



Credit Trader Suite User/Developer Guide

Lakshmi Krishnamurthy
v2.2 14 August 2013

Introduction

[Credit Trader Suite](#) of libraries aims to provide open source analytics and trading/valuation system solution suite for credit and fixed income products. To this end, it implements its functionality in a single library over 5 main components – [CreditProduct](#), [CreditAnalytics](#), [SplineLibrary](#), [FixedPointFinder](#), and [RegressionSuite](#).

Overview and Problem Space Coverage

The main challenges that [Credit Trader Suite](#) attempts to address are:

- Implementation of day count conventions, holidays calendars, and rule-based period generators
- Abstract the functionality behind curves, parameters, and products onto defined interfaces
- Unified/standardized methods for curve calibrations, parameter and product implementers and constructors
- Environmental system to hold live ticks, closing marks, and reference data containers
- Enhanced analytics testing
- Ease of usage, installation, and deployment

While a number of other libraries - both Open Source implementations such as [Quantlib](#) and its [variants](#) and proprietary systems such as [Fincad](#), [NumeriX](#), [Algorithmics](#), they typically cater to the needs of the wider financial mathematics community, thus diluting their value towards treating credit products. Further, few of them inherently export a curve/product/product models that work well with products quotes, marks, and reference data sources, thereby forcing development teams to build their own integration layers from scratch. Finally, building the components of functional credit-trading system requires additional layers of development over analytics.

[Credit Trader Suite](#) is an attempt to overcome these shortcomings. It aims to bring the aspects mentioned above together in one Open Source implementation that readily integrates onto existing systems.

Main Libraries and their Purpose

The libraries the constitute the Credit Trader Suite are:

- [CreditProduct](#) – Focused on the core analytics, and the curve, the parameter, and the product definitions.
- [CreditAnalytics](#) – Concerned with the construction and the implementation of the interfaces defined in CreditProduct, analytics environment management, and functional distribution.
- [SplineLibrary](#) – Provides the functionality for building, calibrating, and evaluating different kinds of splines.
- [FixedPointFinder](#) – Provides the implementation of all the standard bracketing and open root finding techniques, along with a customizable and configurable framework that separates the initialization/bracketing functionality from the eventual root search.
- [RegressionSuite](#) – This aims to ease testing of analytics, measurement and generation of the execution time distribution, as well as release performance characterization.

CreditProduct Description and Problem Space Coverage

[CreditProduct](#) aims to define the functional and behavioral interfaces behind curves, products, and different parameter types (market, valuation, pricing, and product parameters). To facilitate this, it implements various day count conventions, holiday sets, period generators, and calculation outputs.

[CreditProduct](#) library achieves its design goal by implementing its functionality over several packages:

- Dates and holidays coverage: Covers a variety of day count conventions, 120+ holiday locations, as well as custom user-defined holidays
- Curve and analytics definitions: Defines the base functional interfaces for the variants of discount curves, credit curves, and FX curves
- Market Parameter definitions: Defines quotes, component/basket market parameters, and custom scenario parameters
- Valuation and Pricing Parameters: Defines valuation, settlement/work-out, and pricing parameters of different variants
- Product and product parameter definitions: Defines the product creation and behavior interfaces for Cash/EDF/IRS (all rates), bonds/CDS (credit), and basket bond/CDS, and their feature parameters.
- Output measures container: Defines generalized component and basket outputs, as well customized outputs for specific products

CreditAnalytics Description and Problem Space Coverage

[CreditAnalytics](#) provides the functionality behind creation, calibration, and implementation of the curve, the parameter, and the product interfaces defined in [CreditProduct](#). It also implements a curve/parameter/product/analytics management environment, and has packaged samples and testers.

[CreditAnalytics](#) library achieves its design goal by implementing its functionality over several packages:

- Curve calibration and creation: Functional implementation and creation factories for rates curves, credit curves, and FX curves of all types
- Market Parameter implementation and creation: Implementation and creation of quotes, component/basket market parameters, as well as scenario parameters.
- Product implementation and creation: Implementation and creation factories for rates products (cash/EDF/IRS), credit products (bonds/CDS), as well as basket products.

- Reference data/marks loaders: Loaders for bond/CDX, as well a sub-universe of closing marks
- Calculation Environment Manager: Implementation of the market parameter container, manager for live/closing curves, stub/client functionality for serverization/distribution, input/output serialization.
- Samples: Samples for curve, parameters, product, and analytics creation and usage
- Unit functional testers: Detailed unit scenario test of various analytics, curve, parameter, and product functionality.

SplineLibrary Description and Problem Space Coverage

[SplineLibrary](#) provides the functionality for building, calibrating, and evaluating different kinds of splines. It implements the functionality behind spline design, spline constructions, customization, calibration, and evaluation of a wide variety of spline types and basis functions.

[SplineLibrary](#) achieves its design goals by implementing its functionality over several packages the perform the following:

- Spline Calibration Framework: Schematic spline segment/span partitioning, elastic/inelastic entity-variate separation, mathematical calibration formulation framework with constraints/boundary incorporation, sensitivity Jacobian estimator setup, Hard/Soft (Truthness/Smoothness) partitioning.
- Shape Preserving Spline Suite: B-splines, Polynomial Splines and Variants (Bernstein, Kakkis-Pandelis, Manni), Integrated Tension Splines (exponential/hyperbolic), partitioned tension splines (explicit shape control using rational splines).
- Space Spline Curves: Hermitian, Catmull-Rom, Cardinal Cubic, Space Loops.
- Spline Calibration: Explicit segment formulation frame work, Optimizing shape preserving splines, curvature (or criterion-defined) penalty minimizing mu-splines, segment/span self-Jacobian and micro-Jacobian.

- Spline Evaluation: Metric'ed evaluation of monotonicity, convexity, smoothness, smoothness, locality, and approximation order.
- Spline Extensions: Variational Smoothing, density smoothing technique, alternate smoothener customization extension, multi-dimensional (including thin plate) splines.

FixedPointFinder Description and Problem Space Coverage

[RootFinder](#) achieves its design goal by implementing its functionality over several packages:

- Framework: Framework to accommodate bracketing/open convergence initialization, execution customization/configuration, iterative variate evolution, and search termination detection
- Bracket Initialization Techniques: Implementation of the different techniques for the initial bracket extraction.
- Open Method Convergence Zone Initialization Techniques: Implementation of the different techniques for the convergence zone starting variate extraction.
- Iterative Open Methods: Implementation of the iterative Open Methods - Newton-Raphson and Secant Methods
- Iterative Bracketing Primitive Methods: Implementation of the iterative bracketing primitives – Bisection, False Position, Quadratic Interpolation, Inverse Quadratic Interpolation, and Ridder.
- Iterative Bracketing Compound Methods: Implementation of the iterative bracketing compound methodologies – Brent's and Zheng's methods.
- Search Initialization Heuristics: Implementation of a number of search heuristics to make the search targeted and customized.
- Samples: Samples for the various bracketing and the open methods, their customization, and configuration.
- Documentation: Literature review, framework description, mathematical and formulation details of the different components, root finder synthetic knowledge unit (SKU) composition, and module and API usage guide.

- Regression Tests: Statistical regression analysis and dispersion metric evaluation for the initialization and the iteration components of the different bracketing and open root finder methodologies.

RegressionSuite Description and Problem Space Coverage

[RegressionSuite](#) aims to incorporate measurement of the startup lag, measurement of accurate execution times, generating execution statistics, customized input distributions, and processable regression specific details as part of the regular unit tests.

[RegressionSuite](#) library achieves its design goal by implementing its functionality over several packages:

- Regression Engine: Provides control for distribution set, invocation strategy, and load.
- Unit Regression Executor: Framework that implements set up and tear-down, as well as generate run details
- Regression Statistics: Execution time distribution, start-up and other event delay measurements, and system load monitoring
- Regression Output: Fine grained regressor level output, module aggregated output, sub-element execution time estimation.
- Regressor Set: Module containing set of regressors, group level turn on/off and execution control
- Regression Utilities: Formatting and tolerance checking.

Design Objectives

This section covers the design objectives across several facets – functional, software, system, usage, and deployment aspects.

Financial Feature design attributes

The chief design aims from a financial functionality angle are:

- Interface representations of curve, parameter, and products
- Separation of the creation and the implementation modules from the exposed functional interface behavior
- Re-usable functional and behavioral abstractions around financial products
- Provide “open” public implementations of the standard analytics functionality such as day count conventions, holidays, date representations, rule based period generation etc
- Abstraction of the quote, the market parameter and the pricing structures
- Abstraction and implementation of the standard curve calibration

Software Feature design attributes

The chief design aims from a software design angle are:

- Logical functionality identification/localization and functional group partitioning
- Clearly defined interface structure
- Implementation and creation factory bodies
- Reach and interaction through interfaces only

System Feature Design Attributes

The key system design aims are:

- Functionality needs to be readily serverizable and distributable
- Provide built in serialization, marshalling, and persistence of all the main components
- Management containers around the products, the curves, and the parameter sets, and establishing the execution control environment

Analytics Usage Design Objectives

The key usage design goals are:

- The analytics modules should provide comprehensive credit product risk, valuation, and pricing functionality from a set of functional API
- Ease of use
- Flexible
- When direct object access is needed, use only through the object model interface (and amend the interface as appropriate)

Test Design Objectives

The key testing design goals in this area are:

- Comprehensive unit testing of curve, parameters, and product implementation
- Extensive composite scenario testing
- Environment and server components testing
- Release time performance characterization and execution time and execution resource statistics calculation

Installation, Dependency, and Deployment Design Objectives

The key design goals in this area are:

- Minimize dependency on external modules
- Ease of installation and deployment
- Customizability – for non-standard setups – through the supplied configuration file.

Credit Product

Credit Product Library consists of the following 14 packages:

1. Date & Time Manipulators: This contains functionality for creating, manipulating, and adjusting dates, as well as time instants (to nano-second granularity).
2. Day-count Parameters, Conventions, and Date Adjustment Operations: This contains the functionality for day count generation and date adjustment according to specific rules. It also holds parameters needed for specific day count conventions.
3. Location Specific Standard Holiday Set: This contains all the non-weekend holidays that correspond to a specific location jurisdiction, and its description. Each location implements its holidays in a separate class.
4. Custom Holidays: This provides the ability to specify custom holidays, if the standard ones provided earlier are insufficient. Different types of holidays can be added – variable, fixed, static, as well as weekends for a given location.
5. Curve Analytics Definitions: This provides the definition of all the curve objects – the base curve, the rates curves (discount curves and zero curves), credit curves, and the FX curves (FX basis and FX forward curves).
6. Cash flow Period: This contains the cash flow period functionality, as well as place holders for the period related different curve factors.
7. Analytics Support Utilities: This contains utility functions for manipulating the core Credit Product modules, generic utility functions, and a logger.

8. Quotes, Market, and Scenario Parameters Definitions: This contains the classes that implement the definitions for all parameters except product feature parameters – quotes, calibration parameters, market parameters, tweak parameters, and the scenario curves.
9. Pricer Parameters: This contains the pricing parameters corresponding to a given product and model.
10. Quoting Parameters: This contains the quoting parameters needed to interpret a product quote.
11. Valuation Parameters: This contains all the non-market and non-product parameters needed for valuing a product at a given date.
12. Product Definitions: This contains interface definitions for all products, along with definitions for credit, rates, and FX components and specific credit/rates/FX products, and baskets.
13. Product Parameters: This contains the implementations of the features required for a complete construction of an instance of the product.
14. Product RV and Batch Calculation Outputs: This contains the bulk results of pricing and relative value calculation for the products.
15. Serializer: This interface defines the core object serialization methods – serialization into and de-serialization out of byte arrays, as well as the object serializer version.

Credit Product: Date Time Manipulators

[Date Time Manipulators](#) are implemented in the package [org.drip.analytics.date](#). It contains functionality for creating, manipulating, and adjusting dates, as well as time instants (to nano-second granularity).

The functionality is implemented in 2 classes: [DateTime](#) and [JulianDate](#), and both are serializable.

JulianDate

This class provides a comprehensive representation of Julian date and date manipulation functionality. It exports the following functionality:

- Explicit date construction, as well as date construction from several input string formats
- Date Addition/Adjustment, add/subtract days/weeks/months/years and tenor codes
- Leap Year Functionality (number of leap days in the given interval, is the given year a leap year etc.)
- Generate the subsequent IMM date (EDF/CME IMM date, CDS/Credit ISDA IMM date etc)
- Export the date to a variety of date formats.

DateTime

[DateTime](#) contains nano-second level instantiation-time snap of date/time instant.

Credit Product: Day Count Parameters, Conventions, and Date Adjustment Operations

[Day Count Calculators](#) are implemented in the package [org.drip.analytics.daycount](#). It contains the functionality for day count generation and date adjustment according to specific rules. It also holds parameters needed for specific day count conventions.

The functionality is implemented across 18 classes: [ActActDCParams](#), [Convention](#), and [DateAdjustParams](#), [DateEOMAdjustment](#), [DC30_360](#), [DC30_365](#), [DC30_Act](#), [DC30E_360](#), [DCAct_360](#), [DC30_364](#), [DC30_365](#), [DC30_365L](#), [DCAct_Act_ISDA](#), [DCAct_Act](#), [DCFCalculator](#), [DCNL_360](#), [DCNL_364](#), and [DCNL_Act](#).

ActActDCParams

This class contains parameters to represent the Act/Act day count - the frequency, and the reference period start/end dates.

