Credit Analytics User/Developer Guide

Lakshmi Krishnamurthy

v1.4, 1 May 2012

Introduction

Overview

CreditAnalytics is a full-featured financial fixed income credit analytics, trading, and risk library, developed with a special focus towards the needs of the credit products community.

In particular, CreditAnalytics provides analytics to value liquid products (CDS, CDX, CDO, and bonds of all types and variants), liquid and standardized index products, and custom products (single credit forwards and options, and portfolio credit forwards, options, tranches, and other structures).

CreditAnalytics Features

CreditAnalytics captures the valuation, the analytics, and the risk measures calculation for the full set of liquid and semi-liquid credit products. The following is a comprehensive suite of credit products that it handles:

- Single name credit default swaps (with amortizing coupon and notional schedules, and custom recovery schedules)
- Portfolio credit default basket swaps (in particular, it covers the full range of liquid, legacy, and custom CDX/iTRAXX across sectors, LCDX/SovX/Trac-X/LevX/TRACERS, etc), again with variable coupon, notional, and custom recovery schedules – please check the CDX Coverage for the comprehensive list
- Liquid structured credit products – standard CDX NA/EU/Asia/EM IG/HY/XO/HVOL, and sector and ratings based indexes and tranches of all attachments/detachments

- Bespoke structured credit products such as nth-to-default basket and its full set of variants, tranches on the standard indices as well as bespoke baskets, squared/cubed structured variants, in both funded and un-funded forms, as well as deterministic amortizing coupon and principal pay down structures.
- Merton type single name CDS "fundamental" value determination from related equity parameters.
- Comprehensive coverage of all bond types – fixed/floating rate bonds, support for different rate indices and fixings, amortizing/capitalizing bonds, perpetual bonds, European/Bermudan/American embedded option schedules and their variants, fix-to-float on exercise, custom bonds with principal, coupon, and recovery schedules.

Optionally, CreditAnalytics also installs an initial set of bond reference data, bond marks, and IR, treasury, and credit curve closes. Once installed, it can also connect to this database to run analytics and valuation on CDS and bond positions.

CreditAnalytics also contains the following set of curve calibration functionality from market quotes:

- Although not its primary function, CreditAnalytics can bootstrap discount curve from a variety of IR instruments and their quotes – cash/money market instruments, futures (e.g., EDSF), swaps, and treasury quotes.
- Comprehensive calibration routines for single name credit curves such from CDS, bonds, or a mixture of quotes. Inputs can be in one of CDS quoted measures (fixed coupon flat spreads, upfront points, or fair premium/par spreads), one of bond quotes (e.g., yield/Z Spread, asset swap spread, spread to treasury, I Spread, G Spread, Bond Basis, Discount Margin, Option Adjusted Spread, Credit Basis, and other measures to an effective exercise date, or to a specific work-out, or to maturity), or a mixture of any instrument and their corresponding measure.
- For basket products, CreditAnalytics provides a comprehensive set of basket basis calibration routines for the credit indices, correlation calibration routines for standard/bespoke tranches, as well as a suite of advanced correlation calibration functionality (such as multi-factor and random-factor correlation calibration, base

correlation surface set up, and calibration to the Merton model), and conditional/unconditional portfolio loss, pay down, and default distributions over time.

Finally, CreditAnalytics also calculates an elaborate sequence of measures relevant to each product. It is built with an enhanced sequence of standard scenario curves that can be used to generate very customized scenario measures.

Documentation

Apart from the information provided in this user guide, additional documentation of CreditAnalytics functionality and release notes may be found in the CreditAnalytics website. Consult the javadoc for elaborate API usage information.

Installation and Dependencies

The core modules of CreditAnalytics are just two jars:

- **Drip.jar**: This contains the complete suite of the entire CreditAnalytics analytics. Download and install this in your class-path.
- **Ojdbc14.jar**: This contains the Oracle JDBC drivers needed for access to the reference data (optional). Download and install this in your class-path.

Configuration

All the configuration entries are maintained in the provided Config.xml file. Configuration includes information on the location to the day count files, data tables for the bond static reference data, bond closing marks data, and IR/CDS/treasury closing quotes. Each of this information is optional, and the consequence of not providing a configuration file is that the defaults will be used. The following are defaults:

- Day count entries absent – CreditAnalytics uses a comprehensive set of built in day count conventions and holiday calendars across the overwhelming majority of locations, so day count entries are mostly not needed (unless specifically to overwrite the CreditAnalytics 's day count/holiday calendar).
- Bond reference data tables absent – Will not be able to access a bond by its ISIN/CUSIP or any of the identifiers. You will still be able to create user-defined bonds, ranging from simple fixed coupon bonds to complex ones such as amortization/capitalization, floaters, and bonds with embedded options (see next section).
- Closing curve mark tables absent: Will not be able to retrieve closing IR/credit/treasury/FX curve marks and create those curves. Will still be able to calibrate user-defined curves from custom quotes (see next section).

Getting Started

Once you have downloaded and installed CreditAnalytics, the first step is to set up the configuration by altering the entries provided in the Config.xml file (you can rename it, as long you identify the full path in the initializer – see below). Of course, you don't even need a configuration file – in which case the settings default to the values provided in the previous section.

Any of the samples in the examples folder would be the place to start. They contain a comprehensive set of illustrated usage of all the CreditAnalytics API calls.

```
String strConfig;

boolean bFIInit = FI.Init (strConfig);
```

The call FI.Init initializes the CreditAnalytics library - it takes the optional configuration file as an input. If the initialization is not successful, certain CreditAnalytics functionality will not be available, as the sample demonstrates.

DRIP Class Layout and Package Hierarchy

The full set of CreditAnalytics functionality is implemented in a set of 27 packages. The subsequent sections describe each of the package in detail. The layout and the hierarchy of the packages is shown below.

Functional Group	**Functional Sub-group / Package / Module**	**Class**
org.drip.analytics	core	• Serializer
org.drip.analytics	curve	• CreditCurve • DiscountCurve • FXBasis • FXCurve • ZeroCurve
org.drip.analytics	daycount	• ActActDCParams • DateAdjustParams • DayCount • FixedHoliday • FloatingHoliday • Holiday • LocHolidays • StaticHoliday • WeekendHoliday
org.drip.analytics	holset	• LocHolidays
org.drip.analytics	period	• CouponPeriod • Period • ProductCouponPeriodCurveMeasures

		• ProductLossPeriodCurveMeasures
org.drip.calc	output	• BasketOutput • BondCouponMeasures • BondOutput • BondRVMeasures • BondWorkoutMeasures • ComponentOutput
org.drip.chart	surface	• BuildSurface • Contour3D • ContourPlots • GeneratedDelaunaySurface • Histogram • MultiColorScatter • Scatter4D
org.drip.curve	calibration	• Bootstrapable • ComponentCalibrator • ComponentCalibratorBracketing • ComponentCalibratorNR • CreditCurveScenarioGenerator • IRCurveScenarioGenerator
org.drip.feed	historical	• LoadCreditFeeds
org.drip.feed	reference	• LoadBondFeed
org.drip.param	config	• XMLConfigReader
org.drip.param	market	• BasketMarketParamRef • BasketMarketParams • ComponentMarketParamRef • ComponentMarketParams • CreditCurveScenarioContainer • IRCurveScenarioContainer

		• MarketParamsContainer • NodeTweakParams
org.drip.param	pricer	• CalibrationParams • PricerParams
org.drip.param	product	• BondCFTerminationEvent • BondCouponParams • BondCurrencyParams • BondFixedPeriodGenerationParams • BondFloaterParams • BondIdentifierParams • BondIRValuationParams • BondNotionalParams • BondPeriodGenerationParams • BondTSYParams • CDXIdentifier • CompCRValParams • CurrencyPair • EmbeddedOptionSchedule • FactorSchedule • TsyBmkSet
org.drip.param	valuation	• CashSettleParams • NextExerciseInfo • QuotingParams • ValuationParams • WorkoutInfo
org.drip.product	common	• Component • CalibratableComponent
org.drip.product	creator	• BondBuilder • BondProductBuilder

		• BondRefDataBuilder • CDXRefDataBuilder
org.drip.product	credit	• BasketBond • BasketDefaultSwap • BasketProduct • Bond • CreditComponent • CreditDefaultSwap • StandardCDXManager • StandardCDXParams
org.drip.product	fx	• FXForward • FXSpot
org.drip.product	quote	• ComponentQuote • LiveCurve • Quote
org.drip.product	rates	• Cash • EDFuture • InterestRateSwap
org.drip.service	api	• FI
org.drip.service	env	• BondManager • CDSManager • EnvManager • EODCurves • RatesManager • StaticBACurves
org.drip.service	external	• AnalyticsClient • AnalyticsServer
org.drip.service	sample	• BondAnalyticsAPISample • BondBasketAPISample

		• BondLiveAndEODAPISample • BondStaticAPISample • CDSAnalyticsAPISample • CDSBasketAPISample • CDSLiveAndEODAPISample • DayCountAndCalendarAPISample • FXAPISample • RatesAnalyticsAPISample • RatesLiveAndEODAPISample
org.drip.tester	product	• BondTestSuite • FIFull • FuncTestSuite • SerializerTestSuite
org.drip.util	common	• FIGen • Validatable
org.drip.util	date	• DateTime • JulianDate
org.drip.util	internal	• FIUtil • Logger

Package org.drip.analytics.core

This package implements the core interfaces/abstract classes that provide functionality common across much of DRIP. Currently, this has only one abstract class – Serializer.

