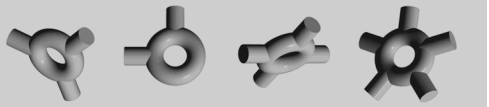
elements



Basic Rates Package

TARMS Inc.

September 07, 2000

Typeset in LaTeX.

Contents

List of Figures

List of Tables

Package Description

Rates represent two aspects of financial modeling. In the first instance, they represent the transformation of an amount of one commodity into an amount of another commodity. In the second instance, they represent the price that someone is willing to exchange a good for. These aspects are related, but reflect two different functions of rates: the first is to express an amount of one commodity in terms of another, the second is to act as a quotation mechanism.

There are also entities that are traditionally regarded as rates, but which represent the transformation of rates into other rates. For example, volatilities can be used to transform an underlying price into an optional price. As another example, premiums transform a rate into another rate of the same kind, with a different price.

The basic rates package covers a number of areas: point rates, rate curves (a generalized term that is applied to any n-dimensional rate curve) quotation mechanisms, derivation mechanisms, components, pieces, . . . [1]

Components

A rate supplied by the marketplace may have several *components* representing the rates quoted for different purposes. The most obvious example of rate components is the bid/ask spread, representing the difference between what something will be bought for and what it will be sold for. Other components include such items as the last quoted rate or the rate at close of trading. Within this model, components may be used to represent special-purpose rates.

[1] **A note on reading order.** This package is very abstract; it provides an infrastructural base for more concrete expressions of rates. On an initial reading, it may pay to skim one of the more concrete packages, such as FX rates or interest rates before examining this package.

Components are always named. A single point rate may contain several components, representing different rates for different purposes. The *bid* and *ask* components are usually assumed to be present in market rates. These rates represent what another party will buy something for and what another party will sell something for respectively. The *mid* component is the average of the bid and ask components.[2] Other common components include the *last component* — the last rate at which a transaction took place — and the *close component* — the last rate at the close of a trading day.

Pieces

Each rate component is constructed from a number of *pieces*. As an example, an interest rate may be quoted as a risk-free rate, plus a premium representing the counterparty risk, plus a premium representing the profit margin. As another example, a forward FX rate may be quoted as a spot rate, plus a forward margin.

Components are usually constructed from a *base rate* and a series of *margins* that represent additions to the base rate.

Quotation Methods

A *quotation method* describes how the bare number of a rate is intended to be interpreted. For example, a bond price might be quoted as "Price per Hundred Face Value" meaning that the figure represents the price paid for 100 units of the principal amount printed on the bond. As another example, an interest rate might be quoted as being "annualized with a 30/360 date basis" meaning that the figure represents an interest payment that compounds yearly, with the elapsed time being calculated using a 30/360 day/year-count convention.

Rate pieces may used different quotation conventions. In particular, margins are often quoted as a number of "points" over some base rate; a point might be 0.01% for an interest rate or 0.0001 for an FX rate.

Quotation methods may be freely changed, since they do not affect the underlying data. However, when a quotation method changes, the number representing the rate usually changes as well.

Rate Derivation

Rates have a derivation path. The most basic derivation path is a basic rate, a rate that is supplied by some outside source. More complex rates can be implied from

[2] This rate may not be a simple arithmetic mean of the two rates; non-linear effects may come into play.

sets of basic rates. As an example, a bond price can be implied from the definition of the bond and a suitable yield curve.

Basic rates have a *rate name*, a string which identifies the rate to an external source of some sort: a Reuters RIC box, a Telerate feed or a spreadsheet.

Rate Specification

A rate is largely independent of the quotation mechanism and derivation path. A logical rate expresses the contractual nature of the rate, without requiring any particular quotation style or derivation.

Point Rates and Prices

At the base level in the model, rates are models which represent the transformation of some amount into another amount in terms of unit contracts. A *point rate* is a rate where everything needed to be known about the transformation is fixed: the date on which it occurs, the commodities involved, etc. A *price* is a particular use of a point rate, expressing the transformation of some commodity into the amount paid for it.

Rate Curves

A *curve* is a mathematical object which represents something that is locally like a line, plane or other linear space. In the terms of this model, rate curves represent the complex curves, surfaces, solids, etc. that can be used to represent rates. As examples: a FX curve is a curve used to get FX rates at future dates, a volatility curve describes volatilities at forward dates and market prices.

The approach taken in this model is to describe complex rate structures as n-dimensional curves. A curve *segment* consists of a set of polynomials, one for each piece of each component in the rate. Disjoint segments are joined together into a tree structure. Getting a rate from a curve involves using the tree structure to get the appropriate segment and then using the polynomials to get the rate.

Curve can be converted into other curve by restricting them. For example, a volatility surface can be converted into a smile curve by restricting the date parameter to a single value.

A note on the use cases. The use cases supplied in this package do not reflect the abstract focus of the package. In some senses, the use cases should be relegated to the packages handling the various instruments that they reflect. However, these cases provide some insight into the general aim of the basic rates package.

Euro Compliance. The triangulation rules required by the EMU require that in-currency FX rates be fixed to a particular rate and that rate not be inverted. The basic rate model accomplishes this by always retaining the components of a derivation wherever possible and following the derivation route.

1 Use Cases

1.1 Simple Interest Rate

A simple interest rate is either quoted as a yield or a discount rate and applies over a fixed term, from d_1 to d_2. The rate is usually quoted as an annualized amount, with a date basis. If a_1 is the amount of some currency at d_1, a_2 is the amount of the same currency at d_2, d is the day count between d_1 and d_2 and y is the year length, then

$$a_2 = a_1(1 + \frac{id}{y})$$

or

$$a_1 = a_2(1 - \frac{rd}{y})$$

where i is the yield (interest rate) or r is the discount rate[3, 2].

1.2 Compound Interest Rate

A compound interest rate is usually quoted as a yield and applies over a fixed term, from d_1 to d_2. The rate is usually quoted as an annualized amount, with a date basis and a number of compoundings per annum.

If a_1 is the amount of some currency at d_1, a_2 is the amount of the same currency at d_2, n is the number of compoundings per annum, d is the day count between d_1 and d_2 and y is the year length, then

$$a_2 = a_1(1 + \frac{i}{n})^{\frac{nd}{y}}$$

where i is the yield (interest rate)[3].

1.3 Continuous Interest Rate

A compound interest rate is usually expressed in terms of the number of terms per annum, n, that interest compounds over. The annualized rate is given by $(1+i/n)^n$,

where i is the interest rate. A continuous interest rate takes compounding to its logical conclusion by setting the annualized rate to be $\lim_{n\to\infty}(1+i/n)^n$.

