RiskAnalysisTool

version 1.0.2

Generated by Doxygen 1.8.9.1

Wed May 13 2015 01:08:52

Contents

Chapter 1

Test List

Class QuantLib::GenericArrayStatistics< StatisticsType > (p. **41**)

the correctness of the returned values is tested by checking them against numerical calculations.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

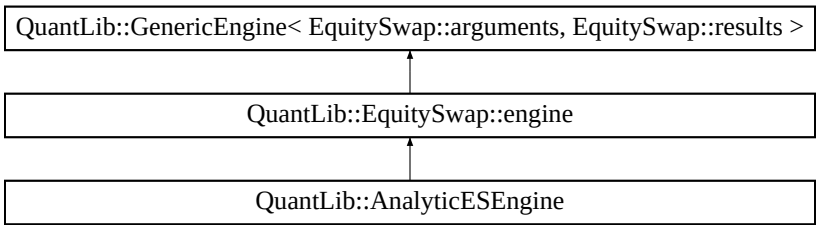
Class Documentation

5.1 QuantLib::AnalyticESEngine Class Reference

Analytic engine of equity swap.

`#include <analyticequityswapengine.hpp>`

Inheritance diagram for QuantLib::AnalyticESEngine:



Public Member Functions

Constructors

- **AnalyticESEngine** ()
- void **calculate** () const

  Calculate the npv of the equity swap.

5.1.1 Detailed Description

Analytic engine of equity swap.

This class is the analytic engine of equity swap. The cash expected cash flow from the equity leg and the fixed coupon leg is discounted back to reference date.

5.1.2 Member Function Documentation

5.1.2.1 void AnalyticESEngine::calculate ( ) const

Calculate the npv of the equity swap.

The swap leg is already constructed the calculation is just discounting the cash flow back to the current time.

The documentation for this class was generated from the following files:

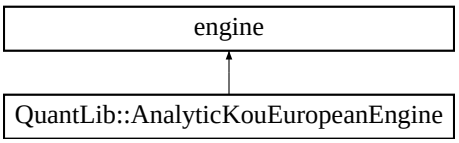
- **analyticequityswapengine.hpp**
- **analyticequityswapengine.cpp**

5.2 QuantLib::AnalyticKouEuropeanEngine Class Reference

Analytic engine for european option, whose underlying stock process is a kou process.

`#include <analytickoueuropeanengine.hpp>`

Inheritance diagram for QuantLib::AnalyticKouEuropeanEngine:



Constructor

- **AnalyticKouEuropeanEngine** (const boost::shared_ptr< **KouProcess** > &process, const Real &tolerance=0.0001)
- void **calculate** () const override

  calculate the european option price

5.2.1 Detailed Description

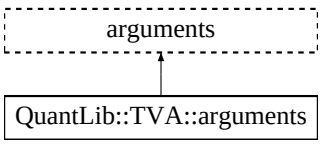
Analytic engine for european option, whose underlying stock process is a kou process.

The documentation for this class was generated from the following files:

- **analytickoueuropeanengine.hpp**
- **analytickoueuropeanengine.cpp**

5.3 QuantLib::TVA::arguments Class Reference

Inheritance diagram for QuantLib::TVA::arguments:



Public Member Functions

- virtual void **validate** () const

The documentation for this class was generated from the following file:

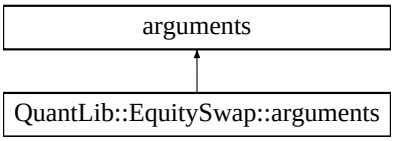
- **tva.hpp**

5.4 QuantLib::EquitySwap::arguments Class Reference

Arguments for equity swap calculation

`#include <equityswap.hpp>`

Inheritance diagram for QuantLib::EquitySwap::arguments:



Public Member Functions

- void **validate** () const

Public Attributes

- boost::shared_ptr< QuantLib::YieldTermStructure > **discountCurve_**
- **Type type_**
- QuantLib::Date **referenceDate_**
- QuantLib::Date **startDate_**
- QuantLib::Real **spotPrice_**
- QuantLib::Real **startPrice_**
- QuantLib::Integer **amount_**
- QuantLib::Rate **fixedRate_**
- QuantLib::Rate **dividend_**
- QuantLib::Time **maturity_**
- QuantLib::Real **sigma_**
- QuantLib::Real **cumulativeCoupoon_**
- QuantLib::Real **cumulativeDividend_**
- QuantLib::Frequency **freq_**
- boost::shared_ptr< QuantLib::Schedule > **schedule_**
- size_t **count_**
- std::vector< QuantLib::Time > **yearfraction_**

5.4.1 Detailed Description

Arguments for equity swap calculation

The documentation for this class was generated from the following file:

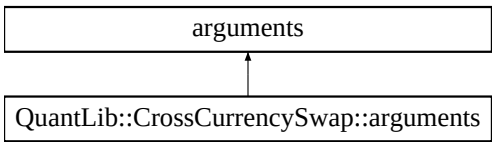
- **equityswap.hpp**

5.5 QuantLib::CrossCurrencySwap::arguments Class Reference

Arguments for cross currency swap calculation

`#include <crosscurrencyswap.hpp>`

Inheritance diagram for QuantLib::CrossCurrencySwap::arguments:



Public Member Functions

- void **validate** () const

Public Attributes

- **Type type**
- Date **referenceDate**
- Rate **fxRate**
- Volatility **fxVol**
- Real **payNominal**
- Real **receiveNominal**
- DayCounter **payDayCount**
- DayCounter **receiveDayCount**
- std::vector< Date > **payDates**
- std::vector< Date > **receiveDates**
- std::vector< Real > **payCoupons**
- std::vector< Real > **receiveCoupons**

5.5.1 Detailed Description

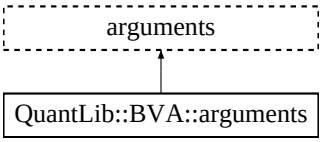
Arguments for cross currency swap calculation

The documentation for this class was generated from the following files:

- **crosscurrencyswap.hpp**
- crosscurrengyswap.cpp

5.6 QuantLib::BVA::arguments Class Reference

Inheritance diagram for QuantLib::BVA::arguments:



Public Member Functions

- void **validate** () const

Public Attributes

- boost::shared_ptr< const **Counterparty** > **self**
- boost::shared_ptr< const **Counterparty** > **counterparty**
- std::vector< boost::shared_ptr< const Instrument > > **portfolio**
- Handle< YieldTermStructure > **discountCurve**

The documentation for this class was generated from the following file:

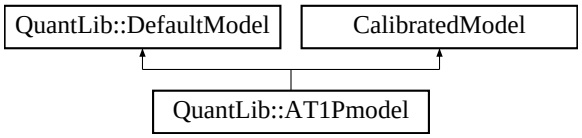
- **bva.hpp**

5.7 QuantLib::AT1Pmodel Class Reference

Analytic tractable first default model.

`#include <at1pdefaultmodel.hpp>`

Inheritance diagram for QuantLib::AT1Pmodel:



Public Member Functions

- const QuantLib::Probability **defaultProbability** (QuantLib::Time T) const override

  Calculate the default probability according to AT1P model.
- const QuantLib::Time **defaultTime** (const Path &path) const override

  Return the default time of the path according to AT1P model.
- boost::shared_ptr< StochasticProcess1D > **process** () const override

  Return the process of counterparty after calibration.
- 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 >(), const std::vector< bool > &fixParameters=std::vector< bool >()) override

  Calibrate the at1p model volatility using sets of CDSs as calibration helper.

destructors

- ~**AT1Pmodel** ()

Constructors

- **AT1Pmodel** (boost::shared_ptr< const std::vector< QuantLib::Period >> periods, boost::shared_ptr< QuantLib::YieldTermStructure > discountcurve, const QuantLib::Real dividend=0, const QuantLib::Real H_V=0.4, const QuantLib::Real B=0, const QuantLib::Real x0=1, boost::shared_ptr< QuantLib::Array > volatility_=nullptr)

5.7.1 Detailed Description

Analytic tractable first default model.

The class support the calculations relating to the analytical tractable first passage time model under the AT1P setting the volatility of the the firm value, is piecewise constant.

The documentation for this class was generated from the following files:

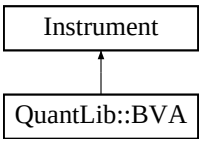
- **at1pdefaultmodel.hpp**
- **at1pdefaultmodel.cpp**

5.8 QuantLib::BVA Class Reference

Bilateral value adjustment manager class.

`#include <bva.hpp>`

Inheritance diagram for QuantLib::BVA:



Classes

- class **arguments**
- class **engine**
- class **results**

Public Member Functions

Constructors & Destructors

- **BVA** (const boost::shared_ptr< **Counterparty** > &self, const boost::shared_ptr< **Counterparty** > &counterparty, const std::vector< boost::shared_ptr< const Instrument >> &portfolio, const Matrix &correlation, const Handle< YieldTermStructure > &discountCurve)

Inspectors

- boost::shared_ptr< **Counterparty** > **self** ()
- boost::shared_ptr< **Counterparty** > **counterparty** ()
- Handle< YieldTermStructure > **discountCurve** ()

Public interface

- const Real **CVA** () const

  Calculate the CVA and return the result, or return the result directly if it has calculated.
- const Real **DVA** () const

  Calculate the DVA and return the result, or return the result directly if it has calculated.
- void **setupArguments** (PricingEngine::arguments *args) const override
- void **fetchResults** (const PricingEngine::results *) const
- bool **isExpired** () const

Protected Attributes

- std::vector< boost::shared_ptr< const Instrument > > **portfolio_**
- const boost::shared_ptr< **Counterparty** > **self_**
- const boost::shared_ptr< **Counterparty** > **counterparty_**
- Matrix **correlation_**
- Handle< YieldTermStructure > **discountCurve_**
- Real **CVA_**
- Real **DVA_**

5.8.1 Detailed Description

Bilateral value adjustment manager class.

This class inherits from Instrument class, responsible for calculation bilateral value adjustment (includes CV↩ A, DVA) of a single instrument or portfolio. The arguments are the information of two counterparties (issuer and investor) using the **Counterparty** (p. 27) class and the information of underlying instrument or portfolio using their corresponding model class. The Pricingengine uses tvaengine based on Monte Carlo simulation framework. The results are including CVA, DVA.

See also

counterparty.hpp (p. 75)
mctvamodel.hpp (p. 86)
mcbvaengine.hpp (p. 81)

5.8.2 Constructor & Destructor Documentation

5.8.2.1 QuantLib::BVA::BVA (const boost::shared_ptr< Counterparty > & *self*, const boost::shared_ptr< Counterparty > & *counterparty*, const std::vector< boost::shared_ptr< const Instrument >> & *portfolio*, const Matrix & *correlation*, const Handle< YieldTermStructure > & *discountCurve*) [inline]

Creates an instance of **BVA** (p. 20) manager using two counterparties, instruments (single instrument or portfolio), correlation matrix of the counterparties and instruments, and the yield term structure.

The documentation for this class was generated from the following file:

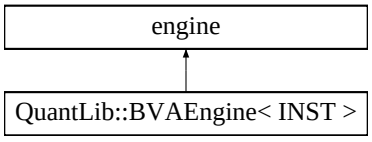
- **bva.hpp**

5.9 QuantLib::BVAEngine< INST > Class Template Reference

Base class for bva engine.

`#include <bvaengine.hpp>`

Inheritance diagram for QuantLib::BVAEngine< INST >:



Public Member Functions

Constructors & Destructors

- **BVAEngine** (Time endTime, const std::vector< boost::shared_ptr< const **Counterparty** >> &counterparties, const std::vector< boost::shared_ptr< const **MCTVAModel** >> &bvaModels, const Handle< YieldTermStructure > &discountCurve, const boost::shared_ptr< **ShortRateTermStructure** > &shortRateDynamics)
- virtual ~**BVAEngine** ()

Inspector

- std::vector< boost::shared_ptr< const **Counterparty** > > **counterparties** () const
- const Time **endTime** () const
- const Handle< YieldTermStructure > & **discountCurve** () const
- const boost::shared_ptr< **ShortRateTermStructure** > & **shortRateDynamics** () const
- const std::vector< boost::shared_ptr< const **MCTVAModel** > > & **models** () const

Protected Attributes

- std::vector< boost::shared_ptr< const **Counterparty** > > **counterparties_**
- Handle< YieldTermStructure > **discountCurve_**
- boost::shared_ptr< **ShortRateTermStructure** > **shortRateDynamics_**
- std::vector< boost::shared_ptr< const **MCTVAModel** > > **bvaModels_**
- Time **endTime_**

5.9.1 Detailed Description

template<typename INST>class QuantLib::BVAEngine< INST >

Base class for bva engine.

This class is base class for bva engine, which is used to set some basic information. Each derived class should implement its own method to calculate bva.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 template<typename INST> QuantLib::BVAEngine< INST >::BVAEngine (Time *endTime,* const std::vector< boost::shared_ptr< const Counterparty >> & *counterparties,* const std::vector< boost::shared_ptr< const MCTVAModel >> & *bvaModels,* const Handle< YieldTermStructure > & *discountCurve,* const boost::shared_ptr< ShortRateTermStructure > & *shortRateDynamics*) [inline]

Create an instance of **BVA** (p. 20) Engine using the information of the two counterparties using **Counterparty** (p. 27) class and underlying instruments using instrument model class.

The documentation for this class was generated from the following file:

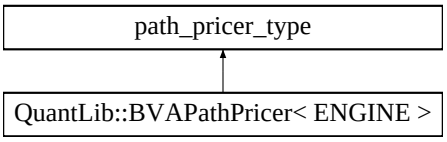
- **bvaengine.hpp**

5.10 QuantLib::BVAPathPricer< ENGINE > Class Template Reference

Path pricer class used to calculate the **BVA** (p. 20) on each Multi-path for **BVAEngine** (p. 21) class.

`#include <bvapathpricer.hpp>`

Inheritance diagram for QuantLib::BVAPathPricer< ENGINE >:



Public Types

- typedef ENGINE::path_pricer_type::argument_type **argument_type**
- typedef ENGINE::path_pricer_type::result_type **result_type**

Public Member Functions

Constructors

- **BVAPathPricer** (const boost::shared_ptr< const ENGINE > &engine)

Destructors

- ~**BVAPathPricer** ()

Operator overload

- result_type **operator()** (const argument_type &multiPath) const
  *Calculate the **TVA** (p. 62) (includes CVA, DVA and FVA) given the Multi-path.*

Protected Attributes

- const boost::weak_ptr< const ENGINE > **engine_**

5.10.1 Detailed Description

template<typename ENGINE>class QuantLib::BVAPathPricer< ENGINE >

Path pricer class used to calculate the **BVA** (p. 20) on each Multi-path for **BVAEngine** (p. 21) class.

This class is the core part for **BVAEngine** (p. 21) class, which is used to calculate the **BVA** (p. 20) (includes CVA, DVA) on each Multi-path given the instruments (single instrument or portfolio), based on the definition of **BVA** (p. 20) (includes CVA, DVA) in Brigo's book: "Counterparty Credit Risk, Collateral and Funding".

If the underlying is a portfolio, netting is considered.

See also

MCBVAEngine (p. 45)

The documentation for this class was generated from the following file:

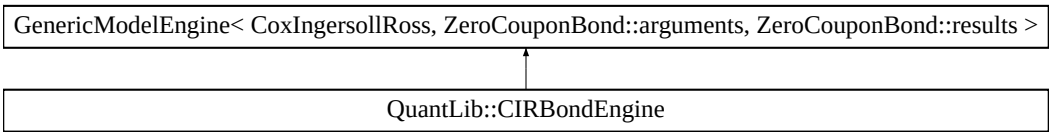
- **bvapathpricer.hpp**

5.11 QuantLib::CIRBondEngine Class Reference

A **CIRBondEngine** (p. 24) is a bond pricing engine based on CIR model.

`#include <cirbondengine.hpp>`

Inheritance diagram for QuantLib::CIRBondEngine:



Public Member Functions

- void **calculate** () const
  Calculates the model value of bond price using CIR model.

Constructors & Destructors

- **CIRBondEngine** (const boost::shared_ptr< CoxIngersollRoss > &model, const Handle< YieldTerm↩ Structure > &termStructure)
  *Creates an instance of **CIRBondEngine** (p. 24) using a CIR model instance and a yield term structure.*

5.11.1 Detailed Description

A **CIRBondEngine** (p. 24) is a bond pricing engine based on CIR model.