Serializer

The Serializer abstract class defines the core object serializer methods – serialization of the object state onto a byte-array, and object construction through de-serialization out of byte arrays.

Methods that the serializer implements provide the current serializer version, as well as object trailers, and delimiters for fields, collection records, collection key values, and multi-level key-value collections. Derived implementations over-ride these fields as appropriate.

Package org.drip.analytics.curve

This package contains the “curve” functionality for the different kinds of curves – credit curve, discount curve, and FX forward curve. As defined here, curve simply holds the term structure of a calibrated or a cooked market parameter. Period measures available for different times are available directly as member of the corresponding curve API. Calibration of the curve is covered in another package – however, the curve may (optionally) return the instruments, the quotes, and their corresponding measures from the curve object.

CreditCurve

The CreditCurve class serves as the baseline hazard curve holder object, and provides the time-dependent survival probabilities and recovery rates. It contains term structure for recovery, the calibration instruments, calibration measures, calibration quotes, and parameters.

CreditCurve maybe created in several ways:

- Survival/recovery curve from start date, a solitary hazard rate and a solitary recovery.
- Survival/recovery curve from start date, survival nodes, and a solitary recovery.
- From start date, single hazard node and solitary recovery.
- From start date, hazard node array, and solitary recovery.
- From start date, hazard node array and recovery node array. Node point sets need not match for hazards and recoveries.

The curve calibration process may set the calibration instruments, calibration valuation parameters, calibration pricing parameters, calibration market parameters (discount curves/fixings), calibration measure, and calibration type. The CreditCurve implementation then retrieves the entire set of calibration input instruments, or the quote for a given input calibration instrument. Further, if the CreditCurve implements the LiveCurve interface, it may also be able to retrieve the tick value for the given instrument.

Using the set of calibration instruments retained, the CreditCurve provides scenario curves based off of one of the following shifts:

- Calibrate and create a new parallel hazard-shifted curve.
- Calibrate and create a new parallel quote-shifted curve.
- Calibrate and create a new parallel SNAC quote shifted curve.

Since the CreditCurve is bootstrapable, it provides functionality to get the master credit curve name, set calibration node quote value, bump the calibration node quote, or assign a flat hazard node value across all tenors.

Credit curve also provides the following core credit functionality:

- Calculate survival from the start date to the given date/tenor
- Calculate time weighted survival from the start date to dates within a pair of start and end dates/tenors
- Calculate the hazard rate between two dates/tenors
- Calculate hazard rate to the specified date.
- Calculate recovery rate at the specified date/tenor.
- Calculate time weighted recovery rate from the start date to dates within a pair of start and end dates/tenors

Finally, credit curve implements the serialization interface to serialize the credit curve instance into byte arrays, as well as de-serialize and populate a credit curve instance from an input byte array.

DiscountCurve

The DiscountCurve class holds the bootstrapped term structure of the forward rates and their corresponding maturity nodes. It provides all the regular discount curve functionality – e.g., the implied rate between two dates, and the discount factor between a pair of dates. It may also optionally contain the calibration instruments, the calibration measures, the calibration quotes, and other relevant calibration parameters.

The DiscountCurve maybe built in one of the following ways:

- From the start date, the currency, and the array of date nodes and discount factors.
- From the start date, the currency, and a discount rate.

During the calibration process, the calibrator may set the discount curve's calibration instruments, calibration valuation parameters, calibration pricing parameters, calibration market parameters (fixings), calibration measure, and calibration type. In this case, the DiscountCurve may be able to retrieve the full set of calibration instruments, or the quote for a given input calibration instrument. Further, if the DiscountCurve implements the LiveCurve interface, it may also be able to retrieve the tick value for the given instrument.

Using the set of calibration instruments retained, the CreditCurve provides scenario curves based off of one of the following shifts:

- Calibrate and create a new parallel quote-shifted curve.
- Calibrate and create a new parallel rate-shifted curve.
- Calibrate a shifted curve using a maturity-mismatched basis.

Since the DiscountCurve is bootstrapable, it provides functionality to get the master discount curve name, get the discount curve currency, set calibration node quote value, bump the calibration node quote, and assign a flat hazard node value across all tenors.

Discount curve also provides the following core discounting/PV'ing functionality:

- Get the discount factor to a given date/tenor.
- Calculate time weighted discount factor from the start date to dates within a pair of start and end dates/tenors.
- Imply a rate between two dates/tenors.

Finally, discount curve implements the serialization interface to serialize the discount curve instance into byte arrays, as well as de-serialize and populate a discount curve instance from an input byte array.

FXBasis

The FXBasis curve contains the term structure of FX basis. Basis can be full or bootstrapped. It is constructed from the currency pair, the spot date, the spot FX, the tenor dates/tenor basis array, and a flag indicating whether the basis has been bootstrapped.

As part of its core functionality, the FXBasis object calculates the term structure of the FX forward (as either outright or as PIP) from a pair of domestic/foreign discount curves, from either domestic or foreign basis. It also retrieves the currency pair, the spot date, the spot FX, and the flag that indicates whether the FX basis is bootstrapped

Finally, FXBasis implements the serialization interface to serialize the FXBasis instance into byte arrays, as well as de-serialize and populate a FXBasis instance from an input byte array.

FXCurve

The FXCurve object contains the term structure of dates/times, the corresponding FX forwards (PIP/outright), and the spot FX info for the given currency pair. It is constructed from the array of dates, array of tenors, array of FX forwards, the PIP flag indicator array, the currency pair, and the spot FX info.

The FXCurve provides the following functionality:

1. Calculate the full basis across the entire term of the forward curve give the domestic and foreign rates curves, for the input valuation parameters.
2. Boot-strap the constant forward basis across the entire term of the forward curve given the domestic and foreign rates curves, for the input valuation parameters.
3. Boot-strap a new domestic or foreign discount curve using the entire term of the forward curve given the domestic and foreign rates curves, for the input valuation parameters.

The FXCurve object provides functionality for retrieving the currency pair, the spot date, and the Spot FX.

Finally, FXCurve implements the serialization interface to serialize the FXCurve instance into byte arrays, as well as de-serialize and populate a FXCurve instance from an input byte array.

ZeroCurve

The ZeroCurve contains the baseline zero discount curve holder object. It is primarily used for calibrating the Z Spread of a bond. It maintains term structure for the spot zero rates, the current discount factors, the accrual fractions, and the corresponding dates. It is constructed from the bond’s cash flow list, the exercise parameters, the reference discount curve, the yield quoting convention, and the optional flat bump to the discount rate.

ZeroCurve overrides the getDF function of the DiscountCurve, and provides the ability to get the zero rate at one of the given pre-set nodes.

Package org.drip.analytics.daycount

This package implements the day count, the date adjustment/roll, location specific holiday calendars, and different kinds of holidays. It also contains explicit set of non-weekend holidays implemented for most locations – these can be over-written using externally specified days.

ActActDCParams

This class contains the date parameters corresponding to the actual/actual reference period. ActActDCParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate an ActActDCParams instance from an input byte array.

DateAdjustParams

This class contains the parameters needed for adjusting dates – holiday calendar and adjustment type. DateAdjustParama implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a DateAdjustParams instance from an input byte array.