If a_1 is the amount of some currency at d_1, a_2 is the amount of the same currency at d_2, n is the number of compoundings per annum, d is the day count between d_1 and d_2 and y is the year length, then

$$a_2 = a_1 e^{r\frac{d}{y}}$$

where r is the continuously compounded rate[3].

1.4 Interest Rate

An interest rate represents the amount of money paid on a loan or deposit over a certain period. There are a number of ways of quoting an interest rate. An interest rate is related to a discount factor, in that if p is the principal of the loan or deposit and i is the interest, then the discount factor is given by $p/(p+i)$.

1.5 Discount Factor

A discount factor represents the change in value of an amount of money over time. If an amount a_1 is deposited and, later, an amount a_2 is retrieved, then the discount factor is given by a_1/a_2.

1.6 FX Rate

An FX rate is the rate at which one currency, the commodity currency, is exchanged for another currency, the counter currency. The rate is normally the amount of units of the counter currency, one unit of the commodity currency will buy.

The rate is usually expressed in terms of a standardized currency pair, with one currency being the commodity currency and one the counter currency. For example, the following currency pairs are standard: GBP/USD, USD/FRF, EUR/USD. Since the actual commodity and counter currencies may be in the reverse order to these standardized pairs, the quotation may be *indirect*, indicating that the rate is expressed in terms of amount of commodity currency needed to buy one unit of the counter currency.[3]

1.7 FX Forward Rate

A forward rate for FX is usually quoted in terms of a number of *points* over the spot rate. The size of a point depends on the two currencies being exchanged but is usually 0.0001.

For example, given a USD/DEM spot rate of 1.8989 and a 3-month forward rate of -120 points, the three month exchange rate is 1.8869.

1.8 FX Cross Rate

An FX cross rate is constructed from two FX rates with a common currency. The cross rate is built by combining the two FX rates across the common currency, giving the same effect as first exchanging the commodity currency for the common currency and then the common currency for the counter currency.

For example, if the USD/SGD rate is 1.6975 and the USD/HKD rate is 7.2210 then a cross rate for SGD/HKD is $7.2210/1.6975 = 4.253902798233 = 4.2539$.

Within the EMU, amounts being converted must follow the following pattern. The commodity currency is converted to a EUR amount, using the fixed exchange rate. The EUR amount is rounded to not less than 3 decimal places of accuracy. The rounded EUR amount is converted to the counter currency amount using the fixed exchange rate. Note that the intermediate rounding step means that amounts are not directly scalable and that calculated cross rates may not be used for some purposes.

1.9 FX Spot Rate

An FX spot rate is an FX rate for the *spot date*, usually two business days hence from the current date.

1.10 Bond Price

A bond price reflects the amount of some currency — usually the currency that the bond is denominated in — needed to purchase some quantity of the bond.

Prices are usually quoted in terms of *price per hundred face value*, the amount of currency needed to buy a face value of 100 of that bond. Alternately, prices may be quoted in terms of *yield*, where the price is expressed as a constant yield; the cost of the bond is then based on valuing the coupon and principal payments of the bond against that yield.

Bond prices may be quoted as *clean* or *dirty*. The dirty price essentially reflects the value of the future cashflows. However, the current holder of the bond will expect to receive a portion of the current coupon, based on the portion of the coupon period that has elapsed. The clean price subtracts that portion from the dirty price.[3].

1.11 Futures Price

Futures prices express the exchange of some amount of currency for a given future. Although the amounts paid to the exchange are usually expressed in terms of margins, the futures price is usually expressed in terms of an implied interest rate on the assets that are being traded.

Futures are often expressed in terms of a *hundred minus yield* convention, where the interest rate, as a percentage yield is subtracted from 100. As an example, if the implied yield is 6.78% then the futures price will be 93.22.

Other conventions include *hundred minus discount*, where a discount rate, rather than a yield is used, as well as straight yields and discount rates. Alternately, futures may be expressed in price per hundred face value terms on the value of the underlying securities.[3]

1.12 Option Premium

An *option premium* is the price paid for the purchase of an option on some underlying transaction. The term "premium" reflects the insurance-like, risk limiting aspects of options. The price paid is usually in the same currency as the underlying instrument of the option.

There are a number of different quotation methods: a straight cash price, a price per hundred face value of the underlying instrument, a price expressed in ticks on the underlying instrument or a price expressed in basis points against the implied yield on the underlying instrument.

1.13 Commodity Price

Commodity prices usually express the price of a commodity in terms of the amount of some currency that a standard amount of the commodity can be exchanged for. As an example, the gold price is often expressed in terms of USD per ounce.

Some commodity-like financial instruments have more complex ways of expressing the price.

1.14 Volatility

A volatility represents the tendency to change in some underlying instrument. Volatilities are usually used, in combination with prices for underlying instruments, to calculate prices for optional transactions.

Volatilities are usually expressed in terms of the annualized standard deviation of the logarithm of the relative price movements.[3]

1.15 Premium

A premium is an additional amount added to a rate, reflecting either some additional level of risk or a profit margin.

Interest rate premiums are usually expressed in terms of *basis points*, units of 0.01%. FX rate premiums are usually expressed in terms of points similar to the ordinary forward FX points — usually, although not always, units of 0.0001.

1.16 Yield Curve

A yield curve provides the interest rates and discount factors that apply between two dates.[3] A yield curve can, therefore, be used to reduce a series of forward cashflows to net present value.

Basic yield curves usually give the base deposit rates for a given currency. These curves may be further manipulated to add risk estimates, etc.

1.17 Exchange Rate Curve

An exchange rate curve gives the exchange rate between two currencies CUA/CUB for any date. Exchange rate curves are usually expressed as a forward rate constructed from a spot rate and forward points calculated from the date and the curve.

1.18 Implied (FX) Yield Curve

An implied yield curve, for a currency CUA is a yield curve built by combining a yield curve for another currency, CUB, and and exchange rate curve for CUA/CUB.

The discount factor for CUA between the dates d_1 and d_2 can be implied by assuming an arbitrage-free swap with currency CUB. Assume an amount a_1^{CUA} at date d_1 and an amount a_2^{CUA} at date d_2, CUA/CUB exchange rates of x_1 and x_2 for d_1 and d_2 and a discount factor of f_{12}^{CUB} for CUB between d_1 and d_2, then

$$f_{12}^{CUA} = \frac{a_1^{CUA}}{a_2^{CUA}} = f_{12}^{CUB}\frac{x_2}{x_1}$$

since $a_1^{CUB} = x_1 a_1^{CUA}$ and $a_2^{CUB} = x_2 a_2^{CUA}$.

