, 798
 - fetchResults, 798
 - performCalculations, 799
 - setPricingEngine, 798
 - setupArguments, 798
 - setupExpired, 799
- QuantLib::IntegralEngine, 800
- QuantLib::InterestRate, 801
 - compoundFactor, 802
 - discountFactor, 802
 - equivalentRate, 803
 - impliedRate, 803
 - operator<<, 803
- QuantLib::InterestRateIndex, 804
 - fixing, 805
 - name, 805
 - update, 805
- QuantLib::InterpolatedDiscountCurve, 806
- QuantLib::InterpolatedForwardCurve, 808
 - zeroYieldImpl, 809
- QuantLib::InterpolatedZeroCurve, 810
- QuantLib::Interpolation, 812
- QuantLib::Interpolation2D, 814
- QuantLib::Interpolation2D::Impl, 816
- QuantLib::Interpolation2D::templateImpl, 817
- QuantLib::Interpolation::Impl, 818
- QuantLib::Interpolation::templateImpl, 819
- QuantLib::IntervalPrice, 820
- QuantLib::InverseCumulativeNormal, 821
- QuantLib::InverseCumulativePoisson, 822
- QuantLib::InverseCumulativeRng, 823
- QuantLib::InverseCumulativeRsg, 824
- QuantLib::IQDCurrency, 825
- QuantLib::IRRCurrency, 826
- QuantLib::ISKCurrency, 827
- QuantLib::Italy, 828
 - Exchange, 829
 - Market, 829
 - Settlement, 829
- QuantLib::ITLCurrency, 830
- QuantLib::JamshidianSwaptionEngine, 831
- QuantLib::Japan, 832
- QuantLib::JarrowRudd, 833
- QuantLib::Jibar, 834
- QuantLib::JointCalendar, 835
- QuantLib::JPYCurrency, 836
- QuantLib::JPYLibor, 837
- QuantLib::JumpDiffusionEngine, 838
- QuantLib::JuQuadraticApproximationEngine, 839
- QuantLib::KnuthUniformRng, 840
 - KnuthUniformRng, 840
 - next, 840
- QuantLib::KRWCurrency, 841
- QuantLib::KWDCurrency, 842
- QuantLib::Lattice, 843
 - partialRollback, 844
 - rollback, 843
- QuantLib::LatticeShortRateModelEngine, 845
 - update, 845
- QuantLib::LazyObject, 846
 - calculate, 847
 - freeze, 847

- performCalculations, [847](#)
- recalculate, [847](#)
- unfreeze, [847](#)
- update, [846](#)
- QuantLib::LeastSquareFunction, [848](#)
- QuantLib::LeastSquareProblem, [849](#)
 - targetValueAndGradient, [849](#)
- QuantLib::LecuyerUniformRng, [850](#)
 - LecuyerUniformRng, [850](#)
 - next, [850](#)
- QuantLib::LeisenReimer, [851](#)
- QuantLib::LevenbergMarquardt, [852](#)
- QuantLib::LexicographicalView, [853](#)
- QuantLib::LfmCovarianceParameterization, [855](#)
- QuantLib::LfmCovarianceProxy, [856](#)
- QuantLib::LfmHullWhiteParameterization, [857](#)
- QuantLib::LfmSwaptionEngine, [858](#)
- QuantLib::Libor, [859](#)
- QuantLib::LiborForwardModel, [860](#)
- QuantLib::LiborForwardModelProcess, [862](#)
 - apply, [863](#)
 - covariance, [863](#)
 - evolve, [863](#)
- QuantLib::Linear, [864](#)
- QuantLib::LinearInterpolation, [865](#)
 - LinearInterpolation, [865](#)
- QuantLib::LinearLeastSquaresRegression, [866](#)
- QuantLib::LineSearch, [867](#)
- QuantLib::LmConstWrapperVolatilityModel, [869](#)
- QuantLib::LmCorrelationModel, [870](#)
- QuantLib::LmExponentialCorrelationModel, [871](#)
- QuantLib::LmExtLinearExponentialVolModel, [872](#)
- QuantLib::LmLinearExponentialCorrelationModel, [873](#)
- QuantLib::LmLinearExponentialVolatilityModel, [874](#)
- QuantLib::LMMCurveState, [875](#)
- QuantLib::LMMDriftCalculator, [876](#)
 - computePlain, [876](#)
 - computeReduced, [876](#)
- QuantLib::LMMNormalDriftCalculator, [877](#)
 - computePlain, [877](#)