DayCountBasis

The DayCountBasis provides the core set of static day-count, holiday, and date roll/adjustment functionality. The class contains flags that indicate where the holidays are loaded from, as well as the holiday types and load rules.

DayCountBasis loads the holiday calendar from the specified location, or from the specified location holiday file set in the configuration file. It gets the available holiday locations. It gets the weekend and week-days corresponding to a calendar set.

DayCountBasis implements the following core day count functionality:

- It also gets the available day count conventions.
- Calculate the year fraction between 2 days for the given day count.
- Roll the given date according to the date roll convention and the holiday calendar set.
- Check if the given date is a holiday according to the holiday calendar set.
- Calculate the number of business days between two days according to the holiday calendar set.
- Calculate the number of holidays between two days according to the holiday calendar set.
- Adjust the given date forward in accordance with the given holiday calendar set.

FixedHoliday

This class contains the fixed holiday's date and month. Will be generated with an optional adjustment for weekends in a given year.

FixedHoliday implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a FixedHoliday instance from an input byte array.

FloatingHoliday

This class contains the floating holiday's month, day in week, and week in month. Will be generated with an optional adjustment for weekends in a given year.

FloatingHoliday implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a FloatingHoliday instance from an input byte array.

Holiday

The Holiday class provides abstraction around a holiday and its description. It contains an abstract function that generates an optional adjustment for weekends in a given year. Its concrete function rolls the given holiday around a weekend – rolling to a preceding or succeeding day depending upon whether it is a first or second weekend day. It also returns the holiday description.

Holiday implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a Holiday instance from an input byte array.

LocHolidays

This class contains the set of regular holidays and the weekend holidays for a location. Weekends are separately for the week days – weekends are set either an array of days, or as the standard weekend (Saturday and Sunday), and the week days as static, fixed, or floating holidays. LocHolidays are used to retrieves the weekend and the regular holiday set for the given location.

StaticHoliday

This class contains a full date as a fixed holiday. Can be constructed from a stringified date or a Julian date.

StaticHoliday implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a StaticHoliday instance from an input byte array.

WeekendHoliday

WeekendHoliday class contains the dates corresponding to the weekend. Weekends can be set from a date array, or as standard (Saturday and Sunday). WeekendHolidays are used to retrieve the weekend days, and it identifies if the given date is a weekend, and if it is, is it a left or a right weekend.

WeekendHoliday implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a WeekendHoliday instance from an input byte array.

Package org.drip.analytics.period

This package implements different types of periods used, their generation, and the corresponding curve measures used in CreditAnalytics.

CouponPeriod

The CouponPeriod encapsulates the period details related to the coupon period. It extends the period class with day-count specific parameters: frequency, reset date, and accrual day-count convention.

Construction of CouponPeriod happens in one of two ways. For the multi-period CouponPeriod set, a helper function creates a set of coupon periods according generated either backwards or forwards, according to period frequency, and optionally specific date adjustment rules for every date set (start/end dates, accrual start/end dates, pay date, and reset date) – this also allows for accrual DCF different from coupon DCF. For a single period coupon set, another helper function is used to construct a single coupon period from start and end dates (as in a zero coupon bond).

CouponPeriod also gets the accrual fraction to an arbitrary date within the period, as well the period reset date.

CouponPeriod implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a CouponPeriod instance from an input byte array.

Period

Period class serves as the place-holder for the typical period dates and fractions: the period start/end dates, the period accrual start/end dates, the period pay date, and the full period day count fraction.
Period class is used to retrieve the period start/end dates, the period accrual start/end dates, the period pay date, and the full period day count fraction. Period reset date defaults to the period start date. It also gets the accrual fraction to an arbitrary date within the period.

Period implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a Period instance from an input byte array.

ProductCouponPeriodCurveMeasures

The ProductCouponPeriodCurveMeasures class implements the discount curve and the credit curve based coupon period valuation metrics for the given period. It enhances the Period class with the following period measures: the period start/end survival probabilities, the period start/end notionals, and the period end discount factor.

ProductCouponPeriodCurveMeasures retrieves period’s start/end survival probabilities, start/end notionals, and period end discount factor.

ProductCouponPeriodCurveMeasures implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a ProductCouponPeriodCurveMeasures instance from an input byte array.

ProductLossPeriodCurveMeasures

The ProductLossPeriodCurveMeasures class implements the discount curve and the credit curve based loss period valuation metrics for the given period. The class enhances the period class by the following period measures: the period’s start/end survival probabilities, and the period’s effective notional/recovery/discount factor.

ProductLossPeriodCurveMeasures retrieves the period’s start/end survival probabilities, and the period effective notional/recovery/discount factor.

ProductLossPeriodCurveMeasures implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a ProductLossPeriodCurveMeasures instance from an input byte array.

Package org.drip.calc.output

This package implements the analytics scenario measure map for basket and component valuation runs.

BasketOutput

The BasketOutput serves as the main place holder for analytical basket measures, optionally across scenarios. It contains measure maps for unadjusted base IR/credit curve runs, flat delta/gamma bump measure maps for IR/credit/recovery bump curve runs, component/tenor bump double maps for IR/credit curves, and flat/component recovery bumped measure maps for recovery bumped credit curves.

BasketOutput implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BasketOutput instance from an input byte array.

BondCouponMeasures

BondCouponMeasures encapsulates the parsimonious, but complete set of the cash-flow oriented coupon measures generated out of a full run to a set of given work-out parameters. Specifically, it is a placeholder for the “dirty” measures – dirty DV01, dirty coupon PV, and full dirty PV.

BondCouponMeasures provides methods to adjust for settlement, and for clean/dirty accrual. It also uploads the state in the fields onto a named measure map.

BondCouponMeasures implements the serialization interface to serialize its instance into byte arrays, as well as de-serialize and populate a BondCouponMeasures instance from an input byte array.

BondOutput

The BondOutput serves as the place holder for a comprehensive suite of analytical bond measures. It contains the bid/ask sides for price, yield, G spread, Z spread, I spread, spread to treasury benchmark, par asset swap spread, workout dates, workout factors, and credit basis. BondOutput instance can be re-constructed from a string representation, as well as stringify it.

BondOutput implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondOutput instance from an input byte array.

BondRVMeasures

BondRVMeasures encapsulates the comprehensive set of RV measures generated out of a full run to a set of given work-out parameters. It holds the price, the Z Spread, the I Spread, the G Spread, the Spread to treasury benchmark, the bond basis, the par asset swap spread, the credit basis, the duration, the convexity, and the work-out information - its constructor automatically sets the full state.

It also uploads the state in the fields onto a named measure map.

BondRVMeasures implements the serialization interface to serialize its instance into byte arrays, as well as de-serialize and populate a BondRVMeasures instance from an input byte array.

BondWorkoutMeasures

BondWorkoutMeasures encapsulates the comprehensive set of scenario measures generated out of a full run to a set of given work-out parameters. It holds the clean/dirty credit risky and risk-less bond coupon measures, risky and risk-less par and principal PV, recovery PV, expected recovery, default exposure with and without recovery, loss on instantaneous default, dirty accrued 01, first coupon rate, and first index rate - the constructor automatically sets the full state.

It also uploads the state in the fields onto a named measure map.

BondWorkoutMeasures implements the serialization interface to serialize its instance into byte arrays, as well as de-serialize and populate a BondWorkoutMeasures instance from an input byte array.

ComponentOutput

The ComponentOutput serves as the place holder for analytical single component output measures, optionally across scenarios. It contains measure maps for unadjusted base IR/credit curves, flat delta/gamma bump measure maps for IR/credit bump curves, tenor bump double maps for IR/credit curves, and flat/recovery bumped measure maps for recovery bumped credit curves.

ComponentOutput implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a ComponentOutput instance from an input byte array.

Package org.drip.chart.surface

This package implements different kinds of 3D charting surfaces from z’s corresponding to a set of (x, y) inputs.

BuildSurface

BuildSurface class constructs an OpenGL 3D surface chart for the z’s corresponding to a set of (x, y).

Contour3D

Contour3D class constructs an OpenGL 3D mapped contour surface chart for the z’s corresponding to a set of (x, y).

ContourPlots

ContourPlots constructs an ortho-normalized OpenGL 3D color mapped contour surface chart for the z’s corresponding to a set of (x, y).

GeneratedDelaunaySurface

GeneratedDelaunaySurface constructs an OpenGL 3D Delaunay surface chart for the z’s corresponding to a set of (x, y).