[3] Two dates are necessary, as interest rates essentially represent a density function.

1.19 Implied (Yield) Exchange Rate Curve

An implied exchange rate curve between two currencies CUA/CUB is constructed from a spot rate, x, and a pair of yield curves for each currency.

If the discount factor between spot and a date d for CUA is f^{CUA} and the discount factor between spot and d for CUB is f^{CUB}, then an implied forward rate can be calculated by assuming an arbitrage-free swap between the two currencies. If we have spot amounts of a^{CUA} and $a^{CUB} = xa^{CUA}$ and forward amounts of b^{CUA} and b^{CUB} then the forward exchange rate, x' is given by

$$x' = \frac{b^{CUB}}{b^{CUA}} = x\frac{f^{CUA}}{f^{CUB}}$$

and forward points given by $x' - x$.

1.20 Cross Exchange Rate Curve

A cross exchange rate curve for a pair of currencies CUA/CUB is constructed from two exchange rate curves CUA/CUC and CUB/CUC with a common currency CUC.

If, at the date d, the CUA/CUC rate is x_1 and the CUB/CUC rate is x_2 then the cross rate is calculated as

$$x = \frac{x_1}{x_2}$$

Note that the EMU conventions actually require the above calculation, for traded amounts, to be calculated in terms of an intermediately rounded actual amount.

1.21 Market Yield Curve

A market yield curve is built from a series of point rates. The point rates are a set of interest rates, discount factors or prices for interest rate instruments, such as FRAs or securities. The interest rates can be used directly. Instrument prices need to have their equivalent yields calculated on the basis of a partially constructed yield curve; in the case of bonds, this process is known as *coupon stripping*.

1.22 Premium Shifted Yield Curve

Yield curves can be modified by the addition of premiums to an underlying yield curve.

Premiums usually reflect the transition between some risk-free yield curve and a curve that reflects the risk associated with a country or a counterparty. As an example, bonds denominated in USD may be issued by countries other than the US; these bonds have a greater level of risk than US bonds issued by domestic issuers and need to be valued against a yield curve that reflects that risk.

1.23 Premium Curve

A premium curve is a curve that gives a premium to apply to some underlying yield, exchange rate, volatility or other curve at a given date, strike price or other variable. Combining a premium curve and an underlying curve gives a curve of the same type as the underlying curve, with the quoted rates shifting in some direction.

1.24 Implied Curve

Implied curves are constructed from other curves – source curves – rather than market data.

Market data curves are usually built by interpolating between the rate sample supplied to the curve. In theory, implied curves could be constructed by evaluating the source curves at various points and then interpolating in a manner similar to market data curves. The differences in interpolation between the various curves will, however, lead to an artificial arbitrage caused by differences in interpolation points.

As an example, imagine two exchange rate curves for USD/SGD and USD/HKD and linear interpolation. The USD/SGD curve is built from spot, 1 month and 6 month points of 1.7018, $+100$ and $+120$. The USD/HKD curve is built from spot, 1 month and 1 year points of 7.7595, $+40$ and $+80$. A cross exchange rate curve could be built from implied points of 4.5596, -242, -285 and -336 — one for each source point. Using the interpolated cross rate curve, the 4 month exchange rate is $4.5596 - 258$. Using the individual rates, the 4 month exchange rate is $4.5596 - 259$, a difference of \$100 in a \$1 million exchange.

For this reason, points on implied curves cannot be interpolated, but must be directly calculated from the source curves.

1.25 Market Exchange Rate Curve

A market exchange rate curve is built from a series of point rates. The point rate usually consist of a spot rate, and a series of forward points for various periods. The curve is then constructed to quote in terms of spot+forward for a given date.

1.26 Market Curve

A market curve is built from a series of point rates, interpolated and extrapolated according to some agreed convention. The point rates are supplied from some external source, either a commercial market data feed, such as Reuters or Telerate, or from a database or spreadsheet of internally chosen rates.

Between the supplied rates, the point rates must be *interpolated*. Interpolation can take many forms. Examples are linear interpolation of interest rates or forward points, exponential interpolation of discount factors or Lagrangian and spline approximations.[1].

Outside the range of the supplied rates, the values supplied by the curve need to be *extrapolated*. Example extrapolations include flat or linear extrapolation of rates or simply generating an error.

1.27 Volatility Smile Curve

A volatility "smile" curve is a curve that contains the volatility of some instrument at a particular delivery date for various strike prices for that instrument.

1.28 Volatility Surface

A volatility surface is a surface that contains the volatility of some instrument for various delivery delivery dates and for various strike prices for that instrument.

2 Interfaces

2.1 ActualRateParameter

An actual parameter provides a value for a formal parameter.

More commonly, the actual parameter is some constant value.

An instance which realizes this interface provides a RateFunctionSpecifier §2.21 with values for the unit contract that the specifier encodes.

An example Date Actual Parameter is one which has a LogicalRateDateFormalParameter of
<"to-date",
"end date for an interest rate",
"Date",
continuous,
30/360>

and a value of
“12-Jul-2001”.

Another example would have the same formal parameter, but a value of
$from - date + FX\,Spot$
meaning that the value of the to-date is derived by adding the FX Spot period to the from-date parameter.

2.1.1 Relationships

	Class	Description	Notes
⇑	Comparable		
↓	ActualRateParameterModel §4.1		

⇑:Inherits ↓:Realized by

2.1.2 Operations

FormalRateParameter formalParameter() formalParameter

The formal parameter that this parameter instantiates.

Object value() value
Raises: RateSpecificationException

The parameter value. Returns the parameter value, possibly derived from the other parameters in the rate specification.

Boolean equal(Comparable arg) equal
arg: Comparable The comparable to test for equality.

Equality test. Two actual rate parameters are equal if they have the same formal parameter and their actual values are equal.

2.2 BasicRateCurveTree

An interface that allows the segmentation of a BasicRateCurve into various polynomial segments.

A tree contains a set of BasicRateCurveSegment §2.4 instances that are the leaves of the tree. These segments are used to construct rates. Above the segments, forming a binary tree, are BasicRateCurveNode §2.3 instances. A node splits the curve into two parts, one “inside” and one “outside” a region. The regions can be

any shape desired. To get a value, the tree is navigated until a segment is reached, the segment then interpolates to produce the appropriate rate.