Convention

This class contains flags that indicate where the holidays are loaded from, as well as the holiday types and load rules. It exports the following date related functionality:

- Add business days according to the specified calendar
- The Year Fraction between any 2 days given the day count type and the holiday calendar
- Adjust/roll to the next working day according to the adjustment rule

- Holiday Functions - is the given day a holiday/business day, the number and the set of holidays/business days between 2 days.
- Calendars and Day counts - Available set of day count conventions and calendars, and the weekend days corresponding to a given calendar.

DateAdjustParams

This class contains the parameters needed for adjusting dates – holiday calendar and adjustment type.

DateEOMAdjustment

This class holds the applicable anterior and posterior EOM adjustments for a given date pair.

DCFCalculator

This interface is the stub for all the day count convention functionality. It exposes the base/alternate day count convention names, the year-fraction and the days accrued.

DC30_360

This class implements the 30/360 day count convention.

DC30_365

This class implements the 30/365 day count convention.

DC30 Act

This class implements the 30/Act day count convention.

DC30E 360

This class implements the 30E/360 day count convention.

DCAct 360

This class implements the Act/360 day count convention.

DCAct 364

This class implements the Act/364 day count convention.

DCAct 365

This class implements the Act/365 day count convention.

DCAct 365L

This class implements the Act/365L day count convention.

DCAct_Act_ISDA

This class implements the Act/Act ISDA day count convention.

DCAct_Act

This class implements the Act/Act day count convention.

DCNL_360

This class implements the NL/360 day count convention.

DCNL_365

This class implements the NL/365 day count convention.

DCNL_Act

This class implements the NL/Act day count convention.

Credit Product: Location Specific Standard Holiday Set

[Location Specific Holidays](#) are implemented in the package [org.drip.analytics.holset](#). It contains all the holidays that correspond to a specific location jurisdiction, and its description.

The functionality is implemented in its own location qualified class instance - each of which is an instance of the [LocationHoliday](#) interface.

LocationHoliday

LocationHoliday is an interface that is implemented by all the Location Holiday classes. It exposes the specific holiday location, as well as the set of location-specific holidays.

Other classes in this package provide explicit holidays and the locale name. So far, [Credit Product](#) has about 130 locales implemented – please consult the [Credit Analytics site](#) for what they are.

Credit Product: Custom Holidays

[Custom Holiday creators](#) are implemented in the package [org.drip.analytics.holiday](#). It provides the ability to add holidays, if the standard ones provided earlier are insufficient. Different types of holidays can be added – [variable](#), [fixed](#), [static](#), as well as [weekends](#) for a given location.

Different holiday types are implemented in their own classes – they are [Static](#), [Fixed](#), and [Variable](#), each of which extends the [Base holiday class](#). [Weekend](#) is implemented in a separate class. All holiday instances for a given [Locale](#) are maintained on a named holiday container.

Base

Base is an abstraction around holiday and description. Abstract function generates an optional adjustment for weekends in a given year.

Weekend

Weekend holds the left and the right weekend days

Static

Static implements a complete date as a specific holiday.

Fixed

Fixed contains the fixed holiday's date and month. Holidays are generated on a per-year basis by applying the year, and by adjusting the date generated.

Variable

Variable class contains the rule characterizing the variable holiday's month, day in week, week in month, and the weekend days. Specific holidays in the given year are generated using these rules.

Locale

Locale contains the set of regular holidays and the weekend holidays for a location. It also provides the functionality to add custom holidays and weekends.

Credit Product: Curve Analytics Definitions

[Curve Analytics Curve definitions](#) are implemented in the package [org.drip.analytics.definition](#). It provides the definition of all the curve objects – the base curve, the rates curves (discount curves and zero curves), credit curves, and the FX curves (FX basis and FX forward curves).

Functionality in this package is implemented across 6 classes – [Curve](#), [DiscountCurve](#), [ZeroCurve](#), [CreditCurve](#), [FXBasisCurve](#), and [FXForwardCurve](#).

Curve

Curve implements the interfaces across all curve instances. It exposes the following functionality:

- Set the effective/start date
- Set up calibration run framework (initialize the build run, build the interpolator etc.)
- Generate scenario curves from the base curve (flat/parallel/custom)
- Set/retrieve the components and their quotes

DiscountCurve

DiscountCurve is the stub for the discount curve functionality. It extends the Curve object by exposing the following functions:

- Forward Rate to a specific date/tenor, and effective rate between a date interval.
- Discount Factor to a specific date/tenor, and effective discount factor between a date interval.

- Zero Rate to a specific date/tenor.
- Generate scenario curves from the base discount curve (flat/parallel/custom).
- Value Jacobian for Forward rate, discount factor, and zero rate.
- Cross Jacobian between each of Forward rate, discount factor, and zero rate.
- Quote Jacobian to Forward rate, discount factor, and zero rate.

ZeroCurve

ZeroCurve exposes the node set containing the zero curve node points. In addition to the discount curve functionality that it automatically provides by extension, it provides the functionality to calculate the zero rate.

CreditCurve

CreditCurve is the stub for the survival curve functionality. It extends the Curve object by exposing the following functions:

- Recovery to a specific date/tenor, and effective recovery between a date interval.
- Hazard Rate to a specific date/tenor, and effective hazard rate between a date interval.
- Survival to a specific date/tenor, and effective survival between a date interval.
- Set/unset date of specific default.
- Generate scenario curves from the base credit curve (flat/parallel/custom)

FXBasisCurve

FXBasisCurve implements the curve representing the FXBasis nodes. It extends the Curve class, and exposes the following functionality:

- Retrieve the spot parameters (FX Spot, Spot Date, and the currency pair).

- Indicate if the basis has been bootstrapped.
- Calculate the Full set of FX Forward corresponding to each basis node.

FXForwardCurve

FXForwardCurve implements the curve representing the FXForward nodes. It extends the Curve class, and exposes the following functionality:

- Retrieve the spot parameters (FX Spot, Spot Date, and the currency pair).
- Calculate the Full set of FX Basis/Zero Rate nodes corresponding to each basis node.
- Bootstrap basis points/discount curves corresponding to the FXForward node set.
- Imply the zero rate to a given date from the FXForward curve.

Credit Product: Cash flow Period

[Cash flow period](#) functionality is implemented in the package [org.drip.analytics.period](#). It contains the cash flow period functionality, as well as place holders for the period related different curve factors.

Functionality in this package is implemented across 4 classes – [Period](#), [CouponPeriod](#), [CouponPeriodCurveFactors](#), and [LossPeriodCurveFactors](#).

Period

Period serves as a holder for the period dates: period start/end, period accrual start/end, pay, and full period day count fraction.

CouponPeriod

CouponPeriod extends the period class with the following coupon day-count specific parameters: frequency, reset date, and accrual day-count convention. It also exposes static methods to construct coupon period sets starting backwards/forwards, as well as merge coupon periods.

CouponPeriodCurveFactors

CouponPeriodCurveFactors is an enhancement of the period class using the following period measures: start/end survival probabilities, start/end notionals, and period start/end discount factor.

LossPeriodCurveFactors

LossPeriodCurveFactors is an implementation of the period class enhanced by the following period measures:

- Start/end survival probabilities
- Period effective notional/recovery/discount factor

Credit Product: Analytics Support Utilities

[Analytics Support](#) functionality is implemented in the package [org.drip.analytics.support](#). It contains utility functions for manipulating the [Credit Product](#) modules, case insensitive maps, and a logger.

Functionality in this package is implemented across 4 classes – [AnalyticsHelper](#), [CaseInsensitiveHashMap](#), [CaseInsensitiveTreeMap](#), and [Logger](#).

AnalyticsHelper

AnalyticsHelper contains the collection of the analytics related utility functions used by the modules. The following is the functionality that it exposes:

- Yield to Discount Factor, and vice versa.
- Map Bloomberg Codes to CreditAnalytics Codes.
- Generate rule-based curve bumped nodes.
- Generate loss periods using a variety of different schemes.
- Aggregate/disaggregate coupon period lists.

CaseInsensitiveHashMap

CaseInsensitiveMap implements a case insensitive key in a hash map.

CaseInsensitiveTreeMap

CaseInsensitiveMap implements a case insensitive key in a hash map

Logger

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

Credit Product: Quote, Market, and Scenario Parameters

[Quote, Market, Tweak, and Scenario parameter definitions](#) are specified in the package [org.drip.param.definition](#). It contains the classes that implement the definitions for all parameters except product feature parameters – quotes, calibration parameters, market parameters, tweak parameters, and the scenario curves.

Functionality in this package is implemented across 10 classes and 5 groups – [CalibrationParams](#), the quote parameters group ([Quote](#), and [ComponentQuote](#)), the tweak parameters group ([NodeTweakParams](#), and [CreditNodeTweakParams](#)), and the scenario curves group ([RatesScenarioCurve](#) and [CreditScenarioGroup](#)), and the market parameters group ([ComponentMarketParams](#), [BasketMarketParams](#), [MarketParams](#)).

CalibrationParams

CalibrationParams the calibration parameters - the measure to be calibrated, the type/nature of the calibration to be performed, and the work-out date to which the calibration is done.

Quote

Quote interface contains the stubs corresponding to a product quote. It contains the quote value and quote instant for the different quote sides (bid/ask/mid).

ComponentQuote

ComponentQuote abstract class holds 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).

NodeTweakParams

NodeTweakParams is the place-holder for the scenario tweak parameters, for either a specific curve node, or the entire curve (flat). Parameter bumps can be parallel or proportional.

CreditNodeTweakParams

CreditNodeTweakParams is the place-holder for the credit curve scenario tweak parameters: the measure, the curve node, and the nodal calibration type (entire curve/flat or a given tenor point).

RatesScenarioCurve

RatesScenarioCurve abstract class exposes the interface the constructs scenario discount curves. The following curve construction scenarios are supported:

- Base, flat/tenor up/down by arbitrary bumps.
- Tenor bumped discount curve set - keyed using the tenor.
- NTP-based custom scenario curves.

CreditScenarioCurve

CreditScenarioCurve abstract class exposes the bump parameters and the curves for the following credit curve scenarios:

- Base, Flat Spread/Recovery bumps.
- Spread/Recovery Tenor bumped up/down credit curve sets keyed using the tenor.
- NTP-based custom scenario curves.

ComponentMarketParams

ComponentMarketParams abstract class provides stub for the ComponentMarketParamsRef interface. It is a place-holder for the market parameters needed to value the component object – the discount curve, the forward curve, the treasury curve, the EDSF curve, the credit curve, the component quote, the treasury quote map, and the fixings map.

BasketMarketParams

BasketMarketParams class extends the BasketMarketParamsRef for a specific scenario. It provides access to maps holding named discount curves, named credit curves, named treasury quote, named component quote, and fixings object.

MarketParams

MarketParams is the place-holder for the comprehensive suite of the market set of curves for the given date. It exports the following functionality:

- Add/remove/retrieve scenario discount curve.
- Add/remove/retrieve scenario zero curve.

- Add/remove/retrieve scenario credit curve.
- Add/remove/retrieve scenario recovery curve.
- Add/remove/retrieve scenario FXForward curve.
- Add/remove/retrieve scenario FXBasis curve.
- Add/remove/retrieve scenario fixings.
- Add/remove/retrieve Treasury/component quotes.
- Retrieve scenario CMP/BMP.
- Retrieve map of flat rates/credit/recovery CMP/BMP.
- Retrieve double map of tenor rates/credit/recovery CMP/BMP.
- Retrieve rates/credit scenario generator.

Credit Product: Pricing Parameters

[Pricing parameter](#) is implemented in the package [org.drip.param.pricer](#). It contains the pricing parameters corresponding to a given product and model.

Currently only the credit-pricing model is implemented – it is implemented in [PricerParams](#).

PricerParams

PricerParams contains the pricer parameters - the discrete unit size, calibration mode on/off, survival to pay/end date, and the discretization scheme.

Credit Product: Quoting Parameters

[Pricing parameter](#) is implemented in the package [org.drip.param.quoting](#). This contains the quoting parameters needed to interpret a product quote.

Functionality in this package is implemented across 3 classes – [MeasureInterpreter](#), [QuotedSpreadInterpreter](#), and [YieldInterpreter](#).

MeasureInterpreter

MeasureInterpreter is the abstract shell stub class from which all product measure quoting parameters are derived. It contains fields needed to interpret a measure quote.

QuotedSpreadInterpreter

QuotedSpreadInterpreter holds the fields needed to interpret a Quoted Spread Quote. It contains the contract type and the coupon.

YieldInterpreter

YieldInterpreter holds the fields needed to interpret a Yield Quote. It contains the quote day count, quote frequency, quote EOM Adjustment, quote Act/Act parameters, and quote Calendar.

Credit Product: Valuation Parameters

[Valuation parameters](#) are implemented in the package [org.drip.param.valuation](#). It contains all the non-market and non-product parameters needed for valuing a product at a given date.

Functionality in this package is implemented across 4 classes – [QuotingParams](#), [CashSettleParams](#), [WorkoutInfo](#), and [ValuationParams](#).

CashSettleParams

CashSettleParams is the place-holder for the cash settlement parameters for a given product. It contains the cash settle lag, the calendar, and the date adjustment mode.

QuotingParams

QuotingParams holds the parameters needed to interpret the input quotes. It contains the quote day count, quote frequency, quote EOM Adjustment, quote Act/Act parameters, and quote Calendar. It also indicates if the native quote is spread based.

ValuationParams

ValuationParams is the place-holder for the valuation parameters for a given product. It contains the valuation and the cash pay/settle dates, as well as the calendar. It also exposes a number of methods to construct standard valuation parameters.

WorkoutInfo

WorkoutInfo is the place-holder for the work-out parameters. It contains the date, the factor, the type, and the yield of the work-out.

Credit Product: Product Definitions

[Product definitions](#) are implemented in the package [org.drip.product.definition](#). It contains interface definitions for all products, along with definitions for credit, rates, and FX components and specific credit/rates/FX products, and baskets.