Histogram

Histogram constructs an OpenGL 3D histogram for the z’s corresponding to a set of (x, y).

MultiColorScatter

MultiColorScatter constructs a 3D depth color-mapped multi-color scatter surface chart for the z’s corresponding to a set of (x, y).

Scatter4D

Scatter4D constructs a 4D depth color-mapped multi-color scatter surface chart for the z’s corresponding to a set of (x, y).

Package org.drip.curve.calibration

This package contains a set of calibrators that use different calibration routines to calibrate the corresponding rates from the component's input measure and quote value.

Bootstrapper

Bootstrapper is the basic interface that defines the core bootstrapping methods – setting/bumping specific nodes, setting flat values across all nodes, and retrieving specific/collective instrument/node quotes.

ComponentCalibrator

The ComponentCalibrator interface defines the curve calibration methods from component market values and measures – by bootstrapping/flat-strapping the discount rate and the hazard rate from the individual component quotes. Calibration can be node-by-node (true bootstrapping) or flat.

ComponentCalibratorBracketing

ComponentCalibratorBracketing uses a bracketing technique to find the discount rate/hazard rate that corresponds to the specific node tenor. Calibration produces either the implied piece-wise constant forward or the flat root across all the nodes.

ComponentCalibratorNR

ComponentCaibratorNR uses the Newton-Raphson method to find the discount rate/hazard rate that corresponds to the specific node tenor. Calibration produces either the implied piece-wise constant forward or the flat root across all the nodes.

CreditCurveScenarioGenerator

CreditCurveScenarioContainer contains the credit calibration instruments to be used with the component calibrator to produce scenario credit curves.

CreditCurveScenarioContainer performs two types of calibration: a) It calibrates and creates a bootstrapped or flat credit curve from valuation parameters, recovery rate, discount curves, and fixings, or b) it calibrates an array/tenor map of bootstrapped or flat tenor bumped credit curves from valuation parameters, recovery rate, discount curves, and fixings.

IRCurveScenarioGenerator

IRCurveScenarioContainer holds the IR calibration instruments to be used with the component calibrator to produce scenario discount curves.

IRCurveScenarioContainer performs two types of discount curve calibration: a) It calibrates and creates a bootstrapped or flat discount curve from valuation parameters, treasury/EDF curves, and fixings, or b) it calibrates an array/tenor map of bootstrapped or flat tenor bumped discount curves from valuation parameters, treasury/EDF curves, and fixings.

Package org.drip.param.config

This package contains implementations that parse input configurations and configure the CreditAnalytics system at start-up. Currently only XML based configuration is implemented.

XMLConfigReader

XMLConfigReader parses the XML configuration file and extracts the information, tag pairs – such as holiday sets for different locations, logger location, analytics server connection strings, database server connection strings. Depending upon the flag in the configuration setting, holiday sets are loaded from either directly from the configuration files, or from database setting.

Package org.drip.param.market

This package contains the market parameters named container objects for various purposes – discount curve/credit curve/fixings and other market parameters needed to price components and baskets. It also issuer/domain named scenario curves, as well as scenario discount curve and scenario credit curve containers.

BasketMarketParamRef

BasketMarketParamRef implements the base market params interface to provide stubs for component IR and credit curves that constitute the basket. All basket market parameter classes implement this interface.

BasketMarketParams

BasketMarketParamRef contains the market parameters needed to price the given basket. It implements the BasketMarketParamsRef interface for a specific scenario. It also contains maps holding named discount curves, named credit curves, named treasury quote, named component quote, and fixings object.

Specifically, BasketMarketParams provides the ability to add and retrieve a named discount curve and a named credit curve. It can also build a ComponentMarketParams object from a component given its ComponentMarketParamsRef object.

BasketMarketParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BasketMarketParams instance from an input byte array.

ComponentMarketParamRef

ComponentMarketParamsRef implements the base market params interface to provide stubs for component name, IR curve, credit curve, TSY curve, and EDSF curve needed to value the component.

ComponentMarketParams

ComponentMarketParams provides the place holder for the market parameters needed to value the component object – discount curve, treasury curve, EDSF curve, credit curve, component quote, treasury quote map, and fixings map. It provides implementation of the ComponentMarketParamsRef interface.

ComponentMarketParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a ComponentMarketParams instance from an input byte array.

CreditCurveScenarioContainer

CreditCurveScenarioContainer serves as the place holder for the bump parameters and the curves for the different credit curve scenarios. It contains the spread and the recovery bumps, and the credit curve scenario generator object that wraps the calibration instruments.

It holds the base credit curve, spread bumped up/down credit curves, recovery bumped up/down credit curves, and the tenor mapped up/down credit curves. Depending upon the scenario creation mode, CreditCurveScenarioContainer cooks the curves that correspond to the scenarios above, and retrieves them.

IRCurveScenarioContainer

IRCurveScenarioContainer serves as the placeholder for the different IR scenario curves. It contains the IR curve scenario generator object that wraps the calibration instruments. IRCurveScenarioContainer holds the base IR curve, spread bumped up/down IR curves, and tenor mapped up/down credit curves. Depending upon the scenario creation mode, cooks the curves that correspond to the scenarios above, and retrieves them.

MarketParamsContainer

MarketParamsContainer is the principal placeholder for the comprehensive suite of the market set of curves for the given date. It contains treasury quote map, fixings map, IR scenario curve map, credit curve scenario set map, and component quote map.

MarketParamsContainer provides an extensive set of functionality for setting/getting scenario curves and quotes. It also provides a collection of function to retrieve scenario market parameters for the given component or basket product:

- Sets/gets scenario discount curve set and scenario credit curve set for a given IR currency/credit curve.
- Sets/gets quote for a given treasury benchmark or for a full set of treasury benchmarks.
- Get/set the fixings map.
- Sets/gets quote for a given component or for a full set of components.
- Get component market parameters for the given component and scenario.
- Get the map of tenor component market parameters for the given component across each of the IR tenors.

- Get the map of tenor component market parameters for the given component across each of the credit tenors.
- Get the basket market parameters for the given basket and scenario.
- Get the map of flat IR bumped basket market parameters for the given basket across each of the IR curves.
- Get the map of flat credit bumped basket market parameters for the given basket across each of the credit curves.
- Get the map of flat recovery bumped basket market parameters for the given basket across each of the credit curves.
- Get the double map of the tenor IR bumped basket market parameters for the given basket across each of the IR curves and their tenors.
- Get the double map of the tenor credit bumped basket market parameters for the given basket across each of the credit curves and their tenors.
- Get the double map of the tenor recovery bumped basket market parameters for the given basket across each of the credit curves and their tenors.

NodeTweakParams

NodeTweakParams serves as the placeholder for the scenario tweak mode, for either the whole curve, or for segments of it. It contains the tweak type (parallel/proportional), tweak node, or tweak amount.

NodeTweakParams implements the serialization interface to serialize its instance into byte arrays, as well as de-serialize and populate a NodeTweakParams instance from an input byte array.

Package org.drip.param.pricer

This package contains all the pricing parameters. Currently it only implements the non-correlation pricing parameter.

PricerParams

PricerParams is a place holder for the credit pricing parameters across all credit product classes and pricing methods. It contains the loss step discretization scheme, time domain unit size, whether survival is to be calculated to the period accrual date or period pay date, and whether current pricing is a calibration operation or not.

PricerParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a PricerParams instance from an input byte array.

CalibrationParams

CalibrationParams serves as the placeholder for the calibration parameters – the measure to be calibrated, and the type and the nature of the calibration to be done, and the exercise parameters to which the calibration is set.

CalibrationParams implements the serialization interface to serialize its instance into byte arrays, as well as de-serialize and populate a CalibrationParams instance from an input byte array.

Package org.drip.param.product

This package implements an extensive set of classes that encapsulate groups of product contract details for cash flow generation, and valuation.

BondCFTerminationParams

BondCFTerminationParams contains the termination static/status parameters for the bond. Specifically, indicates whether the bond is perpetual, has been called, or has defaulted.

BondCFTerminationParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondCFTerminationParams instance from an input byte array.

BondCouponParams

BondCouponParams contains the coupon parameters for the bond. Indicates what the bond coupon/spread is, and contains the coupon schedule and the coupon type.

BondCouponParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondCouponParams instance from an input byte array.