2.2.1 Relationships

	Class	Description	Notes
⇓	BasicRateCurveNode §2.3		
⇓	BasicRateCurveSegment §2.4		
↔	BasicRateCurveModel §4.3	constructor 0..n	
↔	BasicRateCurveNodeModel §4.4	inside 0..n	
↔	BasicRateCurveNodeModel §4.4	outside 0..n	

⇓:Inherited by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

2.2.2 Operations

PointRate value(Collection<ActualRateParameter> parameters) value

parameters: Collection<ActualRateParameter> The parameters that identify the position on the rate curve.

Get the value at some point on the curve. The behavior of this method is subinterface defined.

2.3 BasicRateCurveNode

A non-leaf node on the curve tree. This node uses a LogicalRateParameterRegion §2.7 to split the curve's domain into two branches, so that the correct segment can be identified.

2.3.1 Relationships

	Class	Description	Notes
⇑	BasicRateCurveTree §2.2		
↓	BasicRateCurveNodeModel §4.4		

⇑:Inherits ↓:Realized by

2.3.2 Operations

LogicalRateParameterRegion region() region

The region for the split. Returns the region that is used to determine which branch of the tree to take.

BasicRateCurveTree insideBranch() insideBranch

The branch to take if the point is inside the region. Returns the tree branch to take when the point is inside the region.

BasicRateCurveTree outsideBranch() outsideBranch

The branch to take if the point is outside the region. Returns the tree branch to take when the point is outside the region.

PointRate value(Collection<ActualRateParameter> parameters) value

parameters: Collection<ActualRateParameter> The parameters that identify the position on the rate curve.

Raises: RateConstructorException

Get the value at some point on the curve. If the parameters when supplied to the region's inside operation return true, then return the result of the insideBranch operation, valued with the supplied parameters. Otherwise, return the result of the outsideBranch operation, valued with the supplied parameters.

If there is no inside or outside branch, raise a RateConstructorException.

2.4 BasicRateCurveSegment

A leaf node on the curve tree. A segment of the rate curve that can be represented by a multivariate polynomial. A value is calculated by using the polynomial to interpolate or extrapolate the parameters from some fixed point.

Each rate piece for each component of the rate is calculated separately. For example, if a rate has bid and ask components and each component has base, forward margin and corporate margin pieces, then 6 interpolations need to be made — one for each combination of component and piece.

2.4.1 Relationships

	Class	Description	Notes
⇑	BasicRateCurveTree §2.2		
↓	BasicRateCurveSegmentModel §4.5		

⇑:Inherits ↓:Realized by

2.4.2 Operations

Collection<ActualRateParameter> basePoint() basePoint

The base point of parameters. Returns a collection of actual parameters that are used as the base point for interpolation/extrapolation.

Dictionary coefficients(String component, String piece) coefficients

component: String The component that is being interpolated. (Eg., bid, ask, etc.)

piece: String The piece of the component that is being interpolated. (Eg., base, forward margin, etc.).

The polynomial coefficients. Returns a coefficient map that maps power indices of the various parameters onto coefficients for the supplied component and piece of component. If no coefficient map is explicitly supplied, the it returns nil.

For example, suppose the parameters were x and y and the polynomial represented was $2x + xy - 5y + 3x^2 + 1$ and if we assume that the pair i, j represents the power of x and power of y, respectively, then the coefficient map would be:
$(0, 0) \rightarrow 1$,
$(0, 1) \rightarrow -5$,
$(0, 2) \rightarrow 0$,
$(1, 0) \rightarrow 2$,
$(1, 1) \rightarrow 1$,
$(1, 2) \rightarrow 0$,
$(2, 0) \rightarrow 3$,
$(2, 1) \rightarrow 0$,
$(2, 2) \rightarrow 0$

PointRate value(Collection<ActualRateParameter> parameters) value

parameters: Collection<ActualRateParameter> The parameters that identify the position on the rate curve.

Get the value at some point on the curve.

Suppose there are n parameters, labelled $x_1, \ldots, x_n$ and that the baseVector operation returns a matching set of parameters, $x_1^0, \ldots, x_n^0$. Also suppose that the coefficients for component c, piece p are given by $C^{cp}_{i_1 \ldots i_n}$, where i_j is the power of the jth parameter. Then, if we set $z_i = x_i - x_i^0$, the value of component c, piece p is given by

$$\Sigma_{i_1,\ldots,i_n} C^{cp}_{i_1 \ldots i_n} z_1^{i_1} \cdots z_n^{i_n}$$

2.5 FormalRateParameter

A description of a parameter required by a RateFunctionSpecifier §2.21. Logical rate parameters describe the nature of a parameter, where it fits into a logical rate specifier and the type of the parameter.

Parameters may be either *continuous* or *discrete*. Continuous parameters take values that can have ordinary arithmetic operations performed upon them. Discrete parameters are enumerations that take on a range of discrete values. An example continuous parameter is a date.[4] An example discrete parameter is a currency.

FormalRateParameters obey value semantics, making them useful attributes and keys.

2.5.1 Relationships

	Class	Description	Notes
⇑	Comparable		
⇑	ValueSemantics		
⇑	Identifiable		
⇑	Validatable		
⇓	LogicalRateDateFormalParameter §2.6		
↓	FormalRateParameterModel §4.7		
↔	ActualRateParameterModel §4.1	formalParameter	

⇑:Inherits ⇓:Inherited by ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

2.5.2 Operations

String identifier() identifier

The parameter identifier. Returns a formal parameter name for the parameter. This name should match the regular expression `[A-Za-z_][A-Za-z0-9_]*`.

String description() description

[4] Note that continuous parameters do not follow the normal mathematical definition of "continuous." Dates are integer-like, but they can be added and subtracted and, therefore, form the basis of some interpolation method.

Long description of the parameter. Returns a long description of the parameter. This description is intended as a human-friendly description of what the parameter is intended for.

String type() type

The type of the parameter. Returns the string identifying the type of the parameter. This "type" refers to the behavior that the parameter is expected to exhibit. In strongly-typed languages, such as C++ or Java, the type is the interface or class of the parameter. In languages with no formal typing system, such as Smalltalk, the type is essentially the set of methods that the parameter should respond to — either a class or some more abstract, class-like entity. This string can be used for "type" checking parameters, however in a language such as C++ this will require extra work to get the class of the actual parameter.