[Product definitions](#) are implemented in different groups – base component group ([ComponentMarketParamsRef](#), [Component](#), [CalibrateComponent](#)), base basket group ([BasketMarketParamRef](#), [BasketProduct](#)), [RatesComponent](#), Credit Component Group ([CreditComponent](#), [CreditDefaultSwap](#), [BondProduct](#), [Bond](#)), and FX Component group ([FXSpot](#) and [FXForward](#)).

ComponentMarketParamRef

ComponentMarketParamRef interface provides stubs for component name, IR curve, forward curve, credit curve, TSY curve, and EDSF curve needed to value the component.

Component

Component abstract class extends ComponentMarketParamRef and provides the following methods:

- Get the component's initial notional, notional, and coupon.
- Get the Effective date, Maturity date, First Coupon Date.
- List the coupon periods.
- Set the market curves - discount, TSY, forward, Credit, and EDSF curves.
- Retrieve the component's settlement parameters.

- Value the component using standard/custom market parameters.
- Retrieve the component's named measures and named measure values.

CalibratableComponent

CalibratableComponent abstract class provides implementation of Component's calibration interface. It exposes stubs for getting/setting the component's calibration code, generate calibrated measure values from the market inputs, and compute micro-Jacobians (QuoteDF and PVDF micro-Jacks).

BasketMarketParamRef

BasketMarketParamRef interface provides stubs for component's IR and credit curves that constitute the basket.

BasketProduct

BasketProduct abstract class extends BasketMarketParamRef. It provides methods for getting the basket's components, notional, coupon, effective date, maturity date, coupon amount, and list of coupon periods.

RatesComponent

RatesComponent is the abstract class that extends CalibratableComponent on top of which all rates components are implemented.

CreditComponent

CreditComponent is the base abstract class on top of which all credit components are implemented. Its methods expose Credit Valuation Parameters, and coupon/loss cash flows.

CreditDefaultSwap

CreditDefaultSwap is the base abstract class implements the pricing, the valuation, and the RV analytics functionality for the CDS product.

BondProduct

BondProduct interface implements the product static data behind bonds of all kinds. Bond static data is captured in a set of 11 container classes – BondTSYParams, BondCouponParams, BondNotionalParams, BondFloaterParams, BondCurrencyParams, BondIdentifierParams, ComponentValuationParams, ComponentRatesValuationParams, ComponentCreditValuationParams, ComponentTerminationEvent, BondFixedPeriodParams, and one EmbeddedOptionSchedule object instance each for the call and the put objects. Each of these parameter sets can be set separately.

Bond

Bond abstract class implements the pricing, the valuation, and the RV analytics functionality for the bond product.

FXSpot

FXSpot is the abstract class exposes the functionality behind the FXSpot Contract. Derived implementations return the spot date and the currency pair.

FXForward

FXForward is the abstract class exposes the functionality behind the FXForward Contract. Derived implementations expose the primary/secondary codes, the effective/maturity dates, the currency pair, imply the discount curve basis and the FX Forward from a set of market parameters. The value function carries out a full valuation.

Credit Product: Product Parameters

[Product parameter definitions](#) are implemented in the package [org.drip.product.params](#). It contains the implementations of the features required for a complete construction of an instance of the product.

[Product parameters](#) are implemented across 20 classes. [Validatable](#) is the base interface that underpins most of them. Others are identifier parameters ([CDXIdentifier](#), [CDXRefDataParams](#), [IdentifierSet](#), [StandardCDXParams](#)), [CouponSetting](#), [CreditSetting](#), [CurrencySet](#), [EmbeddedOptionSchedule](#), [FactorSchedule](#), [NotionalSetting](#), [PeriodGenerator](#), [PeriodSet](#), [FloaterSetting](#), [RatesSetting](#), [TerminationSetting](#), [QuoteConvention](#), Treasury Parameters ([TreasuryBenchmark](#), [TsyBmkSet](#)), and [CurrencyPair](#).

Validatable

Validatable interface defines the validate function, which validates the current object state.

CDXIdentifier

CDXIdentifier implements the creation and the static details of the all the NA, EU, SovX, EMEA, and ASIA standardized CDS indexes. It contains the index, the tenor, the series, and the version of a given CDX.

CDXRefDataParams

CDXRefDataParams contains the complete set of reference data that corresponds to the contract of a standard CDX. It consists of the following category and fields:

- Descriptive => Index Label, Index Name, Curve Name, Index Class, Index Group Name, Index Short Group.
- Name, Index Short Name, Short Name.
- Issuer ID => Curve ID, Red ID, Series, Version, Curvy Curve ID, Location, Bloomberg Ticker.
- Quote Details => Quote As CDS.
- Date => Issue Date, Maturity Date.
- Coupon Parameters => Coupon Rate, Currency, Day Count, Full First Stub, Frequency.
- Component Details => Original Count, Defaulted Count.
- Payoff Details => Knock out on Default, Pay Accrued Amount, Recovery on Default.
- Other => Index Life Span, Index Factor

IdentifierSet

IdentifierSet contains the component's identifier parameters - ISIN, CUSIP, ID, and ticker.

StandardCDXParams

StandardCDXParams implements the parameters used to create the standard CDX - the coupon, the number of components, and the currency.

CouponSetting

CouponSetting contains the coupon type, schedule, and the coupon amount for the component. If available, the floor and/or the ceiling may also be applied to the coupon, in a pre-determined order of precedence.

CreditSetting

CreditSetting contains the credit related valuation parameters - use default pay lag, use curve or the component recovery, component recovery, credit curve name, and whether there is accrual on default.

CurrencySet

CurrencySet contains the component's trade, the coupon, and the redemption currencies.

EmbeddedOptionSchedule

EmbeddedOptionSchedule is a place-holder for the embedded option schedule for the component. It contains the schedule of exercise dates and factors, the exercise notice period, and the option is to call or put. Further, if the option is of the type fix-to-float on exercise, contains the post-exercise floater index and floating spread. If the exercise is not discrete (American option), the exercise dates/factors are discretized according to a pre-specified discretization grid.

FactorSchedule

FactorSchedule contains the array of dates and factors.

NotionalSetting

NotionalSetting contains the product's notional schedule and the amount. It also incorporates hints on how the notional factors are to be interpreted - off of the original or the current notional. Further flags tell whether the notional factor is to be applied at the start/end/average of the coupon period.

PeriodGenerator

PeriodGenerator generates the component coupon periods from flexible inputs. Periods can be generated forwards or backwards, with long/short stubs. For customization, date adjustment parameters can be applied to each cash flow date of the period - effective, maturity, period start start/end, accrual start/end, pay and reset can each be generated according to the date adjustment rule applied to nominal period start/end.

PeriodSet

PeriodSet is the place holder for the component's period generation parameters. It contains the component'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.

FloaterSetting

FloaterSetting contains the component's floating rate parameters. It holds the rate index, floater day count, and one of either the coupon spread or the full current coupon.

RatesSetting

RatesSetting contains the rate related valuation parameters - the discount curves to be used for discounting the coupon, the redemption, the principal, and the settle cash flows.

TerminationSetting

TerminationSetting class contains the current "liveness" state of the component, and, if inactive, how it entered that state.

QuoteConvention

QuoteConvention contains the Component Market Convention Parameters - the quote convention, the calculation type, the first settle date, and the redemption amount.

TreasuryBenchmark

TreasuryBenchmark contains component treasury benchmark parameters - the treasury benchmark set, and the names of the treasury and the EDSF IR curves.

TsyBmkSet

TsyBmkSet contains the treasury benchmark set - the primary treasury benchmark, and an array of secondary treasury benchmarks.

CurrencyPair

CurrencyPair class contains the numerator currency, the denominator currency, the quote currency, and the PIP Factor.

Credit Product: Product RV and Batch Calculation Outputs

[Product bulk outputs](#) are implemented in the package [org.drip.analytics.output](#). It contains the bulk results of pricing and relative value calculation for the products.

Outputs are implemented in 6 classes – [ComponentMeasures](#), bond specific calculation outputs ([ExerciseInfo](#), [BondCouponMeasures](#), [BondWorkoutMeasures](#), [BondRVMeasures](#)), and [BasketMeasures](#).

ComponentMeasures

ComponentMeasures is the place-holder for analytical single component output measures, optionally across scenarios. It contains measure maps for the following scenarios:

- Unadjusted Base IR/credit curves
- Flat delta/gamma bump measure maps for IR/credit bump curves
- Tenor bump double maps for IR/credit curves
- Flat/recovery bumped measure maps for recovery bumped credit curves
- Measure Maps generated for Custom Scenarios

ExerciseInfo

ExerciseInfo is a place-holder for the full set of exercise information. It contains the exercise date, the exercise factor, and the exercise type.

BondCouponMeasures

This class encapsulates the parsimonious but complete set of the cash-flow oriented coupon measures generated out of a full bond analytics run to a given work-out. These are:

- DV01
- PV Measures (Coupon PV, Index Coupon PV, PV)

BondWorkoutMeasures

BondWorkoutMeasures encapsulates the parsimonious yet complete set of measures generated out of a full bond analytics run to a given work-out. It contains the following:

- Credit Risky/Credit Riskless Clean/Dirty Coupon Measures
- Credit Risky/Credit Riskless Par/Principal PV
- Loss Measures such as expected Recovery, Loss on instantaneous default, and default exposure with/without recovery
- Unit Coupon measures such as Accrued 01, first coupon/index rate

BondRVMeasures

BondRVMeasures encapsulates the comprehensive set of RV measures calculated for the bond to the appropriate exercise:

- Workout Information.
- Price, Yield, and Yield01.
- Spread Measures: Asset Swap/Credit/G/I/OAS/PECS/TSY/Z.
- Basis Measures: Bond Basis, Credit Basis, Yield Basis.
- Duration Measures: Macaulay/Modified Duration, Convexity

BasketMeasures

BasketMeasures is the place holder for the analytical basket measures, optionally across scenarios. It contains the following scenario measure maps:

- Unadjusted Base Measures
- Flat delta/gamma bump measure maps for IR/credit/RR bump curves
- Component/tenor bump double maps for IR/credit/RR curves
- Flat/component recovery bumped measure maps for recovery bumped credit curves
- Custom scenario measure map

Credit Product: Serializer

[Serializer interface](#) are implemented in the package [org.drip.service.stream](#). The interface defines methods for serializing out of and de-serializing into a byte stream, as well as the object serialization version.

There is just one interface in this package – [Serializer](#).

Serializer

Serializer interface defines the core object serializer methods – serialization into and de-serialization out of byte arrays, as well as the object version.

Credit Analytics Library

Credit Analytics Library consists of the following 16 packages:

1. Curve Implementations: This contains the curve objects implemented using one of the many ways of calibration, and contains concrete implementations of Discount Curve, Zero Curve, Credit Curve, and FX Curves.
2. Curve Calibrators: This contains the curve calibrators that use different calibration schemes, and the curve scenario generators.
3. Curve Creators: This contains the curve object factories for the different curves using the calibration parameters.
4. Reference Data Loaders: This package contains functionality that loads the bond and the CDS reference data, as well as closing marks for a few date ranges.
5. Analytics Configurator: This package contains functionality to configure various aspects of Credit Analytics.
6. Market, Quote, and Scenario Parameter Implementations: This contains the implementations of the Credit Product interfaces representing the quotes, the basket/component market parameters, and the scenario curve containers.
7. Market, Quote, and Scenario Parameter Creators: This contains the builder factories for the quotes, market parameters, and the scenario curves.

8. [Rates Component Implementations](#): This contains the implementations of the [Credit Product interfaces](#) for Cash, Euro-dollar future, fixed/floating streams, interest rate swap instruments, and rates basket products.
9. [Credit Product Implementations](#): This contains the implementations of the [Credit Product interfaces](#) for Bonds, CDS, basket CDS, and bond baskets.
10. [FX Product Implementations](#): This contains the implementation of the [Credit Product interface](#) for FX products.
11. [Product Creators](#): This contains the creators for the various rates, credit, and FX component and basket products.
12. [Analytics Environment Manager](#): This provides functionality for loading products from their reference data and managing them, as well as creating/accessing live/closing curves.
13. [Analytics Bridge](#): This provides the stub and proxy functionality for invoking [Credit Analytics](#) functionality in a remote server and extracting the results.
14. [Analytics API](#): This provides a unified and comprehensive functional, static interface of all the main [Credit Analytics](#) functionality.
15. [Analytics Samples](#): This provides samples illustrating the functionality provided by [Credit Analytics](#) – samples demonstrating the creation and usage of curves and products across rates, credit, and FX components and baskets. Examples are also provided on how to compare against standard analytics vendors/suppliers (e.g., Bloomberg).

16. [Functional Testers](#): This contains a fairly extensive set of unit and composite testers for the curve, products, serialization, and analytics functionality provided by the [Credit Analytics suite](#), with a special focus on bonds.

Credit Analytics: Curve Implementations

[Credit Product curve definitions](#) are implemented in the package [org.drip.analytics.curve](#).

This package contains the curve objects implemented using one of the many ways of calibration, and contains concrete implementations of Discount Curve, Zero Curve, Credit Curve, and FX Curves.

Functionality in this package is implemented over 8 classes – [ConstantForwardHazard](#), [ConstantForwardRate](#), [DerivedFXBasis](#), [DerivedFXForward](#), [DerivedZeroRate](#), [HyperbolicTensionForwardRate](#), [PolynomialForwardRate](#), and [PolynomialSplineDF](#).

ConstantForwardHazard

This class contains the constant hazard rate based credit curve holder object. It exports the following functionality:

- Calculate survival probability / survival probability Jacobian
- Calculate recovery / recovery Jacobian
- Calculate hazard rate / hazard rate Jacobian
- Construct tweaked curve instances (parallel/tenor/custom tweaks)
- Optionally provide the calibration instruments and quotes used to build the curve.