BondCurrencyParams

BondCurrencyParams contains the currency parameters for the bond. It contains the trade, the coupon, and the redemption currencies.

BondCurrencyParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondCurrencyParams instance from an input byte array.

BondFixedPeriodGenerationParama

BondFixedPeriodGenerationParams contains the period generation parameters for the bond. It contains the frequency, accrual/coupon day-count conventions, effective, maturity, and final maturity dates, and maturity type. The coupon periods generated are and kept in a list.

BondFixedPeriodGenerationParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondFixedPeriodGenerationParams instance from an input byte array.

BondFloaterParams

BondFloaterParams is the placeholder for the bond’s floating rate parameters. It contains the floater flag, the rate index, the index spread, floater day count, and the current coupon.

BondFloaterParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondFloaterParams instance from an input byte array.

BondIdentiferParams

BondIdentifierParams is the placeholder for the bond’s identifier parameters. It contains the ISIN, CUSIP, bond ID, and the ticker.

BondIdentifierParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondIdentitiferParams instance from an input byte array.

BondIRValuationParams

BondIRValuationParams serves as the placeholder for the bond’s IR valuation parameters. It contains the bond’s IR curve, the quoting convention, the calculation type, the first settle date, and the redemption value.

BondIRValuationParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondIRValuationParams instance from an input byte array.

BondNotionalParams

BondNotionalParams serves as the placeholder for the bond’s notional parameters. It contains the bond’s current notional and the outstanding notional schedule.

BondNotionalParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondNotionalParams instance from an input byte array.

BondPeriodGenerationParams

BondPeriodGenerationParams is the placeholder for the generic bond's period generation parameters. It contains the bond's date adjustment parameters for period start/end, period accrual start/end, effective, maturity, pay and reset, first coupon date, and interest accrual start date. Validation of the BondPeriodGenerationParams class results in the generation of the list of periods according to the date generation rules – invalid/inconsistent parameters result in no such list being created.

BondPeriodGenerationParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondPeriodGenerationParams instance from an input byte array.

BondTSYParams

BondTSYParams is the placeholder for the bond's treasury related parameters. It contains the bond's treasury benchmark set, government treasury benchmark, and EDSF short-end benchmark identifiers.

BondTSYParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondTSYParams instance from an input byte array.

CDXIdentifier

CDXIdentifier serves as the placeholder for the identifier parameters for a given standard CDX contract. It contains the index, the tenor, the series, and the version for the given contract.

CDXIdentifier implements the serialization interface to serialize the CDXIdentifier object instance into byte arrays, as well as de-serialize and populate a CDXIdentifier instance from an input byte array.

CompCRValParams

CompCRValParams is the placeholder for the credit component's credit valuation parameters. It contains the credit curve, the component's recovery, and whether that is usable, the loss pay lag, and whether the accrual is to be applied on default.

CompCRValParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a CompCRValParams instance from an input byte array.

CurrencyPair

CurrencyPair is the placeholder for the currency pair object for a given FX contract. It contains the numerator currency, the denominator currency, and the quoting currencies, and the PIP factor for the given contract.

CurrencyPair implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a CurrencyPair instance from an input byte array.

EmbeddedOptionSchedule

EmbeddedOptionSchedule is the placeholder for the embedded option schedule for the bond. It contains the schedule of exercise dates and factors, the exercise notice period, and whether the option is to call or put. Further, if the option is of the type fix-to-float on exercise, it contains the post-exercise floater index and floating spread.

If the exercise date is not discrete (American option), the exercise dates/factors are discretized according to a pre-specified discretization grid.

Several helper functions in EmbeddedOptionSchedule help create the schedule from string date/factor arrays.

EmbeddedOptionSchedule provides functions to retrieve whether the option is for a put, whether it is fix-to-float on exercise, and the date/factor array.

EmbeddedOptionSchedule implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a EmbeddedOptionSchedule instance from an input byte array.

FactorSchedule

FactorSchedule serves as the place holder for factor index schedule of all types – in particular coupon/principal factors. It contains matched date/factor arrays.

FactorSchedule provides static creator/factory functions that create a FactorSchedule instance from the arrays of dates/factors, string arrays of dates/factors, and even bullet schedules.

FactorSchedule provides functionality to retrieve the factor/factor index for a given date, as well as the array of factors/dates. It also returns the effective time-weighted factor between two arbitrary dates.

FactorSchedule implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a FactorSchedule instance from an input byte array.

TsyBmkSet

TsyBmkSet contains the applicable treasury benchmarks on the given bond. It holds the primary treasury benchmark, and none or many secondary treasury benchmarks.

TsyBmkSet implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a TsyBmkSet instance from an input byte array.

Package org.drip.param.valuation

This package holds the valuation related classes, typically based off of the given value date.

CashSettleParams

CashSettleParams is the placeholder for the cash settlement parameters for a given product or a settlement structure. It holds the settle delay, settlement adjustment calendar, and the cash settle adjustment modes.

CashSettleParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a CashSettleParams instance from an input byte array.

NextExerciseInfo

NextExerciseInfo serves as the placeholder for the next exercise information fields for a given product, starting from the given valuation date. It contains the exercise type, the exercise factor, and the exercise date.

NextExercise Info Implements the serialization interface to serialize the NextExerciseInfo object instance into byte arrays, as well as de-serialize and populate a NextExerciseInfo instance from an input byte array.

QuotingParams

QuotingParams is the placeholder for the quoting parameters for a given product or a settlement. Contains the quoting yield day count convention, quoting yield frequency, whether EOM adjustment is to be applied or not, the adjustment calendar, and whether the product is spread quoted or price quoted.

QuotingParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a QuotingParams instance from an input byte array.

ValuationParams

ValuationParams is the placeholder for the valuation parameters for a given product. It contains the valuation and the cash pay/settle dates.

ValuationParams implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a ValuationParams instance from an input byte array.

WorkoutInfo

WorkoutInfo serves as the placeholder for the workout parameters. It contains the date, the factor, and the yield of the workout.

WorkoutInfo implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a WorkoutInfo instance from an input byte array.

Package org.drip.product.common

This package contains the abstract classes from which all components (calibratable or not) are built.

CalibratableComponent

CalibratableComponent is an abstract class that exposes functionality that makes the component calibratable. It implements the Component interface, provides stub for getting the component's primary code, and optionally provides multiple secondary codes.

CalibratableComponent also defines the calibMeasures method – which calculates and returns the requested measure for the given set of inputs.

Component

Component is an abstract class that exposes functionality that defines component products of al types. It extends the ComponentMarketParamRef interface, and provides methods for getting the component's notional, coupon, effective date, maturity date, coupon amount, and list of coupon periods.

Component also calculates the full set of component measures across a set of specified scenarios, and holds the results in ComponentOutput. Functions are available to retrieve the specified measure for the component.

Package org.drip.product.creator

This package provides the functionality needed to create the full-featured (primarily credit) product, in particular bonds. It provides functionality to load the definitions for the valuation as well as informational reference data for bonds. It also provides functionality to create custom bonds of all variants found in standard bonds.

BondBuilder

BondBuilder contains the suite of helper functions for creating user defined bonds, optionally with custom cash flows and embedded option schedules (European or American).

Specifically, BondBuilder creates full-featured bonds from custom cash flows (coupon/principal flows or amortization/capitalization schedules), simple fixed rate bonds, simple floater bonds, and bonds from explicitly specified coupon flows and principal flows (principal flows can be specified as either principal pay down/up or outstanding notional).

BondProductBuilder

BondProductBuilder contains the comprehensive set of static parameters of the bond product needed for the full bond valuation.

Specifically, the BondProductBuilder contains the bond identifier parameters (ISIN, CUSIP), the issuer level parameters (Ticker, SPN or the credit curve string), the coupon parameters (coupon rate, coupon frequency, coupon type, day count), the maturity parameters (maturity date, maturity type, final maturity, redemption value), the date parameters (announce, first settle, first coupon, interest accrual start, and issue dates), the embedded option parameters (callable, putable, has been exercised), the currency parameters (trade, coupon, and redemption currencies), the floater parameters (floater flag, floating coupon convention, current coupon, rate index, spread), and the fags indicating whether the bond is perpetual or has defaulted.

BondProductBuilder is created from resultset of a bond ref data entry, from an SQL query. On construction, it is validated to ensure internal consistency.