 - computeReduced, [877](#)
- QuantLib::LmVolatilityModel, [878](#)
- QuantLib::LocalConstantVol, [879](#)
- QuantLib::LocalVolCurve, [880](#)
 - localVolImpl, [881](#)
- QuantLib::LocalVolSurface, [882](#)
- QuantLib::LocalVolTermStructure, [884](#)
 - LocalVolTermStructure, [885](#)
- QuantLib::LogLinear, [886](#)
- QuantLib::LogLinearInterpolation, [887](#)
 - LogLinearInterpolation, [887](#)
- QuantLib::LogNormalCmSwapRatePc, [888](#)
- QuantLib::LogNormalCotSwapRatePc, [889](#)
- QuantLib::LogNormalFwdRateEuler, [890](#)
- QuantLib::LogNormalFwdRateEulerConstrained, [891](#)
- QuantLib::LogNormalFwdRateIpc, [892](#)
- QuantLib::LogNormalFwdRatePc, [893](#)
- QuantLib::LongstaffSchwartzPathPricer, [894](#)
- QuantLib::LTLCurrency, [895](#)
- QuantLib::LUFCurrency, [896](#)
- QuantLib::LVLCurrency, [897](#)
- QuantLib::MakeCapFloor, [898](#)
- QuantLib::MakeCms, [899](#)
- QuantLib::MakeMCAmericanEngine, [900](#)
- QuantLib::MakeMCDigitalEngine, [901](#)
- QuantLib::MakeMCEuropeanEngine, [902](#)
- QuantLib::MakeMCEuropeanHestonEngine, [903](#)
- QuantLib::MakeMCHullWhiteCapFloorEngine, [904](#)
- QuantLib::MakeMCVarianceSwapEngine, [905](#)
- QuantLib::MakeSchedule, [906](#)
- QuantLib::MakeVanillaSwap, [907](#)
- QuantLib::MarketModel, [908](#)
- QuantLib::MarketModelCapFloorEngine, [909](#)
 - update, [909](#)
- QuantLib::MarketModelComposite, [910](#)
- QuantLib::MarketModelEvolver, [912](#)
- QuantLib::MarketModelFactory, [913](#)
- QuantLib::MarketModelMultiProduct, [914](#)
- QuantLib::Matrix, [915](#)
 - CholeskyDecomposition, [918](#)
 - inverse, [918](#)
 - operator *, [918](#)
 - operator +, [918](#)
 - operator +=, [917](#)
 - operator -, [918](#)
 - operator /, [918](#)
 - operator <<, [918](#)
 - outerProduct, [918](#)
 - pseudoSqrt, [918](#)
 - rankReducedSqrt, [919](#)
 - swap, [918](#)
 - transpose, [918](#)
- QuantLib::MCAmericanBasketEngine, [920](#)
- QuantLib::MCAmericanEngine, [921](#)
- QuantLib::MCBarrierEngine, [922](#)
- QuantLib::MCBasketEngine, [924](#)
- QuantLib::McCliquetOption, [926](#)
- QuantLib::MCDigitalEngine, [927](#)

- QuantLib::MCDiscreteArithmeticAPEngine, 928
- QuantLib::McDiscreteArithmeticASO, 930
- QuantLib::MCDiscreteAveragingAsianEngine, 931
- QuantLib::MCDiscreteGeometricAPEngine, 933
- QuantLib::MCEuropeanEngine, 934
- QuantLib::MCEuropeanHestonEngine, 935
- QuantLib::McEverest, 936
- QuantLib::McHimalaya, 937
- QuantLib::MCHullWhiteCapFloorEngine, 938
 - update, 939
- QuantLib::MCLongstaffSchwartzEngine, 940
- QuantLib::McMaxBasket, 942
- QuantLib::McPagoda, 943
- QuantLib::McPerformanceOption, 944
- QuantLib::McPricer, 945
- QuantLib::McSimulation, 946
- QuantLib::MCVanillaEngine, 948
- QuantLib::MCVarianceSwapEngine, 949
- QuantLib::MersenneTwisterUniformRng, 951
 - MersenneTwisterUniformRng, 951
 - next, 951
- QuantLib::Merton76Process, 952
 - apply, 953
 - time, 953
- QuantLib::Mexico, 954
 - BMV, 954
 - Market, 954
- QuantLib::MixedScheme, 955
- QuantLib::Money, 957
 - AutomatedConversion, 958
 - BaseCurrencyConversion, 958
 - close, 959
 - close_enough, 959
 - ConversionType, 958
 - NoConversion, 958
 - operator *, 959
 - operator!=, 959
 - operator+, 959
 - operator-, 959
 - operator/, 959