Boolean isContinuous() isContinuous

The parameter described is continuous? Return true if this parameter is a continuous, as opposed to discrete, parameter.

Boolean equal(Comparable arg) equal

arg: Comparable

Equality test. Formal parameters are equal if their identifiers and types are equal.

Reportable validate() validate

Is valid if identifier() matches the regular expression `[A-Za-z_][A-Za-z0-9_]*`.

2.6 LogicalRateDateFormalParameter

An extension of the formal parameter structure for dates. Dates are continuous and must have an attached date basis for day count calculations.

2.6.1 Relationships

	Class	Description	Notes
⇑	FormalRateParameter §2.5		

⇑:Inherits

2.6.2 Operations

DateBasis dateBasis() dateBasis

The date basis for interpolation. Return the DateBasis that is to be used when calculating day-counts, etc. for interpolation purposes.

Boolean isContinuous() isContinuous

The parameter described is continuous? Return true.

2.7 LogicalRateParameterRegion

Parameter regions are used to break a BasicRateCurve §2.16 into a series of pieces. The BasicRateCurveTree §2.2 uses a region to break the curve's domain into two pieces, which can then, in turn, be further broken into pieces until a segment is reached.

Despite the name "region" and the use of the term "inside", instances that implement this interface will, most likely, represent a cut along a line or a plane. Those points to the left of the line being inside the region and those to the right being outside.

2.7.1 Relationships

	Class	Description	Notes
⇓	Interval §2.8		
↓	RectangularRegionModel §4.24		
↔	BasicRateCurveNodeModel §4.4	region 0..n	
↔	Interval §2.8	intervals 1..n	→

⇓:Inherited by ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

2.7.2 Operations

Boolean inside(Collection<ActualRateParameter> parameters) inside
parameters: Collection<ActualRateParameter>

Are these parameters inside the region? Test parameters to see whether the parameter set falls inside the specified region or outside it. If the parameters fall inside the region then return true, otherwise return false.

2.8 Interval

2.8.1 Relationships

	Class	Description	Notes
⇑	LogicalRateParameterRegion §2.7		
⇑	Identifiable		
↓	IntervalModel §4.9		
↔	LogicalRateParameterRegion §2.7	intervals 1..n	
↔	RectangularRegionModel §4.24	intervals	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.9 Price

A price is a general interface for rates which are intended to be quoted as prices for deals. In practice, almost all point rates can be used to quote a price. The Price interface provides a mechanism for grouping any special quotation mechanisms specific to the use of a rate in a deal.

2.9.1 Relationships

	Class	Description	Notes
⇓	PointRate §2.10		

⇓:Inherited by

2.9.2 Operations

Instrument buy(Instrument quantity) buy
quantity: Instrument The quantity to convert.
Raises: RateConversionException

Buy one instrument by paying some other instrument. This operation transforms one instrument into an equivalent instrument at the rate set by this price. This operation is bi-directional, since a price essentially represents an exchange between an amount of the instrument and the secondary instrument. If argument quantity is of the primary instrument, then the quantity is converted into the secondary instrument, and vice-versa.

"Buying" indicates that the user of this price is exchanging the other instrument for the instrument provided in quantity. This distinction has relevance when a price has a bid/ask spread.

A RateConversionException §5.2 is raised if this rate cannot convert the instrument.

Instrument sell(Instrument quantity) sell

quantity: Instrument The quantity to convert.

Raises: RateConversionException

Sell one instrument by paying another instrument. This operation transforms one instrument into an equivalent instrument at the rate set by this price. This operation is bi-directional, since a price essentially represents an exchange between an amount of the instrument and the secondary instrument. If argument quantity is of the primary instrument, then the quantity is converted into the secondary instrument, and vice-versa.

"Selling" indicates that the user of this price is exchanging the instrument provided in quantity for the other instrument. This distinction has relevance when a price has a bid/ask spread.

A RateConversionException §5.2 is raised if this rate cannot convert the instrument.

2.10 PointRate

A point rate is a rate for a single contract. A point rate can be used to convert a suitable amount of one commodity into an amount of another commodity.

Point rates are built out of several components. Generally, there will always be a bid and ask component, reflecting the spread between buy and sell. The mid component is a derived component that reflects the mid-point between the bid and ask rates. There may be other components for quotation purposes or other requirements.

Point rates exchange between two commodities. Although the exchange is notionally symmetrical, the point rate distinguishes between the *primary commodity* and the *secondary commodity*. Which commodity is the primary and which is the secondary is determined by the conventions of usage: the primary commodity is the commodity that would normally be regarded as that being traded; the secondary commodity is the commodity that is being used to pay for the primary commodity. For example, a bond price has the bond as a primary commodity and the currency exchanged for it as the secondary commodity.

2.10.1 Relationships

	Class	Description	Notes
⇑	Rate §2.14		
⇑	Price §2.9		
⇓	CrossPointRate §2.12		
⇓	BasicPointRate §2.11		

⇑:Inherits ⇓:Inherited by

2.10.2 Operations

RateQuote bid() bid

The bid component. Return the bid component, the price at which the item being traded will be bought at.

RateQuote ask() ask

The ask component. Return the ask component, the price at which the item being traded will be sold at.

RateQuote mid() mid

The mid component. Return the mid component. The mid component is the rate "halfway" between the bid and ask components. However, since the quotation methods for rates can disguise non-linear effects (eg., interest rates), the mid rate is not (necessarily) the mean of the two rates.

Instead, if amount x of the primary commodity in the rate relationship will buy y_{bid} of the secondary commodity at the bid rate and y_{ask} of the second commodity at the ask rate, then the mid rate is that rate for which x of the primary commodity will buy $(y_{bid} + y_{ask})/2$ of the secondary commodity.

RateQuote quote(String quoteName) quote
quoteName: String

Get an arbitrary rate component.

Return the named component, if such a component is held or can be derived. If no such component exists, return nil.

Commodity primaryCommodity() primaryCommodity

The primary commodity. Return the commodity that this rate uses as the primary commodity.

Commodity secondaryCommodity() secondaryCommodity

The secondary commodity. Return the commodity that this rate uses as the secondary commodity.

Instrument buy(Instrument quantity) buy
quantity: Instrument The quantity to convert.
Raises: RateConversionException

Buy one quantity of an instrument by paying some other quantity of the instrument. This operation transforms one quantity of a commodity into an equivalent quantity of another commodity at the rate set by this rate. This operation is bidirectional, since a rate essentially represents an exchange between an amount of the primary commodity and the secondary commodity. If argument quantity is of the primary commodity, then the quantity is converted into the secondary commodity, and vice-versa.