BondProductBuilder provides functionality to get either the individual fields or the corresponding bond parameter set.

BondProductBuilder implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondProductBuilder instance from an input byte array.

BondRefDataBuilder

BondRefDataBuilder contains the entire set of static parameters for the bond product.

Specifically, it holds the bond identifier parameters (ISIN, CUSIP, BBG ID, name short name), the issuer level parameters (Ticker, category, industry, issue type, issuer country, issuer country code, collateral type, description, security type, unique Bloomberg ID, long company name, issuer name, SPN or the credit curve string), issue parameters (issue amount, issue price, outstanding amount, minimum piece, minimum increment, par amount, lead manager, exchange code, country of incorporation, country of guarantor, country of domicile, industry sector, industry group, industry sub-group, senior/sub), coupon parameters (coupon rate, coupon frequency, coupon type, day count), maturity parameters (maturity date, maturity type, final maturity, redemption value), date parameters (announce, first settle, first coupon, interest accrual start, next coupon, previous coupon, penultimate coupon, and issue dates), embedded option parameters (callable, putable, has been exercised), currency parameters (trade, coupon, and redemption currencies), floater parameters (floater flag, floating coupon convention, current coupon, rate index, spread), trade status, ratings (S & P, Moody, and Fitch), and whether the bond is private placement, is registered, is a bearer bond, is reverse convertible, is a structured note, can be unit traded, is perpetual or has defaulted.

BondRefDatatBuilder is typically created from the SQL query resultset of a bond ref data entry. It provides functionality to get either the individual fields or the corresponding bond parameter set.

BondRefDataBuilder implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BondRefDataBuilder instance from an input byte array.

CDXRefDataBuilder

CDXRefDataBuilder contains the entire set of static and valuation parameters for the named Standard CDX Product.

Specifically, CDXRefDataBuilder holds the Index Curve ID, Index Curve SPN, Index Label, Index Name, Index Issue Date, Index Maturity Date, Index Coupon, Index Currency, Index Full First Stub, Index Day Count Convention, Index Recovery, Index Coupon Frequency, Index Reference Entity ID, Index Class, Index Series, Index Group Name, Index Short Group Name, Index Version, Index Life Span, Index Curvy Curve ID, Index Factor, Original Component Count, Defaulted Component Count, Index Location, whether the index pays accrued on default, whether the index knocks out on default, whether the index is quoted as a CDS, Index Bloomberg Ticker, and Index Short name.

Individual functions in CDXRefDataBuilder validate each one of its above-mentioned fields.

Finally, CDXRefDataBuilder also has functionality to create the actual index basket product (along with the corresponding cash-flows).

Package org.drip.product.credit

This package implements all the liquid and the bespoke components and basket credit products – credit default swap, basket credit default swap, and bond.

BasketBond

BasketBond serves as the placeholder for the bond basket/bond ETF product contract details. It contains the basket name, the basket notional, the component bonds, and their weights.

BasketBond contains methods to retrieve the notional at different times, the coupon, the effective/maturity dates, the flow periods, the component IR and the component credit curves. Its value method returns a full set mapped set of all the product measures for a given set of scenario curves. Using this in conjunction with the calcMeasures of the BondBasket produces a comprehensive set of all the mapped scenario measures.

BasketBond implements the serialization interface to serialize the BondBasket object instance into byte arrays, as well as de-serialize and populate a BondBasket instance from an input byte array.

BasketDefaultSwap

BasketDefaultSwap serves as the placeholder for the basket default swap product contract details. It contains the effective date, the maturity date, the coupon, the coupon day count, the coupon frequency, the basket components, the basket notional, the loss pay lag, and optionally the outstanding notional schedule and the flat basket recovery.

BasketDefaultSwap provides a number of helper functions to create named CDX from an effective date, maturity date, coupon, IR curve, and a set of components – in a few different forms.

BasketDefaultSwap also implements the following core basket functionality:

- It retrieves the notional at different times, the coupon, the effective/maturity dates, the flow periods, the component IR and the component credit curves.
- The value method returns a full set mapped set of all the product measures for a given set of scenario curves. Using this in conjunction with the calcMeasures of the BasketProduct produces a comprehensive set of all the mapped scenario measures.

BasketDefaultSwap implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a BasketDefaultSwap instance from an input byte array.

BasketProduct

BasketProduct is the master base class for all the basket credit products. It is an abstract class that extends the BasketMarketParamRef interface. It provides methods for getting the basket’s components, notional, coupon, effective date, maturity date, coupon amount, and list of coupon periods.

BasketProduct calculates the full set of basket measures across a set of specified scenarios, and holds the results in BasketOutput. It retrieves the specified measure for the basket.

Bond

Bond is the concrete base class that implements the foundation functionality behind all kinds of bonds. Like other credit components, bond extends the CreditComponent abstract class.

The static data behind the bond class is captured in a set of 11 container classes – BondTSYParams, BondCouponParams, BondNotionalParams, BondFloaterParams, BondCurrencyParams, BondIdentifierParams, BondIRValuationParams, CompCRValParams, BondCFTerminationEvent, BondFixedPeriodGenerationParams, and one EmbeddedOptionSchedule object instance each for the call and the put objects. Each of these parameter sets can be set separately.

Bond class provides functionality to retrieve al the valuation related bond contract details:

- It gets the bond’s different identifiers – ISIN, CUSIP, primary/secondary codes, and bond’s canonical name.
- It gets the bond’s coupon at different times, and the IR/credit/EDSF/TSY benchmark names.
- It retrieves the notional at different times, effective/maturity dates, coupon/loss flow periods, and component credit valuation parameters.
- It retrieves floater parameters (rate index, current coupon, float spread, floating coupon convention), embedded option schedule for call/put, whether the bond is amortizing, defaulted, is perpetual, coupon/accrual day-count, coupon type, coupon frequency, trade/redemption/coupon currency.

The bond class implements a comprehensive set of analytics calculations given different kinds of inputs. Calculations are done for: reset date for the period given by the valuation date, accrual, theoretical price from a discount/credit curve, price/yield/Z spread/credit basis/treasury spread/G Spread/I Spread/par asset swap spread/duration/convexity from price/yield/Z spread/credit basis/treasury spread/G Spread/I Spread/par asset swap spread inputs. Calculations are done to either a specified work-out date, or maturity/optimal exercise date.

The bond’s value method returns a complete mapped set of all the bond product measures for a given set of scenario curves. Using this in conjunction with the calcMeasures of the Component produces a comprehensive set of all the mapped scenario measures.

Bond also implements calibMeasures method, which calculates and returns the specific measure requested.

Bond also implements a calibrator that calibrates the yield, the credit basis, or the Z spread for the bond given the price input. Calibration happens via either Newton-Raphson method, or via bracketing/root searching.

Bond implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a Bond instance from an input byte array.

CreditComponent

CreditComponent implements the base abstract class over which all credit components are implemented. It extends the CalibratableComponent abstract class.

CreditComponent contains methods to get the credit valuation parameters, the coupon and the loss flows, and recovery at either a single date, or time weighted average between two dates (component recovery, if available, precedes curve recovery).

CreditDefaultSwap

CreditDefaultSwap is the place-holder for the credit default swap product contract details. It contains the effective date, the maturity date, the coupon, the coupon day count, the coupon frequency, the contingent credit, the currency, the basket notional, the credit valuation parameters, and optionally the outstanding notional schedule.

CreditDefaultSwap provides a number of helper functions to create named CDS from an effective date, maturity date/tenor, coupon, IR curve, credit valuation parameters, and comprehensive coupon period generation parameters – in a few different forms.

The CreditDefaultSwap class retrieves the notional at different times, the coupon, the effective/maturity dates, the flow periods, the component IR/Credit curves, the component credit valuation parameters, the CDS recovery, and the CDS primary/secondary codes.

The value method returns a fully mapped set of all the product measures for a given set of scenario curves. Using this in conjunction with the calcMeasures of the Component produces a comprehensive set of all the mapped scenario measures.

CreditDefaultSwap also implements a flat spread calibrator that calibrates a flat hazard curve off of the current CDS contract from a given market price.

CreditDefaultSwap implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a CreditDefaultSwap instance from an input byte array.

StandardCDXManager

StandardCDXManager serves as the placeholder for all the standard CDX/iTRAXX named product details. It contains the named CDX/iTRAXX series identifier (series, version, and index), and maps that retrieve the first coupon date or the series from the other, given an index.