This class is to calculate bond prices using CIR model, which are used by the calibration helpers of CoxIngersoll↩ Ross model. Internally, this class calculate the bond price using

$$e^{b(t)-a(t)r_t}$$

See also

CoxIngersollRoss
ZerocouponbondHelper (p. 68)

The documentation for this class was generated from the following files:

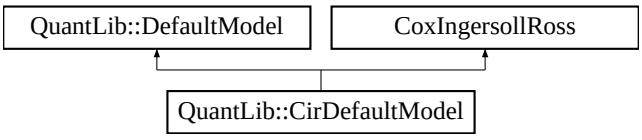
- **cirbondengine.hpp**
- **cirbondengine.cpp**

5.12 QuantLib::CirDefaultModel Class Reference

Intensity default model based on the CIR process.

`#include <cirdefaultmodel.hpp>`

Inheritance diagram for QuantLib::CirDefaultModel:



Public Member Functions

Constructors & Destructors

- **CirDefaultModel** (Rate r0=0.05, Real theta=0.1, Real k=0.1, Real sigma=0.1)

  Creates an instance of CIR default model by setting parameters as default initial values.
- ∼**CirDefaultModel** ()

Public interface

- const Probability **defaultProbability** (Time t) const override

  Calculate the default probability according to CIR dedault model.
- const Time **defaultTime** (const Path &path) const override
- boost::shared_ptr< StochasticProcess1D > **process** () const override

  Return the process of counterparty after calibration.
- 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 >(), const std::vector< bool > &fixParameters=std::vector< bool >()) override

  Calibrate the CIR default model volatility using sets of CDSs as calibration helper.

5.12.1 Detailed Description

Intensity default model based on the CIR process.

The model uses CIR process to describe defalut intensity, it should be calibrated by CDS data.

5.12.2 Member Function Documentation

5.12.2.1 const Time CirDefaultModel::defaultTime (const Path & *path*) const [override], [virtual]

Return the default time of the path according to CIR dedault model, based on Monte Carlo method

Implements **QuantLib::DefaultModel** (p. 33).

The documentation for this class was generated from the following files:

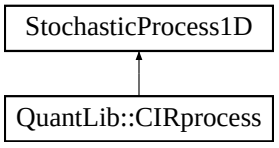
- **cirdefaultmodel.hpp**
- **cirdefaultmodel.cpp**

5.13 QuantLib::CIRprocess Class Reference

CIR process class.

`#include <cirprocess.hpp>`

Inheritance diagram for QuantLib::CIRprocess:



Public Types

- enum **Discretization** { **Euler**, **Milstein**, **ImplicitMilstein**, **NonCentralChiSquareVariance** }

  An enum type for Discretization method.

Public Member Functions

Constructors & Destructors

- **CIRprocess** (Real mean, Real speed, Volatility sigma, Real x0=0.0, **Discretization** discretization=**Implicit↩ Milstein**)
- ∼**CIRprocess** ()

Inspectors

- Real **x0** () const
- Real **revertSpeed** () const
- Real **revertLevel** () const
- Volatility **volatility** () const

Public interface

- Real **drift** (Time t, Real x) const

  Return the drift term of the process.
- Real **diffusion** (Time t, Real x) const

  Return the diffusion of the process.
- Real **evolve** (Time t0, Real x0, Time dt, Real dw) const

  Return simulation value of the process at time t0+dts.

5.13.1 Detailed Description

CIR process class.

This class describes a square-root process governed by

$$dx = a(b - x_t)dt + \sigma\sqrt{x_t}dW_t.$$

The process is used to model CIR process which provides more precise evolve method.

Remarks

The Implict Milstein discretization scheme is recommended to use for efficient simulation. Implict Milstein is got from classical Milstein, in which we replace the drift term $a(b - r_t)dt$ with $a(b - r_{t+dt})dt$. We then obtain:

$$x_{t+dt} = \frac{x_t + abdt + \sigma\sqrt{x_t}\sqrt{dt}dw + \frac{1}{4}\sigma^2 dt(dw^2 - 1)}{1 + adt}$$

5.13.2 Member Enumeration Documentation

5.13.2.1 enum QuantLib::CIRprocess::Discretization

An enum type for Discretization method.

Enumerator

Euler Euler Discretization Scheme.

Milstein Milstein Discretization Scheme.

ImplicitMilstein Implicit Milstein Discretization Scheme.

NonCentralChiSquareVariance Non-Central Chi-Sqaure Distribution.

5.13.3 Constructor & Destructor Documentation

5.13.3.1 CIRprocess::CIRprocess (Real *mean,* Real *speed,* Volatility *sigma,* Real *x0 =* 0.0, Discretization *discretization =* ImplicitMilstein)

Creates an instance of CIR process using the four parameters long-term mean, revert speed, volatility and initial value

The documentation for this class was generated from the following files:

- **cirprocess.hpp**
- cirprocess.cpp

5.14 QuantLib::Counterparty Class Reference

Counterparty (p. 27) class.

`#include <counterparty.hpp>`

Public Member Functions

- double **fundingSpread** (Time t) const
  Return the funding spread at time t.

Constructors & Destructors

- **Counterparty** (int settlementDays, double recoveryRate, const boost::shared_ptr< const std::vector< double >> &cdsSpreads, const boost::shared_ptr< const std::vector< Period >> &cdsTenors, const boost::shared_ptr< YieldTermStructure > &discountCurve, const boost::shared_ptr< **Default**↩ **Model** > &model_, const QuantLib::Frequency &freq=QuantLib::Quarterly, const QuantLib::Business↩ DayConvention &busDayConvention=QuantLib::Following, const QuantLib::DateGeneration::Rule &dateGenerationRule=QuantLib::DateGeneration::TwentiethIMM, const QuantLib::Date &reference↩ Date=QuantLib::Date(), const QuantLib::DayCounter &dayCounter=QuantLib::Actual365Fixed(), const QuantLib::Calendar &calendar=QuantLib::TARGET(), const boost::shared_ptr< YieldTermStructure > &defaultdiscountCurve=0)
  *Creates an instance of **Counterparty** (p. 27) using its CDS information and the default model.*
- ~**Counterparty** ()

Public interface

- const Real **getDefaultProb** (const Time &t) const
  Return default probability within time t based on the default model.
- const Time **getDefaultTime** (const Path &path) const
  Return default time given a path describing the evolution of default state variable.
- const Time **defaultTimeGenerator** (const Probability &prob, const double &epsilon=1e-2) const
  Return random number of default time.
- void **modelCalibrate** (OptimizationMethod &method, const EndCriteria &endCriteria, bool need↩ Calibrate=true, const Constraint &constraint=Constraint(), const std::vector< Real > &weights=std↩ ::vector< Real >(), const std::vector< bool > &fixParameters=std::vector< bool >())
  Calibrate default model.

Inspectors

- const boost::shared_ptr< const **DefaultModel** > & **getModel** () const
  Return shared pointer of default model.
- const double **getRecoveryRate** () const
- boost::shared_ptr< YieldTermStructure > **discountCurve** () const
- const Date **referenceDate** () const
- boost::shared_ptr< StochasticProcess1D > **createDefaultProcess** () const
  Return shared pointer of process describing default state variable.

5.14.1 Detailed Description

Counterparty (p. 27) class.

This class is used to describe defalut behavior of investor and issuer involved in a transaction, based on the default model.

The documentation for this class was generated from the following files:

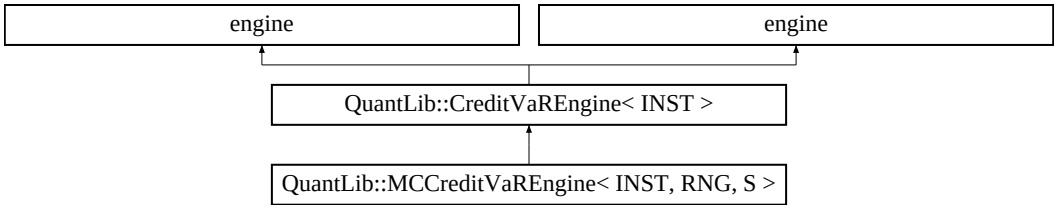
- **counterparty.hpp**
- **counterparty.cpp**

5.15 QuantLib::CreditVaREngine< INST > Class Template Reference

base class for credit VaR engine.

`#include <creditvarengine.hpp>`

Inheritance diagram for QuantLib::CreditVaREngine< INST >:



Public Member Functions

- **CreditVaREngine** (Time endTime, const std::vector< boost::shared_ptr< const **Counterparty** >> &counterparties, const std::vector< boost::shared_ptr< const **MCTVAModel** >> &creditvarModels, const Handle< YieldTermStructure > &discountCurve, const boost::shared_ptr< **ShortRateTermStructure** > &shortRateDynamics)
- std::vector< boost::shared_ptr< const **Counterparty** > > **counterparties** () const
- const Time **endTime** () const
- const Handle< YieldTermStructure > & **discountCurve** () const
- const boost::shared_ptr< **ShortRateTermStructure** > & **shortRateDynamics** () const
- const std::vector< boost::shared_ptr< const **MCTVAModel** > > & **models** () const
- Time **endTime** () const

Constructors & Destructors

- **CreditVaREngine** (const boost::shared_ptr< const INST > instrument, const boost::shared_ptr< const **Counterparty** > issuer, const Time &endTime, const Handle< YieldTermStructure > &discountCurve)
- virtual ~**CreditVaREngine** ()

Protected Attributes

- std::vector< boost::shared_ptr< const **Counterparty** > > **counterparties_**
- Handle< YieldTermStructure > **discountCurve_**
- boost::shared_ptr< **ShortRateTermStructure** > **shortRateDynamics_**
- std::vector< boost::shared_ptr< const **MCTVAModel** > > **CreditVarModels_**
- Time **endTime_**
- const Time **endTime_**
- const boost::shared_ptr< const INST > **instrument_**
- const boost::shared_ptr< const **Counterparty** > **issuer_**
- const Handle< YieldTermStructure > **discountCurve_**

5.15.1 Detailed Description

template<typename INST>class QuantLib::CreditVaREngine< INST >

base class for credit VaR engine.

This class is base class for credit VaR calculation, which is used to set some basic information. Each derived class should implement its own method to calculate credit VaR.

The documentation for this class was generated from the following file:

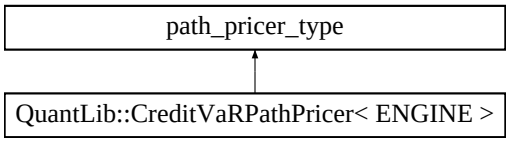
- pricingengines/credit/creditvarengine.hpp

5.16 QuantLib::CreditVaRPathPricer< ENGINE > Class Template Reference

Abstract base class used to calculate default value of each path.

`#include <creditvarpathpricer.hpp>`

Inheritance diagram for QuantLib::CreditVaRPathPricer< ENGINE >:



Public Types

- typedef ENGINE::path_pricer_type::argument_type **argument_type**
- typedef ENGINE::path_pricer_type::result_type **result_type**

Public Member Functions

Destructors

- ~**CreditVaRPathPricer** ()

Operator overload

- result_type **operator()** (const argument_type &multiPath) const

Protected Member Functions

Constructors

- **CreditVaRPathPricer** (const boost::shared_ptr< const ENGINE > &engine)

Protected interface

- const boost::weak_ptr< const ENGINE > **engine_**

  Return weak pointer of pricing engine.
- virtual Real **defaultNPV** (const MultiPath &path, Time defaultTime) const =0

  Return default NPV at each path, must be implemented in derived class.

5.16.1 Detailed Description

template<typename ENGINE>class QuantLib::CreditVaRPathPricer< ENGINE >

Abstract base class used to calculate default value of each path.

This class is used to calculate default value of each path. To use this class, derived class must implement the pure virtual function defaultNPV.

The documentation for this class was generated from the following file:

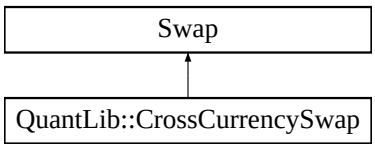
- **creditvarpathpricer.hpp**

5.17 QuantLib::CrossCurrencySwap Class Reference

fixed-fixed cross currency swap class

`#include <crosscurrencyswap.hpp>`

Inheritance diagram for QuantLib::CrossCurrencySwap:



Classes

- class **arguments**

  Arguments for cross currency swap calculation
- class **engine**

  base class for cross currency swap pricing engine
- class **results**

  Results for cross currency swap calculation

Public Types

- enum **Type** { **payDomestic** = 0, **payForeign** }

Public Member Functions

Constructors & Destructors

- **CrossCurrencySwap** (**Type type**, Rate fxRate, Volatility fxVol, const Date &referenceDate, Real pay↩ Nominal, Real receiveNominal, const Schedule &paySchedule, Rate payRate, const DayCounter &pay↩ DayCount, const Schedule &receiveSchedule, Rate receiveRate, const DayCounter &receiveDayCount, boost::shared_ptr< Currency > payCurrency, boost::shared_ptr< Currency > receiveCurrency, boost↩ ::optional< BusinessDayConvention > paymentConvention=boost::none)
- ~**CrossCurrencySwap** ()

Instrument interface

- void **setupArguments** (PricingEngine::arguments *args) const
- void **fetchResults** (const PricingEngine::results *r) const

Inspectors

- **Type type** () const

  Return type of cross currency swap, which should be payDomestic or payForeign.
- const Rate **fxRate** () const
- const Volatility **fxVolatility** () const
- const Date & **referenceDate** () const
- Real **payNominal** () const
- Real **receiveNominal** () const
- const Schedule & **paySchedule** () const
- Rate **payRate** () const
- const DayCounter & **payDayCount** () const
- const Schedule & **receiveSchedule** () const
- Rate **receiveRate** () const
- const DayCounter & **receiveDayCount** () const
- BusinessDayConvention **paymentConvention** () const
- const Leg & **payLeg** () const
- const Leg & **receiveLeg** () const

5.17.1 Detailed Description

fixed-fixed cross currency swap class

This class is used to describe fixed-fixed cross currency swap

5.17.2 Member Enumeration Documentation

5.17.2.1 enum QuantLib::CrossCurrencySwap::Type

An enum type for Swap Type. The documentation block cannot be put after the enum!

Enumerator

payDomestic pay

payForeign receive

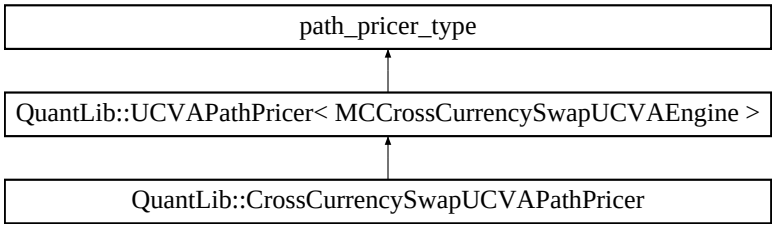
The documentation for this class was generated from the following files:

- **crosscurrencyswap.hpp**
- crosscurrengyswap.cpp

5.18 QuantLib::CrossCurrencySwapUCVAPathPricer Class Reference

UCVA Path Pricer for fixed-fixed cross currency swap.

Inheritance diagram for QuantLib::CrossCurrencySwapUCVAPathPricer:



Public Member Functions

Constructors & Destructors

- **CrossCurrencySwapUCVAPathPricer** (const boost::shared_ptr< const **MCCrossCurrencySwapUC↩ VAEngine** > &engine)
- ~**CrossCurrencySwapUCVAPathPricer** ()

Protected Member Functions

- virtual Real **defaultNPV** (const MultiPath &path, Time defaultTime) const

  Calculate the NPV of the remaining cash flow at default time τ.

Additional Inherited Members

5.18.1 Detailed Description

UCVA Path Pricer for fixed-fixed cross currency swap.

The documentation for this class was generated from the following file:

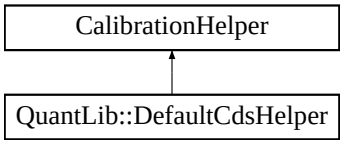
- **mccrosscurrencyswapucvaengine.cpp**

5.19 QuantLib::DefaultCdsHelper Class Reference

CDS helpers to calibrate default model.

`#include <defaultcdshelper.hpp>`

Inheritance diagram for QuantLib::DefaultCdsHelper:



Public Member Functions

Constructors & Destructors

- **DefaultCdsHelper** (Time cdsTime, const Handle< Quote > &impliedDefaultProb, const Handle< Yield↩ TermStructure > &termStructure, const boost::shared_ptr< CalibratedModel > &model)
- ~**DefaultCdsHelper** ()

Public interface

- Real **modelValue** () const override

  Calculate default probability based on default model.
- Real **blackPrice** (Volatility volatility) const override

  Return default probability boostrapped from CDS data.

5.19.1 Detailed Description

CDS helpers to calibrate default model.