 - operator<, 959
 - operator<<, 959
 - operator<=, 959
 - operator==, 959
 - operator>, 959
 - operator>=, 959
- QuantLib::MonotonicCubicSpline, 960
 - MonotonicCubicSpline, 960
- QuantLib::MonteCarloModel, 961
- QuantLib::MoreGreeks, 962
- QuantLib::MoroInverseCumulativeNormal, 963
- QuantLib::MTBrownianGenerator, 964
- QuantLib::MtlCurrency, 965
- QuantLib::MultiAssetOption, 966
 - fetchResults, 967
 - setupExpired, 967
- QuantLib::MultiAssetOption::arguments, 968
- QuantLib::MultiAssetOption::results, 969
- QuantLib::MultiCubicSpline, 970
- QuantLib::MultiPath, 971
- QuantLib::MultiPathGenerator, 972
- QuantLib::MultiProductComposite, 973
- QuantLib::MultiProductMultiStep, 974
- QuantLib::MultiProductOneStep, 975
- QuantLib::MultiVariate, 976
- QuantLib::MXNCurrency, 977
- QuantLib::NaturalCubicSpline, 978
 - NaturalCubicSpline, 978
- QuantLib::NaturalMonotonicCubicSpline, 979
 - NaturalMonotonicCubicSpline, 979
- QuantLib::NeumannBC, 980
 - applyAfterApplying, 980
 - applyAfterSolving, 981
 - applyBeforeApplying, 980
 - applyBeforeSolving, 980
 - setTime, 981
- QuantLib::Newton, 982
- QuantLib::NewtonSafe, 983
- QuantLib::NewZealand, 984
- QuantLib::NLGCurrency, 985
- QuantLib::NoConstraint, 986
- QuantLib::NOKCurrency, 987
- QuantLib::NonLinearLeastSquare, 988
- QuantLib::NormalDistribution, 989
- QuantLib::NormalFwdRatePc, 990
- QuantLib::Norway, 991
- QuantLib::NPRCurrency, 992
- QuantLib::Null, 993
- QuantLib::NullCalendar, 994
- QuantLib::NullCondition, 995
- QuantLib::NullParameter, 996
- QuantLib::NZDCurrency, 997
- QuantLib::NZDLibor, 998
- QuantLib::Observable, 999
 - notifyObservers, 999
 - operator=, 999
- QuantLib::ObservableValue, 1000
- QuantLib::Observer, 1001
 - update, 1001
- QuantLib::OneAssetOption, 1002
 - fetchResults, 1003
 - impliedVolatility, 1003
 - setupExpired, 1004

- QuantLib::OneAssetOption::arguments, 1005
- QuantLib::OneAssetOption::results, 1006
- QuantLib::OneAssetStrikedOption, 1007
 - fetchResults, 1008
 - setupExpired, 1008
- QuantLib::OneDayCounter, 1009
- QuantLib::OneFactorAffineModel, 1010
- QuantLib::OneFactorModel, 1011
- QuantLib::OneFactorModel::ShortRateDynamics, 1012
- QuantLib::OneFactorModel::ShortRateTree, 1013
- QuantLib::OperatorFactory, 1014
- QuantLib::OptimizationMethod, 1015
- QuantLib::Option, 1016
 - operator<<, 1017
- QuantLib::Option::arguments, 1018
- QuantLib::OrnsteinUhlenbeckProcess, 1019
 - expectation, 1019
 - stdDeviation, 1020
 - variance, 1020
- QuantLib::Parameter, 1021
- QuantLib::Parameter::Impl, 1022
- QuantLib::Path, 1023
- QuantLib::PathGenerator, 1024
- QuantLib::PathPricer, 1025
- QuantLib::Payoff, 1026
 - name, 1026
- QuantLib::PercentageStrikePayoff, 1027
 - name, 1027
- QuantLib::Period, 1028
 - operator *, 1029
 - operator!=, 1029
 - operator-, 1029
 - operator<, 1029
 - operator<<, 1029
 - operator<=, 1029
 - operator==, 1029
 - operator>, 1029
 - operator>=, 1029
- QuantLib::PiecewiseConstantParameter, 1030
- QuantLib::PiecewiseYieldCurve, 1031
 - update, 1032
- QuantLib::PiecewiseZeroSpreadedTermStructure, 1033
 - update, 1034
- QuantLib::PKRCurrency, 1035
- QuantLib::PlainVanillaPayoff, 1036
 - name, 1036
- QuantLib::PLNCurrency, 1037
- QuantLib::PoissonDistribution, 1038