If this rate is mine, the commodity is the primary commodity and the quantity is greater than zero, use the bid rate. Any change of one of the listed characteristics flips from bid to ask. Another change flips back from ask to bid.

The Instruments, in this case, must be SimpleCashflows. A RateConversionException §5.2 is raised if this rate cannot convert the instrument.

Instrument sell(Instrument quantity) sell
quantity: Instrument The quantity to convert.
Raises: RateConversionException

Sell one quantity of an instrument by paying some other quantity of the instrument. This operation transforms one quantity of a commodity into an equivalent quantity of another commodity at the rate set by this rate. This operation is bidirectional, since a rate essentially represents an exchange between an amount of the primary commodity and the secondary commodity. If argument quantity is of the primary commodity, then the quantity is converted into the secondary commodity, and vice-versa.

If this rate is mine, the commodity is the primary commodity and the quantity is greater than zero, use the ask rate. Any change of one of the listed characteristics flips from ask to bid. Another change flips back from bid to ask.

The Instruments, in this case, must be SimpleCashflows. A RateConversionException §5.2 is raised if this rate cannot convert the instrument.

2.11 BasicPointRate

A basic point rate is a point rate where the rate is directly specified, rather than as a chain of commodity transformations, as in the CrossPointRate §2.12.

2.11.1 Relationships

	Class	Description	Notes
⇑	PointRate §2.10		
↓	BasicPointRateModel §4.2		

⇑:Inherits ↓:Realized by

2.12 CrossPointRate

A cross point rate is a rate defined in terms of two other point rates with a common commodity. When buying or selling commodities, the commodity is first transformed into an amount of the common commodity. The common commodity amount is rounded to an intermediate value, and the intermediate value is then transformed into an amount of the target commodity.

2.12.1 Relationships

	Class	Description	Notes
⇑	PointRate §2.10		

⇑:Inherits

2.12.2 Operations

Commodity commonCommodity() commonCommodity

The common commodity. Return the commodity that is common to both the primary leg and the secondary leg.

PointRate primaryLeg() primaryLeg

The primary/common leg. Return the rate that transforms between the primary commodity and the common commodity.

PointRate secondaryLeg() secondaryLeg

The secondary/common leg. Return the rate that transforms between the secondary commodity and the common commodity.

2.13 QuotationMethod

A quotation method provides enough information to interpret the actual value of a rate, in conjunction with a RateFunctionSpecifier §2.21. Quotation methods are

usually rate type specific. For example, discount rates normally apply to interest rates, forward margins to FX rates, etc.

Quotation methods come in two forms: *full* and *margin*. Full quotation methods mean that the rate pieces can be valued independently, as complete rates. Margin quotation methods mean that the rate pieces must be combined with another rate for a complete rate to be built.

Each type of rate — interest rate, FX rate, etc. — has a *canonical quotation method*, a standardized method for representing the rate.

2.13.1 Relationships

	Class	Description	Notes
⇑	Comparable		
↔	RatePieceModel §4.20	quotation method 0..n	
↔	BasicRateDerivationSpecifierModel §4.6	quotationMethod	

⇑:Inherits ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.13.2 Operations

Boolean isMargin() isMargin

Is this rate in margin form? Return true if this rate is a margin over another rate, false otherwise.

Boolean isCanonical() isCanonical

Is this the canonical representation? Return true if this quotation method represents the canonical quotation method for this rate type.

String type() type

The type of rate that this rate is for. Returns a string giving the type of rate this this quotation method can be used for.

See the RateFunctionSpecifier §2.21 interface.

parse(InputStream stream, Boolean loose, RateFunctionSpecifier specifier) parse

stream: InputStream The stream to read the value from.
loose: Boolean Perform “loose” parsing. The default value is true.

specifier: RateFunctionSpecifier The specifier to use when interpreting this rate.
Raises: ParseException

Read in a text description of a rate and convert it into an appropriately quoted rate. Read in a value from an input stream in whatever form this quotation method accepts. Raise a ParseException if it is not possible to read the value.

If loose is true, then "loosely" parse the input stream; initial white space is ignored, additional accuracy is accepted and sensibly inferable elements are inferred.

printRate(OutputStream stream, Number rate, Boolean loose, RateFunctionSpecifier specifier) printRate
stream: OutputStream The stream to print onto.
rate: Number The rate to print.
loose: Boolean Print the rate in "loose" format. The default value is false.
specifier: RateFunctionSpecifier The specifier to use when interpreting this rate.

Print a rate piece on an output stream. Print the rate in a form parseable by the parse operation. If loose is true, then additional accuracy, above that normally expected, can be printed.

Number asCanonical(Number rate, RateFunctionSpecifier specifier) asCanonical
rate: Number
specifier: RateFunctionSpecifier The specifier to use when interpreting this rate.
Raises: RateQuotationException

Convert a rate into the equivalent canonical quotation method. Convert the supplied rate, assumed to be quoted in the form given by this quotation method into an equivalent amount in the canonical quotation method. Raise a RateQuotationException if it is not possible to convert the rate.

Number fromCanonical(Number rate, RateFunctionSpecifier specifier)
fromCanonical
rate: Number
specifier: RateFunctionSpecifier The specifier to use when interpreting this rate.
Raises: RateQuotationException

Convert a rate from canonical form into this quotation form. Convert a rate supplied in the canonical quotation method into an equivalent rate in this quotation

method. Raise a RateQuotationException if it is not possible to convert the rate.

Boolean equal(Comparable arg) equal

arg: Comparable The comparable to test for equality.

Equality test. This equality test is defined by the concrete realizations of this interface.

2.14 Rate

The rate interface covers anything that, abstractly, might be regarded as a rate. This definition includes such "rates" as curves, surfaces, etc.

Rates are quoted at a specific time and have an expiry time. After the expiry time, the rate is considered to be *stale*. Stale rates should be refreshed, if possible.

Rates are either *yours* or *mine*, reflecting who is doing the quotation. If yours, the rate has been supplied from some outside source. If mine, the rate has been supplied from within the system.

2.14.1 Relationships

	Class	Description	Notes
⇓	PointRate §2.10		
⇓	RateCurve §2.15		
↔	ImpliedRateCurveModel §4.8	sources 0..n	◊

⇓:Inherited by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.14.2 Operations

RateFunctionSpecifier specifier() specifier

The rate specifier for this rate. Return the rate specifier that describes this rate.