ConstantForwardRate

This class contains the constant forward rate based discount curve holder object. It exports the following functionality:

- Calculate discount factor / discount factor Jacobian

- Calculate implied forward rate / implied forward rate Jacobian
- Construct tweaked curve instances (parallel/tenor/custom tweaks)
- Optionally provide the calibration instruments and quotes used to build the curve.

DerivedFXBasis

This class contains the constant forward basis based FX Basis Curve holder object. It exports the following functionality:

- Calculate FX Forward / FX Forward Jacobian
- Calculate Forward Basis / Forward Basis Jacobian
- Construct tweaked curve instances (parallel/tenor/custom tweaks)
- Optionally provide the calibration instruments and quotes used to build the curve.

DerivedFXForward

This class contains the constant forward based FX Forward Curve holder object. It exports the following functionality:

- Calculate FX Forward / FX Forward Jacobian
- Calculate Forward Basis / Forward Basis Jacobian
- Construct tweaked curve instances (parallel/tenor/custom tweaks)
- Optionally provide the calibration instruments and quotes used to build the curve.

DerivedZeroRate

This class implements the zero rate curve. It exports the following functionality:

- Calculate discount factor / discount factor Jacobian
- Calculate zero/forward rate / zero/forward rate Jacobian

- Construct tweaked curve instances (parallel/tenor/custom tweaks)
- Optionally provide the calibration instruments, the quotes, the base discount curve, and the zero bump used to build the curve.

HyperbolicTensionForwardRate

This class contains the polynomial spline discount factor based discount curve holder object. The class constructs the discount curve using generic polynomial splines (arbitrary degree, variable shape control, custom segment knot constraints, user specified variational penalty optimization, and segment tension). It exports the following functionality:

- Calculate discount factor / discount factor Jacobian
- Calculate implied forward rate / implied forward rate Jacobian
- Construct tweaked curve instances (parallel/tenor/custom tweaks)
- Optionally provide the calibration instruments and quotes used to build the curve.

PolynomialForwardRate

This class contains the polynomial forward rate based discount curve holder object. The class constructs the discount curve using generic polynomial splines (arbitrary degree, variable shape control, custom segment knot constraints, user specified variational penalty optimization, and segment tension). It exports the following functionality:

- Calculate discount factor / discount factor Jacobian
- Calculate implied forward rate / implied forward rate Jacobian
- Construct tweaked curve instances (parallel/tenor/custom tweaks)
- Optionally provide the calibration instruments and quotes used to build the curve.

PolynomialSplineDF

This class contains the polynomial spline discount factor based discount curve holder object. The class constructs the discount curve using generic polynomial splines (arbitrary degree, variable shape control, custom segment knot constraints, user specified variational penalty optimization, and segment tension). It exports the following functionality:

- Calculate discount factor / discount factor Jacobian
- Calculate implied forward rate / implied forward rate Jacobian
- Construct tweaked curve instances (parallel/tenor/custom tweaks)
- Optionally provide the calibration instruments and quotes used to build the curve.

Credit Analytics: Curve Calibrators

[Curve calibrators](#) are implemented in the package [org.drip.analytics.calibration](#). This package contains the curve calibrators that use different calibration schemes, and the curve scenario generators.

Functionality in this package is implemented over 3 classes – [CurveCalibrator](#), [RatesCurveScenarioGenerator](#), and [CreditCurveScenarioGenerator](#).

CurveCalibrator

CurveCalibrator calibrates the discount and credit/hazard curves from the components and their quotes.

CurveCalibrator employs a set of techniques for achieving this calibration.

- It bootstraps the nodes in sequence to calibrate the curve.
- In conjunction with splining interpolation techniques, it may also be used to perform dual sweep calibration. The inner sweep achieves the calibration of the segment spline parameters, while the outer sweep calibrates iteratively for the targeted boundary conditions.

CurveCalibrator bootstraps/cooks both discount curves and credit curves.

RatesCurveScenarioGenerator

This class contains the interest rate calibration instruments to be used with the component calibrator to produce scenario interest rate curves.

RatesCurveScenarioGenerator typically first constructs the actual curve calibrator instance to localize the intelligence around curve construction. It then uses this curve calibrator instance to build individual curves or the sequence of node bumped scenario curves. The curves in the set may be an array, or tenor-keyed.

CreditCurveScenarioGenerator

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

CreditCurveScenarioGenerator typically first constructs the actual curve calibrator instance to localize the intelligence around curve construction. It then uses this curve calibrator instance to build individual curves or the sequence of node bumped scenario curves. The curves in the set may be an array, or tenor-keyed.

Credit Analytics: Curve Creators

[Curve creators](#) are implemented in the package [org.drip.analytics.creator](#). This contains the curve object factories for the different curves using the calibration parameters.

Functionality in this package is implemented over 5 classes – [DiscountCurveBuilder](#), [ZeroCurveBuilder](#), [CreditCurveBuilder](#), [FXBasisCurveBuilder](#), and [FXForwardCurveBuilder](#).

CreditCurveBuilder

This class contains the builder functions that construct the credit curve (comprising both survival and recovery) instance. It contains static functions that build different types of credit curve from 3 major types of inputs:

- From a variety of ordered credit-sensitive calibration instruments and their quotes
- From an array of ordered survival probabilities
- From a serialized byte stream of the credit curve instance

DiscountCurveBuilder

This class contains the builder functions that construct the discount curve (comprising both the rates and the discount factors) instance. It contains static functions that build different types of discount curve from 3 major types of inputs:

- From a variety of ordered DF-sensitive calibration instruments and their quotes
- From an array of ordered discount factors
- From a serialized byte stream of the discount curve instance

FXBasisCurveBuilder

This class contains the baseline FX Basis curve builder object. It contains static functions that build FX Basis curves from the 3 major inputs:

- An ordered array of Forward FX
- An ordered array of Forward Basis Points
- A byte Stream representing the serialized instance of the FXBasisCurve.

FXForwardCurveBuilder

This class contains the baseline FX Forward curve builder object. It contains static functions that build FX Forward curves from the 3 major inputs:

- An ordered array of Forward FX
- An ordered array of Forward Basis Points
- A byte Stream representing the serialized instance of the FXForwardCurve.

ZeroCurveBuilder

This class contains the baseline zero curve builder object. It contains static functions that build zero curves from cash flows, discount curves, and other input curves/instruments.

Credit Analytics: Reference Data Loaders

[Data loaders](#) are implemented in the package [org.drip.feed.loaders](#). This package contains functionality that loads the bond and the CDS reference data, as well as closing marks for a few date ranges.

Functionality in this package is implemented over 3 classes – [BondRefData](#), [CDXRefData](#), and [CreditStaticAndMarks](#).

BondRefData

BondRefData contains functionality to load a variety of Bond Product reference data and closing marks. It exposes the following functionality:

- Load the bond valuation-based reference data, amortization schedule and EOS
- Build the bond instance entities from the valuation-based reference data
- Load the bond non-valuation-based reference data

BondRefData assumes the appropriate connections are available to load the data.

CDXRefData

CDXRefData contains the functionality to load the standard CDX reference data and definitions, and create compile time static classes for these definitions.

CreditStaticAndMarks

CreditStaticAndMarks contains functionality to load a variety of Credit and Rates Product reference data and closing marks. It exposes the following functionality:

- Load the bond reference data, static data, amortization schedule and EOS.
- Build the bond instance entities from the reference data.
- Load the bond, CDS, and Rates product Closing Marks.
- Load and build the Holiday Calendars.

CreditStaticAndMarks assumes the appropriate connections are available to load the data.

Credit Analytics: Analytics Configurator

[Credit Analytics configurator](#) is implemented in the package [org.drip.param.config](#). This package contains functionality to configure various aspects of [Credit Analytics](#).

Functionality in this package is implemented in a single class – [ConfigLoader](#).

ConfigLoader

ConfigLoader implements the configuration functionality. It exposes the following:

- Parses the XML configuration file and extract the tag pairs information.
- Retrieve the logger.
- Load the holiday calendars and retrieve the location holidays.
- Connect to analytics server and the database server.

Depending on the configuration setting, ConfigLoader loads the above from either a file or the specified database.

Credit Analytics: Market Parameters, Quotes, and Scenario Parameter Implementations

[Quotes and Market Parameters](#) are implemented in the package [org.drip.param.market](#). This contains the implementations of the [Credit Product interfaces](#) representing the quotes, the basket/component market parameters, and the scenario curve containers.

Functionality in this package is implemented over 8 classes – [MultiSidedQuote](#), [ComponentTickQuote](#), [ComponentMultiMeasureQuote](#), [RatesCurveScenarioContainer](#), [CreditCurveScenarioContainer](#), [ComponentMarketParamsSet](#), [BasketMarketParamSet](#), and [MarketParamsContainer](#).

MultiSidedQuote

MultiSidedQuote implements the Quote interface, which contains the stubs corresponding to a product quote. It contains the quote value and the quote time-snap for the different quote sides (bid/ask/mid).

ComponentTickQuote

ComponentTickQuote holds the tick related component parameters - it contains the product ID, the quote composite, the source, the counter party, and whether the quote can be treated as a mark.

ComponentMultiMeasureQuote

ComponentMultiMeasureQuote holds 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).

RatesCurveScenarioContainer

RatesCurveScenarioContainer implements the RatesScenarioCurve abstract class that exposes the interface the constructs scenario discount curves. The following curve construction scenarios are supported:

- Base, flat/tenor up/down by arbitrary bumps
- Tenor bumped discount curve set - keyed using the tenor
- NTP-based custom scenario curves

CreditCurveScenarioContainer

CreditCurveScenarioContainer is the place-holder for the bump parameters and the curves for the different credit curve scenarios. Contains the spread and the recovery bumps, and the credit curve scenario generator object that wraps the calibration instruments. It also contains the base credit curve, spread bumped up/down credit curves, recovery bumped up/down credit curves, and the tenor mapped up/down credit curves.

ComponentMarketParamSet

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

[BasketMarketParamSet](#)

BasketMarketParamSet provides an implementation of BasketMarketParamsRef for a specific scenario. It contains maps holding named discount curves, named credit curves, named component quote, and fixings object.

[MarketParamsContainer](#)

MarketParamsContainer extends MarketParams abstract class, and is the place-holder for the comprehensive suite of the market set of curves for the given date. It exports the following functionality:

- Add/remove/retrieve scenario discount curve.
- Add/remove/retrieve scenario zero curve.
- Add/remove/retrieve scenario credit curve.
- Add/remove/retrieve scenario recovery curve.
- Add/remove/retrieve scenario FXForward curve.
- Add/remove/retrieve scenario FXBasis curve.
- Add/remove/retrieve scenario fixings.
- Add/remove/retrieve Treasury/component quotes.
- Retrieve scenario CMP/BMP.
- Retrieve map of flat rates/credit/recovery CMP/BMP.
- Retrieve double map of tenor rates/credit/recovery CMP/BMP.
- Retrieve rates/credit scenario generator.

Credit Analytics: Market Parameters, Quotes, and Scenario Parameter Creators

Builders for quotes, market parameters, and scenario curves are implemented in the package [org.drip.param.creator](#). This contains the builder factories for the quotes, market parameters, and the scenario curves.

Functionality in this package is implemented over 8 classes – [QuoteBuilder](#), [ComponentQuoteBuilder](#), [ComponentTickQuoteBuilder](#), [RatesScenarioCurveBuilder](#), [CreditScenarioCurveBuilder](#), [ComponentMarketParamsBuilder](#), [BasketMarketParamsBuilder](#), [MarketParamsBuilder](#).

QuoteBuilder

QuoteBuilder contains the quote builder object. It contains static functions that build two-sided quotes from inputs, as well as from a byte stream.

ComponentQuoteBuilder

ComponentQuoteBuilder contains the component quote builder object. It contains static functions that build component quotes from the quote inputs, as well as from byte streams.

ComponentTickQuoteBuilder

ComponentTickQuoteBuilder implements the component tick quote builder object. It contains static functions that build component quotes from the inputs, as well as from byte array.

RatesScenarioCurveBuilder

RatesScenarioCurveBuilder implements the the construction of the scenario discount curve using the input discount curve instruments.

CreditScenarioCurveBuilder

CreditScenarioCurveBuilder implements the construction, de-serialization, and building of the custom scenario based credit curves.

ComponentMarketParamsBuilder

ComponentMarketParamsBuilder implements the various ways of constructing, de-serializing, and building the Component Market Parameters.

BasketMarketParamsBuilder

BasketMarketParamsBuilder implements the various ways of constructing, de-serializing, and building the Basket Market Parameters.

MarketParamsBuilder

MarketParamsBuilder implements the functionality for constructing, de-serializing, and building the Market Universe Curves Container.

Credit Analytics: Rates Component Implementations

[Rates components](#) are implemented in the package [org.drip.product.rates](#). This contains the implementations of the [Credit Product interfaces](#) for Cash, Euro-dollar future, fixed/floating streams, interest rate swap instruments, and rates basket products.

Functionality in this package is implemented over 6 classes – [CashComponent](#), [EDFComponent](#), [FixedStream](#), [FloatingStream](#), [IRSComponent](#), and [RatesBasket](#).

CashComponent

CashComponent contains the implementation of the Cash product and its contract/valuation details.

EDFComponent

EDFComponent contains the implementation of the Euro-dollar future contract/valuation (EDF).

FixedStream

FixedStream contains an implementation of the fixed leg cash flow stream product.

FloatingStream

FloatingStream contains an implementation of the Floating leg cash flow stream.