Calculate default probability based on default and market data, it will be used to calibrate default model.

The documentation for this class was generated from the following files:

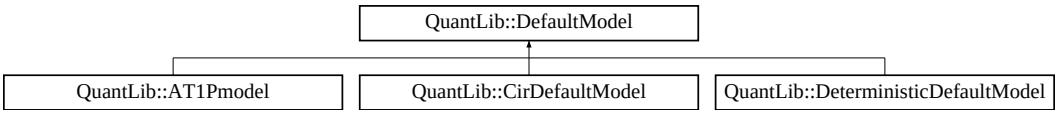
- **defaultcdshelper.hpp**
- defaultcdshelper.cpp

5.20 QuantLib::DefaultModel Class Reference

abstract base class for default model

`#include <defaultmodel.hpp>`

Inheritance diagram for QuantLib::DefaultModel:



Public Member Functions

Constructors & Destructors

- **DefaultModel** ()
- virtual ∼**DefaultModel** ()

Publice interface

- virtual const Probability **defaultProbability** (Time t) const =0

  Calculate the default probability at time t according to the given dedault model.
- virtual const Time **defaultTime** (const Path &path) const =0
- virtual boost::shared_ptr< StochasticProcess1D > **process** () const =0

  Return stochastic process describing default state variable.

5.20.1 Detailed Description

abstract base class for default model

Abstract base class for default model, define pure virtual functions need to be implemented in derived class.

5.20.2 Member Function Documentation

5.20.2.1 virtual const Time QuantLib::DefaultModel::defaultTime (const Path & *path*) const [pure virtual]

Return the default time of the path according to the given dedault model, based on Monte Carlo method

Implemented in **QuantLib::AT1Pmodel** (p. 19), **QuantLib::CirDefaultModel** (p. 25), and **QuantLib::↩ DeterministicDefaultModel** (p. 34).

The documentation for this class was generated from the following file:

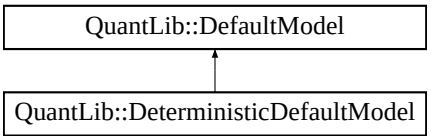
- **defaultmodel.hpp**

5.21 QuantLib::DeterministicDefaultModel Class Reference

Intensity default model based on deterministic default intensity.

`#include <deterministicdefaultmodel.hpp>`

Inheritance diagram for QuantLib::DeterministicDefaultModel:



Public Member Functions

Constructors & Destructors

- **DeterministicDefaultModel** ()
- ∼**DeterministicDefaultModel** ()

Inspectors

- void **setHazardRateStructure** (boost::shared_ptr< QuantLib::PiecewiseDefaultCurve< QuantLib::↩ HazardRate, QuantLib::BackwardFlat > > newHarzardRateStructure)
- virtual const Probability **defaultProbability** (Time t) const override

  Calculate the default probability.
- virtual const Time **defaultTime** (const Path &path) const override

  Return the default time of the path based on Monte Carlo method.
- virtual boost::shared_ptr< StochasticProcess1D > **process** () const override

  Return a geometric brownian motion with zero drift and zero volatility.

5.21.1 Detailed Description

Intensity default model based on deterministic default intensity.

The model uses deterministic function to describe defalut intensity, it is boostrapped from CDS data.

The documentation for this class was generated from the following files:

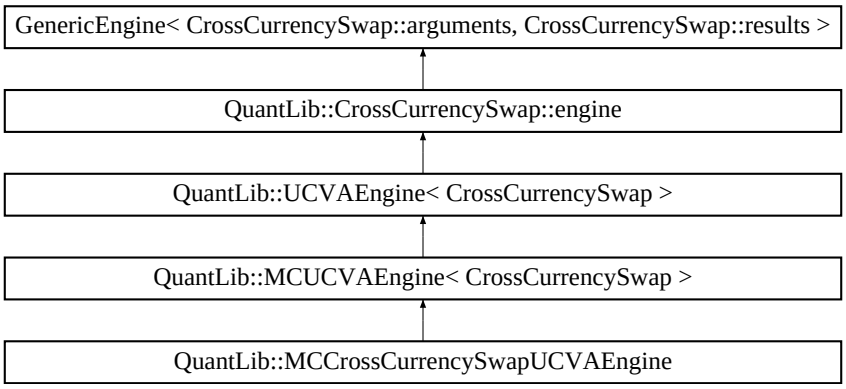
- **deterministicdefaultmodel.hpp**
- **deterministicdefaultmodel.cpp**

5.22 QuantLib::CrossCurrencySwap::engine Class Reference

base class for cross currency swap pricing engine

`#include <crosscurrencyswap.hpp>`

Inheritance diagram for QuantLib::CrossCurrencySwap::engine:



5.22.1 Detailed Description

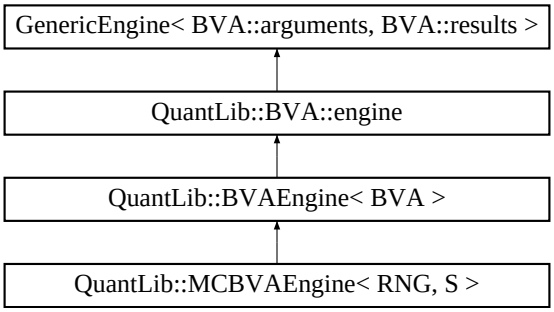
base class for cross currency swap pricing engine

The documentation for this class was generated from the following file:

- **crosscurrencyswap.hpp**

5.23 QuantLib::BVA::engine Class Reference

Inheritance diagram for QuantLib::BVA::engine:



The documentation for this class was generated from the following file:

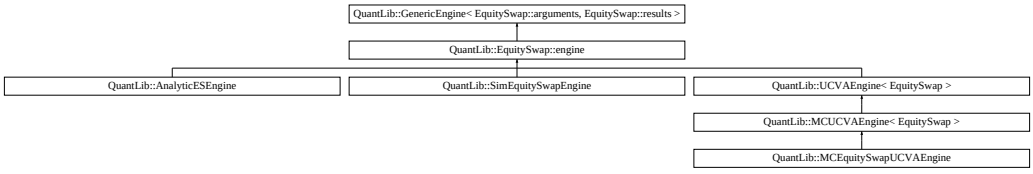
- **bva.hpp**

5.24 QuantLib::EquitySwap::engine Class Reference

equity option engine base class

`#include <equityswap.hpp>`

Inheritance diagram for QuantLib::EquitySwap::engine:



5.24.1 Detailed Description

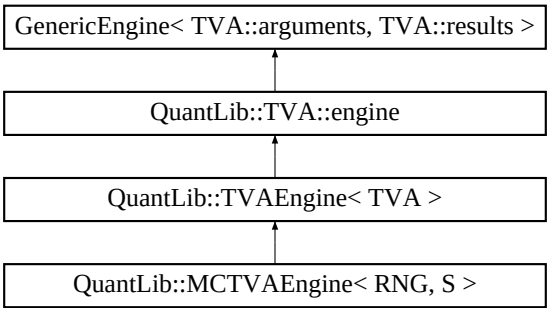
equity option engine base class

The documentation for this class was generated from the following file:

- **equityswap.hpp**

5.25 QuantLib::TVA::engine Class Reference

Inheritance diagram for QuantLib::TVA::engine:



The documentation for this class was generated from the following file:

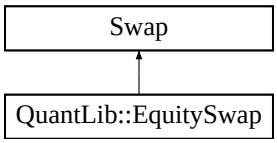
- **tva.hpp**

5.26 QuantLib::EquitySwap Class Reference

equity swap class

`#include <equityswap.hpp>`

Inheritance diagram for QuantLib::EquitySwap:



Classes

- class **arguments**

  Arguments for equity swap calculation
- class **engine**

  equity option engine base class
- class **results**

  Results class of equity swap

Public Types

- enum **Type** { **Receiver** = -1, **Payer** = 1 }

Public Member Functions

- size_t **Count** () const
  return the number of payoffs
- Real **getMaturity** () const
  return the maturity
- Real **getDividendYield** () const
  return the continuous dividend yield of the stock
- Real **getSpotPrice** () const
  return the current spot price of the stock
- Real **getVolatility** () const
  return the constant volatility of the stock
- Calendar **getCalendar** () const
  return the calendar of used in the calculation

Destructor

- ∼**EquitySwap** ()

Constructors

- **EquitySwap** (**Type** type, QuantLib::Date startdate_, QuantLib::Date referencedate_, QuantLib::Real startprice_, QuantLib::Real spotprice_, QuantLib::Integer amount_, QuantLib::Real cumulativecoupoon↩_, QuantLib::Real cumulativedividend_, QuantLib::Rate fixedrate_, QuantLib::Rate dividend_, Quant↩Lib::Time maturity, QuantLib::Real sigma_, boost::shared_ptr< QuantLib::YieldTermStructure > discountcurve_, const QuantLib::Frequency &freq=QuantLib::Semiannual, const QuantLib::Business↩DayConvention &busDayConvention=QuantLib::Following, const QuantLib::DateGeneration::Rule &rule↩_=QuantLib::DateGeneration::Backward, const QuantLib::Calendar &calendar=TARGET())
- **EquitySwap** (const **EquitySwap** &)

5.26.1 Detailed Description

equity swap class

This class defines the instrument equity swap Equity swap is a contract between two parties, which one party pays fixed rate coupon on the notional, the other party pays dividend return from the stock. At maturity, the notional is exchanged with the stock. For simplicity, the dividend yield and the volatility of the stock is supposed constant.

See **Counterparty** (p. 27) Credit Risk, Collateral and Funding(Brigo 2013) page 169 for more detailed descriptions.

The two legs of the equity swap is initialized using the private "initialize" method, whenever an equity swap is constructed.

5.26.2 Member Enumeration Documentation

5.26.2.1 enum QuantLib::EquitySwap::Type

An enum type. The documentation block cannot be put after the enum!

Enumerator

Receiver receive fixed coupon

Payer pay fixed coupon

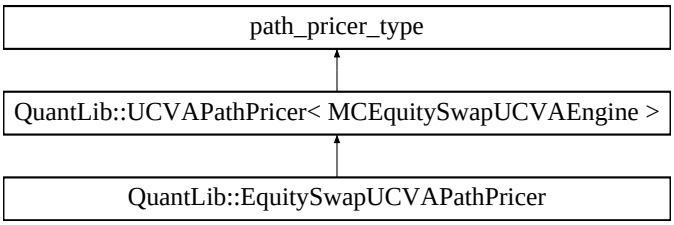
The documentation for this class was generated from the following files:

- **equityswap.hpp**
- **equityswap.cpp**

5.27 QuantLib::EquitySwapUCVAPathPricer Class Reference

UCVA Path Pricer for equity swap.

Inheritance diagram for QuantLib::EquitySwapUCVAPathPricer:



Public Member Functions

Constructors & Destructors

- **EquitySwapUCVAPathPricer** (const boost::shared_ptr< const **MCEquitySwapUCVAEngine** > &engine)
- ~**EquitySwapUCVAPathPricer** ()

Protected Member Functions

- virtual Real **defaultNPV** (const MultiPath &path, Time defaultTime) const

  Calculate the NPV of the remaining cash flow at default time τ.

Additional Inherited Members

5.27.1 Detailed Description

UCVA Path Pricer for equity swap.

The documentation for this class was generated from the following file:

- **mcequityswapucvaengine.cpp**

5.28 QuantLib::ExactSimulation< SIM, RNG, S, C > Class Template Reference

The documentation for this class was generated from the following file:

- **exactsimulation.hpp**

5.29 QuantLib::ExposureModel Class Reference

Public Member Functions

- virtual Real **NPV** (const MultiPath &path, Time defaultTime) const =0
- virtual std::vector< boost::shared_ptr< StochasticProcess1D > > **processes** () const =0

The documentation for this class was generated from the following file:

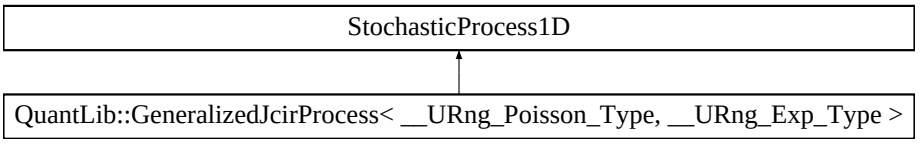
- exposuremodel.hpp

5.30 QuantLib::GeneralizedJcirProcess< __URng_Poisson_Type, __URng_Exp_Type > Class Template Reference

Jump diffusion CIR process class.

`#include <jcirprocess.hpp>`

Inheritance diagram for QuantLib::GeneralizedJcirProcess< __URng_Poisson_Type, __URng_Exp_Type >:



Public Member Functions

Constructors & Destructors

- **GeneralizedJcirProcess** (Real mean, Real speed, Real jumpIntensity, Real jumpMean, Volatility sigma, Real x0=0.0, const __URng_Poisson_Type &URng_Poisson=__URng_Poisson_Type(Seed↩ Generator::instance().get()), const __URng_Exp_Type &URng_Exp=__URng_Poisson_Type(Seed↩ Generator::instance().get()), const boost::shared_ptr< discretization > &d=boost::make_shared< Euler↩ Discretization >())

Public interface

- const Real **drift** (Time t, Real x) const

  Return the drift term of the process.
- const Real **diffusion** (Time t, Real x) const

  Return the diffusion of the process.
- const Real **evolve** (Time t0, Real x0, Time dt, Real dw) const

  Return simulation value of the process at time t0+dt.
- const Real **x0** () const
- const Real **mean** () const
- const Real **speed** () const
- const Real **jumpIntensity** () const
- const Real **jumpMean** () const
- const Volatility **volatility** () const

5.30.1 Detailed Description

template<typename __URng_Poisson_Type, typename __URng_Exp_Type>class QuantLib::GeneralizedJcirProcess< __U↩ Rng_Poisson_Type, __URng_Exp_Type >

Jump diffusion CIR process class.

This class describes a jump diffusion CIR process governed by

$$dx = a(b - x_t)dt + \sigma\sqrt{x_t}dW_t + dJ_t.$$

It is an extension of the classical CIR model. The jumps of the JCIR are introduced with the help of a pure-jump Levy process J_t.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 template<typename __URng_Poisson_Type , typename __URng_Exp_Type > QuantLib::GeneralizedJcir↩ Process< __URng_Poisson_Type, __URng_Exp_Type >::GeneralizedJcirProcess (Real *mean,* Real *speed,* Real *jumpIntensity,* Real *jumpMean,* Volatility *sigma,* Real *x0 =* 0.0, const __URng_Poisson_Type & *URng_Poisson =* __URng_Poisson_Type(SeedGenerator::instance().get()), const __URng_Exp_Type & *URng_Exp =* __URng_Poisson_Type(SeedGenerator::instance().get()), const boost::shared_ptr< discretization > & *d =* boost::make_shared<EulerDiscretization>())

Creates an instance of CIR process using the four parameters: long-term mean, revert speed, volatility, initial value and the additional jump information

The documentation for this class was generated from the following file:

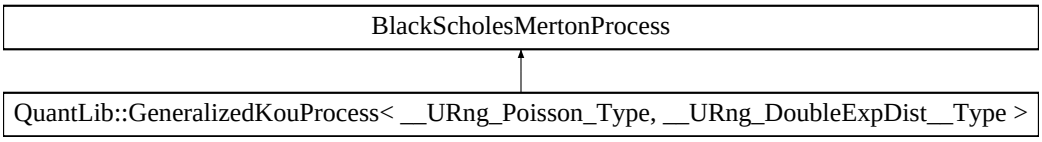
- **jcirprocess.hpp**

5.31 QuantLib::GeneralizedKouProcess< __URng_Poisson_Type, __URng_Double↩ ExpDist__Type > Class Template Reference

Kou process class.

`#include <kouprocess.hpp>`

Inheritance diagram for QuantLib::GeneralizedKouProcess< __URng_Poisson_Type, __URng_DoubleExpDist_↩ _Type >:



Public Member Functions

Constructors & Destructors

- **GeneralizedKouProcess** (const Handle< Quote > &x0, const Handle< YieldTermStructure > &dividendTS, const Handle< YieldTermStructure > &riskFreeTS, const Handle< BlackVolTermStructure > &blackVolTS, const Real jumpIntensity, const Real posProbability, const Real posJumpMean, const Real negJumpMean, const __URng_Poisson_Type &URng_Poisson=__URng_Poisson_Type(Seed↩ Generator::instance().get()), const __URng_DoubleExpDist__Type &URng_DoubleExpDist=__URng_↩ Poisson_Type(SeedGenerator::instance().get()), const boost::shared_ptr< discretization > &d=boost↩ ::shared_ptr< discretization >(new EulerDiscretization))

Public interface

- Real **evolve** (Time t0, Real x0, Time dt, Real dw) const override

  Return simulation value of the process at time t0+dt.
- const Real **jumpIntensity** () const
- const Real **posProbability** () const
- const Real **posJumpMean** () const
- const Real **negJumpMean** () const

5.31.1 Detailed Description

template<typename __URng_Poisson_Type, typename __URng_DoubleExpDist__Type = __URng_Poisson_Type>class QuantLib::GeneralizedKouProcess< __URng_Poisson_Type, __URng_DoubleExpDist__Type >

Kou process class.