- QuantLib::Poland, 1039
- QuantLib::PositiveConstraint, 1040
- QuantLib::PricingEngine, 1041
- QuantLib::PrimeNumbers, 1042
- QuantLib::Problem, 1043
 - reset, 1044
- QuantLib::ProjectedCostFunction, 1045
- QuantLib::PTECurrency, 1046
- QuantLib::QuantoEngine, 1047
 - underlyingArgs, 1048
- QuantLib::QuantoForwardVanillaOption, 1049
- QuantLib::QuantoOptionArguments, 1050
- QuantLib::QuantoOptionResults, 1051
- QuantLib::QuantoTermStructure, 1052
- QuantLib::QuantoVanillaOption, 1054
 - fetchResults, 1055
 - setupExpired, 1055
- QuantLib::Quote, 1056
- QuantLib::RandomizedLDS, 1057
 - nextRandomizer, 1058
- QuantLib::RandomSequenceGenerator, 1059
- QuantLib::RatchetMaxPayoff, 1060
 - name, 1060
- QuantLib::RatchetMinPayoff, 1061
 - name, 1061
- QuantLib::RatchetPayoff, 1062
 - name, 1062
- QuantLib::RateHelper, 1063
 - earliestDate, 1064
 - latestDate, 1064
 - setTermStructure, 1064
 - update, 1064
- QuantLib::RelativeDateRateHelper, 1065
 - update, 1065
- QuantLib::RelinkableHandle, 1066
 - linkTo, 1066
 - RelinkableHandle, 1066
- QuantLib::ReplicatingVarianceSwapEngine, 1067
- QuantLib::Ridder, 1068
- QuantLib::ROLCurrency, 1069
- QuantLib::RONCurrency, 1070
- QuantLib::Rounding, 1071
 - Ceiling, 1072
 - Closest, 1072
 - Down, 1072
 - Floor, 1072
 - None, 1072
 - Rounding, 1072
 - Type, 1071
 - Up, 1072
- QuantLib::SABR, 1073
- QuantLib::SABRInterpolation, 1074
- QuantLib::SalvagingAlgorithm, 1075
- QuantLib::Sample, 1076
- QuantLib::SampledCurve, 1077
 - firstDerivativeAtCenter, 1078

- secondDerivativeAtCenter, 1078
- valueAtCenter, 1078
- QuantLib::SARCurrency, 1079
- QuantLib::SaudiArabia, 1080
 - Market, 1080
 - Tadawul, 1080
- QuantLib::Schedule, 1081
- QuantLib::Secant, 1082
- QuantLib::SeedGenerator, 1083
- QuantLib::SegmentIntegral, 1084
- QuantLib::SEKCurrency, 1085
- QuantLib::Settings, 1086
 - evaluationDate, 1086
- QuantLib::Settlement, 1088
- QuantLib::SGDCurrency, 1089
- QuantLib::ShortRateModel, 1090
- QuantLib::ShoutCondition, 1091
- QuantLib::SimpleCashFlow, 1092
 - amount, 1092
- QuantLib::SimpleDayCounter, 1093
- QuantLib::SimpleLocalEstimator, 1094
- QuantLib::SimpleQuote, 1095
- QuantLib::Simplex, 1096
 - Simplex, 1096
- QuantLib::SimpsonIntegral, 1097
- QuantLib::Singapore, 1098
 - Market, 1099
 - SGX, 1099
- QuantLib::SingleAssetOption, 1100
 - impliedVolatility, 1101
- QuantLib::SingleProductComposite, 1102
- QuantLib::Singleton, 1103
- QuantLib::SingleVariate, 1104
- QuantLib::SITCurrency, 1105
- QuantLib::SKKCurrency, 1106
- QuantLib::Slovakia, 1107
 - BSSE, 1108
 - Market, 1108
- QuantLib::SmileSection, 1109
- QuantLib::SMMDriftCalculator, 1110
- QuantLib::SobolBrownianGenerator, 1111
 - Diagonal, 1111
 - Factors, 1111
 - Ordering, 1111
 - Steps, 1111
- QuantLib::SobolRsg, 1112
 - skipTo, 1113
 - SobolRsg, 1113
- QuantLib::SoftCallability, 1114
- QuantLib::Solver1D, 1115
 - setMaxEvaluations, 1116
 - solve, 1116
- QuantLib::SouthAfrica, 1117
- QuantLib::SouthKorea, 1118
 - KRX, 1119
 - Market, 1119
- QuantLib::SquareRootProcess, 1120
- QuantLib::StatsHolder, 1121
- QuantLib::SteepestDescent, 1122
- QuantLib::step_iterator, 1123
 - make_step_iterator, 1123
- QuantLib::StepCondition, 1124
- QuantLib::StepConditionSet, 1125