StandardCDXManager provides the functionality to retrieve the CDX/iTRAXX index given the index, the series, and the tenor, as well as retrieve the on-the-run index corresponding to the given date and tenor.

Finally, StandardCDXManager also provides functionality to retrieve the pre-set and pre-loaded CDX names, their first coupon dates, and descriptions.

StandardCDXParams

StandardCDXParams serves as the placeholder for the given standard CDX/iTRAXX index. It contains the components, the currency, and the coupon for the given index.

Package org.drip.product.fx

This package implements the core vanilla FX products – the FX spot and FX forward. Both these products may be quoted in different forms, and the conversions may be applied using the functionality available in those classes.

FXForward

The FXForward class serves as the place-holder for the FX forward product contract details. It contains the effective date, the maturity date, the currency pair and the product code.

FXForward class provides static helper functions that create named FXForward object from an effective date, maturity date/tenor, and currency pair. Its instance member functions retrieve the product currency pair, the effective/maturity dates, and the primary/secondary codes.

FXForward implements a number of the core FX forward functionality:

- Implies the FX forward as either an outright or a PIP from the input valuation parameters, FX spot, numerator/denominator discount curves based on either the numerator discount curve or the denominator.
- Calibrates the discount curve basis from the input valuation parameters, FX spot/forward, numerator/denominator discount curves based on either the numerator discount curve or the denominator.
- Implements a flat spread calibrator that calibrates a flat curve off of the current CDS contract from a given market price.

The value method returns a full set mapped set of all the product measures for a given set of scenario curves.

FXForward implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a FXForward instance from an input byte array.

Finally FXForward also implements a FX basis calibrator that calibrates a basis to either the numerator or the denominator curve off of the current FX forward through the Newton Raphson method.

FXSpot

The FXSpot class serves as the place-holder for the FX spot contract details – the spot date and the currency pair.

FXSpot implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a FXSpot instance from an input byte array.

Package org.drip.product.quote

This package contains classes the provide the implementation behind different kinds of quotes – sided quotes, component quotes, and live ticking quotes.

ComponentQuote

The ComponentQuote serves as the placeholder for the different types of quotes for a given component. It contains a single market field/quote pair, but multiple alternate named quotes (to accommodate quotes on different measures for the component).

ComponentQuote implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a ComponentQuote instance from an input byte array.

LiveQuote

LiveQuote is the interface exposing the “refresh” method to be implemented by any derived product quote.

Quote

The Quote class contains the details corresponding to a product quote. It contains the quote value, quote instant for the different quote sides (bid/ask/mid).

Quote implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a Quote instance from an input byte array.

Package org.drip.product.rates

This class provides the implementation behind the interest rate products of all kinds – cash/money market products, euro-dollar futures, and the interest rate swap.

Cash

The Cash class serves as the placeholder for the cash product contract details. It contains the effective date, the maturity date, the coupon, the coupon day count, the coupon frequency, the currency, the basket notional, and optionally the outstanding notional schedule.

Cash class provides a number of static, helper factory functions that create Cash instance from the effective date, the maturity date/tenor, and the IR curve. Its instance members retrieve the notional at different times, the coupon, the effective/maturity dates, the flow periods, the IR curves, and the product primary/secondary codes.

The value method returns a full set mapped set of all the product measures for a given set of scenario curves. Using this in conjunction with the calcMeasures of the Component produces a comprehensive set of all the mapped scenario measures.

Cash implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a Cash instance from an input byte array.

EDFuture

The EDFuture class serves as the placeholder for the euro-dollar product contract details. It holds the effective date, the maturity date, the coupon, the coupon day count, the coupon frequency, the currency, the basket notional, and optionally the outstanding notional schedule.

Static helper functions of EDFuture create EDF instance from effective date, maturity date/tenor, EDF code, and IR curve, as well create a euro-dollar pack from a given date. Instance member functions retrieve the notional at different times, the coupon, the effective/maturity dates, the flow periods, the IR curves, and the product primary/secondary codes.

The value method returns a full set mapped set of all the product measures for a given set of scenario curves. Using this in conjunction with the calcMeasures of the Component produces a comprehensive set of all the mapped scenario measures.

EDFuture implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a EDFuture instance from an input byte array.

InterestRateSwap

InterestRateSwap is the placeholder for the interest-rate-swap product contract details. It contains the effective date, the maturity date, the coupon, the coupon/accrual day count, the coupon frequency, the currency, the basket notional, the floating rate index, and, optionally, the outstanding notional schedule.

Static helper functions create IRS instance from an effective date, maturity date/tenor, IRS code, and IR curve. Instance members retrieve the notional at different times, the coupon, the effective/maturity dates, the flow periods, the IR curves, and the product primary/secondary codes.

The value method returns a full set mapped set of all the product measures for a given set of scenario curves. Using this in conjunction with the calcMeasures of the Component produces a comprehensive set of all the mapped scenario measures.

InterestRateSwap implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a InterestRateSwap instance from an input byte array.

Package org.drip.service.api

This package implements the static client level and service level API through which the stateless (or limited stateful) interactions with the CreditAnalytics functionality is to be made.

FI

The FI class exposes all the CreditAnalytics API to clients – this class serves as the main functional interface. On initialization, it retains an instance of the current date/time, the bond valuation data, and the bond ref data caches, and the SQL connections to all the databases.

The following are the extensive and comprehensive high-level description of the functionality provided by FI (please refer to FI javadoc and the source for more details):

- Initializes the holiday calendars, the data-base connections, and the current date/times.
- Gets all the holiday locations, available day counts, week day/weekend/combined holidays for the location set and year, the weekend days, and whether the given day is a holiday or not.
- Calculates the year fraction contained in the start/end dates, day count convention, and holiday calendar set; also adjusts/rolls the given date to the next business day in accordance with the holiday calendar set and adjustment convention.
- Gets the set of on-the-run treasury ISINs and their yields.
- Creates a component from the code, and makes a component set.
- Retrieves the available IR/TSY curve names, loads the live/EOD IR/TSY curve for a given EOD or a set of EODs.
- Loads the live/EOD Cash/EDF/IRS curve(s) for a given EOD or a set of EODs.

- Retrieves the available CDS curve names, loads the live/EOD CDS/bond/combined credit curve for an issuer for a given EOD or a set of EODs.
- Add/remove/retrieve bond/bond ref data from bond ID (ISIN/CUSIP etc)
- Get the bond's call/put schedules (optionally, given a date)
- Create a fixing for the given bond at the given period, get all available tickers, and get all the ISINs for the given ticker, and their outstanding notional.
- Get outstanding issuer notional aggregated by pre-specified maturity buckets.
- Calculate the bond's price/yield, theoretical credit price, spread to treasury benchmark, Z Spread, I Spread, par asset swap spread, and credit basis given the bond's yield, bond's price, or spread to a given treasury benchmark.
- Get the bond's named boolean, string, integer, double, or date field.
- Create a CDX product given the effective and the maturity dates.

Package org.drip.service.env

This package provides the functionality that implements the container for EOD set of credit curves, IR curves, bond product objects, and bond marks holder.

BondManager

BondManager serves as the container that holds the EOD and bond static information for the given issuer. It holds bond EOD bid/ask marks, ticker-wise maturity sorted set of bonds, map containing bond instance to ID (ISIN/CUSIP), flags for different formats of runs and corresponding displays/calculations.

BondManager provides the following functionality (please see javadoc and the source for the precise usage):

- Loads the option schedule for all bonds.
- Calculates the bond measures and generates the formatted runs for a pair of bid/ask price for the given bond.
- Generates formatted runs of measures for all the bonds for a given EOD.
- Retrieves and/or loads the bid/ask/mid levels for a single bond or for all the bonds on a given EOD.
- Builds and retrieves the bond object associated with a ResultSet and a MarketParamsContainer object.
- Loads the bond from the given bond ID, MarketParamsContainer, the SQL statement, and an optional date to determine when the discretization of an American option should start.
- Loads the entire ref data for a bond given the ID and the SQL statement.
- Gets all the available issuer tickers in the database.

- Retrieves all the available ISINs for the given issuer ticker.
- Loads the full set of bonds and commits them to memory.
- Calculates the measures for all the bonds associated with a ticker maturity after the valuation parameters, given the bid/ask prices and the market parameters.
- Calculates the measures for all the bonds associated with a ticker maturity after the valuation parameters, given the closing bid/ask prices and the closing market parameters.
- Calculates and saves/loads all the bond measures for all the bonds corresponding to the given closing date.