IRSComponent

IRSComponent contains the implementation of the Interest Rate Swap product contract/valuation details. It is made off one fixed stream and one floating stream.

RatesBasket

RatesBasket contains the implementation of the Basket of Rates Component legs. RatesBasket is made from zero/more fixed and floating streams.

Credit Analytics: Credit Product Implementations

[Credit product definitions](#) are implemented in the package [org.drip.product.credit](#). This contains the implementations of the [Credit Product interfaces](#) for Bonds, CDS, basket default swaps, and bond baskets.

Functionality in this package is implemented over 4 classes – [BondComponent](#), [BondBasket](#), [CDSComponent](#), and [CDSBasket](#).

BondComponent

BondComponent is the base class that extends CreditComponent abstract class and implements the functionality behind bonds of all kinds. Bond static data 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.

BondBasket

BondBasket implements the bond basket product contract details. Contains the basket name, basket notional, component bonds, and their weights.

CDSComponent

CDSComponent implements the credit default swap product contract details. Contains effective date, maturity date, coupon, coupon day count, coupon frequency, contingent credit, currency, basket notional, credit valuation parameters, and optionally the outstanding notional schedule.

CDSBasket

CDSBasket implements the basket default swap product contract details. Contains effective date, maturity date, coupon, coupon day count, coupon frequency, basket components, basket notional, loss pay lag, and optionally the outstanding notional schedule and the flat basket recovery.

Credit Analytics: FX Component Implementations

[FX components](#) are implemented in the package [org.drip.product.fx](#). This contains the implementations of the [Credit Product interfaces](#) for FX spot and forward contracts.

Functionality in this package is implemented over 2 classes – [FXSpotContract](#) and [FXForwardContract](#).

FXForwardContract

FXForwardContract contains the FX forward product contract details - the effective date, the maturity date, the currency pair and the product code.

FXSpotContract

FXSpotContract contains the FX spot contract parameters - the spot date and the currency pair.

Credit Analytics: Product Creators

[Product creators](#) are implemented in the package [org.drip.product.creator](#). This contains the creators for the various rates, credit, and FX component and basket products.

Functionality in this package is implemented over 12 classes – [CashBuilder](#), [EDFutureBuilder](#), [RatesStreamBuilder](#), [CDSBuilder](#), bond creator classes ([BondRefDataBuilder](#), [BondProductBuilder](#), [BondBuilder](#)), CDS basket creator classes ([CDSBasketBuilder](#), [CDXRefDataHolder](#)), [BondBasketBuilder](#), and FX product builder classes ([FXSpotBuilder](#) and [FXForwardBuilder](#)). Of these [CDXRefDataHolder](#) is generated from the CDX reference/static information.

BondBasketBuilder

BondBasketBuilder contains the suite of helper functions for creating the bond Basket Product from different kinds of inputs and byte streams.

BondBuilder

BondBuilder contains the suite of helper functions for creating simple fixed/floater bonds, user defined bonds, optionally with custom cash flows and embedded option schedules (European or American). It also constructs bonds by de-serializing the byte stream.

BondProductBuilder

BondProductBuilder holds the static parameters of the bond product needed for the full bond valuation. It contains:

- Bond identifier parameters (ISIN, CUSIP)
- Issuer level parameters (Ticker, SPN or the credit curve string)
- 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, and issue dates)
- Embedded option parameters (callable, puttable, has been exercised)
- Currency parameters (trade, coupon, and redemption currencies)
- Floater parameters (floater flag, floating coupon convention, current coupon, rate index, spread)
- Whether the bond is perpetual or has defaulted

BondRefDataBuilder

BondRefDataBuilder holds the entire set of static parameters for the bond product. In particular, it contains

- Bond identifier parameters (ISIN, CUSIP, BBG ID, name short name)
- 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),
- 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.

CashBuilder

CashBuilder contains the suite of helper functions for creating the Cash product from the parameters/codes/byte array streams.

CDSBasketBuilder

CDSBasketBuilder contains the suite of helper functions for creating the CDS Basket Product from different kinds of inputs and byte streams.

CDSBuilder

CDSBuilder contains the suite of helper functions for creating the CreditDefaultSwap product from the parameters/byte array streams. It also creates the standard EU, NA, ASIA contracts, CDS with amortization schedules, and CDS from product codes/tenors.

CDXRefDataHolder

[CDXRefDataHolder](#) contains all the generated standard CDX Products, returned as instances of [CreditProduct's BasketProduct](#) interface. Since this is a generated file, please do not delete this.

EDFutureBuilder

EDFutureBuilder contains the suite of helper functions for creating the EDFuture product from the parameters/codes/byte array streams.

FXForwardBuilder

FXForwardBuilder contains the suite of helper functions for creating the FXForwardBuilder product from the parameters/byte array streams.

FXSpotBuilder

FXSpotBuilder contains the suite of helper functions for creating the FXSpot from the corresponding parameters/byte array streams.

RatesStreamBuilder

RatesStreamBuilder contains the suite of helper functions for creating the Stream-based Rates Products from different kinds of inputs. In particular, it demonstrates the following:

- Construction of the custom/standard fixed/floating streams from parameters.
- Construction of the custom/standard IRS from parameters.
- Construction of the fixed/floating streams and IRS from byte arrays.

Credit Analytics: Analytics Environment Manager

[Analytics Environment Manager](#) component are implemented in the package [org.drip.service.env](#). This contains the creators for the various rates, credit, and FX component and basket products.

Functionality in this package is implemented over 7 classes – [BondManager](#), [CDSManager](#), [EnvManager](#), [EODCurves](#), [RatesManager](#), [StandardCDXManager](#), and [StaticBACurves](#).

BondManager

BondManager implements a container that holds the EOD and bond static information on a per issuer basis. It exposes the following functionality:

- Retrieve the available tickers, and all the ISIN's per ticker.
- Load the full set of bond reference data, embedded option schedules, and amortization schedules.
- Load the full set of bond marks.
- Calculate the bond RV/Value measures for a ticker/full bond set, given the EOD and the appropriate curves and market measures.
- Save the computed measures for a given EOD.
- (Optionally) Generate a Bond Creator File.

CDSManager

CDSManager is the container that retrieves the EOD and CDS/credit curve information on a per-issuer basis and populates the MPC.

EnvManager

EnvManager sets the environment and connection parameters, and populates the market parameters (quotes, curves, and fixings) for a given EOD.

EODCurves

EODCurves that creates the closing curves from the closing marks available in the DB for a given EOD and populates them onto the MPC. It builds the following:

- Discount Curve (from cash/future/swap - typical sequence), EDSF Curve, and TSY Curve
- Credit Curve from CDS quotes
- On-the-run TSY yield quotes

RatesManager

RatesManager manages the creation/loading of the rates curves of different kinds for a given EOD.

StaticBACurves

StaticBACurves that creates the closing curves from custom/user defined marks for a given EOD and populates them onto the MPC. It builds the following:

- Discount Curve (from cash/future/swap - typical sequence), EDSF Curve, and TSY Curve
- Credit Curve from CDS quotes
- On-the-run TSY yield quotes

StandardCDXManager

StandardCDXManager implements the creation and the static details of the all the NA, EU, SovX, EMEA, and ASIA standardized CDS indices. It exposes the following functionality:

- Retrieve the full set of pre-set/pre-loaded CDX names/descriptions.
- Retrieve all the CDX's given an index name.
- Get the index, index series, and the effective/maturity dates for a given CDX.
- Get all the on-the-runs for an index, date, and tenor.
- Retrieve the full basket product corresponding to NA/EU/ASIA IG/HY/EM and other available standard CDX.
- Build a custom CDX product.

Credit Analytics: Analytics Bridge

[Analytics Bridge](#) is implemented in the package [org.drip.service.bridge](#). This provides the stub and proxy functionality for invoking [Credit Analytics functionality](#) in a remote server and extracting the results.

Functionality in this package is implemented over 4 classes – [CreditAnalyticsRequest](#), [CreditAnalyticsResponse](#), [CreditAnalyticsStub](#), and [CreditAnalyticsProxy](#).

CreditAnalyticsRequest

CreditAnalyticsRequest contains the requests for the Credit Analytics server from the client. It contains the following parameters:

- The GUID and the time-stamp of the request.
- The component that is being valued.
- The valuation, the pricer, and the quoting parameters.
- The market parameters assembled in the ComponentMarketParams.

Typical usage is: Client fills in the entities in the request, serializes them, and sends them to the server, and receives a serialized response back from the server.

CreditAnalyticsResponse

CreditAnalyticsResponse contains the response from the Credit Analytics server to the client. It contains the following parameters:

- The GUID and of the request.

- The type and time-stamp of the response.
- The string version of the response body.

CreditAnalyticsProxy

CreditAnalyticsProxy captures the requests for the Credit Analytics server from the client, formats them, and sends them to the Credit Analytics Stub.

CreditAnalyticsStub

CreditAnalyticsStub serves as a sample server that hosts the Credit Analytics functionality. It receives requests from the analytics client as a serialized message, and invokes the CreditAnalytics functionality, and sends the client the serialized results.

Credit Analytics: Analytics API

[Analytics API](#) is implemented in the package [org.drip.service.api](#). This provides a unified and comprehensive functional, static interface of all the main [Credit Analytics functionality](#).

Functionality in this package is implemented over a single class – [CreditAnalytics](#).

CreditAnalytics

CreditAnalytics exposes all the CreditAnalytics API to clients – this class is the main functional interface. The functions exposed are too numerous to list, and can be roughly grouped into the following:

- Product Creation
- Curve Construction from Market Instruments
- Product Reference Data Examination
- Product Valuation from the Market Parameters
- Product Measure Extraction
- Product RV Measure Computation
- General Finance Math calculation (day count, date adjust etc.)
- Closing points extraction

Credit Analytics: Samples

[Credit Analytics samples](#) are available in the package [org.drip.service.sample](#). This provides samples illustrating the functionality provided by [Credit Analytics](#) – samples demonstrating the creation and usage of curves and products across rates, credit, and FX components and baskets. Examples are also provided on how to compare against standard analytics vendors/suppliers (e.g., Bloomberg).

Functionality in this package is implemented over 18 classes – [BloombergCDSW](#), [BloombergSWPM](#), [BloombergYAS](#), [BondAnalyticsAPI](#), [BondBasketAPI](#), [BondLiveAndEODAPI](#), [BondRVMeasuresAPI](#), [BondStaticAPI](#), [CDSBasketAPI](#), [CDSLIVEAndEODAPI](#), [CreditAnalyticsAPI](#), [DayCountAndCalendarAPI](#), [FXAPI](#), [MultiLegSwapAPI](#), [RatesAnalyticsAPI](#), [RatesLiveAndEODAPI](#), [StandardCDXAPI](#), and [TreasuryCurveAPI](#).

BloombergCDSW

BloombergCDSW contains the sample demonstrating the replication of Bloomberg's CDSW functionality.

BloombergSWPM

BloombergSWPM contains the sample demonstrating the replication of Bloomberg's SWPM functionality.

BloombergYAS

BloombergYAS contains the sample demonstrating the replication of Bloomberg's YAS functionality.

BondAnalyticsAPI

BondAnalyticsAPI contains a demo of the bond analytics API Sample. It generates the value and the RV measures for essentially the same bond (with identical cash flows) constructed in 3 different ways:

- As a fixed rate bond.
- As a floater.
- As a bond constructed from a set of custom coupon and principal flows.

It shows these measures reconcile where they should.

BondBasketAPI

BondBasketAPI contains a demo of the bond basket API Sample. It shows the following:

- Build the IR Curve from the Rates' instruments.
- Build the Component Credit Curve from the CDS instruments.
- Create the basket market parameters and add the named discount curve and the credit curves to it.
- Create the bond basket from the component bonds and their weights.
- Construct the Valuation and the Pricing Parameters.
- Generate the bond basket measures from the valuation, the pricer, and the market parameters.

BondLiveAndEODAPI

BondLiveAndEODAPI contains the comprehensive sample class demonstrating the usage of the EOD and Live Curve Bond API functions.

BondRVMeasuresAPI

BondRVMeasuresAPI is a Simple Bond RV Measures API Sample demonstrating the invocation and usage of Bond RV Measures functionality. It shows the following:

- Create the discount/treasury curve from rates/treasury instruments.
- Compute the work-out date given the price.
- Compute and display the base RV measures to the work-out date.
- Compute and display the bumped RV measures to the work-out date.

BondStaticAPI

BondStaticAPI contains a demo of the bond static API Sample. The Sample demonstrates the retrieval of the bond's static fields.

CDSBasketAPI

CDSBasketAPI contains a demo of the CDS basket API Sample. It shows the following:

- Build the IR Curve from the Rates' instruments.
- Build the Component Credit Curve from the CDS instruments.
- Create the basket market parameters and add the named discount curve and the credit curves to it.

- Create the CDS basket from the component CDS and their weights.
- Construct the Valuation and the Pricing Parameters.
- Generate the CDS basket measures from the valuation, the pricer, and the market parameters.

[CDSLIVEANDEODAPI](#)

CDSLIVEANDEODAPI is a fairly comprehensive sample demo'ing the usage of the EOD and Live CDS Curve API functions. It demonstrates the following:

- Retrieves all the CDS curves available for the given EOD.
- Retrieves the calibrated credit curve from the CDS instruments for the given CDS curve name, IR curve name, and EOD. Also shows the 10Y survival probability and hazard rate.
- Displays the CDS quotes used to construct the closing credit curve.
- Loads all available credit curves for the given curve ID built from CDS instruments between 2 dates and displays the corresponding 5Y quote.
- Calculate and display the EOD CDS measures for a spot starting CDS based off of a specific credit curve.