- QuantLib::StickyMaxPayoff, 1126
 - name, 1126
- QuantLib::StickyMinPayoff, 1127
 - name, 1127
- QuantLib::StickyPayoff, 1128
 - name, 1128
- QuantLib::StochasticProcess, 1129
 - apply, 1131
 - covariance, 1130
 - evolve, 1130
 - expectation, 1130
 - stdDeviation, 1130
 - time, 1131
 - update, 1131
- QuantLib::StochasticProcess1D, 1132
 - apply, 1133
 - evolve, 1133
 - expectation, 1133
 - stdDeviation, 1133
 - variance, 1133
- QuantLib::StochasticProcess1D::discretization, 1134
- QuantLib::StochasticProcess::discretization, 1135
- QuantLib::StochasticProcessArray, 1136
 - apply, 1137
 - covariance, 1137
 - evolve, 1137
 - expectation, 1137
 - stdDeviation, 1137
 - time, 1137
- QuantLib::Stock, 1139
 - performCalculations, 1139
- QuantLib::StrikedTypePayoff, 1140
- QuantLib::StulzEngine, 1141
- QuantLib::SuperFundPayoff, 1142
 - name, 1142
- QuantLib::SuperSharePayoff, 1143
 - name, 1143
- QuantLib::Surface, 1145
- QuantLib::SVD, 1146
- QuantLib::Swap, 1147
 - performCalculations, 1148
 - setupExpired, 1148
 - Swap, 1148

- QuantLib::SwapIndex, [1149](#)
 - underlyingSwap, [1149](#)
- QuantLib::SwapRateHelper, [1150](#)
 - setTermStructure, [1151](#)
- QuantLib::Swaption, [1152](#)
- QuantLib::Swaption::arguments, [1154](#)
- QuantLib::Swaption::engine, [1155](#)
- QuantLib::SwaptionConstantVolatility, [1156](#)
- QuantLib::SwaptionHelper, [1158](#)
- QuantLib::SwaptionVolatilityCube, [1159](#)
 - update, [1161](#)
- QuantLib::SwaptionVolatilityMatrix, [1162](#)
 - performCalculations, [1164](#)
 - SwaptionVolatilityMatrix, [1163](#)
 - update, [1164](#)
- QuantLib::SwaptionVolatilityStructure, [1165](#)
 - SwaptionVolatilityStructure, [1167](#)
- QuantLib::Sweden, [1168](#)
- QuantLib::Switzerland, [1169](#)
- QuantLib::SymmetricSchurDecomposition, [1170](#)
 - SymmetricSchurDecomposition, [1170](#)
- QuantLib::TabulatedGaussLegendre, [1171](#)
- QuantLib::Taiwan, [1172](#)
 - Market, [1173](#)
 - TSEC, [1173](#)
- QuantLib::TARGET, [1174](#)
- QuantLib::TermStructure, [1175](#)
 - TermStructure, [1176](#)
 - update, [1176](#)
- QuantLib::TermStructureConsistentModel, [1177](#)
- QuantLib::TermStructureFittingParameter, [1178](#)
- QuantLib::THBCurrency, [1179](#)
- QuantLib::Thirty360, [1180](#)
- QuantLib::Tian, [1181](#)
- QuantLib::Tibor, [1182](#)
- QuantLib::TimeBasket, [1183](#)
- QuantLib::TimeGrid, [1184](#)
 - TimeGrid, [1185](#)
- QuantLib::TimeSeries, [1186](#)
 - TimeSeries, [1187](#)
- QuantLib::TqrEigenDecomposition, [1188](#)
- QuantLib::TransformedGrid, [1189](#)
- QuantLib::TrapezoidIntegral, [1190](#)
- QuantLib::Tree, [1191](#)
- QuantLib::TreeCapFloorEngine, [1192](#)
- QuantLib::TreeLattice, [1193](#)
 - partialRollback, [1194](#)
 - rollback, [1194](#)
- QuantLib::TreeLattice1D, [1195](#)
- QuantLib::TreeLattice2D, [1196](#)
- QuantLib::TreeSwaptionEngine, [1197](#)
- QuantLib::TreeVanillaSwapEngine, [1198](#)
- QuantLib::TridiagonalOperator, [1199](#)
- QuantLib::TridiagonalOperator::TimeSetter, [1201](#)
- QuantLib::Trigeorgis, [1202](#)
- QuantLib::TrinomialTree, [1203](#)
- QuantLib::TRLCurrency, [1204](#)
- QuantLib::TRLibor, [1205](#)
- QuantLib::TRYCurrency, [1206](#)
- QuantLib::TsiveriotisFernandesLattice, [1207](#)