This class describes a double exponential jump process, initiated by Steven KOU, which is a compromise between reality and tractability. It gives an explanation of the two empirical phenomena which received much attention in financial markets: the asymmetric leptokurtic feature and the volatility smile. It permits to obtain analytical solutions to the prices of many derivatives : European call and put options; interest rate derivatives, such as swaptions, caps, floors, and bond options; as well as path-dependant options, such as perpetual American options, barrier, and lookback options .

5.31.2 Constructor & Destructor Documentation

5.31.2.1 template<typename __URng_Poisson_Type , typename __URng_DoubleExpDist__Type > QuantLib::GeneralizedKouProcess< __URng_Poisson_Type, __URng_DoubleExpDist__Type >::GeneralizedKouProcess (const Handle< Quote > & *x0,* const Handle< YieldTermStructure > & *dividendTS,* const Handle< YieldTermStructure > & *riskFreeTS,* const Handle< BlackVolTermStructure > & *blackVolTS,* const Real *jumpIntensity,* const Real *posProbability,* const Real *posJumpMean,* const Real *negJumpMean,* const __URng_Poisson_Type & *URng_Poisson* = `__URng_Poisson_Type(SeedGenerator::instance().get())`, const __URng_DoubleExpDist__Type & *URng_DoubleExpDist* = `__URng_Poisson_Type(SeedGenerator::instance().get())`, const boost::shared_ptr< discretization > & *d* = `boost::shared_ptr<discretization>(new EulerDiscretization)`)

Creates an instance of Kou process using the four parameters: initial value, risk-free term structure, dividend term structure, black volatility term structure and the additional jump information

The documentation for this class was generated from the following file:

- **kouprocess.hpp**

5.32 QuantLib::GenericArrayStatistics< StatisticsType > Class Template Reference

Statistics analysis of N-dimensional (sequence) data.

`#include <arraystatistics.hpp>`

Public Types

- typedef StatisticsType **statistics_type**
- typedef Array **value_type**

Public Member Functions

- **GenericArrayStatistics** (Size dimension=0)

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

- Array **mean** () const
- Array **variance** () const
- Array **standardDeviation** () const
- Array **downsideVariance** () const
- Array **downsideDeviation** () const
- Array **semiVariance** () const
- Array **semiDeviation** () const
- Array **errorEstimate** () const
- Array **skewness** () const
- Array **kurtosis** () const
- Array **min** () const
- Array **max** () const
- Array **gaussianPercentile** (Real y) const
- Array **percentile** (Real y) const
- Array **gaussianPotentialUpside** (Real percentile) const
- Array **potentialUpside** (Real percentile) const
- Array **gaussianValueAtRisk** (Real percentile) const
- Array **valueAtRisk** (Real percentile) const
- Array **gaussianExpectedShortfall** (Real percentile) const
- Array **expectedShortfall** (Real percentile) const
- Array **regret** (Real target) const
- Array **gaussianShortfall** (Real target) const
- Array **shortfall** (Real target) const
- Array **gaussianAverageShortfall** (Real target) const
- Array **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_**
- Array **results_**
- Matrix **quadraticSum_**

5.32.1 Detailed Description

template<class StatisticsType>class QuantLib::GenericArrayStatistics< 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.

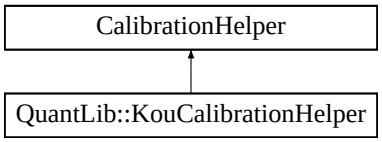
Test the correctness of the returned values is tested by checking them against numerical calculations.

The documentation for this class was generated from the following file:

- arraystatistics.hpp

5.33 QuantLib::KouCalibrationHelper Class Reference

Inheritance diagram for QuantLib::KouCalibrationHelper:



The documentation for this class was generated from the following file:

- koucalibrationhelper.hpp

5.34 QuantLib::KouHelper Class Reference

Kou process helper.

`#include <analytickoueuropeanengine.hpp>`

Static Public Member Functions

- static const double **Gamma** (double mu, double sigma, double lambda, double p, double eta1, double eta2, double a, double T, double tolerance)

5.34.1 Detailed Description

Kou process helper.

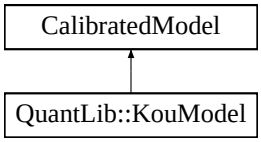
support functions to analytically calculate the option price under kou process assumption

The documentation for this class was generated from the following files:

- **analytickoueuropeanengine.hpp**
- **analytickoueuropeanengine.cpp**

5.35 QuantLib::KouModel Class Reference

Inheritance diagram for QuantLib::KouModel:



The documentation for this class was generated from the following file:

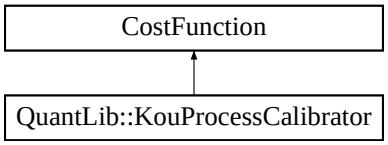
- koumodel.hpp

5.36 QuantLib::KouProcessCalibrator Class Reference

A Kou process calibrator class which is used to do calibration using sets of vanilla options.

`#include <kouprocesscalibrator.hpp>`

Inheritance diagram for QuantLib::KouProcessCalibrator:



Public Member Functions

Constructors & Destructors

- **KouProcessCalibrator** (const QuantLib::Date &referenceDate, const QuantLib::Calendar &calendar, const QuantLib::DayCounter &dayCounter, double riskFreeRate, double spotPrice, double dividend, std::shared_ptr< std::vector< double >> strike, std::shared_ptr< std::vector< QuantLib::Date >> maturityDate, std::shared_ptr< std::vector< double >> optionPrice, std::shared_ptr< QuantLib::End↩ Criteria > endcriteria)

Public interface

- QuantLib::BlackConstantVol **getVol** ()

  Return the black volatility term structure.
- double **getPosJumpMean** ()
- double **getNegJumpMean** ()
- double **getPosProb** ()
- double **getJumpIntensity** ()
- void **calibrate** ()

  Calibrate the Kou process.
- QuantLib::Real **value** (const QuantLib::Array &x) const

  Sum the all difference term given from values method.
- QuantLib::Disposable< QuantLib::Array > **values** (const QuantLib::Array &x) const

  Caculate the difference between the model value and market value.

5.36.1 Detailed Description

A Kou process calibrator class which is used to do calibration using sets of vanilla options.

This class is used to calibrate kou process using sets of vanilla options with different tenors on the underlying stock, it is implemented by inheriting the CostFunction and using the Optimizing framework in QuantLib.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 KouProcessCalibrator::KouProcessCalibrator (const QuantLib::Date & *referenceDate,* const QuantLib::Calendar & *calendar,* const QuantLib::DayCounter & *dayCounter,* double *riskFreeRate,* double *spotPrice,* double *dividend,* std::shared_ptr< std::vector< double >> *strike,* std::shared_ptr< std::vector< QuantLib::Date >> *maturityDate,* std::shared_ptr< std::vector< double >> *optionPrice,* std::shared_ptr< QuantLib::EndCriteria > *endcriteria*)

Creates an instance of Kou process calibrator using sets of vanilla options information and some market observable information of underlying stock

The documentation for this class was generated from the following files:

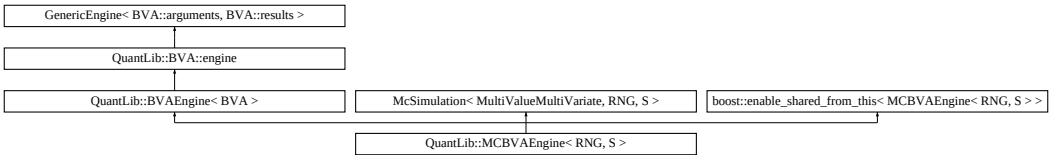
- **kouprocesscalibrator.hpp**
- **kouprocesscalibrator.cpp**

5.37 QuantLib::MCBVAEngine< RNG, S > Class Template Reference

Pricing engine class for **BVA** (p. 20) calculation via Monte Carlo Simulation.

`#include <mcbvaengine.hpp>`

Inheritance diagram for QuantLib::MCBVAEngine< RNG, S >:



Public Types

- typedef **MultiValueMultiVariate**< RNG >::path_type **path_type**
- typedef McSimulation< **MultiValueMultiVariate**, RNG, S >::stats_type **stats_type**
- typedef McSimulation< **MultiValueMultiVariate**, RNG, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< **MultiValueMultiVariate**, RNG, S >::path_generator_type **path_generator_type**

Public Member Functions

Constructors & Destructors

- **MCBVAEngine** (const std::vector< boost::shared_ptr< const **Counterparty** >> &counterparties, const std::vector< boost::shared_ptr< const **MCTVAModel** >> &instrumentModels, const Matrix &correlationMatrix, const Handle< YieldTermStructure > &discountCurve, const boost::shared_ptr< **ShortRateTermStructure** > &shortRateDynamics, Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, Time endTime, BigNatural seed=SeedGenerator::instance().get())
- virtual ~**MCBVAEngine** ()

Public interface

- void **calculate** () const

  calculate bva by monte carlo simulation and store results in pricing engine
- boost::shared_ptr< StochasticProcessArray > **process** () const

  Return shared pointer to stochastic procss array containing underlying process as well as default process.

Protected Member Functions

Protected interface

- virtual TimeGrid **timeGrid** () const

  Return time grid used by Monte Carlo Simulation.
- virtual boost::shared_ptr< path_generator_type > **pathGenerator** () const

  Return shared pointer to path generator who generates sample pathes in simulation.
- virtual boost::shared_ptr< path_pricer_type > **pathPricer** () const
- virtual std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const

  Return shared pointer to stochastic procss array containing underlying process.

Additional Inherited Members

5.37.1 Detailed Description

template<typename RNG = PseudoRandom, typename S = GenericArrayStatistics<GaussianStatistics>>class QuantLib::MC↩ BVAEngine< RNG, S >

Pricing engine class for **BVA** (p. 20) calculation via Monte Carlo Simulation.

This class is the pricing engine for the "instrument" **BVA** (p. 20), responsible for the calculation of **BVA** (p. 20) (includes CVA, DVA) via Monte Carlo Simulation framework. The information of the two counterparties (issuer and investor) and the information of the underlying instruments (single instrument or portfolio) should be giving.

See also

BVA (p. 20)
BVAEngine (p. 21)
Counterparty (p. 27)
MCTVAModel (p. 54)

5.37.2 Constructor & Destructor Documentation

5.37.2.1 **template<typename RNG = PseudoRandom, typename S = GenericArrayStatistics<GaussianStatistics>> QuantLib::MCBVAEngine< RNG, S >::MCBVAEngine (const std::vector< boost::shared_ptr< const Counterparty >> & *counterparties,* const std::vector< boost::shared_ptr< const MCTVAModel >> & *instrumentModels,* const Matrix & *correlationMatrix,* const Handle< YieldTermStructure > & *discountCurve,* const boost::shared_ptr< ShortRateTermStructure > & *shortRateDynamics,* Size *timeSteps,* Size *timeStepsPerYear,* bool *antitheticVariate,* Size *requiredSamples,* Real *requiredTolerance,* Size *maxSamples,* Time *endTime,* BigNatural *seed* = `SeedGenerator::instance().get()`) [inline]**

Creates an instance of MC **BVA** (p. 20) engine using two counterparties, instruments (single instrument or portfolio), correlation matrix of the counterparties and instruments, and the yield term structure (deterministic or stochastic), these information may be got from the "instrument" **BVA** (p. 20).

The documentation for this class was generated from the following file:

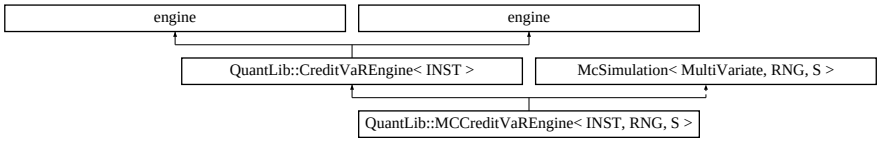
- **mcbvaengine.hpp**

5.38 QuantLib::MCCreditVaREngine< INST, RNG, S > Class Template Reference

Abstract base class for Credit VaR calculation via Monte Carlo Simulation.

`#include <mccreditvarengine.hpp>`

Inheritance diagram for QuantLib::MCCreditVaREngine< INST, RNG, S >:



Public Types

- typedef MultiVariate< RNG >::path_type **path_type**
- typedef McSimulation< MultiVariate, RNG, S >::stats_type **stats_type**
- typedef McSimulation< MultiVariate, RNG, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< MultiVariate, RNG, S >::path_generator_type **path_generator_type**

Public Member Functions

Constructors & Destructors

- **MCCreditVaREngine** (const boost::shared_ptr< const INST > &instrument, const boost::shared_ptr< const **Counterparty** > &issuer, const boost::shared_ptr< const **Counterparty** > &investor, const Matrix &correlation, Time endTime, const Handle< YieldTermStructure > &discountCurve, Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size max↩Samples, BigNatural seed)
- virtual ~**MCCreditVaREngine** ()

Public interface

- void **calculate** () const

  calculate credit VaR by monte carlo simulation and store results in pricing engine
- boost::shared_ptr< StochasticProcessArray > **process** () const

Protected Member Functions

- virtual boost::shared_ptr< path_pricer_type > **pathPricer** () const =0
- virtual std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const =0

Protected interface

- virtual TimeGrid **timeGrid** () const

  Return time grid used by Monte Carlo Simulation.
- virtual boost::shared_ptr< path_generator_type > **pathGenerator** () const

Additional Inherited Members

5.38.1 Detailed Description

template<typename INST, typename RNG = PseudoRandom, typename S = Statistics>class QuantLib::MCCreditVaREngine< INST, RNG, S >

Abstract base class for Credit VaR calculation via Monte Carlo Simulation.

This class is used to calculate Credit VaR via Monte Carlo Simulation.

The documentation for this class was generated from the following file:

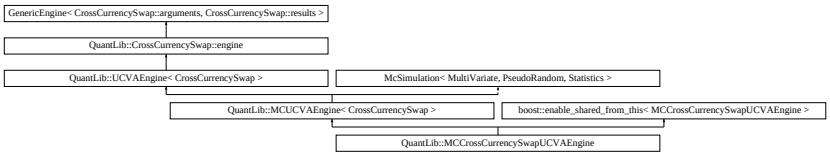
- **mccreditvarengine.hpp**

5.39 QuantLib::MCCrossCurrencySwapUCVAEngine Class Reference

Monte Carlo pricing engine for UCVA calculation of fixed-fixed cross currency swap.

`#include <mccrosscurrencyswapucvaengine.hpp>`

Inheritance diagram for QuantLib::MCCrossCurrencySwapUCVAEngine:



Public Member Functions

Constructors & Destructors

- **MCCrossCurrencySwapUCVAEngine** (const Calendar &calender, const DayCounter &dayCounter, Date referenceDate, const Handle< YieldTermStructure > &riskFreeTermStructure, const boost::shared_ptr< const **CrossCurrencySwap** > &swap, const boost::shared_ptr< const **Counterparty** > &issuer, Rate domesticRate, Real domesticRateSpeed, Real domesticRateMean, Volatility domesticRateVol, Rate foreignRate, Real foreignRateSpeed, Real foreignRateMean, Volatility foreignRateVol, const Matrix &corr, Size timeStepsPerYear=360, bool antitheticVariate=true, Size requiredSamples=50000, Real requiredTolerance=0.0001, Size maxSamples=QL_MAX_INTEGER, BigNatural seed=SeedGenerator::instance().get())
- ~**MCCrossCurrencySwapUCVAEngine** ()

Protected Member Functions

Protected interface

- virtual std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const
  Override function in base class. Return the processes of interest, which will be used by path generator.
- virtual boost::shared_ptr< path_pricer_type > **pathPricer** () const
  Return shared pointer of path pricer.

Additional Inherited Members

5.39.1 Detailed Description

Monte Carlo pricing engine for UCVA calculation of fixed-fixed cross currency swap.

This class is used to calculate UCVA of fixed-fixed cross currency swap via Monte Carlo simulation, giving the information of the **Counterparty** (p. 27) and the underlying instrument.

The kernel calculation part for UCVA is implemented in its pathpricer class.