CDSManager

CDSManager is the container that creates, maintains, and saves the EOD and CDS/credit curve information on a per-issuer basis.

CDSManager retrieves all the credit curves available for the given EOD from the database. It also saves the EOD credit measures that correspond to the credit curve represented by the SPN ID from the input market parameters and the EOD tenor quotes from the database – it also saves all the EOD credit measures all the curves in the given MPC.

Finally, it loads all the credit curves corresponding to a given EOD from the database.

EnvManager

The EnvManager class sets the environment/connection parameters, and populates the market parameters for the given EOD.

EODCurves

The EODCurves container exposes the functionality to create the set of closing IR and credit curves for a given EOD.

EODCurves builds the named EOD credit curve for the given SPN ID/discount curve combination by retrieving the EOD quotes from the database, and using the specified discount curve.

EODCurves also builds the named EOD IR curve of a specific type (cash/EDF/IRS) for the given discount curve name and currency by retrieving the EOD quotes from the database. It also builds the named full EOD IR curve using instruments of all types (cash/EDF/IRS) for the given instrument set type (swaps/treasuries), discount curve name, and currency by retrieving the corresponding EOD quotes from the database. Finally, it loads the requested type of discount curve (cash/EDF/IRS/full/TSY) for the given currency and the EOD, and optionally adds it to the market params container.

RatesManager

The RatesManager constructs, and gets the discount curve names of the requested type (or the full discount curve) for the given EOD. It also loads the full closing discount curve for the given EOD.

StaticBACurves

StaticBACurves class creates a set of discount/credit curves from customuser-defined marks for a given EOD. It creates a set of USD treasury bonds/ED futures/IR swaps and their quotes, builds a discount curve from them, and loads it to the market parameters container. It creates a set of credit default swaps and their quotes, builds a credit curve from them, and loads it to the market parameters container.

Package org.drip.service.external

This package implements the server level analytics functionality provided using CreditAnalytics. It also implements an analytics client that dies a protocol based interaction with the analytics server.

AnalyticsClient

The AnalyticsClient class captures the requests for the FI server from the GUI client, formats them, and sends them to the AnalyticsServer.

AnalyticsServer

The AnaytcsServer class receives the requests from the analytics client, and invokes the FI functionality, and sends the client the results.

Package org.drip.service.sample

This package contains classes the illustrate the full usage of the functionality provided by FI – the inputs, the calculation results, the product creation and the product state management details, extraction of the different product measures, curve construction from user-supplied/live/EOD measures/quotes for discount curves, credit curves, FX/treasury curves, as well as calculation of the several RV/valuation measures.

BondAnalyticsAPISample

This class contains a demo of the bond analytics API sample. It has functions illustrating the creation and the usage of the principal and coupon schedules, custom fixed, floaters, and custom bonds from arbitrary cash flows, bond with option schedules, run through of the full set of bond relative value analytics, and calibration of credit curves from CDS/bond prices and their corresponding measures.

BondBasketAPISample

This class contains a demo of the bond basket API sample. It has functions illustrating the creation and the usage of the bond basket from its components, and a run through of the full set of bond basket value.

BondLiveAndEODAPISample

This class contains a demo of the live and EOD API calls for bonds. It displays the full set of RV, interest rate, and credit measures for a given bond identifier (ISIN/CUSIP) given closing price/yield/spread to treasury, full set of RV measures for all the bonds in a given ticker, their individual and bucketed outstanding notional, full set of valuation and relative value measures, coupon flows, and loss flows for a given identifier.

BondStaticAPISample

This class contains a demo of the API calls for bond static fields. It displays the full set of static fields for a given bond identifier (ISIN/CUSIP).

CDSAnalyticsAPISample

This class contains a demo of the CDS analytics API sample. It has functions illustrating the creation and the usage of CDS, display of their coupon and loss flows, and calibration of credit curves from CDS prices of all types, and their corresponding measures, as well as from the survival probabilities and hazard rates.

CDSBasketAPISample

This class contains a demo of the CDS basket API sample. It has functions illustrating the creation and the usage of pre-set and pre-loaded CDS basket, the full set of named CDX indices and their descriptions, and a run through of the full set of bond basket value.

CDSLiveAndEODAPISample

This class contains a demo of the live and EOD API calls for CDS. It displays the full set of RV, interest rate, and credit measures for a given CDS given several forms of CDS quotes, and the creation of the calibrated credit curve for the given CDS curve name and EOD.

DayCountAndCalendarAPISample

This class contains a demo of the day count and the calendar analytics API sample. It has functions display of the built-in holiday locations, week days, weekends, and holiday sets between a start date and an end date for a set of holiday calendars, available day count calendars, and samples to adjust/roll according to holiday sets.

FXAPISample

This class contains a demo of the FX analytics API sample. It has functions to create and use FX spot, FX forward curves (as either outright or as PIP), create basis curves both as spot-starting basis as well as forward basis to the domestic/foreign rates curves, and recovers the forward curve (in differing formats) from the corresponding basis.

RatesAnalyticsAPISample

This class contains a demo of the rates analytics API sample. It has functions illustrating the creation and the usage of different rates instruments (cash/futures/swaps), display of their coupon flows, and calibration of rates curves from discount factors and forward rates, and their corresponding measures.

RatesLiveAndEODAPISample

This class contains a demo of the live and EOD API calls for rates instruments. It displays the full set cash/EDF/swaps/treasury/composite types of rates curves available on a EOD, their calibration, creation and usage.

Package org.drip.util.external

This package provides the set of interfaces and static functionality exposable to the external CreditAnalytics API users.

FIGen

FIGen provides a collection of the static generic utility functions used by the internal and external DRIP modules.

The following lists the set of functionality that FIGen provides (consult the javadoc and the source for more details on the corresponding APIs):

- Initializes DRIP<->Bloomberg holiday code map, and rates curve switcher map.
- Utility functions to parse Bloomberg holiday codes, and the get tenor/month code from frequency.
- Construct the default rate index string from the currency and the coupon frequency.
- Create a DRIP Julian Date from java date.
- Convert string to Boolean.
- Make Oracle date from YYYYMMDD string/ Bloomberg date string.
- Make double array from string tokenizer.
- Format input number into different requested formats (with optional padding).
- Create a fixings object for the bond, value date, and the fix coupon.
- Generate a GUID string.
- Generate the credit products loss period in accordance with the parameters specified in the pricer.

Validatable

Validatable is the interface that exposes the “validate” method. This method is needed to validate the state of the implementing object.

Package org.drip.util.date

This package implements the Julian date and time functionality used internally by CreditAnalytics.

DateTime

DateTime class provides complete representation of the instantiation-time date and time objects. It retrieves the date and time at the instance of creation (or an arbitrarily set values).

DateTime implements the serialization interface to serialize the instance into byte arrays, as well as de-serialize and populate a DateTime instance from an input byte array.

JulianDate

JulianDate class privides a comprehensive representation of the Julian Date and the date manipulation functionality. Julian date is canonically represented as a double.

As a generic implementation of the date class, JulianDate provides the following functionality:

- Converts year, month, and day to Julian double.
- Converts the Julian double to Y/M/D string.
- Extracts the year, month, or day from the input Julian double.
- Computes days elapsed in the year, days remaining in the year, or is the current day a leap year.
- Finds out how many “leap days” are between two given dates – right/left inclusive.

- Characters corresponding to the month/day in a few formats (regular/Oracle), and vice versa.
- Number of days in the given month, whether the given Julian double corresponds to the end-of-month.
- Different ways of constructing the Julian date object – Today, from YMD, from DDMMMYYYY, or from simply a Julian double.
- Gets the object’s Julian double.
- Add/subtract days/business days/months/years/tenor.
- Gets the first EDF start date (based on the EDF roll months).
- Difference to/equal to/compare with another date,
- Get Oracle string representation, and the hash code.

Package org.drip.util.internal

This package implements the internal utilities used by other CreditAnalytics modules.

FIUtil

FIUtil provides a set of utility functions meant primarily for other internal CreditAnalytics modules. Its functions calculate the yield DF given the yield and time fraction, and the treasury benchmark identifier from the valuation date and the maturity date.

Logger

The logger implements level-set logging, backed by either the screen or a file. Logging always includes time-stamps, and happens according to the level requested.