 - partialRollback, [1207](#)
 - rollback, [1207](#)
- QuantLib::TTDCurrency, [1209](#)
- QuantLib::Turkey, [1210](#)
- QuantLib::TWDCurrency, [1211](#)
- QuantLib::TwoFactorModel, [1212](#)
- QuantLib::TwoFactorModel::ShortRateDynamics, [1213](#)
- QuantLib::TwoFactorModel::ShortRateTree, [1214](#)
- QuantLib::TypePayoff, [1215](#)
- QuantLib::Ukraine, [1216](#)
 - Market, [1217](#)
 - USE, [1217](#)
- QuantLib::UnitedKingdom, [1218](#)
 - Exchange, [1219](#)
 - Market, [1219](#)
 - Settlement, [1219](#)
- QuantLib::UnitedStates, [1220](#)
 - GovernmentBond, [1222](#)
 - Market, [1222](#)
 - NERC, [1222](#)
 - NYSE, [1222](#)
 - Settlement, [1222](#)
- QuantLib::UpperBoundEngine, [1223](#)
- QuantLib::UpRounding, [1224](#)
- QuantLib::USDCurrency, [1225](#)
- QuantLib::USDLibor, [1226](#)
- QuantLib::VanillaOption, [1227](#)
- QuantLib::VanillaOption::engine, [1228](#)
- QuantLib::VanillaSwap::arguments, [1229](#)
- QuantLib::VanillaSwap::results, [1230](#)
- QuantLib::VarianceSwap, [1231](#)
 - fetchResults, [1232](#)
 - performCalculations, [1232](#)
 - setupExpired, [1232](#)
- QuantLib::VarianceSwap::arguments, [1233](#)
- QuantLib::VarianceSwap::engine, [1234](#)
- QuantLib::VarianceSwap::results, [1235](#)
- QuantLib::Vasicek, [1236](#)
- QuantLib::Vasicek::Dynamics, [1238](#)
- QuantLib::VEBCurrency, [1239](#)
- QuantLib::Visitor, [1240](#)
- QuantLib::YieldTermStructure, [1241](#)

- discount, 1243
- forwardRate, 1243
- parRate, 1243, 1244
- YieldTermStructure, 1243
- zeroRate, 1243
- QuantLib::ZARCurrency, 1245
- QuantLib::ZeroCondition, 1246
- QuantLib::ZeroCouponBond, 1247
- QuantLib::ZeroSpreadedTermStructure, 1248
- QuantLib::ZeroYield, 1250
- QuantLib::ZeroYieldStructure, 1251
 - discountImpl, 1252
- QuantLib::Zibor, 1253
- Quanto option engines, 122
- Quarterly
 - datetime, 110
- rankReducedSqrt
 - QuantLib::Matrix, 919
- Rate
 - types, 103
- Real
 - types, 102
- recalculate
 - QuantLib::LazyObject, 847
- regret
 - QuantLib::GenericRiskStatistics, 741
- RelinkableHandle
 - QuantLib::RelinkableHandle, 1066
- removeHoliday
 - QuantLib::Calendar, 287
- reset
 - QuantLib::DiscretizedAsset, 425
 - QuantLib::DiscretizedDiscountBond, 427
 - QuantLib::DiscretizedOption, 428
 - QuantLib::Problem, 1044
- rho
 - QuantLib::BlackCalculator, 242
- rollback
 - QuantLib::FiniteDifferenceModel, 641
 - QuantLib::Lattice, 843
 - QuantLib::TreeLattice, 1194
 - QuantLib::TsiveriotisFernandesLattice, 1207
- Rounding
 - QuantLib::Rounding, 1072
- SecondDerivative
 - QuantLib::CubicSpline, 389
- secondDerivativeAtCenter
 - QuantLib::SampledCurve, 1078
- Semiannual
 - datetime, 110
- semiDeviation
 - QuantLib::GenericRiskStatistics, 741
- semiVariance
 - QuantLib::GenericRiskStatistics, 741
- sequencestatistics.hpp
 - DEFINE_SEQUENCE_STAT_CONST_-METHOD_DOUBLE, 1489
 - DEFINE_SEQUENCE_STAT_CONST_-METHOD_VOID, 1489
- setMaxEvaluations
 - QuantLib::Solver1D, 1116
- setPricingEngine
 - QuantLib::Instrument, 798
- setTermStructure
 - QuantLib::DepositRateHelper, 412
 - QuantLib::FixedCouponBondHelper, 644
 - QuantLib::FraRateHelper, 687
 - QuantLib::RateHelper, 1064
 - QuantLib::SwapRateHelper, 1151
- setTime
 - QuantLib::BoundaryCondition, 272
 - QuantLib::DirichletBC, 416