See also

MCUCVAEngine (p. 56)
CrossCurrencySwapUCVAPathPricer (p. 31)
MCVanillaSwapUCVAEngine (p. 57)
MCEquitySwapUCVAEngine (p. 49)

The documentation for this class was generated from the following files:

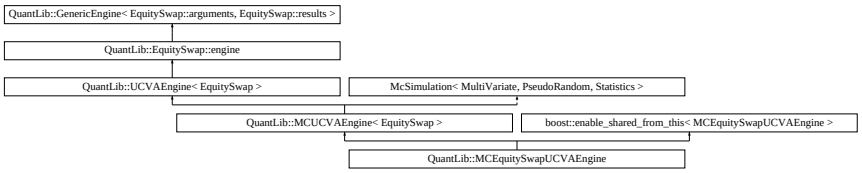
- **mccrosscurrencyswapucvaengine.hpp**
- **mccrosscurrencyswapucvaengine.cpp**

5.40 QuantLib::MCEquitySwapUCVAEngine Class Reference

Monte Carlo pricing engine for UCVA calculation of equity swap.

`#include <mcequityswapucvaengine.hpp>`

Inheritance diagram for QuantLib::MCEquitySwapUCVAEngine:



Public Member Functions

Constructor

- **MCEquitySwapUCVAEngine** (Handle< YieldTermStructure > riskFreeTermStructure, const boost↩ ::shared_ptr< const **EquitySwap** > &swap, const boost::shared_ptr< const **Counterparty** > &issuer, const Matrix &corr, Size timeStepsPerYear=360, bool antitheticVariate=true, Size required↩ Samples=50000, Real requiredTolerance=0.0001, Size maxSamples=QL_MAX_INTEGER, BigNatural seed=QuantLib::SeedGenerator::instance().get())

Destructor

- ~**MCEquitySwapUCVAEngine** ()
- virtual boost::shared_ptr< path_pricer_type > **pathPricer** () const
- std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const

  return the underlying stock process

Additional Inherited Members

5.40.1 Detailed Description

Monte Carlo pricing engine for UCVA calculation of equity swap.

This class is used to calculate UCVA of equity swap via Monte Carlo simulation, giving the information of the **Counterparty** (p. 27) and the underlying instrument.

The kernel calculation part for UCVA is implemented in its pathpricer class.

See also

MCUCVAEngine (p. 56)
EquitySwapUCVAPathPricer (p. 38)
MCCrossCurrencySwapUCVAEngine (p. 48)
MCVanillaSwapUCVAEngine (p. 57)

The documentation for this class was generated from the following files:

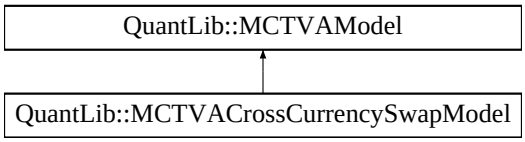
- **mcequityswapucvaengine.hpp**
- **mcequityswapucvaengine.cpp**

5.41 QuantLib::MCTVACrossCurrencySwapModel Class Reference

TVA (p. 62) model for cross currency swap.

`#include <mctvacrosscurrencyswapmodel.hpp>`

Inheritance diagram for QuantLib::MCTVACrossCurrencySwapModel:



Public Member Functions

Constructors & Destructors

- **MCTVACrossCurrencySwapModel** (const Calendar &calendar, const DayCounter &dayCounter, Date referenceDate, Handle< YieldTermStructure > riskFreeTermStructure, const boost::shared_ptr< const **CrossCurrencySwap** > &**instrument**, Rate domesticRate, Real domesticRateSpeed, Real domestic↩ RateMean, Volatility domesticRateVol, Rate foreignRate, Real foreignRateSpeed, Real foreignRateMean, Volatility foreignRateVol)

Public interface

- std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const

  Return the underlying instrument processes of the cross currency swap.
- Real **exposure** (const MultiPath &path, const Time issuerDefaultTime, const Time investorDefaultTime, const Handle< YieldTermStructure > disTS) const

  Return the exposure of cash flow at given time and paths information.

Additional Inherited Members

5.41.1 Detailed Description

TVA (p. 62) model for cross currency swap.

This class implements tva model for cross currency swap, working as tva calculation helper, responsible for providing information of underlying instrument and calculation the exposure of cash flow at given time. It's used in the calculation of CVA, DVA and FVA of underlring instrument under Monte Carlo Simulation framework.

Remarks

The exposure of cross currency swap at given time t, NPV(t), is calculated as the discounted cash flow. The exchange rate $X_s, s \in [t, T]$, using in calculation of NPV(t), is giving as the expectation

$$X_s = X_t e^{\mu(s-t)}$$

See also

MCTVAVanillaSwapModel (p. 55)
MCTVAEquitySwapModel (p. 53)

5.41.2 Constructor & Destructor Documentation

5.41.2.1 QuantLib::MCTVACrossCurrencySwapModel::MCTVACrossCurrencySwapModel (const Calendar & *calendar,* const DayCounter & *dayCounter,* Date *referenceDate,* Handle< YieldTermStructure > *riskFreeTermStructure,* const boost::shared_ptr< const CrossCurrencySwap > & *instrument,* Rate *domesticRate,* Real *domesticRateSpeed,* Real *domesticRateMean,* Volatility *domesticRateVol,* Rate *foreignRate,* Real *foreignRateSpeed,* Real *foreignRateMean,* Volatility *foreignRateVol*) [inline]

Creates an instance of cross currency swap model using the information of the contract including the calendar, day counter, reference date, yield term structure, and the instrument.

The documentation for this class was generated from the following files:

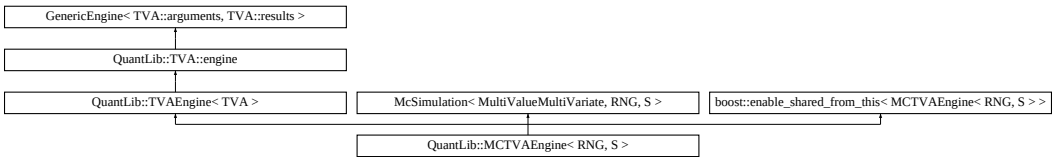
- **mctvacrosscurrencyswapmodel.hpp**
- **mctvacrosscurrencyswapmodel.cpp**

5.42 QuantLib::MCTVAEngine< RNG, S > Class Template Reference

Pricing engine class for **TVA** (p. 62) calculation via Monte Carlo Simulation.

`#include <mctvaengine.hpp>`

Inheritance diagram for QuantLib::MCTVAEngine< RNG, S >:



Public Types

- typedef **MultiValueMultiVariate**< RNG >::path_type **path_type**
- typedef McSimulation< **MultiValueMultiVariate**, RNG, S >::stats_type **stats_type**
- typedef McSimulation< **MultiValueMultiVariate**, RNG, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< **MultiValueMultiVariate**, RNG, S >::path_generator_type **path_generator_type**

Public Member Functions

Constructors & Destructors

- **MCTVAEngine** (const std::vector< boost::shared_ptr< const **Counterparty** >> &counterparties, const std::vector< boost::shared_ptr< const **MCTVAModel** >> &instrumentModels, const Matrix &correlationMatrix, const Handle< YieldTermStructure > &discountCurve, const boost::shared_↩ ptr< **ShortRateTermStructure** > &shortRateDynamics, Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, Time endTime, BigNatural seed=SeedGenerator::instance().get())
- virtual ~**MCTVAEngine** ()

Public interface

- void **calculate** () const
  Calculate bva by monte carlo simulation and store results in pricing engine.
- boost::shared_ptr< StochasticProcessArray > **process** () const
  Return shared pointer to stochastic procss array containing underlying process as well as default process.

Protected Member Functions

Protected interface

- virtual TimeGrid **timeGrid** () const
  Return time grid used by Monte Carlo Simulation.
- virtual boost::shared_ptr< path_generator_type > **pathGenerator** () const
  Return shared pointer to path generator who generates sample pathes in simulation.
- virtual boost::shared_ptr< path_pricer_type > **pathPricer** () const
- virtual std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const
  Return shared pointer to stochastic procss array containing underlying process.

Additional Inherited Members

5.42.1 Detailed Description

template<typename RNG = PseudoRandom, typename S = GenericArrayStatistics<GaussianStatistics>>class QuantLib::MC↩ TVAEngine< RNG, S >

Pricing engine class for **TVA** (p. 62) calculation via Monte Carlo Simulation.

This class is the pricing engine for the "instrument" **TVA** (p. 62), responsible for the calculation of **TVA** (p. 62) (includes CVA, DVA and FVA) via Monte Carlo Simulation framework. The information of the two counterparties (issuer and investor) and the information of the underlying instruments (single instrument or portfolio) should be giving.

See also

TVA (p. 62)
Counterparty (p. 27)
MCTVAModel (p. 54)

5.42.2 Constructor & Destructor Documentation

5.42.2.1 template<typename RNG = PseudoRandom, typename S = GenericArrayStatistics<GaussianStatistics>> QuantLib::MCTVAEngine< RNG, S >::MCTVAEngine (const std::vector< boost::shared_ptr< const Counterparty >> & *counterparties,* const std::vector< boost::shared_ptr< const MCTVAModel >> & *instrumentModels,* const Matrix & *correlationMatrix,* const Handle< YieldTermStructure > & *discountCurve,* const boost::shared_ptr< ShortRateTermStructure > & *shortRateDynamics,* Size *timeSteps,* Size *timeStepsPerYear,* bool *antitheticVariate,* Size *requiredSamples,* Real *requiredTolerance,* Size *maxSamples,* Time *endTime,* BigNatural *seed* = `SeedGenerator::instance().get()`) `[inline]`

Creates an instance of MC **TVA** (p. 62) engine using two counterparties, instruments (single instrument or portfolio), correlation matrix of the counterparties and instruments, and the yield term structure (deterministic or stochastic),

these information may be got from the "instrument" **TVA** (p. 62).

The documentation for this class was generated from the following file:

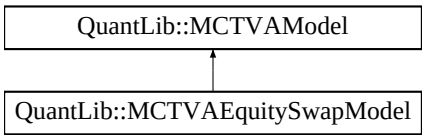
- **mctvaengine.hpp**

5.43 QuantLib::MCTVAEquitySwapModel Class Reference

TVA (p. 62) model for equity swap.

`#include <mctvaequityswapmodel.hpp>`

Inheritance diagram for QuantLib::MCTVAEquitySwapModel:



Public Member Functions

Constructors & Destructors

- **MCTVAEquitySwapModel** (const Calendar &calendar, const DayCounter &dayCounter, Date reference↩ Date, Handle< YieldTermStructure > riskFreeTermStructure, const boost::shared_ptr< const **Equity**↩ **Swap** > &**instrument**)

Public interface

- std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const
  Return the underlying instrument processes of the equity swap.
- Real **exposure** (const MultiPath &path, const Time issuerDefaultTime, const Time investorDefaultTime, const Handle< YieldTermStructure > disTS) const
  Return the exposure of cash flow at given time and paths information.

Additional Inherited Members

5.43.1 Detailed Description

TVA (p. 62) model for equity swap.

This class implements tva model for equity swap, working as tva calculation helper, responsible for providing information of underlying instrument and calculation the exposure of cash flow at given time. It's used in the calculation of CVA, DVA and FVA of underlring instrument under Monte Carlo Simulation framework.

Remarks

The exposure of equity swap at given time t, NPV(t), is calculated as the discounted cash flow. The payoff of floating leg at time t_i is

$$\frac{q(E(S_{t_i}) - E(S_{t_{i-1}}))}{r - q}$$

where $E(S_{t_i}) = S_0 e^{(r-q)t_i}$.

See also

MCTVAVanillaSwapModel (p. 55)
MCTVACrossCurrencySwapModel (p. 50)

5.43.2 Constructor & Destructor Documentation

5.43.2.1 QuantLib::MCTVAEquitySwapModel::MCTVAEquitySwapModel (const Calendar & *calendar,* const DayCounter & *dayCounter,* Date *referenceDate,* Handle< YieldTermStructure > *riskFreeTermStructure,* const boost::shared_ptr< const **EquitySwap** > & *instrument*) [inline]

Creates an instance of equity swap model using the information of the contract including the calendar, day counter, reference date, yield term structure, and the instrument.

The documentation for this class was generated from the following files:

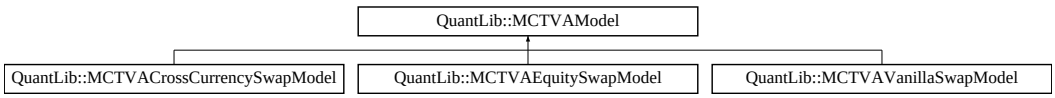
- **mctvaequityswapmodel.hpp**
- **mctvaequityswapmodel.cpp**

5.44 QuantLib::MCTVAModel Class Reference

base class for tva model.

`#include <mctvamodel.hpp>`

Inheritance diagram for QuantLib::MCTVAModel:



Public Member Functions

Constructors & Destructors

- **MCTVAModel** (const Calendar &calendar, const DayCounter &dayCounter, Date referenceDate, Handle< YieldTermStructure > riskFreeTermStructure, const boost::shared_ptr< const Instrument > &**instrument**)
- ~**MCTVAModel** ()

Public interface

- boost::shared_ptr< const Instrument > **instrument** () const

  Return shared pointer to underlying instrument.
- virtual std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const =0

  Return shared pointer to stochastic procss array containing underlying process.
- virtual Real **exposure** (const MultiPath &path, const Time issuerDefaultTime, const Time investorDefault↩ Time, const Handle< YieldTermStructure > disTS) const =0

  Calculate the exposure of cash flow at given time and multi-paths information.

Protected Attributes

- Calendar **calendar_**
- DayCounter **dayCounter_**
- Date **referenceDate_**
- const Handle< YieldTermStructure > **riskFreeTermStructure_**
- const boost::shared_ptr< const Instrument > **instrument_**

5.44.1 Detailed Description

base class for tva model.

This class is a base class for tva model, working as tva calculation helper, responsible for providing information of underlying instrument and calculation the exposure of cash flow at given time

The documentation for this class was generated from the following file:

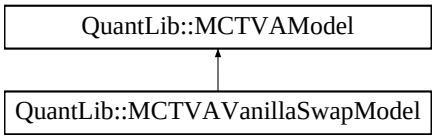
- **mctvamodel.hpp**

5.45 QuantLib::MCTVAVanillaSwapModel Class Reference

TVA (p. 62) model for vanilla swap.

`#include <mctvavanillaswapmodel.hpp>`

Inheritance diagram for QuantLib::MCTVAVanillaSwapModel:



Public Member Functions

Constructors & Destructors

- **MCTVAVanillaSwapModel** (const Calendar &calendar, const DayCounter &dayCounter, Date reference↩ Date, Handle< YieldTermStructure > riskFreeTermStructure, const boost::shared_ptr< const Cox↩ IngersollRoss > &cirModel, const boost::shared_ptr< VanillaSwap > &**instrument**)

Public interface

- std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const
  Return the underlying instrument processes of the vanilla swap.
- Real **exposure** (const MultiPath &path, const Time issuerDefaultTime, const Time investorDefaultTime, const Handle< YieldTermStructure > disTS) const
  Return the exposure of cash flow at given time and paths information.

Additional Inherited Members

5.45.1 Detailed Description

TVA (p. 62) model for vanilla swap.

This class implements tva model for vanilla swap, working as tva calculation helper, responsible for providing information of underlying instrument and calculation the exposure of cash flow at given time. It's used in the calculation of CVA, DVA and FVA of underlring instrument under Monte Carlo Simulation framework.

Remarks

The exposure of vanilla swap at given time t, NPV(t), is calculated as the discounted cash flow.

See also

MCTVAEquitySwapModel (p. 53)
MCTVACrossCurrencySwapModel (p. 50)

5.45.2 Constructor & Destructor Documentation

5.45.2.1 QuantLib::MCTVAVanillaSwapModel::MCTVAVanillaSwapModel (const Calendar & *calendar,* const DayCounter & *dayCounter,* Date *referenceDate,* Handle< YieldTermStructure > *riskFreeTermStructure,* const boost::shared_ptr< const CoxIngersollRoss > & *cirModel,* const boost::shared_ptr< VanillaSwap > & *instrument*) [inline]

Creates an instance of vanilla swap model using the information of the contract including the calendar, day counter, reference date, yield term structure, and the instrument.