[CreditAnalyticsAPI](#)

CreditAnalyticsAPI contains a demo of the CDS Analytics API Sample. It illustrates the following:

- Credit Curve Creation: From flat Hazard Rate, and from an array of dates and their corresponding survival probabilities.
- Create Credit Curve from CDS instruments, and recover the input measure quotes.
- Create an SNAC CDS, price it, and display the coupon/loss cash flow.

[DayCountAndCalendarAPI](#)

DayCountAndCalendarAPI demonstrates Day-count and Calendar API FUnctionality. It does the following:

- Get all the holiday locations in CreditAnalytics, and all the holidays in the year according to the calendar set.
- Get all the week day/weekend holidays in the year according to the calendar set.
- Calculate year fraction between 2 dates according to semi-annual, Act/360, and USD calendar.
- Adjust the date FORWARD according to the USD calendar.
- Roll to the PREVIOUS date according to the USD calendar.

[FXAPI](#)

FXAPI contains a demo of the FX API Sample. It shows the following:

- Create a currency pair, FX SPot, and FX Forward.
- Calculate the FX forward PIP/outright.
- Calculate the DC Basis on the domestic and the foreign curves.
- Create an FX curve from the spot, and the array of nodes, FX forward, as well as the PIP indicator.
- Calculate the array of the domestic/foreign basis.
- Calculate the array of bootstrapped domestic/foreign basis.
- Re-imply the array of FX Forward from domestic/foreign Basis Curve.

[MultiLegSwapAPI](#)

MultiLegSwapAPI illustrates the creation, invocation, and usage of the MultiLegSwap. It shows how to:

- Create the Discount Curve from the rates instruments.
- Set up the valuation and the market parameters.
- Create the Rates Basket from the fixed/float streams.
- Value the Rates Basket.

[RatesAnalyticsAPI](#)

RatesAnalyticsAPI contains a demo of the Rates Analytics API Usage. It shows the following:

- Build a discount curve using: cash instruments only, EDF instruments only, IRS instruments only, or all of them strung together.
- Re-calculate the component input measure quotes from the calibrated discount curve object.
- Compute the PVDF Wengert Jacobian across all the instruments used in the curve construction.

[RatesLiveAndEODAPI](#)

RatesLiveAndEODAPI contains the sample API demonstrating the usage of the Rates Live and EOD functions. It does the following:

- Pulls all the closing rates curve names (of any type, incl. TSY) that exist for a given date.
- Load the full IR curve created from all the single currency rate quotes (except TSY) for the given currency and date.
- Calculate the discount factor to an arbitrary date using the constructed curve.

- Retrieve the components and their quotes that went into constructing the curve, and display them.
- Load all the rates curves available between the dates for the currency specified, and step through.
- Load all the Cash quotes available between the dates for the currency specified, and step through.
- Load all the EDF quotes available between the dates for the currency specified, and step through.
- Load all the IRS quotes available between the dates for the currency specified, and step through.
- Load all the TSY quotes available between the dates for the currency specified, and step through.

[StandardCDXAPI](#)

StandardCDXAPI contains a demo of the CDS basket API Sample. It shows the following:

- Construct the CDX.NA.IG 5Y Series 17 index by name and series.
- Construct the on-the-run CDX.NA.IG 5Y Series index.
- List all the built-in CDX's - their names and descriptions.
- Construct the on-the run CDX.EM 5Y corresponding to T - 1Y.
- Construct the on-the run ITRAXX.ENERGY 5Y corresponding to T - 7Y.
- Retrieve the full set of date/index series set for ITRAXX.ENERGY.

[TreasuryCurveAPI](#)

TreasuryCurveAPI contains a demo of construction and usage of the treasury discount curve from government bond inputs. It shows the following:

- Create on-the-run TSY bond set.
- Calibrate a discount curve off of the on-the-run yields and calculate the implied zeroes and DF's.
- Price an off-the-run TSY.

Credit Analytics: Functional Testers

[Credit Analytics functional testers](#) are available in the package [org.drip.testers.functional](#). This contains a fairly extensive set of unit and composite testers for the curve, products, serialization, and analytics functionality provided by the [Credit Analytics suite](#), with a special focus on bonds.

Functionality in this package is implemented over 4 classes – [BondTestSuite](#), [CreditAnalyticsTestSuite](#), [ProductTestSuite](#), and [SerializerTestSuite](#).

BondTestSuite

BondTestSuite tests more-or-less the full suite of bond functionality exposed in CreditAnalytics API.

CreditAnalyticsTestSuite

CreditAnalyticsTestSuite tests more-or-less the full suite of functionality exposed in CreditAnalytics API across all products, curves, quotes, outputs, and parameters, and their variants.

ProductTestSuite

ProductTestSuite tests more-or-less the full suite of the product valuation functionality exposed in CreditAnalytics API. The following variants are tested.

- Full suite of products - rates, credit and FX, both components and baskets.
- Base flat/tenor bumped scenario tests.

SerializerTestSuite

SerializerTestSuite tests the serialization functionality across all products, curves, quotes, outputs, and parameters, and their variants.

Regression Suite Library

[Regression Suite Library](#) consists of the following 2 packages:

1. [Core Regression Library](#): This contains the full set of [Regression Suite's](#) core framework and the set of extensible interfaces.
2. [Curve Regression Suite](#): This contains the suite for regression testing of the Curve construction and usage functionality.
3. [Curve Jacobian Regression Suite](#): This contains the suite for regression testing of the Curve Jacobian extraction and usage functionality.
4. [Fixed Point Finder Regression Suite](#): This contains the suite for regression testing of the non-linear univariate fixed-point finder.
5. [Basis Spline Regression Suite](#): This contains the suite for regression testing of the basis spline set construction, usage, and the span interpolation functionality.

Regression Suite: Core

The [core functionality of the regression suite library](#) is implemented in the package [org.drip.regression.core](#). This contains the full set of [Regression Suite](#)'s core framework and the set of extensible interfaces.

Functionality in this package is implemented over 7 classes – [RegressionEngine](#), [RegressionRunDetail](#), [RegressionRunOutput](#), [RegressorSet](#), [UnitRegressionExecutor](#), [UnitRegressionStat](#), and [UnitRegressor](#).

RegressionEngine

RegressionEngine provides the control and frame-work functionality for the General Purpose Regression Suite. It invokes the following steps as part of the execution:

- Initialize the regression environment. This step sets up the regression sets, and adds individual regressors to the set.
- Invoke the regressors in each set one by one.
- Collect the results and details of the regression runs.
- Compile the regression statistics.
- Optionally display the regression statistics.

RegressionRunDetail

RegressionRunDetail contains named field level detailed output of the regression activity.

RegressionRunOutput

RegressionRunOutput contains the output of a single regression activity. It holds the following:

- The execution time
- The Success/failure status of the run
- The regression scenario that was executed
- The Completion time for the regression module
- The Regression Run Detail for the regression run

RegressorSet

RegressorSet interface provides the Regression set stubs. It contains a set of regressors and is associated with a unique name. It provides the functionality to set up the contained regressors.

UnitRegressionExecutor

UnitRegressionExecutor implements the UnitRegressor, and splits the regression execution into pre-, execute, and post-regression. It provides default implementations for pre-regression and post-regression. Most typical regressors only need to over-ride the execRegression method.

UnitRegressionStat

UnitRegressionStat creates the statistical details for the Unit Regressor. It holds the following:

- Execution Initialization Delay
- Execution time mean, variance, maximum, and minimum
- The full list of individual execution times

UnitRegressor

UnitRegressor provides the stub functionality for the Individual Regressors. Its derived classes implement the actual regression run. Individual regressors are named.

Curve Regression Suite

The core functionality of the [curve regression library](#) is implemented in the package [org.drip.regression.curve](#). This contains the suite for regression testing of the Curve construction and usage functionality.

Functionality in this package is implemented over 5 classes – [DiscountCurveRegressor](#), [ZeroCurveRegressor](#), [CreditCurveRegressor](#), [FXCurveRegressor](#), and [CreditAnalyticsRegressionEngine](#).

DiscountCurveRegressor

DiscountCurveRegressor implements the regression set analysis for the Discount Curve. DiscountCurveRegressor regresses 11 scenarios:

- #1: Create the discount curve from a set 30 instruments (cash/future/swap).
- #2: Create the discount curve from a flat discount rate.
- #3: Create the discount curve from a set of discount factors.
- #4: Create the discount curve from the implied discount rates.
- #5: Extract the discount curve instruments and quotes.
- #6: Create a parallel shifted discount curve.
- #7: Create a rate shifted discount curve.
- #8: Create a basis rate shifted discount curve.
- #9: Create a node tweaked discount curve.
- #10: Compute the effective discount factor between 2 dates.
- #11: Compute the effective implied rate between 2 dates.

ZeroCurveRegressor

ZeroCurveRegressor implements the regression analysis set for the Zero Curve. The regression tests do the following:

- Build a discount curve, followed by the zero curve.
- Regressor #1: Compute zero curve discount factors.
- Regressor #2: Compute zero curve zero rates.

CreditCurveRegressor

CreditCurveRegressor implements the regression set analysis for the Credit Curve.

CreditCurveRegressor regresses 12 scenarios:

- #1: Create an SNAC CDS.
- #2: Create the credit curve from a set of CDS instruments.
- #3: Create the credit curve from a flat hazard rate.
- #4: Create the credit curve from a set of survival probabilities.
- #5: Create the credit curve from an array of hazard rates.
- #6: Extract the credit curve instruments and quotes.
- #7: Create a parallel hazard shifted credit curve.
- #8: Create a parallel quote shifted credit curve.
- #9: Create a node tweaked credit curve.
- #10: Set a specific default date on the credit curve.
- #11: Compute the effective survival probability between 2 dates.
- #12: Compute the effective hazard rate between 2 dates.

FXCurveRegressor

FXCurveRegressor implements the regression analysis set for the FX Curve.

FXCurveRegressor implements 3 regression tests:

- #1: FX Basis and FX Curve Creation: Construct a FX forward Curve from an array of FX forward nodes and the spot.
- #2: Imply the FX Forward given the domestic and foreign discount curves.
- #3a: Compute the domestic and foreign basis given the market FX forward.
- #3b: Build the domestic/foreign basis curve given the corresponding basis nodes.
- #3c: Imply the array of FX forward points/PIPs from the array of basis and domestic/foreign discount curves.

CreditAnalyticsRegressionEngine

CreditAnalyticsRegressionEngine implements the RegressionEngine for the curve regression. It adds the CreditCurveRegressor, DiscountCurveRegressor, FXCurveRegressor, and ZeroCurveRegressor, and launches the regression engine.

Curve Jacobian Regression Suite

The core functionality of the [curve Jacobian regression library](#) is implemented in the package [org.drip.regression.curveJacobian](#). This contains the suite for regression testing of the Curve construction and usage functionality.

Functionality in this package is implemented over 5 classes – [CashJacobianRegressorSet](#), [EDFJacobianRegressorSet](#), [IRSJacobianRegressorSet](#), [DiscountCurveJacobianRegressorSet](#), and [CurveJacobianRegressionEngine](#).

CashJacobianRegressorSet

CashJacobianRegressorSet implements the regression analysis set for the Cash product related Sensitivity Jacobians. Specifically, it computes the PVDF micro-Jack.

EDFJacobianRegressorSet

EDFJacobianRegressorSet implements the regression analysis set for the EDF product related Sensitivity Jacobians. Specifically, it computes the PVDF micro-Jack.

IRSJacobianRegressorSet

IRSJacobianRegressorSet implements the regression analysis set for the IRS product related Sensitivity Jacobians. Specifically, it computes the PVDF micro-Jack.

DiscountCurveJacobianRegressorSet

DiscountCurveJacobianRegressorSet implements the regression analysis for the full discount curve (built from cash/future/swap) Sensitivity Jacobians. Specifically, it computes the PVDF micro-Jack.

CurveJacobianRegressionEngine

CurveJacobianRegressionEngine implements the RegressionEngine for the curve Jacobian regression. It adds the CashJacobianRegressorSet, the EDFJacobianRegressorSet, the IRSJacobianRegressorSet, and the DiscountCurveJacobianRegressorSet, and launches the regression engine.

Fixed-Point Finder Regression Suite

The core functionality of the [non-linear fixed-point finder regression library](#) is implemented in the package [org.drip.regression.fixedpointfinder](#). This contains the suite for regression testing of the non-linear univariate fixed-point finder.

Functionality in this package is implemented over 4 classes – [OpenRegressorSet](#), [BracketingRegressorSet](#), [CompoundBracketingRegressorSet](#), and [FixedPointFinderRegressionEngine](#).

OpenRegressorSet

OpenRegressorSet implements the regression run for the Open (i.e., Newton) Fixed Point Search Method.

BracketingRegressorSet

BracketingRegressorSet implements regression run for the Primitive Bracketing Fixed Point Search Method. It implements the following 4 primitive bracketing schemes: Bisection, False Position, Quadratic, and Inverse Quadratic.

CompoundBracketingRegressorSet

CompoundBracketingRegressorSet implements regression run for the Compound Bracketing Fixed Point Search Method. It implements the following 2 compound bracketing schemes: Brent and Zheng.

FixedPointFinderRegressionEngine

FixedPointFinderRegressionEngine implements the RegressionEngine for the Fixed Point Finder regression. It adds the OpenRegressorSet, the BracketingRegressorSet, and the CompoundBracketingRegressorSet, and launches the regression engine.

Basis Spline Regression Suite

The core functionality of the [basis spline regression library](#) is implemented in the package [org.drip.regression.spline](#). This contains the suite for regression testing of the basis spline set construction, usage, and the span interpolation functionality.

Functionality in this package is implemented over 3 classes – [BasisSplineRegressor](#), [BasisSplineRegressorSet](#), and [BasisSplineRegressionEngine](#).