 - QuantLib::NeumannBC, 981
- Settlement
 - QuantLib::Brazil, 276
 - QuantLib::Germany, 748
 - QuantLib::Italy, 829
 - QuantLib::UnitedKingdom, 1219
 - QuantLib::UnitedStates, 1222
- settlementDate
 - QuantLib::ForwardRateAgreement, 675
- setupArguments
 - QuantLib::Instrument, 798
- setupExpired
 - QuantLib::Instrument, 799
 - QuantLib::MultiAssetOption, 967
 - QuantLib::OneAssetOption, 1004
 - QuantLib::OneAssetStrikedOption, 1008
 - QuantLib::QuantoVanillaOption, 1055
 - QuantLib::Swap, 1148
 - QuantLib::VarianceSwap, 1232
- SGX
 - QuantLib::Singapore, 1099
- Short-rate modelling framework, 130
- short_date
 - manips, 156
- short_period
 - manips, 156
- short_weekday
 - manips, 156
- shortest_weekday
 - manips, 156
- shortfall
 - QuantLib::GenericRiskStatistics, 742
- Side

- QuantLib::BoundaryCondition, 271
- Simplex
 - QuantLib::Simplex, 1096
- Size
 - types, 102
- skewness
 - QuantLib::GeneralStatistics, 732
 - QuantLib::IncrementalStatistics, 787
- skipTo
 - QuantLib::SobolRsg, 1113
- SobolRsg
 - QuantLib::SobolRsg, 1113
- solve
 - QuantLib::Solver1D, 1116
- spotIncome
 - QuantLib::FixedRateBondForward, 650
 - QuantLib::ForwardRateAgreement, 676
- spotValue
 - QuantLib::ForwardRateAgreement, 676
- Spread
 - types, 103
- Sqrt
 - QuantLib::Array, 198
- standardDeviation
 - QuantLib::GeneralStatistics, 732
 - QuantLib::IncrementalStatistics, 787
- standardDeviations
 - QuantLib::CovarianceDecomposition, 380
- std, 161
- stdDeviation
 - QuantLib::G2ForwardProcess, 697
 - QuantLib::G2Process, 699
 - QuantLib::HullWhiteForwardProcess, 767
 - QuantLib::HullWhiteProcess, 769
 - QuantLib::OrnsteinUhlenbeckProcess, 1020
 - QuantLib::StochasticProcess, 1130
 - QuantLib::StochasticProcess1D, 1133
 - QuantLib::StochasticProcessArray, 1137
- Steps
 - QuantLib::SobolBrownianGenerator, 1111
- Stochastic processes, 144
- strikeSensitivity
 - QuantLib::BlackCalculator, 242
- Swap
 - QuantLib::Swap, 1148
- swap
 - QuantLib::Array, 198
 - QuantLib::Clone, 331
 - QuantLib::Matrix, 918
- Swaption engines, 123
- SwaptionVolatilityMatrix
 - QuantLib::SwaptionVolatilityMatrix, 1163
- SwaptionVolatilityStructure
 - QuantLib::SwaptionVolatilityStructure, 1167
- SymmetricSchurDecomposition
 - QuantLib::SymmetricSchurDecomposition, 1170
- Tadawul
 - QuantLib::SaudiArabia, 1080
- targetValueAndGradient
 - QuantLib::LeastSquareProblem, 849
- Template capabilities, 153
- templateMacros
 - QL_TYPENAME, 153
- Term structures, 146
- TermStructure
 - QuantLib::TermStructure, 1176
- theta
 - QuantLib::BlackCalculator, 242
 - QuantLib::BlackScholesCalculator, 250
- thetaPerDay
 - QuantLib::BlackCalculator, 242
 - QuantLib::BlackScholesCalculator, 250
- Time
 - types, 102
- time
 - QuantLib::GeneralizedBlackScholesProcess, 730
 - QuantLib::HestonProcess, 757
 - QuantLib::Merton76Process, 953
 - QuantLib::StochasticProcess, 1131
 - QuantLib::StochasticProcessArray, 1137
- TimeGrid
 - QuantLib::TimeGrid, 1185
- TimeSeries
 - QuantLib::TimeSeries, 1187
- TimeUnit
 - datetime, 110
- topPercentile
 - QuantLib::GeneralStatistics, 733
- transpose
 - QuantLib::Matrix, 918
- TSEC
 - QuantLib::Taiwan, 1173
- Type
 - QuantLib::ExchangeRate, 619
 - QuantLib::Rounding, 1071
- types
 - BigInteger, 102