The documentation for this class was generated from the following files:

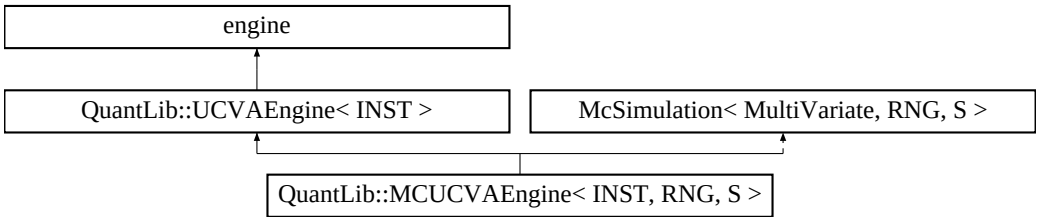
- **mctvavanillaswapmodel.hpp**
- **mctvavanillaswapmodel.cpp**

5.46 QuantLib::MCUCVAEngine< INST, RNG, S > Class Template Reference

Abstract base class for UCVA calculation via Monte Carlo Simulation.

`#include <mcucvaengine.hpp>`

Inheritance diagram for QuantLib::MCUCVAEngine< INST, RNG, S >:



Public Types

- typedef MultiVariate< RNG >::path_type **path_type**
- typedef McSimulation< MultiVariate, RNG, S >::stats_type **stats_type**
- typedef McSimulation< MultiVariate, RNG, S >::path_pricer_type **path_pricer_type**
- typedef McSimulation< MultiVariate, RNG, S >::path_generator_type **path_generator_type**

Public Member Functions

Constructors & Destructors

- **MCUCVAEngine** (const boost::shared_ptr< const INST > &instrument, const boost::shared_ptr< const **Counterparty** > &issuer, const Matrix &correlation, Time endTime, const Handle< YieldTermStructure > &discountCurve, Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- virtual ~**MCUCVAEngine** ()

Public interface

- void **calculate** () const

  calculate UCVA by monte carlo simulation and store results in pricing engine
- boost::shared_ptr< StochasticProcessArray > **process** () const

  Return shared pointer to stochastic procss array containing underlying process as well as default process.

Protected Member Functions

- virtual boost::shared_ptr< path_pricer_type > **pathPricer** () const =0
- virtual std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const =0

Protected interface

- virtual TimeGrid **timeGrid** () const
  Return time grid used by Monte Carlo Simulation.
- virtual boost::shared_ptr< path_generator_type > **pathGenerator** () const
  Return shared pointer to path generator who generates sample pathes in simulation.

Additional Inherited Members

5.46.1 Detailed Description

template<typename INST, typename RNG = PseudoRandom, typename S = Statistics>class QuantLib::MCUCVAEngine< INST, RNG, S >

Abstract base class for UCVA calculation via Monte Carlo Simulation.

This abstract base class is the base Monte Carlo pricing engine class of instrument for UCVA calculation.

The kernel calculation part is its path pricer class.

The user who wants to calculate the UCVA of a instrument should implement the UCVA engine class and the path pricer class for the instrument.

See also

UCVAPathPricer (p. 66)

The documentation for this class was generated from the following file:

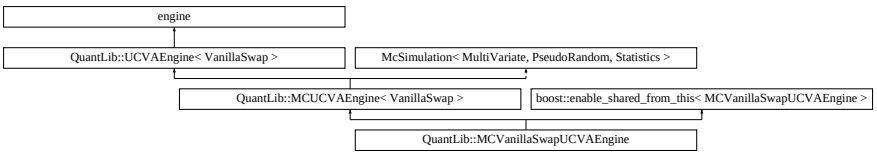
- **mcucvaengine.hpp**

5.47 QuantLib::MCVanillaSwapUCVAEngine Class Reference

Monte Carlo pricing engine for UCVA calculation of vanilla swap.

`#include <mcvanillaswapucvaengine.hpp>`

Inheritance diagram for QuantLib::MCVanillaSwapUCVAEngine:



Public Member Functions

Constructors & Destructors

- **MCVanillaSwapUCVAEngine** (const Calendar &calender, const DayCounter &dayCounter, Date referenceDate, const Handle< YieldTermStructure > &riskFreeTermStructure, const boost::shared_ptr< const VanillaSwap > &swap, const boost::shared_ptr< const **Counterparty** > &issuer, const

boost::shared_ptr< const CoxIngersollRoss > &cirModel, const Matrix &corr, Size timeStepsPer↩ Year=360, bool antitheticVariate=true, Size requiredSamples=50000, Real requiredTolerance=0.0001, Size maxSamples=QL_MAX_INTEGER, BigNatural seed=SeedGenerator::instance().get())
- **MCVanillaSwapUCVAEngine::~MCVanillaSwapUCVAEngine** ()

Public interface

- const boost::shared_ptr< const CoxIngersollRoss > & **Model** () const
  Return the cir model.

Protected Member Functions

Protected interface

- std::vector< boost::shared_ptr< StochasticProcess1D > > **instrumentProcess** () const
  Return the processes of instruments.
- virtual boost::shared_ptr< path_pricer_type > **pathPricer** () const
  Return the shared pointer of path_pricer_type.

Additional Inherited Members

5.47.1 Detailed Description

Monte Carlo pricing engine for UCVA calculation of vanilla swap.

This class is used to calculate UCVA of vanilla swap via Monte Carlo simulation, giving the information of the **Counterparty** (p. 27) and the underlying instrument.

The kernel calculation part for UCVA is implemented in its pathpricer class.

See also

MCUCVAEngine (p. 56)
VanillaSwapUCVAPathPricer (p. 67)
MCCrossCurrencySwapUCVAEngine (p. 48)
MCEquitySwapUCVAEngine (p. 49)

The documentation for this class was generated from the following files:

- **mcvanillaswapucvaengine.hpp**
- **mcvanillaswapucvaengine.cpp**

5.48 QuantLib::MultiValueMultiVariate< RNG > Struct Template Reference

Traits class for multi-state Monte Carlo Simulation.

`#include <multivaluemctraits.hpp>`

Public Types

- enum { **allowsErrorEstimate** = RNG::allowsErrorEstimate }
- typedef RNG **rng_traits**
- typedef MultiPath **path_type**
- typedef PathPricer< path_type, Array > **path_pricer_type**
- typedef RNG::rsg_type **rsg_type**
- typedef MultiPathGenerator< rsg_type > **path_generator_type**

5.48.1 Detailed Description

template<class RNG>struct QuantLib::MultiValueMultiVariate< RNG >

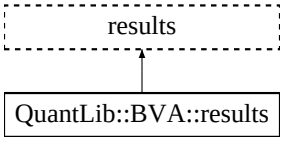
Traits class for multi-state Monte Carlo Simulation.

The documentation for this struct was generated from the following file:

- **multivaluemctraits.hpp**

5.49 QuantLib::BVA::results Class Reference

Inheritance diagram for QuantLib::BVA::results:



Public Member Functions

- void **reset** ()

Public Attributes

- Real **CVA**
- Real **DVA**

The documentation for this class was generated from the following file:

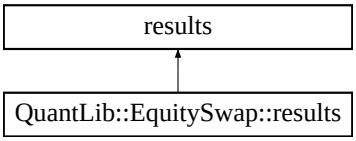
- **bva.hpp**

5.50 QuantLib::EquitySwap::results Class Reference

Results class of equity swap

`#include <equityswap.hpp>`

Inheritance diagram for QuantLib::EquitySwap::results:



5.50.1 Detailed Description

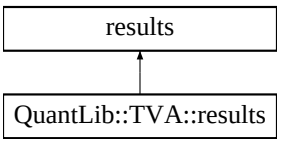
Results class of equity swap

The documentation for this class was generated from the following file:

- **equityswap.hpp**

5.51 QuantLib::TVA::results Class Reference

Inheritance diagram for QuantLib::TVA::results:



Public Member Functions

- virtual void **reset** ()

Public Attributes

- Real **CVA**
- Real **DVA**
- Real **FVA**

The documentation for this class was generated from the following file:

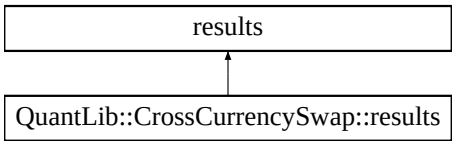
- **tva.hpp**

5.52 QuantLib::CrossCurrencySwap::results Class Reference

Results for cross currency swap calculation

`#include <crosscurrencyswap.hpp>`

Inheritance diagram for QuantLib::CrossCurrencySwap::results:



5.52.1 Detailed Description

Results for cross currency swap calculation

The documentation for this class was generated from the following file:

- **crosscurrencyswap.hpp**

5.53 RiskAnalysisTool::Calculation::Sample::SampleClass Class Reference

Public Member Functions

- double **Sum** (const std::shared_ptr< const std::vector< double >> &data)

The documentation for this class was generated from the following files:

- SampleClass.hpp
- SampleClass.cpp

5.54 RiskAnalysisTool::Calculation::sealed Class Reference

Public Member Functions

- **ComputeEngine** ()

The documentation for this class was generated from the following file:

- ComputeEngine.cpp

5.55 QuantLib::ShortRateTermStructure Class Reference

Interest-rate term structure class based on short-rate model.

`#include <shortratetermstructure.hpp>`

Public Member Functions

Constructors & Destructors

Creates an instance of yield term structure giving the model

- **ShortRateTermStructure** (const Calendar &calendar, const DayCounter &daycounter, const boost↩ ::shared_ptr< OneFactorAffineModel > &model)
- ∼**ShortRateTermStructure** ()

Term structure

These methods return the yield term structure from reference date given max date or max time and the short rate at reference date.

- boost::shared_ptr< TermStructure > **GetTermStructure** (Date referenceDate, Date maxDate, Real rate, Frequency freq=Quarterly)
- boost::shared_ptr< TermStructure > **GetTermStructure** (Date referenceDate, Time Tmax, Real rate, Frequency freq=Quarterly)

5.55.1 Detailed Description

Interest-rate term structure class based on short-rate model.

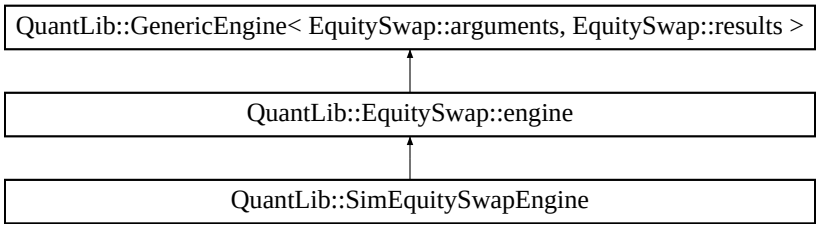
This class is used to generate yield term structure based on the given One factor & Affine short-rate model. The discount factors in the term structure are all expectations and calculated by the corresponding formula of given model.

The documentation for this class was generated from the following files:

- **shortratetermstructure.hpp**
- **shortratetermstructure.cpp**

5.56 QuantLib::SimEquitySwapEngine Class Reference

Inheritance diagram for QuantLib::SimEquitySwapEngine:



Public Member Functions

- **SimEquitySwapEngine** (size_t n_=1000)
- void **calculate** () const

The documentation for this class was generated from the following files:

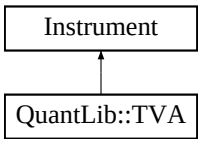
- simequityswapengine.hpp
- simequityswapengine.cpp

5.57 QuantLib::TVA Class Reference

Total value adjustment manager class.

`#include <tva.hpp>`

Inheritance diagram for QuantLib::TVA:



Classes

- class **arguments**
- class **engine**
- class **results**

Public Member Functions

- void **setupArguments** (PricingEngine::arguments ∗) const
- void **fetchResults** (const PricingEngine::results ∗) const
- bool **isExpired** () const
- const Real **CVA** () const
- const Real **DVA** () const
- const Real **FVA** () const

Protected Attributes

- Real **CVA_**
- Real **DVA_**
- Real **FVA_**

5.57.1 Detailed Description

Total value adjustment manager class.

This class inherits from Instrument class, responsible for calculation total value adjustment (includes CVA, DV↩ A, FVA) of a single instrument or portfolio. The arguments are the information of two counterparties (issuer and investor) using the **Counterparty** (p. 27) class and the information of underlying instrument or portfolio using their corresponding model class. The Pricingengine uses tvaengine based on Monte Carlo simulation framework. The results are including CVA, DVA and FVA.

See also

counterparty.hpp (p. 75)
tvamodel.hpp
tvaengine.hpp (p. 89)

The documentation for this class was generated from the following file:

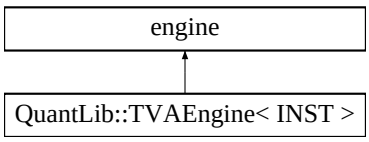
- **tva.hpp**

5.58 QuantLib::TVAEngine< INST > Class Template Reference

Base class for bva engine.

`#include <tvaengine.hpp>`

Inheritance diagram for QuantLib::TVAEngine< INST >:



Public Member Functions

Constructors & Destructors

- **TVAEngine** (Time endTime, const std::vector< boost::shared_ptr< const **Counterparty** >> &counterparties, const std::vector< boost::shared_ptr< const **MCTVAModel** >> &tvaModels, const Handle< YieldTermStructure > &discountCurve, const boost::shared_ptr< **ShortRateTermStructure** > &short↩ RateDynamics)
- virtual ~**TVAEngine** ()

Inspector

- std::vector< boost::shared_ptr< const **Counterparty** > > **counterparties** () const
- const Time **endTime** () const
- const Handle< YieldTermStructure > & **discountCurve** () const
- const boost::shared_ptr< **ShortRateTermStructure** > & **shortRateDynamics** () const
- const std::vector< boost::shared_ptr< const **MCTVAModel** > > & **models** () const

Protected Attributes

- std::vector< boost::shared_ptr< const **Counterparty** > > **counterparties_**
- Handle< YieldTermStructure > **discountCurve_**
- boost::shared_ptr< **ShortRateTermStructure** > **shortRateDynamics_**
- std::vector< boost::shared_ptr< const **MCTVAModel** > > **tvaModels_**
- Time **endTime_**

5.58.1 Detailed Description

template<typename INST>class QuantLib::TVAEngine< INST >

Base class for bva engine.

This class is base class for tva engine, which is used to set some basic information. Each derived class should implement its own method to calculate bva.

5.58.2 Constructor & Destructor Documentation

5.58.2.1 template<typename INST> QuantLib::TVAEngine< INST >::TVAEngine (Time *endTime,* const std::vector< boost::shared_ptr< const Counterparty >> & *counterparties,* const std::vector< boost::shared_ptr< const MCTVAModel >> & *tvaModels,* const Handle< YieldTermStructure > & *discountCurve,* const boost::shared_ptr< ShortRateTermStructure > & *shortRateDynamics*) [inline]

Create an instance of **TVA** (p. 62) Engine using the information of the two counterparties using **Counterparty** (p. 27) class and underlying instruments using instrument model class.

The documentation for this class was generated from the following file:

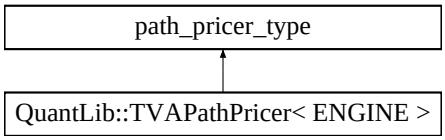
- **tvaengine.hpp**

5.59 QuantLib::TVAPathPricer< ENGINE > Class Template Reference

Path pricer class used to calculate the **TVA** (p. 62) on each Multi-path for **TVAEngine** (p. 63) class.

`#include <tvapathpricer.hpp>`

Inheritance diagram for QuantLib::TVAPathPricer< ENGINE >:



Public Types

- typedef ENGINE::path_pricer_type::argument_type **argument_type**
- typedef ENGINE::path_pricer_type::result_type **result_type**

Public Member Functions

Constructors

- **TVAPathPricer** (const boost::shared_ptr< const ENGINE > &engine)

Destructors

- ~**TVAPathPricer** ()

Operator overload

- result_type **operator()** (const argument_type &multiPath) const
  *Calculate the **TVA** (p. 62) (includes CVA, DVA and FVA) given the Multi-path.*

Protected Attributes

- const boost::weak_ptr< const ENGINE > **engine_**

5.59.1 Detailed Description

template<typename ENGINE>class QuantLib::TVAPathPricer< ENGINE >

Path pricer class used to calculate the **TVA** (p. 62) on each Multi-path for **TVAEngine** (p. 63) class.

This class is the core part for **TVAEngine** (p. 63) class, which is used to calculate the **TVA** (p. 62) (includes CVA, DVA and FVA) on each Multi-path given the instruments (single instrument or portfolio), based on the definition of **TVA** (p. 62) (includes CVA, DVA and FVA) in Brigo's book: "Counterparty Credit Risk, Collateral and Funding".

If the underlying is a portfolio, netting is considered.