Timestamp quoteTime() quoteTime

The time at which this rate was quoted. Return the time at which this rate became current. If this rate is derived from a basic derivation (see BasicRateDerivationSpecifier §2.19), then this is the time of quotation from the source. If this rate is derived from an implied derivation (see ImpliedRateDerivationSpecifier §2.20), then the quote time is when the implied rate was built.

Boolean isYours() isYours

Is this rate an external quotation? Return true if this rate is quoted as-if from an outside source.

Timestamp expiry() expiry

The expiry time. Return the date and time at which this rate becomes stale. Once a rate has become stale, it should be re-requested. Re-requesting applies to one-shot, as well as stream rates (see RateSource §2.26).

Boolean isStale() isStale

Is this rate stale? Return true if the expiry date is not null and the current date and time is after the (non-null) expiry date and time, false otherwise.

2.15 RateCurve

A rate curver represents some rate curve, surface or higher-order object that can be interrogated, with sufficient parameters to give a point rate for some point within its domain. The parameters used to construct rate curves must be continuous parameters (see FormalRateParameter §2.5); discrete parameters must be fixed before a curve is constructed.

Rate curves are completely abstract. Specialized versions deal with the common cases of curves and surfaces.

Rate curve are assumed to be functional, in the sense that the curve will always return the same result from a value operation with the same parameters. The functional condition implies that curve fold results can be cached or tabulated.

2.15.1 Relationships

	Class	Description	Notes
⇑	Rate §2.14		
⇓	BasicRateCurve §2.16		
⇓	ImpliedRateCurve §2.17		

⇑:Inherits ⇓:Inherited by

2.15.2 Operations

OrderedCollection<FormalRateParameter> formalParamaters() formalParamaters

The parameters of the curve. Return an ordered collection of the parameters of the rate curve. This collection is the same as the curve formal parameters that the specification holds.

PointRate value(Collection<ActualRateParameter> parameters) value
parameters: Collection<ActualRateParameter> The parameters that identify the position on the rate curve.

Get the value at some point on the curve . Return the point rate that corresponds to the combination of the point actual parameters from the specifier and the actual parameters supplied by the parameters argument. The parameter combination must fix the specifier.

2.16 BasicRateCurve

Basic rate curve abstractly describe the process by which rates are interpolated or extrapolated into a rate curve. n-dimensional rate curve are piece-wise interpolated by the construction of n-variable polynomials.

2.16.1 Relationships

	Class	Description	Notes
⇑	RateCurve §2.15		
↓	BasicRateCurveModel §4.3		

⇑:Inherits ↓:Realized by

2.16.2 Operations

PointRate value(Collection<ActualRateParameter> parameters) value
parameters: Collection<ActualRateParameter> The parameters that identify the position on the rate curve.

Get the value at some point on the curve.
Construct the point rate by using the associated constructor.

BasicRateCurveTree constructor() constructor

The rate constructor.
Return the constructor tree used to build this rate.

2.17 ImpliedRateCurve

A variety of rate curve where the point rate is constructed by combining two or more point rates given by component rate curves. The source rates need to be combined together to form the resulting point, using a suitable RateConstructor §3.1.

Source rates can be point rates, as well as rate curve. As an example, a forward FX rate implied from two yield curves needs a spot FX rate to use as a base rate.

2.17.1 Relationships

	Class	Description	Notes
⇑	RateCurve §2.15		
↓	ImpliedRateCurveModel §4.8		

⇑:Inherits ↓:Realized by

2.17.2 Operations

PointRate value(Collection<ActualRateParameter> parameters) value

parameters: Collection<ActualRateParameter> The parameters that identify the position on the rate curve.

Get the value at some point on the curve. A point rate is constructed by first getting point rates from the supplied source curves. The parameters for the source curves must match the parameters required by the source rate curves and are constructed by combining the fixed parameters from this rate curve's specifier and the parameters supplied by the parameter argument.

The resulting source point rates are then supplied to the rate constructor's construct operation, to give a resultant point rate. The constructed rate must match the yours/mine convention of the implied curve.

RateConstructor constructor() constructor

The rate constructor. Returns the rate constructor that is used to combine the source point rates together to form the returned rate.

Collection<Rate> sources() sources

The source rates. Returns the source curve or point rates that are used as sources to construct the rate.

Timestamp quoteTime() quoteTime

The time at which this rate was quoted. Return the maximum of the quote times of all the source rates.

Timestamp expiry() expiry

The expiry time. Return the minimum of the expiry dates and times of all the source rates.

2.18 RateDerivationSpecifier

A rate derivation specifier describes how a rate is to be constructed. There are two main streams of derivation: basic rates and implied rates. Basic rates have a name and a source and can be acquired from some external source of data. Implied rates are constructed from other rates, using some construction methodology.

2.18.1 Relationships

	Class	Description	Notes
⇑	Comparable		
⇓	BasicRateDerivationSpecifier §2.19		
⇓	ImpliedRateDerivationSpecifier §2.20		

⇑:Inherits ⇓:Inherited by

2.19 BasicRateDerivationSpecifier

A basic rate derivation specifier acquires a rate by means of a rate source and a rate name. The rate source is some external source of rates. The rate name is a string key to the rate source that uniquely specifies the rate (in logical, contract terms) that is required.

2.19.1 Relationships

	Class	Description	Notes
⇑	RateDerivationSpecifier §2.18		
↓	BasicRateDerivationSpecifier-Model §4.6		

⇑:Inherits ↓:Realized by

2.19.2 Operations

RateName rateName() rateName

The rate name for this rate. Returns the rate name that this rate is identified by.

RateSource rateSource() rateSource

The supplying rate source. Returns the source that supplies this rate.

Boolean equal(Comparable arg) equal

arg: Comparable The comparable to test for equality.

Equality test. Two basic rate derivations are equal if the rate sources and rate names are equal.

QuotationMethod quotationMethod() quotation-Method

The way in which the externally supplied rate will be quoted.

2.20 ImpliedRateDerivationSpecifier

An implied rate derivation specifier indicates that a rate is constructed in terms of a set of source rates, which can be combined together to build another rate. An example implied rate derivation is the building of a bond price from the bond definition and a yield curve.

2.20.1 Relationships

	Class	Description	Notes
⇑	RateDerivationSpecifier §2.18		

⇑:Inherits

2.20.2 Operations

OrderedCollection<RateDefinitionSpecifier> sources() sources

The source rates. Returns a collection of source rate specifiers that this method requires to build the rate. This collection will be the same as the specifiers returned by constructor().sources().