 - Decimal, 102
 - DiscountFactor, 103
 - Integer, 102
 - Natural, 102
 - Rate, 103
 - Real, 102

- Size, 102
- Spread, 103
- Time, 102
- Volatility, 103
- Unadjusted
 - datetime, 110
- underlyingArgs
 - QuantLib::QuantoEngine, 1048
- underlyingIncome_
 - QuantLib::Forward, 665
- underlyingSpotValue_
 - QuantLib::Forward, 665
- underlyingSwap
 - QuantLib::SwapIndex, 1149
- unfreeze
 - QuantLib::LazyObject, 847
- Up
 - QuantLib::Rounding, 1072
- update
 - QuantLib::BlackCapFloorEngine, 243
 - QuantLib::BlackSwaptionEngine, 255
 - QuantLib::CalibratedModel, 293
 - QuantLib::CalibrationHelper, 295
 - QuantLib::CappedFlooredCoupon, 310
 - QuantLib::CapVolatilityVector, 314
 - QuantLib::CmsMarket, 336
 - QuantLib::CompositeQuote, 346
 - QuantLib::DecInterpCapletVolStructure, 409
 - QuantLib::DerivedQuote, 414
 - QuantLib::ExtendedDiscountCurve, 629
 - QuantLib::FlatForward, 654
 - QuantLib::FloatingRateCoupon, 658
 - QuantLib::FloatingRateCouponPricer, 659
 - QuantLib::ForwardValueQuote, 682
 - QuantLib::FuturesConvAdjustmentQuote, 691
 - QuantLib::GeneralizedBlackScholesProcess, 730
 - QuantLib::GenericModelEngine, 739
 - QuantLib::HestonProcess, 757
 - QuantLib::InterestRateIndex, 805
 - QuantLib::LatticeShortRateModelEngine, 845
 - QuantLib::LazyObject, 846
 - QuantLib::MarketModelCapFloorEngine, 909
 - QuantLib::MCHullWhiteCapFloorEngine, 939
 - QuantLib::Observer, 1001
 - QuantLib::PiecewiseYieldCurve, 1032
 - QuantLib::PiecewiseZeroSpreadedTermStructure, 1034
 - QuantLib::RateHelper, 1064
 - QuantLib::RelativeDateRateHelper, 1065
 - QuantLib::StochasticProcess, 1131
 - QuantLib::SwaptionVolatilityCube, 1161
 - QuantLib::SwaptionVolatilityMatrix, 1164
 - QuantLib::TermStructure, 1176
- USE
 - QuantLib::Ukraine, 1217
- Utilities, 148
- valueAtCenter
 - QuantLib::SampledCurve, 1078
- valueAtRisk
 - QuantLib::GenericRiskStatistics, 741
- valueDate_
 - QuantLib::Forward, 665
- Vanilla option engines, 124
- vanillaengines
 - FDAmericanEngine, 125
 - FDDividendAmericanEngine, 126
 - FDDividendEuropeanEngine, 126
 - FDDividendShoutEngine, 126
 - FDShoutEngine, 126
- variance
 - QuantLib::Abcd, 165
 - QuantLib::AbcdFunction, 167
 - QuantLib::EulerDiscretization, 461
 - QuantLib::GeneralStatistics, 732
 - QuantLib::HullWhiteForwardProcess, 767
 - QuantLib::HullWhiteProcess, 769
 - QuantLib::IncrementalStatistics, 787
 - QuantLib::OrnsteinUhlenbeckProcess, 1020
 - QuantLib::StochasticProcess1D, 1133
- variances
 - QuantLib::CovarianceDecomposition, 380
- vega
 - QuantLib::BlackCalculator, 242
- Volatility
 - types, 103
- volatility
 - QuantLib::Abcd, 165
 - QuantLib::AbcdFunction, 167
- Weekday
 - datetime, 110
- Weekly
 - datetime, 110
- Xetra
 - QuantLib::Germany, 748
- Year
 - datetime, 109

- yield
 - QuantLib::Bond, [269](#)
- YieldTermStructure
 - QuantLib::YieldTermStructure, [1243](#)
- yieldtermstructures
 - DiscountCurve, [147](#)
 - ForwardCurve, [147](#)
 - ZeroCurve, [147](#)
- ZeroCurve
 - yieldtermstructures, [147](#)
- zeroRate
 - QuantLib::YieldTermStructure, [1243](#)
- zeroYieldImpl
 - QuantLib::CompoundForward, [348](#)
 - QuantLib::ExtendedDiscountCurve, [630](#)
 - QuantLib::ForwardRateStructure, [678](#)
 - QuantLib::InterpolatedForwardCurve, [809](#)