See also

MCTVAEngine (p. 51)

The documentation for this class was generated from the following file:

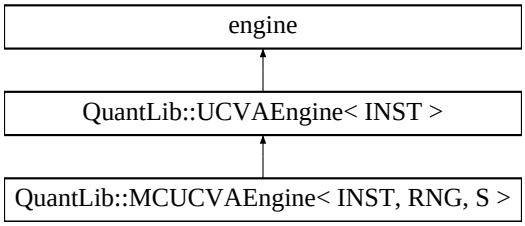
- **tvapathpricer.hpp**

5.60 QuantLib::UCVAEngine< INST > Class Template Reference

Abstract base class for ucva engine.

`#include <ucvaengine.hpp>`

Inheritance diagram for QuantLib::UCVAEngine< INST >:



Public Member Functions

Constructors & Destructors

- **UCVAEngine** (const boost::shared_ptr< const INST > instrument, const boost::shared_ptr< const **Counterparty** > issuer, const Time &endTime, const Handle< YieldTermStructure > &discountCurve)

  Create an instance of UCVA Engine using the information of counterparty and underlying instrument.
- virtual ~**UCVAEngine** ()

Inspector

- const Handle< YieldTermStructure > & **discountCurve** () const
- Time **endTime** () const
- const boost::shared_ptr< const **Counterparty** > & **issuer** () const
- const boost::shared_ptr< const INST > & **instrument** () const

Protected Attributes

- const Time **endTime_**
- const boost::shared_ptr< const INST > **instrument_**
- const boost::shared_ptr< const **Counterparty** > **issuer_**
- const Handle< YieldTermStructure > **discountCurve_**

5.60.1 Detailed Description

template<typename INST>class QuantLib::UCVAEngine< INST >

Abstract base class for ucva engine.

This class is base class for ucva engine, which just contains information as least as possible. Each derived class should implement its own method to calculate ucva.

The documentation for this class was generated from the following file:

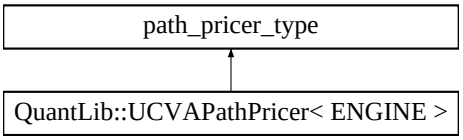
- **ucvaengine.hpp**

5.61 QuantLib::UCVAPathPricer< ENGINE > Class Template Reference

Abstract base class used to calculate cash flow at default time of each path under the setting of UCVA.

`#include <ucvapathpricer.hpp>`

Inheritance diagram for QuantLib::UCVAPathPricer< ENGINE >:



Public Types

- typedef ENGINE::path_pricer_type::argument_type **argument_type**
- typedef ENGINE::path_pricer_type::result_type **result_type**

Public Member Functions

Destructors

- ~**UCVAPathPricer** ()

Operator overload

- result_type **operator()** (const argument_type &multiPath) const
  Calculate the UCVA given the Multi-path.

Protected Member Functions

Constructors

- **UCVAPathPricer** (const boost::shared_ptr< const ENGINE > &engine)

Protected interface

- const boost::weak_ptr< const ENGINE > **engine_**
  Return weak pointer of pricing engine.
- virtual Real **defaultNPV** (const MultiPath &path, Time defaultTime) const =0
  Return default NPV at each path, must be implemented in derived class.

5.61.1 Detailed Description

template<typename ENGINE>class QuantLib::UCVAPathPricer< ENGINE >

Abstract base class used to calculate cash flow at default time of each path under the setting of UCVA.

This class is used to calculate cash flow at default time of each path. To use this class, derived class must implement the pure virtual function defaultNPV.

See also

MCUCVAEngine (p. 56)

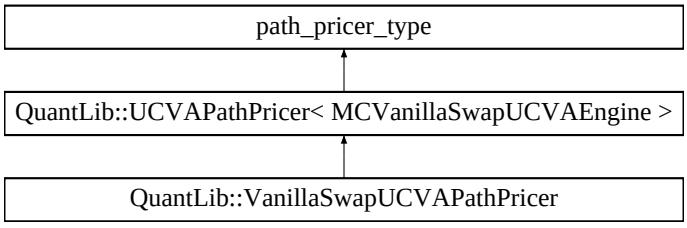
The documentation for this class was generated from the following file:

- **ucvapathpricer.hpp**

5.62 QuantLib::VanillaSwapUCVAPathPricer Class Reference

UCVA Path Pricer for vanilla swap.

Inheritance diagram for QuantLib::VanillaSwapUCVAPathPricer:



Public Member Functions

Constructors & Destructors

- **VanillaSwapUCVAPathPricer** (const boost::shared_ptr< const **MCVanillaSwapUCVAEngine** > &engine)

Protected Member Functions

- virtual Real **defaultNPV** (const MultiPath &path, Time defaultTime) const

  Calculate the NPV of the remaining cash flow at default time τ.

Additional Inherited Members

5.62.1 Detailed Description

UCVA Path Pricer for vanilla swap.

5.62.2 Member Function Documentation

5.62.2.1 virtual Real QuantLib::VanillaSwapUCVAPathPricer::defaultNPV (const MultiPath & *path,* Time *defaultTime*) const [inline], [protected], [virtual]

Calculate the NPV of the remaining cash flow at default time τ.

The calculation method used here is based on CIR model.

Implements **QuantLib::UCVAPathPricer< MCVanillaSwapUCVAEngine >** (p. 67).

The documentation for this class was generated from the following file:

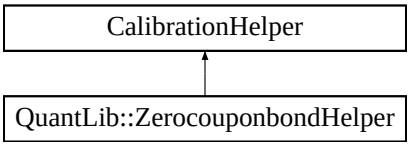
- **mcvanillaswapucvaengine.cpp**

5.63 QuantLib::ZerocouponbondHelper Class Reference

Zero coupon bond calibration helper.

`#include <zerocouponbondhelper.hpp>`

Inheritance diagram for QuantLib::ZerocouponbondHelper:



Public Member Functions

Constructors & Destructors

- **ZerocouponbondHelper** (Natural settlementDays, const Calendar &calendar, Real faceAmount, const Date &maturityDate, BusinessDayConvention paymentConvention, Real redemption, const Date &issueDate, const Handle< Quote > &volatility, const Handle< YieldTermStructure > &termStructure, CalibrationErrorType calibrationErrorType=RelativePriceError)

Creates an instance of zero coupon bond helper using the information of the bond.

Public interface

- virtual void **addTimesTo** (std::list< Time > &times) const
  This method is to be used with tree-based methods.
- virtual Real **modelValue** () const
- virtual Real **blackPrice** (Volatility volatility) const

5.63.1 Detailed Description

Zero coupon bond calibration helper.

One factor affine short rate model calibration helper using zero coupon bond. It can be use to calibrate CIR model and vasicek model, etc.

5.63.2 Member Function Documentation

5.63.2.1 Real ZerocouponbondHelper::blackPrice (Volatility *volatility*) const [virtual]

Calculate black value of zero coupon bond

5.63.2.2 Real ZerocouponbondHelper::modelValue ( ) const [virtual]

Calculate model value of zero coupon bond

The documentation for this class was generated from the following files:

- **zerocouponbondhelper.hpp**
- **zerocouponbondhelper.cpp**

Chapter 6

File Documentation

6.1 analyticequityswapengine.cpp File Reference

```
#include <pch.h>
#include <ql/pricingengines/swap/analyticequityswapengine.hpp>
```

6.2 analyticequityswapengine.hpp File Reference

Analytic engine of equity swap.

```
#include <Calculation/Calculation.h>
#include <ql/instruments/swap/equityswap.hpp>
```

Classes

- class **QuantLib::AnalyticESEngine**

  Analytic engine of equity swap.

6.2.1 Detailed Description

Analytic engine of equity swap.

6.3 analytickoueuropeanengine.cpp File Reference

```
#include <pch.h>
#include <boost/math/distributions/normal.hpp>
#include "analytickoueuropeanengine.hpp"
```

6.4 analytickoueuropeanengine.hpp File Reference

Analytic engine for european option.

```
#include <Calculation/Calculation.h>
#include <boost/shared_ptr.hpp>
#include <ql/instruments/vanillaoption.hpp>
#include <ql/pricingengines/vanilla/mcvanillaengine.hpp>
#include <ql/processes/kouprocess.hpp>
```

Classes

- class **QuantLib::AnalyticKouEuropeanEngine**

  Analytic engine for european option, whose underlying stock process is a kou process.

- class **QuantLib::KouHelper**

  Kou process helper.

6.4.1 Detailed Description

Analytic engine for european option.

6.5 at1pdefaultmodel.cpp File Reference

```
#include <pch.h>
#include <ql/models/default/at1pdefaultmodel.hpp>
```

6.6 at1pdefaultmodel.hpp File Reference

Analytic tractable first passage model.

```
#include <Calculation/Calculation.h>
#include <ql/models/defaultmodel.hpp>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <ql/termstructures/volatility/equityfx/blackvoltermstructure.hpp>
#include <ql/termstructures/defaulttermstructure.hpp>
#include <ql/methods/montecarlo/path.hpp>
#include <ql/processes/blackscholesprocess.hpp>
```

Classes

- class **QuantLib::AT1Pmodel**

  Analytic tractable first default model.

6.6.1 Detailed Description

Analytic tractable first passage model.

6.7 bva.hpp File Reference

Bilateral value adjustment manager class.

```
#include <Calculation/Calculation.h>
#include <boost/shared_ptr.hpp>
#include <ql/types.hpp>
#include <ql/math/matrix.hpp>
#include <ql/instrument.hpp>
#include <ql/pricingengine.hpp>
#include <ql/termstructures/yieldtermstructure.hpp>
#include <ql/credit/counterparty.hpp>
```

Classes

- class **QuantLib::BVA**

  Bilateral value adjustment manager class.
- class **QuantLib::BVA::arguments**
- class **QuantLib::BVA::results**
- class **QuantLib::BVA::engine**

6.7.1 Detailed Description

Bilateral value adjustment manager class.

6.8 bvaengine.hpp File Reference

Base class for bva engine.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <ql/types.hpp>
#include <ql/instrument.hpp>
#include <ql/credit/counterparty.hpp>
#include <ql/models/shortratetermstructure.hpp>
```

Classes

- class **QuantLib::BVAEngine**< **INST** >

  Base class for bva engine.

6.8.1 Detailed Description

Base class for bva engine.

6.9 bvapathpricer.hpp File Reference

Path pricer class used to calculate the BVA on each Multi-path for BVAEngine class.

```
#include <Calculation/Calculation.h>
#include <boost/shared_ptr.hpp>
#include <boost/weak_ptr.hpp>
#include <ql/credit/counterparty.hpp>
```

Classes

- class **QuantLib::BVAPathPricer< ENGINE >**

  *Path pricer class used to calculate the **BVA** (p. 20) on each Multi-path for **BVAEngine** (p. 21) class.*

6.9.1 Detailed Description

Path pricer class used to calculate the BVA on each Multi-path for BVAEngine class.

6.10 cirbondengine.cpp File Reference

```
#include <pch.h>
#include "cirbondengine.hpp"
```

6.11 cirbondengine.hpp File Reference

Bond pricing engine based on CIR model.

```
#include <Calculation/Calculation.h>
#include <ql/instruments/bonds/zerocouponbond.hpp>
#include <ql/pricingengines/genericmodelengine.hpp>
#include <ql/models/shortrate/onefactormodels/coxingersollross.hpp>
```

Classes

- class **QuantLib::CIRBondEngine**

  *A **CIRBondEngine** (p. 24) is a bond pricing engine based on CIR model.*

6.11.1 Detailed Description

Bond pricing engine based on CIR model.

6.12 cirdefaultmodel.cpp File Reference

```
#include <pch.h>
#include "cirdefaultmodel.hpp"
```

6.13 cirdefaultmodel.hpp File Reference

Intensity default model based on the CIR process.

```
#include <Calculation/Calculation.h>
#include <ql/models/defaultmodel.hpp>
```

Classes

- class **QuantLib::CirDefaultModel**

  Intensity default model based on the CIR process.

6.13.1 Detailed Description

Intensity default model based on the CIR process.

6.14 cirprocess.hpp File Reference

CIR process class.

```
#include <Calculation/Calculation.h>
#include <ql/processes/squarerootprocess.hpp>
```

Classes

- class **QuantLib::CIRprocess**

  CIR process class.

6.14.1 Detailed Description

CIR process class.

6.15 counterparty.cpp File Reference

```
#include <pch.h>
#include <ql/credit/counterparty.hpp>
#include <ql/models/default/deterministicdefaultmodel.hpp>
#include <ql/errors.hpp>
#include <boost/cast.hpp>
```

6.16 counterparty.hpp File Reference

Counterparty class.

```
#include <Calculation/Calculation.h>
#include <boost/shared_ptr.hpp>
#include <boost/cast.hpp>
#include <ql/termstructures/credit/piecewisedefaultcurve.hpp>
#include <ql/math/interpolations/backwardflatinterpolation.hpp>
#include <ql/termstructures/yieldtermstructure.hpp>
#include <ql/time/dategenerationrule.hpp>
#include <ql/time/calendars/target.hpp>
#include <ql/models/defaultmodel.hpp>
#include <ql/models/defaultcdshelper.hpp>
```

Classes

- class **QuantLib::Counterparty**

  Counterparty *(*p. *27) class.*

6.16.1 Detailed Description

Counterparty class.

6.17 creditvarpathpricer.hpp File Reference

Abstract base class used to calculate default value of each path.

```
#include <Calculation/Calculation.h>
#include <boost/shared_ptr.hpp>
#include <boost/weak_ptr.hpp>
#include <ql/credit/counterparty.hpp>
#include <ql/methods/montecarlo/multipath.hpp>
```

Classes

- class **QuantLib::CreditVaRPathPricer**< **ENGINE** >

  Abstract base class used to calculate default value of each path.

6.17.1 Detailed Description

Abstract base class used to calculate default value of each path.

This class is used to calculate default value of each path. To use this class, derived class must implement the pure virtual function defaultNPV.

Date

2015-04-15

6.18 crosscurrencyswap.hpp File Reference

fixed-fixed cross currency swap class

```
#include <Calculation/Calculation.h>
#include <ql/instruments/swap.hpp>
#include <ql/currency.hpp>
```

Classes

- class **QuantLib::CrossCurrencySwap**

  fixed-fixed cross currency swap class
- class **QuantLib::CrossCurrencySwap::arguments**

  Arguments for cross currency swap calculation
- class **QuantLib::CrossCurrencySwap::results**

  Results for cross currency swap calculation
- class **QuantLib::CrossCurrencySwap::engine**

  base class for cross currency swap pricing engine

6.18.1 Detailed Description

fixed-fixed cross currency swap class

6.19 defaultcdshelper.hpp File Reference

CDS helpers to calibrate default model.

```
#include <Calculation\Calculation.h>
#include <boost/shared_ptr.hpp>
#include <ql/termstructures/yieldtermstructure.hpp>
#include <ql/models/calibrationhelper.hpp>
```

Classes

- class **QuantLib::DefaultCdsHelper**

  CDS helpers to calibrate default model.

6.19.1 Detailed Description

CDS helpers to calibrate default model.

6.20 defaultmodel.hpp File Reference

Abstract base class for default model.

```
#include <Calculation/Calculation.h>
#include <ql/models/model.hpp>
#include <ql/stochasticprocess.hpp>
```

Classes

- class **QuantLib::DefaultModel**

  abstract base class for default model

6.20.1 Detailed Description

Abstract base class for default model.

6.21 deterministicdefaultmodel.cpp File Reference

```
#include <pch.h>
#include "deterministicdefaultmodel.hpp"
```

6.22 deterministicdefaultmodel.hpp File Reference

intensity default model based on deterministic default intensity

```
#include <ql/models/defaultmodel.hpp>
#include <Calculation/Calculation.h>
```

Classes

- class **QuantLib::DeterministicDefaultModel**

  Intensity default model based on deterministic default intensity.

6.22.1 Detailed Description

intensity default model based on deterministic default intensity

6.23 equityswap.cpp File Reference

```
#include <pch.h>
#include "equityswap.hpp"
```

6.24 equityswap.hpp File Reference

equity swap class

```
#include <Calculation/Calculation.h>
#include <ql/instruments/swap.hpp>
#include <ql/termstructures/yieldtermstructure.hpp>
#include <ql/time/date.hpp>
#include <ql/time/daycounter.hpp>
#include <ql/time/schedule.hpp>
#include <ql/pricingengine.hpp>
#include <boost/shared_ptr.hpp>
#include <ql/processes/blackscholesprocess.hpp>
```

Classes

- class **QuantLib::EquitySwap**

  equity swap class
- class **QuantLib::EquitySwap::arguments**

  Arguments for equity swap calculation
- class **QuantLib::EquitySwap::results**

  Results class of equity swap
- class **QuantLib::EquitySwap::engine**

  equity option engine base class

6.24.1 Detailed Description

equity swap class

6.25 exactsimulation.hpp File Reference

```
#include <ql/types.hpp>
#include <ql/math/copulas/gaussiancopula.hpp>
#include <ql/math/statistics/statistics.hpp>
```

Classes

- class **QuantLib::ExactSimulation**< **SIM, RNG, S, C** >

6.26 jcirprocess.hpp File Reference

JCIR process class.