BasisSplineRegressor

BasisSplineRegressor implements the custom basis spline regressor for the given basis spline. As part of the regression run, it executes the following:

- Calibrate and compute the left and the right Jacobian.
- Reset right node and re-run calibration.
- Compute an intermediate value Jacobian.

BasisSplineRegressorSet

BasisSplineRegressorSet carries out regression testing for the following series of basis splines:

- #1: Polynomial Basis Spline, $n = 2$ basis functions, and $C^k = 0$.
- #2: Polynomial Basis Spline, $n = 3$ basis functions, and $C^k = 1$.
- #3: Polynomial Basis Spline, $n = 4$ basis functions, and $C^k = 1$.
- #4: Polynomial Basis Spline, $n = 4$ basis functions, and $C^k = 2$.
- #5: Polynomial Basis Spline, $n = 5$ basis functions, and $C^k = 1$.
- #6: Polynomial Basis Spline, $n = 5$ basis functions, and $C^k = 2$.

- #7: Polynomial Basis Spline, $n = 5$ basis functions, and $C^k = 3$.
- #8: Polynomial Basis Spline, $n = 6$ basis functions, and $C^k = 1$.
- #9: Polynomial Basis Spline, $n = 6$ basis functions, and $C^k = 2$.
- #10: Polynomial Basis Spline, $n = 6$ basis functions, and $C^k = 3$.
- #11: Polynomial Basis Spline, $n = 6$ basis functions, and $C^k = 4$.
- #12: Polynomial Basis Spline, $n = 7$ basis functions, and $C^k = 1$.
- #13: Polynomial Basis Spline, $n = 7$ basis functions, and $C^k = 2$.
- #14: Polynomial Basis Spline, $n = 7$ basis functions, and $C^k = 3$.
- #15: Polynomial Basis Spline, $n = 7$ basis functions, and $C^k = 4$.
- #16: Polynomial Basis Spline, $n = 7$ basis functions, and $C^k = 5$.
- #17: Bernstein Polynomial Basis Spline, $n = 4$ basis functions, and $C^k = 2$.
- #18: Exponential Tension Spline, $n = 4$ basis functions, Tension = 1., and $C^k = 2$.
- #19: Hyperbolic Tension Spline, $n = 4$ basis functions, Tension = 1., and $C^k = 2$.
- #20: Kaklis-Pandelis Tension Spline, $n = 4$ basis functions, KP = 2, and $C^k = 2$.

[BasisSplineRegressionEngine](#)

BasisSplineRegressionEngine implements the RegressionEngine class for the basis spline functionality.

DRIP Math

[DRIP Math Library](#) consists of the following 9 packages:

1. [DRIP Calculus Routines](#): This contains functionality for mathematical integration and differentiation.
2. [DRIP Common Utility Routines](#): This contains functionality that is common across all DRIP modules.
3. [DRIP Probability Distribution Library](#): This contains functionality to generate/utilize probability distributions.
4. [DRIP Univariate Function Library](#): This contains functionality to generate a wide variety of univariate functions.
5. [DRIP Span/Grid Library](#): This package implements the layout of the n-D grid functionality in accordance with the calibration schema set out in the corresponding specification.
6. [DRIP Linear Algebra Routines](#): This package implements a selected set of linear algebra routines needed by the different segments of Credit Analytics.
7. [DRIP Math Samples](#): This package provides an illustration of the usage of the DRIP math functionality. Samples here demonstrate linear equation set solution/matrix inversion, basis spline set creation and usage, highly customized polynomial spline construction and properties evaluation, fixed point search examples, and span interpolation samples.

8. [DRIP Non-linear Fixed Point Search Library](#): This package implements a number of univariate, non-linear fixed-point search routines. Methodology separates execution initialization from variate iteration. A variety of open and closed variate iteration techniques are implemented, along with simple/complex closed variate iteration techniques.

9. [DRIP Spline Library](#): This package implements the basis set across the different splines – their creation, the segment level calibration, the customization, and segment-level inference values. It customizes the spline construction into spline-independent segment-control parameters, and spline-dependent basis set generator parameters. Finally it implements the spline functionality according to the specification and formulation laid out in detail.

DRIP Calculus Routines

The core functionality of the [DRIP calculus routines](#) is implemented in the package [org.drip.math.calculus](#). This contains functionality for mathematical integration and differentiation. Differentiation is carried out using algorithmic and automated differentiation techniques, and modeule specific Jacobian generation methods are introduced. Several types of integration techniques are introduced too.

Functionality in this package is implemented over 4 classes – [DerivativeControl](#), [Differential](#), [Integrator](#), and [WengertJacobian](#).

DerivativeControl

DerivativeControl provides bumps needed for numerically approximating derivatives. Bumps can be absolute or relative, and they default to a floor.

Differential

Differential holds the incremental differentials for the [variate](#) and the objective function.

Integrator

Integrator implements the following routines for integrating the objective function:

- Mid-Point Scheme
- Trapezoidal Scheme

- Simpson/Simpson38 schemes
- Boole Scheme

WengertJacobian

WengertJacobian contains the Jacobian of the given set of Wengert variables to the set of parameters. It exposes the following functionality:

- Set the Wengert variables
- Accumulate the Partial
- Scale the partial entries
- Merge the Jacobian with another
- Retrieve the WengertJacobian elements

DRIP Common Utility Routines

The core functionality of the [DRIP common utility routines](#) is implemented in the package [org.drip.math.common](#). This contains functionality that is common across all DRIP modules. Utilities include date manipulation, string parsing and formatting, map manipulation, and number operations that are not readily covered.

Functionality in this package is implemented over 5 classes – [DateUtil](#), [FormatUtil](#), [MapUtil](#), [NumberUtil](#), and [StringUtil](#).

DateUtil

DateUtil implements date utility functions those are extraneous to the JulianDate implementation. It exposes the following functionality:

- Retrieve Day, Month, and Year From Java Date.
- Switch between multiple date formats (Oracle Date, BBG Date, different string representations etc).

FormatUtil

FormatUtil implements formatting utility functions. Currently it just exports functions to pad and format.

MapUtil

The MapUtil class implements generic utility functions used in DRIP modules. Some of the functions it exposes are:

- Map Merging Functionality.
- Map Key Functionality - key-value flatteners, key prefixers.
- Decompose/transform List/Set/Array Contents.
- Multi-Dimensional Map Manipulator Routines.

NumberUtil

NumberUtil implements number utility functions. It exposes the following functions:

- Verify number/number array validity, and closeness match.
- Factorial Permutation/Combination functionality
- Dump multi-dimensional array contents

StringUtil

StringUtil implements string utility functions. It exports the following functions:

- Decompose + Transform string arrays into appropriate target type set/array/list, and vice versa.
- General-purpose String processor functions, such as GUID generator, splitter, type converter and input checker.

DRIP Probability Distribution Library

The core functionality of the [DRIP probability distribution library](#) is implemented in the package [org.drip.math.distribution](#). This contains functionality to generate/utilize probability distributions. Supported distributions include the univariate normal distribution and its variants..

Currently, functionality in this package is implemented over 2 classes – [Univariate](#) and [UnivariateNormal](#).

Univariate

Univariate implements the base abstract class behind univariate distributions. It exports methods for incremental, cumulative, and inverse cumulative distribution densities.

UnivariateNormal

UnivariateNormal implements the univariate normal distribution. It implements incremental, cumulative, and inverse cumulative distribution densities.

DRIP Univariate Function Library

The core functionality of the [DRIP Univariate function library](#) is implemented in the package [org.drip.math.function](#). These are currently most commonly used in the fixed point searching schemes, as well as spline approximation functionality.

Supported univariate functions include polynomial variants (regular, Bernstein), Cauchy convergent series (e.g., natural log series), exponential and hyperbolic tension univariates, rational univariates. Also supported are function convolutions and function reflections.

Functionality in this package is implemented over 9 classes – [AbstractUnivariate](#), [BernsteinPolynomial](#), [ExponentialTension](#), [HyperbolicTension](#), [NaturalLogSeriesElement](#), [Polynomial](#), [RationalShapeControl](#), [UnivariateConvolution](#), and [UnivariateReflection](#).

AbstractUnivariate

AbstractUnivariate provides the evaluation of the objective function and its derivatives for a specified variate. Default implementation of the derivatives is for non-analytical black box objective functions.

BernsteinPolynomial

BernsteinPolynomial provides the evaluation of the BernsteinPolynomial and its derivatives for a specified variate.

ExponentialTension

ExponentialTension provides the evaluation of the Exponential Tension Function and its derivatives for a specified variate.

HyperbolicTension

HyperbolicTension provides the evaluation of the Hyperbolic Tension Function and its derivatives for a specified variate.

NaturalLogSeriesElement

NaturalLogSeriesElement implements an element in the natural log series expansion.

Polynomial

Polynomial provides the evaluation of the n-th order Polynomial and its derivatives for a specified variate.

RationalShapeControl

RationalShapeControl implements the deterministic rational shape control functionality on top of interpolating basis splines inside $[0, \dots, 1)$, and globally $[x_0, \dots, x_1)$:

$$y = \frac{1}{1 + \lambda x(1-x)} \text{ where } x \text{ is the normalized ordinate mapped as } x \Rightarrow \frac{x - x_{i-1}}{x_i - x_{i-1}}.$$

UnivariateConvolution

This class provides the evaluation of the Convolution $f_1(x) \times f_2(x)$ and its derivatives for a specified variate.

UnivariateReflection

UnivariateReflection provides the evaluation $f(1-x)$ instead of $f(x)$ for a given f .

DRIP Span/Grid Library

The core functionality of the [DRIP span/grid library](#) is implemented in the package [org.drip.math.grid](#). This package implements the layout of the n-D grid functionality in accordance with the calibration schema set out in the corresponding specification. It lays the basic segment/span separation, partition the segment parameters into elastics and inelastics, and evaluates segment characteristics (monotonicity, variational smoothening effectiveness, etc.).

Functionality in this package is implemented over 5 classes – [Inelastics](#), [Segment](#), [SegmentControlParams](#), [SegmentMonotonicity](#), and [Span](#).

Inelastics

This class the inelastic fields of the given segment – in this case the start/end coordinates.

Segment

This abstract class extends Inelastics, and incorporates segment specific inelastic parameters. Interpolating segment spline functions and their coefficients are implemented/calibrated in the overriding spline classes. It provides functionality for assessing the various segment attributes:

- Segment Monotonicity.
- Interpolated Function Value, the ordered derivative, and the corresponding Jacobian.
- Segment Local/Global Derivative.
- Evaluation of the Segment Micro-Jack.

- Head / Regular Segment calibration - both of the basis function coefficients and the Jacobian.

SegmentControlParams

This class holds the parameters that guide the creation and the behavior of the segment. It holds the segment elastic/inelastic parameters and the named basis function set.

SegmentMonotonicity

This class contains the monotonicity details related to the given segment. Indicates whether the segment is monotonic, and if not, whether it contains a maximum, a minimum, or an inflection.

Span

This class implements the span that spans multiple segments. It holds the ordered segment sequence, the segment control parameters, and, if available, the spanning Jacobian. It exports the following group of functionality:

- Construct adjoining segment sequences in accordance with the segment control parameters
- Calibrate according to a varied set of (i.e., NATURAL/FINANCIAL) boundary conditions
- Interpolate both the value, the ordered derivatives, and the Jacobian at the given ordinate
- Compute the monotonicity details - segment/span level monotonicity, co-monotonicity, local monotonicity.
- Insert knots

Span also exports several static span creation/calibration methods to generate customized basis splines, with customized segment behavior using the segment control.

DRIP Linear Algebra Routines

The core functionality of the [DRIP linear algebra routines](#) is implemented in the package [org.drip.math.linearalgebra](#). This package implements a selected set of linear algebra routines needed by the different segments of Credit Analytics. Functionality it exports includes solving systems of linear equations, matrix manipulation (inversion/diagonalization/components extraction etc), and Jacobian generation.

Functionality in this package is implemented over 4 classes – [LinearizationOutput](#), [LinearSystemSolver](#), [Matrix](#), and [MatrixComplementTransform](#).

LinearizationOutput

LinearizationOutput holds the output of a sequence of linearization operations. It contains the transformed original matrix, the transformed RHS, and the method used for the linearization operation.

LinearSystemSolver

LinearSystemSolver implements the solver for a system of linear equations given by $Ax = B$, where A is the matrix, x the set of variables, and B is the result to be solved for. It exports the following functions:

- Row Regularization and Diagonal Pivoting.
- Check for Diagonal Dominance.
- Solving the linear system using any one of the following: Gaussian Elimination, Gauss Seidel reduction, or matrix inversion.

Matrix

Matrix implements Matrix manipulation routines. It exports the following functionality:

- Matrix Inversion using Closed form solutions (for low-dimension matrices), or using Gaussian elimination
- Matrix Product
- Matrix Diagonalization, Regularization, and Diagonal Pivoting

MatrixComplementTransform

This class holds the results of Matrix transforms on the source and the complement, e.g., during a Matrix Inversion Operation.

DRIP Math Samples

The core functionality of the [DRIP Math Samples](#) is implemented in the package [org.drip.math.sample](#). This package provides an illustration of the usage of the DRIP math functionality. Samples here demonstrate linear equation set solution/matrix inversion, basis spline set creation and usage, highly customized polynomial spline construction and properties evaluation, fixed point search examples, and span interpolation samples.

Functionality in this package is implemented over 5 classes – [BasisSplineSet](#), [LinearAlgebra](#), [PolynomialBasisSpline](#), [RootFinder](#), and [SpanInterpolator](#).

BasisSplineSet

BasisSplineSet implements Samples for the Construction and the usage of various basis spline functions. It demonstrates the following:

- Construction of segment control parameters - polynomial (regular/Bernstein) segment control, exponential/hyperbolic tension segment control, Kaklis-Pandelis tension segment control.
- Control the segment using the rational shape controller, and the appropriate C^k
- Interpolate the node value and the node value Jacobian with the segment, as well as at the boundaries
- Calculate the segment monotonicity

LinearAlgebra

LinearAlgebra implements Samples for Linear Algebra and Matrix Manipulations. It demonstrates the following:

- Compute the inverse of a matrix, and multiply with the original to recover the unit matrix
- Solves system of linear equations using one the exposed techniques

PolynomialBasisSpline

PolynomialBasisSpline implements Samples for the Construction and the usage of polynomial basis spline functions. It demonstrates the following:

- Control the polynomial segment using the rational shape controller, the appropriate C^k , and the basis function.
- Demonstrate the variational shape optimization behavior.
- Interpolate the node value and the node value Jacobian with the segment, as well as at the boundaries.
- Calculate the segment monotonicity.

RootFinder

RootFinder contains a sample illustration of usage of the Root Finder Library. It demonstrates the root extraction using the following techniques:

- Newton-Raphson method
- Bisection Method
- False Position
- Quadratic Interpolation
- Inverse Quadratic Interpolation
- Ridder's method
- Brent's method

- Zheng's method

SpanInterpolator

SpanInterpolator demonstrates the Span builder and usage API. It shows the following:

- Construction of segment control parameters - polynomial (regular/Bernstein) segment control, exponential/hyperbolic tension segment control, Kaklis-Pandelis tension segment control.
- Control the segment using the rational shape controller, and the appropriate C^k
- Construct a calibrated span interpolator
- Insert a knot into the Span
- Interpolate the node value and the node value Jacobian
- Calculate the segment/span monotonicity

DRIP Non-linear Fixed-Point Search Library

The core functionality of the [DRIP non-linear fixed-point search library](#) is implemented in the package [org.drip.math.solver1D](#). This package implements a number of univariate, non-linear fixed-point search routines. Methodology separates execution initialization from variate iteration. A variety of open and closed variate iteration techniques are implemented, along with primitive/complex closed variate iteration techniques.

Functionality in this package is implemented over 19 classes – [BracketingControlParams](#), [BracketingOutput](#), [ConvergenceControlParams](#), [ConvergenceOutput](#), [ExecutionControl](#), [ExecutionControlParams](#), [ExecutionInitializationOutput](#), [ExecutionInitializer](#), [FixedPointFinder](#), [FixedPointFinderBracketing](#), [FixedPointFinderBrent](#), [FixedPointFinderNewton](#), [FixedPoinderOutput](#), [FixedPointFinderZheng](#), [InitializationHeuristics](#), [IteratedBracket](#), [IteratedVariate](#), [VariateIterationSelectionParams](#), and [VariateIteratorPrimitive](#).

BracketingControlParams

BracketingControlParams implements the control parameters for bracketing solutions. BracketingControlParams provides the following parameters:

- The starting variate from which the search for bracketing begins.
- The initial width for the brackets.
- The factor by which the width expands with each iterative search.
- The number of such iterations.

BracketingOutput

BracketingOutput carries the results of the bracketing initialization. In addition to the fields of ExecutionInitializationOutput, BracketingOutput holds the left/right bracket variates and the corresponding values for the objective function.

ConvergenceControlParams

ConvergenceControlParams holds the fields needed for the controlling the execution of Newton's method. ConvergenceControlParams does that using the following parameters:

- The determinant limit below which the convergence zone is deemed to have reached.
- Starting variate from where the convergence search is kicked off.
- The factor by which the variate expands across each iterative search.
- The number of search iterations.

ConvergenceOutput

ConvergenceOutput extends the ExecutionInitializationOutput by retaining the starting variate that results from the convergence zone search. ConvergenceOutput does not add any new field to ExecutionInitializationOutput.

ExecutionControl

ExecutionControl implements the core fixed-point search execution control and customization functionality. ExecutionControl is used for a) calculating the absolute tolerance, and b) determining whether the OF has reached the goal. ExecutionControl determines the execution termination using its ExecutionControlParams instance.

ExecutionControlParams

ExecutionControlParams holds the parameters needed for controlling the execution of the fixed-point finder. ExecutionControlParams fields control the fixed-point search in one of the following ways:

- Number of iterations after which the search is deemed to have failed.
- Relative Objective Function Tolerance Factor which, when reached by the objective function, will indicate that the fixed point has been reached.
- Variate Convergence Factor, factor applied to the initial variate to determine the absolute convergence.
- Absolute Tolerance fall-back, which is used to determine that the fixed point has been reached when the relative tolerance factor becomes zero.
- Absolute Variate Convergence Fall-back, fall-back used to determine if the variate has converged.

ExecutionInitializationOutput

ExecutionInitializationOutput holds the output of the root initializer calculation. The following are the fields held by ExecutionInitializationOutput:

- Whether the initialization completed successfully.
- The number of iterations, the number of objective function calculations, and the time taken for the initialization.
- The starting variate from the initialization

ExecutionInitializer

ExecutionInitializer implements the initialization execution and customization functionality. ExecutionInitializer performs two types of variate initializations:

- Bracketing initialization: This brackets the fixed point using the bracketing algorithm described in <http://www.credit-trader.org>. If successful, a pair of variate/OF coordinate nodes that bracket the fixed-point is generated. These brackets are eventually used by routines that iteratively determine the fixed-point. Bracketing initialization is controlled by the parameters in BracketingControlParams.
- Convergence Zone initialization: This generates a variate that lies within the convergence zone for the iterative determination of the fixed point using the Newton's method. Convergence Zone Determination is controlled by the parameters in ConvergenceControlParams.

ExecutionInitializer behavior can be customized/optimized through several of the initialization heuristics techniques implemented in the InitializationHeuristics class.

FixedPointFinder

FixedPointFinder is the base abstract class that is implemented by customized invocations, e.g., Newton's method, or any of the bracketing methodologies.

FixedPointFinder invokes the core routine for determining the fixed point from the goal. The ExecutionControl determines the execution termination. The initialization heuristics implements targeted customization of the search.

FixedPointFinder main flow comprises of the following steps:

- Initialize the fixed-point search zone by determining either a) the brackets, or b) the starting variate.
- Compute the absolute OF tolerance that establishes the attainment of the fixed point.
- Launch the variate iterator that iterates the variate.
- Iterate until the desired tolerance has been attained.
- Return the fixed-point output.

Fixed point finders that derive from this provide implementations for the following:

- Variate initialization: They may choose either bracketing initializer, or the convergence initializer - functionality is provided for both in this module.
- Variate Iteration: Variates are iterated using a) any of the standard primitive built-in variate iterators (or custom ones), or b) a variate selector scheme for each iteration.

FixedPointFinderBracketing

FixedPointFinderBracketing customizes the FixedPointFinder for bracketing based fixed-point finder functionality.

FixedPointFinderBracketing applies the following customization:

- Initializes the fixed-point finder by computing the starting brackets.
- Iterating the next search variate using one of the specified variate iterator primitives.

By default, FixedPointFinderBracketing does not do compound iterations of the variate using any schemes - that is done by classes that extend it.

FixedPointFinderBrent

FixedPointFinderBrent customizes FixedPointFinderBracketing by applying the Brent's scheme of compound variate selector.

Brent's scheme, as implemented here, is described in <http://www.credit-trader.org>. This implementation retains absolute shifts that have happened to the variate for the past 2 iterations as the discriminant that determines the next variate to be generated.

FixedPointFinderBrent uses the following parameters specified in VariateIterationSelectorParams:

- The Variate Primitive that is regarded as the "fast" method.

- The Variate Primitive that is regarded as the "robust" method.
- The relative variate shift that determines when the "robust" method is to be invoked over the "fast".
- The lower bound on the variate shift between iterations that serves as the fall-back to the "robust".

FixedPointFinderNewton

FixedPointFinderNewton customizes the FixedPointFinder for Open (Newton's) fixed-point finder functionality.

FixedPointFinderNewton applies the following customization:

- Initializes the fixed point finder by computing a starting variate in the convergence zone.
- Iterating the next search variate using the Newton's method.

FixedPointFinderOutput

FixedPointFinderOutput holds the result of the fixed-point search.

FixedPointFinderOutput contains the following fields:

- Whether the search completed successfully
- The number of iterations, the number of objective function base/derivative calculations, and the time taken for the search
- The output from initialization

FixedPointFinderZheng

FixedPointFinderZheng implements the fixed-point locator using Zheng's improvement to Brent's method.

FixedPointFinderZheng overrides the iterateCompoundVariate method to achieve the desired simplification in the iterative variate selection.

InitializationHeuristics

InitializationHeuristics implements several heuristics used to kick off the fixed-point bracketing/search process.

The following custom heuristics are implemented as part of the heuristics based kick-off:

- Custom Bracketing Control Parameters: Any of the standard bracketing control parameters can be customized to kick-off the bracketing search.
- Soft Left/Right Bracketing Hints: The left/right-starting bracket edges are used as soft bracketing initialization hints.
- Soft Mid Bracketing Hint: A mid bracketing level is specified to indicate the soft bracketing kick-off.
- Hard Bracketing Floor/Ceiling: A pair of hard floor and ceiling limits is specified as a constraint to the bracketing.
- Hard Search Boundaries: A pair of hard left and right boundaries is specified to kick-off the final fixed-point search.

These heuristics are further interpreted and developed inside the ExecutionInitializer and the ExecutionControl implementations.

IteratedBracket

IteratedBracket holds the left/right bracket variates and the corresponding values for the objective function during each iteration.

IteratedVariate

IteratedVariate holds the variate and the corresponding value for the objective function during each iteration.

VariateIterationSelectionParams

VariateIterationSelectorParams implements the control parameters for the compound variate selector scheme used in Brent's method.

Brent's method uses the following fields in VariateIterationSelectorParams to generate the next variate:

- The Variate Primitive that is regarded as the "fast" method.
- The Variate Primitive that is regarded as the "robust" method.
- The relative variate shift that determines when the "robust" method is to be invoked over the "fast".
- The lower bound on the variate shift between iterations that serves as the fall-back to the "robust".

VariateIteratorPrimitive

VariateIteratorPrimitive implements the various Primitive Variate Iterator routines.

VariateIteratorPrimitive implements the following iteration primitives:

- Bisection
- False Position
- Quadratic
- Inverse Quadratic
- Ridder

It may be readily enhanced to accommodate additional primitives.

DRIP Spline Library

The core functionality of the [DRIP spline library](#) is implemented in the package [org.drip.math.spline](#). This package implements the basis set across the different splines – their creation, the segment level calibration, the customization, and segment-level inference values. It customizes the spline construction into spline-independent segment-control parameters, and spline-dependent basis set generator parameters. Finally it implements the spline functionality according to the specification and formulation laid out in detail.

Functionality in this package is implemented over 8 classes – [BasisSetParams](#), [ExponentialTensionBasisSetParams](#), [KaklisPandelisBasisSetParams](#), [PolynomialBasisSetParams](#), [SegmentBasisSetBuilder](#), [SegmentCk](#), [SegmentConstraint](#), and [SegmentInelasticParams](#).

BasisSetParams

BasisSetParams is an empty stub class whose derived implementations hold the per-segment basis set parameters.

ExponentialTensionBasisSetParams

ExponentialTensionBasisSetParams implements per-segment parameters for the exponential tension basis set - currently it only contains the tension parameter.

KaklisPandelisBasisSetParams

KaklisPandelisBasisSetParams implements per-segment parameters for the Kaklis Pandelis basis set - currently it only holds the polynomial tension degree.

PolynomialBasisSetParams

PolynomialBasisSetParams implements per-segment basis set parameters for the polynomial basis spline - currently it holds the number of basis functions.

SegmentBasisSetBuilder

This class implements the basis set and spline builder for the following types of splines:

- Exponential basis tension splines
- Hyperbolic basis tension splines
- Polynomial basis splines
- Bernstein Polynomial basis splines
- Kaklis Pandelis basis tension splines

The elastic coefficients for the segment using C^k basis splines inside $[0, \dots, 1)$ - globally $[x_0, \dots, x_1)$ are extracted for $y = \text{Interpolator}(C^k, x) * \text{ShapeController}(x)$ where x is the normalized ordinate mapped as: $x \rightarrow \frac{x - x_{i-1}}{x_i - x_{i-1}}$. The inverse quadratic/rational spline is a typical shape controller spline used.

SegmentCk

This concrete class extends segment, and implements the segment's C^k based spline functionality. It exports the following:

- Calibration => Head Calibration, Regular Calibration
- Estimated Segment Elastics => The Basis Functions and their coefficients, C^k , the shape controller
- Local Point Evaluation => Value, Ordered Derivative
- Local Monotonicity
- Local coefficient/derivative micro-Jack, and value/coefficient micro-Jack
- Local Jacobians => Value Micro Jacobian, Value Elastic Jacobian, Composite Value Jacobian

SegmentConstraint

This class holds the segment coefficient constraints and their values.

SegmentInelasticParams

This class implements basis per-segment elastics parameter set. Currently it contains C^k and the segment specific constraints.

Installation and Deployment Notes

Installation is really simple just drop of each of the jars ([CreditProduct](#), [CreditAnalytics](#), [SplineLibrary](#), [FixedPointFinder](#), and [RegressionSuite](#)) in the class-path.

Configuration is done off of the configuration files corresponding to each of the libraries. For most typical set-ups, the standard configuration should suffice. Please consult the configuration documentation on each of the libraries to configure each of the modules.

Because there is no other dependency, deployment should also be straightforward. Use the regression output as a guide for module capacity estimation.