```
#include <Calculation/Calculation.h>
#include <boost/shared_ptr.hpp>
#include <boost/make_shared.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/processes/eulerdiscretization.hpp>
#include <ql/math/distributions/poissondistribution.hpp>
#include <ql/math/randomnumbers/mt19937uniformrng.hpp>
#include <ql/math/randomnumbers/seedgenerator.hpp>
```

Classes

- class **QuantLib::GeneralizedJcirProcess< __URng_Poisson_Type, __URng_Exp_Type >**

  Jump diffusion CIR process class.

Typedefs

- typedef GeneralizedJcirProcess< MersenneTwisterUniformRng, MersenneTwisterUniformRng > **Quant↩ Lib::JcirProcess**

6.26.1 Detailed Description

JCIR process class.

6.27 kouprocess.hpp File Reference

Kou process class.

```
#include <Calculation/Calculation.h>
#include <ql/math/randomnumbers/seedgenerator.hpp>
#include <ql/processes/blackscholesprocess.hpp>
#include <ql/processes/eulerdiscretization.hpp>
#include <ql/math/randomnumbers/mt19937uniformrng.hpp>
```

Classes

- class **QuantLib::GeneralizedKouProcess< __URng_Poisson_Type, __URng_DoubleExpDist__Type >**

  Kou process class.

Typedefs

- typedef GeneralizedKouProcess< MersenneTwisterUniformRng, MersenneTwisterUniformRng > **Quant↩ Lib::KouProcess**

6.27.1 Detailed Description

Kou process class.

6.28 kouprocesscalibrator.cpp File Reference

```
#include <pch.h>
#include <ql/calibration/kouprocesscalibrator.hpp>
#include <ql/pricingengines/vanilla/analytickoueuropeanengine.hpp>
```

6.29 kouprocesscalibrator.hpp File Reference

Kou process calibrator class.

```
#include <Calculation/Calculation.h>
#include <memory>
#include <vector>
#include <ql/math/optimization/all.hpp>
#include <ql/termstructures/volatility/equityfx/blackconstantvol.hpp>
```

Classes

- class **QuantLib::KouProcessCalibrator**

  A Kou process calibrator class which is used to do calibration using sets of vanilla options.

6.29.1 Detailed Description

Kou process calibrator class.

6.30 mcbvaengine.hpp File Reference

Pricing engine class for BVA calculation via Monte Carlo Simulation.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <boost/make_shared.hpp>
#include <ql/qldefines.hpp>
#include <ql/types.hpp>
#include <ql/math/matrix.hpp>
#include <ql/methods/montecarlo/mctraits.hpp>
#include <ql/math/statistics/arraystatistics.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/pricingengines/mcsimulation.hpp>
#include <ql/credit/counterparty.hpp>
#include <ql/methods/montecarlo/multivaluemctraits.hpp>
#include <ql/pricingengines/credit/mctvamodel.hpp>
#include <ql/instruments/credit/bva.hpp>
#include <ql/pricingengines/credit/bvaengine.hpp>
#include <ql/pricingengines/credit/bvapathpricer.hpp>
#include <boost/enable_shared_from_this.hpp>
```

Classes

- class **QuantLib::MCBVAEngine**< **RNG, S** >

  *Pricing engine class for **BVA** (p. 20) calculation via Monte Carlo Simulation.*

6.30.1 Detailed Description

Pricing engine class for BVA calculation via Monte Carlo Simulation.

6.31 mccreditvarengine.hpp File Reference

Abstract base class for Credit VaR calculation via Monte Carlo Simulation.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <boost/make_shared.hpp>
#include <ql/qldefines.hpp>
#include <ql/types.hpp>
#include <ql/math/matrix.hpp>
#include <ql/methods/montecarlo/mctraits.hpp>
#include <ql/math/statistics/arraystatistics.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/pricingengines/mcsimulation.hpp>
#include <ql/credit/counterparty.hpp>
#include <ql/pricingengines/bvaengine.hpp>
#include <ql/methods/montecarlo/multivaluemctraits.hpp>
#include <ql/pricingengines/credit/creditvarengine.hpp>
```

Classes

- class **QuantLib::MCCreditVaREngine< INST, RNG, S >**

  Abstract base class for Credit VaR calculation via Monte Carlo Simulation.

6.31.1 Detailed Description

Abstract base class for Credit VaR calculation via Monte Carlo Simulation.

This class is used to calculate Credit VaR via Monte Carlo Simulation.

Date

2015-04-15

6.32 mccrosscurrencyswapucvaengine.cpp File Reference

```
#include <pch.h>
#include "mccrosscurrencyswapucvaengine.hpp"
#include "ucvapathpricer.hpp"
```

Classes

- class **QuantLib::CrossCurrencySwapUCVAPathPricer**

  UCVA Path Pricer for fixed-fixed cross currency swap.

6.33 mccrosscurrencyswapucvaengine.hpp File Reference

Monte Carlo pricing engine for UCVA calculation of fixed-fixed cross currency swap.

```
#include <Calculation/Calculation.h>
#include <boost/enable_shared_from_this.hpp>
#include <ql/types.hpp>
#include <ql/time/calendar.hpp>
#include <ql/time/daycounter.hpp>
#include <ql/math/randomnumbers/seedgenerator.hpp>
#include <ql/pricingengines/ucva/mcucvaengine.hpp>
#include <ql/termstructures/yieldtermstructure.hpp>
#include <ql/termstructures/voltermstructure.hpp>
#include <ql/instruments/swap/crosscurrencyswap.hpp>
#include <ql/processes/cirprocess.hpp>
```

Classes

- class **QuantLib::MCCrossCurrencySwapUCVAEngine**

  Monte Carlo pricing engine for UCVA calculation of fixed-fixed cross currency swap.

6.33.1 Detailed Description

Monte Carlo pricing engine for UCVA calculation of fixed-fixed cross currency swap.

6.34 mcequityswapucvaengine.cpp File Reference

```
#include <pch.h>
#include "mcequityswapucvaengine.hpp"
#include <ql/pricingengines/swap/analyticequityswapengine.hpp>
#include <ql/pricingengines/ucva/ucvapathpricer.hpp>
```

Classes

- class **QuantLib::EquitySwapUCVAPathPricer**

  UCVA Path Pricer for equity swap.

6.35 mcequityswapucvaengine.hpp File Reference

Monte Carlo pricing engine for UCVA calculation of equity swap.

```
#include <Calculation/Calculation.h>
#include <boost/enable_shared_from_this.hpp>
#include <ql/instruments/swap/equityswap.hpp>
#include <ql/pricingengines/ucva/mcucvaengine.hpp>
#include <ql/credit/counterparty.hpp>
```

Classes

- class **QuantLib::MCEquitySwapUCVAEngine**

  Monte Carlo pricing engine for UCVA calculation of equity swap.

6.35.1 Detailed Description

Monte Carlo pricing engine for UCVA calculation of equity swap.

6.36 mctvacrosscurrencyswapmodel.cpp File Reference

```
#include <pch.h>
#include "mctvacrosscurrencyswapmodel.hpp"
```

6.37 mctvacrosscurrencyswapmodel.hpp File Reference

TVA model for cross currency swap.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <ql/instruments/swap/crosscurrencyswap.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/pricingengines/credit/mctvamodel.hpp>
#include <ql/termstructures/yieldtermstructure.hpp>
#include <ql/methods/montecarlo/multipath.hpp>
```

Classes

- class **QuantLib::MCTVACrossCurrencySwapModel**

  TVA (p. *62) model for cross currency swap.*

6.37.1 Detailed Description

TVA model for cross currency swap.

6.38 mctvaengine.hpp File Reference

Pricing engine class for TVA calculation via Monte Carlo Simulation.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <boost/make_shared.hpp>
#include <ql/qldefines.hpp>
#include <ql/types.hpp>
#include <ql/math/matrix.hpp>
#include <ql/methods/montecarlo/mctraits.hpp>
#include <ql/math/statistics/arraystatistics.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/pricingengines/mcsimulation.hpp>
#include <ql/credit/counterparty.hpp>
#include <ql/methods/montecarlo/multivaluemctraits.hpp>
#include <ql/pricingengines/credit/mctvamodel.hpp>
#include <ql/instruments/credit/tva.hpp>
#include <ql/pricingengines/credit/tvaengine.hpp>
#include <ql/pricingengines/credit/tvapathpricer.hpp>
#include <boost/enable_shared_from_this.hpp>
```

Classes

- class **QuantLib::MCTVAEngine**< **RNG, S** >

  *Pricing engine class for **TVA** (p. 62) calculation via Monte Carlo Simulation.*

6.38.1 Detailed Description

Pricing engine class for TVA calculation via Monte Carlo Simulation.

6.39 mctvaequityswapmodel.cpp File Reference

```
#include <pch.h>
#include "mctvaequityswapmodel.hpp"
#include <ql/pricingengines/swap/analyticequityswapengine.hpp>
#include <ql/instruments/swap/equityswap.hpp>
```

6.40 mctvaequityswapmodel.hpp File Reference

TVA model for equity swap.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <ql/instruments/swap/equityswap.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/pricingengines/credit/mctvamodel.hpp>
#include <ql/termstructures/yieldtermstructure.hpp>
#include <ql/methods/montecarlo/multipath.hpp>
```

Classes

- class **QuantLib::MCTVAEquitySwapModel**
  TVA *(p. 62) model for equity swap.*

6.40.1 Detailed Description

TVA model for equity swap.

6.41 mctvamodel.hpp File Reference

Base class for tva model.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <ql/instrument.hpp>
#include <ql/pricingengine.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/termstructures/yieldtermstructure.hpp>
#include <ql/methods/montecarlo/multipath.hpp>
#include <ql/time/all.hpp>
```

Classes

- class **QuantLib::MCTVAModel**
  base class for tva model.

6.41.1 Detailed Description

Base class for tva model.

6.42 mctvavanillaswapmodel.cpp File Reference

```
#include <pch.h>
#include "mctvavanillaswapmodel.hpp"
```

6.43 mctvavanillaswapmodel.hpp File Reference

TVA model for vanilla swap.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <ql/instruments/vanillaswap.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/pricingengines/credit/mctvamodel.hpp>
#include <ql/termstructures/yieldtermstructure.hpp>
#include <ql/methods/montecarlo/multipath.hpp>
#include <ql/termstructures/volatility/equityfx/blackconstantvol.hpp>
```

Classes

- class **QuantLib::MCTVAVanillaSwapModel**
  TVA *(p. 62) model for vanilla swap.*

6.43.1 Detailed Description

TVA model for vanilla swap.

6.44 mcucvaengine.hpp File Reference

Abstract base class for UCVA calculation via Monte Carlo Simulation.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <boost/make_shared.hpp>
#include <ql/qldefines.hpp>
#include <ql/types.hpp>
#include <ql/math/matrix.hpp>
#include <ql/methods/montecarlo/mctraits.hpp>
#include <ql/math/statistics/arraystatistics.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/pricingengines/mcsimulation.hpp>
#include <ql/credit/counterparty.hpp>
#include <ql/pricingengines/ucvaengine.hpp>
#include <ql/methods/montecarlo/multivaluemctraits.hpp>
```

Classes

- class **QuantLib::MCUCVAEngine**< **INST, RNG, S** >
  Abstract base class for UCVA calculation via Monte Carlo Simulation.

6.44.1 Detailed Description

Abstract base class for UCVA calculation via Monte Carlo Simulation.

6.45 mcvanillaswapucvaengine.cpp File Reference

```
#include <pch.h>
#include "mcvanillaswapucvaengine.hpp"
#include "ucvapathpricer.hpp"
```

Classes

- class **QuantLib::VanillaSwapUCVAPathPricer**
  UCVA Path Pricer for vanilla swap.

6.46 mcvanillaswapucvaengine.hpp File Reference

Monte Carlo pricing engine for UCVA calculation of vanilla swap.

```
#include <Calculation/Calculation.h>
#include <boost/enable_shared_from_this.hpp>
#include <ql/math/randomnumbers/seedgenerator.hpp>
#include <ql/instruments/vanillaswap.hpp>
#include <ql/pricingengines/ucva/mcucvaengine.hpp>
#include <ql/credit/counterparty.hpp>
#include <ql/processes/cirprocess.hpp>
#include <ql/models/shortrate/onefactormodels/coxingersollross.hpp>
```

Classes

- class **QuantLib::MCVanillaSwapUCVAEngine**

  Monte Carlo pricing engine for UCVA calculation of vanilla swap.

6.46.1 Detailed Description

Monte Carlo pricing engine for UCVA calculation of vanilla swap.

6.47 multivaluemctraits.hpp File Reference

Traits class for multi-state Monte Carlo Simulation.

```
#include <Calculation/Calculation.h>
#include <ql/types.hpp>
#include <ql/methods/montecarlo/mctraits.hpp>
```

Classes

- struct **QuantLib::MultiValueMultiVariate**< **RNG** >

  Traits class for multi-state Monte Carlo Simulation.

6.47.1 Detailed Description

Traits class for multi-state Monte Carlo Simulation.

6.48 shortratetermstructure.cpp File Reference

```
#include <pch.h>
#include "shortratetermstructure.hpp"
```

6.49 shortratetermstructure.hpp File Reference

Interest-rate term structure class based on short-rate model.

```
#include <Calculation/Calculation.h>
#include <ql/models/shortrate/onefactormodel.hpp>
#include <ql/termstructure.hpp>
```

Classes

- class **QuantLib::ShortRateTermStructure**

  Interest-rate term structure class based on short-rate model.

6.49.1 Detailed Description

Interest-rate term structure class based on short-rate model.

6.50 tva.hpp File Reference

Total value adjustment manager class.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <ql/types.hpp>
#include <ql/instrument.hpp>
#include <ql/pricingengine.hpp>
#include <ql/credit/counterparty.hpp>
```

Classes

- class **QuantLib::TVA**

  Total value adjustment manager class.
- class **QuantLib::TVA::arguments**
- class **QuantLib::TVA::results**
- class **QuantLib::TVA::engine**

6.50.1 Detailed Description

Total value adjustment manager class.

6.51 tvaengine.hpp File Reference

Base class for tva engine.

```
#include <Calculation/Calculation.h>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <ql/types.hpp>
#include <ql/instrument.hpp>
#include <ql/credit/counterparty.hpp>
#include <ql/models/shortratetermstructure.hpp>
```

Classes

- class **QuantLib::TVAEngine**< **INST** >

  Base class for bva engine.

6.51.1 Detailed Description

Base class for tva engine.

6.52 tvapathpricer.hpp File Reference

Path pricer class used to calculate the TVA on each Multi-path for TVAEngine class.

```
#include <Calculation/Calculation.h>
#include <boost/shared_ptr.hpp>
#include <boost/weak_ptr.hpp>
```

Classes

- class **QuantLib::TVAPathPricer**< **ENGINE** >

  *Path pricer class used to calculate the **TVA*** (p. *62) on each Multi-path for **TVAEngine*** (p. *63) class.*

6.52.1 Detailed Description

Path pricer class used to calculate the TVA on each Multi-path for TVAEngine class.

6.53 ucvaengine.hpp File Reference

Abstract base class for ucva engine.

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

Classes

- class **QuantLib::UCVAEngine**< **INST** >

  Abstract base class for ucva engine.

6.53.1 Detailed Description

Abstract base class for ucva engine.

6.54 ucvapathpricer.hpp File Reference

Abstract base class used to calculate cash flow at default time of each path under the setting of UCVA.

```
#include <Calculation/Calculation.h>
#include <boost/shared_ptr.hpp>
#include <boost/weak_ptr.hpp>
```

Classes

- class **QuantLib::UCVAPathPricer< ENGINE >**

  Abstract base class used to calculate cash flow at default time of each path under the setting of UCVA.

6.54.1 Detailed Description

Abstract base class used to calculate cash flow at default time of each path under the setting of UCVA.

6.55 zerocouponbondhelper.cpp File Reference

```
#include <pch.h>
#include <ql/pricingengines/bond/discountingbondengine.hpp>
#include "zerocouponbondhelper.hpp"
```

6.56 zerocouponbondhelper.hpp File Reference

Zero coupon bond calibration helper.

```
#include <Calculation/Calculation.h>
#include <ql/models/calibrationhelper.hpp>
#include <ql/instruments/bonds/zerocouponbond.hpp>
```

Classes

- class **QuantLib::ZerocouponbondHelper**

  Zero coupon bond calibration helper.

6.56.1 Detailed Description

Zero coupon bond calibration helper.

Index