RateConstructor constructor() constructor

The rate constructor. Returns the rate constructor that can be used to build the implied rate.

Boolean equal(Comparable arg) equal

arg: Comparable The comparable to test for equality.

Equality test. Two implied rate derivations are equal if the sources are equal and the constructors are identical.

2.21 RateFunctionSpecifier

A rate function specifier is an abstract description of the unit contract that a rate describes. That is, it specifies what the rate can be used to do, without saying anything about how the rate is derived.

A rateFunctionSpecifier is characterised by a 'type' which specifies the type of the rate (exchange rate, interest rate etc), a set of curveParameters, which specify what rate curve the rate is on (e.g. the USD/DEM exchange rate curve or the USD yield curve), and a set of point parameters which specify a specific point along the rate curve (e.g. the 1 month point along the USD yield curve). A rate functionSpecifier must have its type and curve parameters specified. If the point parameters are not specified, then the rateFunctonSpecifier represents a whole rate curve, rather than a single point along it.

RateFunctionSpecifiers are used as part of a rate: Each rate needs to know what kind of rate it is.

Example rate function specifiers:

FX rate Commodity currency, counter currency, forward date. Without a forward date specified, the rate is a curve over a series of forward dates.

Interest rate Currency, location of origin (for bonds), party (for loans), from-date, to-date.

Exchange traded option price Option contract, expiry date, strike price, exchange.

Futures volatility Futures contract, delivery date, price. Without specified delivery dates and prices, this becomes a "smile" surface. Alternately, the price can be defined as a function of the delivery date, eg. $p = 99.23 + (d -$ '21-Jan-2000'$) * 0.01$ to provide a complex forward curve.

2.21.1 Relationships

	Class	Description	Notes
⇑	Comparable		
⇓	RateDefinitionSpecifier §2.22		

⇑:Inherits ⇓:Inherited by

2.21.2 Operations

Collection<FormalRateParameter> curveFormalParameters() curveFormalParameters

The possible parameters for this rate specifier. These formal parameters define the data required to specifiy a curve for the rate. Returns a collection of the formal parameters that give the possible parameters that may be fixed by this specifier.

Collection<ActualRateParameter> curveActualParameters() curveActualParameters

The set of parameters for this specifier. These parameters define a curve for the rate. Returns a collection of parameters that describe this specifier.

String type() type

The type of rate that this specifier specifies. Returns a string identifying the type of rate that this logical rate specifier specifies. Examples are 'Interest Rate' or 'FX Rate'.

If the language being used supports some form of interned string class (eg., Smalltalk's Symbols or Java's intern() method) then return an interned string.

See QuotationMethod §2.13.

Boolean equal(Comparable arg) equal

arg: Comparable The comparable to test for equality.

Equality test. A rate function specifier is equal to another rate function specifier if both refer to the same unit contract. A rate function specifier is equal to a non-rate function specifier if the specification, quotation method and derivations are all equal.

Collection<FormalRateParameter> pointFormalParameters() pointFormalParameters

The possible parameters for this rate specifier. These formal parameters define the data required to specifiy a point on a curve for the rate. Returns a collection of the formal parameters that give the possible parameters that may be fixed by this specifier.

Collection<ActualRateParameter> pointActualParameters() pointActualParameters

The set of parameters for this specifier. These parameters define a point rate on a curve. Returns a collection of parameters that describe this specifier.

2.22 RateDefinitionSpecifier

RateDefinitionSpecifiers are used for two distinct purposes:

(i) They are used to define rates in reference data. For example, a rate may be available from an external source, identified by a string (its rate name). In this case a rateDefinitionSpecifier can be used to specify how that external rate(a simple number) is interpreted. RateDefinitionSpecifiers can also be used to specify implied rates, which are calculated from other rates. if used for this purpose, the derivationMethod must not be null.

(ii) RateDefinitionSpecifiers are also used to request rates from a rate manager. When a rate is required, we may want to specify the rate derivation method, as well as the rate's function. For instance, we may be specifically interested in the JPY yield curve implied by the USD yield curve, and the USD.JPY exchange rate curve, rather than just wanting any available JPY yield curve. For this purpose the curveParameters of the RateFunctionSpecifier must all be non-nil. The point parameters can either all be nil (in which case a curve is requested) or all non-nil. The derivationMethod can be nil, in which case the request may return any appropriate rate, irrespective of how it is derived.

2.22.1 Relationships

	Class	Description	Notes
⇑	RateFunctionSpecifier §2.21		

⇑:Inherits

2.22.2 Operations

RateDerivationSpecifier derivationMethod() derivation-Method

This is the method by which the specified rate is derived. If the specified rate is a point rate supplied by another system, then this will be a BasicRateDerivation-Specifier, specifying the rate source, a rate name (identifier string) and quotation method. If this rate is an implied rate calculated from other rates, then the derivationmethod will be an ImpliedRateDerivationSpecifier which lists the source rates and the calculation method. This RateDefinitionSpecifier may also specify a rate curve, in which case the derivationMethod will also be an ImpliedRateDerivation-Specifier.

RateFunctionSpecifier rateFunctionSpecifier() rateFunction-Specifier

Return the RateFunctionSpecifier implicit in the receiver. That is, construct a RateFunctionSpecifier with the same type and parameters as the receiver.

Code	Description
{	The { character itself
e	Maturity date
d	Maturity day
m	Maturity month
y	Maturity year
p	Maturity (to) period.
f	Start (from) period
s	Strike price
o	Option put/call code

Table 1: Rate Name Codes

2.23 RateName

A rate name is a logical specification of a key, usable by some RateSource §2.26. Rate names map a logical rate description, in the form of a RateFunctionSpecifier §2.21 onto a string key that can be supplied to some external source of rates. Since many financial instruments are described in terms of series of contracts, some form of pattern-based naming is needed.

As a simple example, the USD interest rate for today to 1 year might be given by asking for 'USD1YD=' from some external rate source. As a more complex example, a series of futures contracts might be expressed as 'FJM{m}{y}' where the {y} and {m} elements represent year and month specifications for a particular contract.

RateNames are essentially strings with special properties and, therefore, have value semantics and inherit the ValueSemantics interface. The strings contain embedded escape sequences for elements that need to be filled out by appropriate encoders and decoders. The escape sequences begin with the { character, and end with the } character. Within the {} pair is the code for the portion of the specifier that is to be included. The codes are summarised in table 1.[5]

Two codes may be joined together into a code that combines the characteristics of both elements by placing more that one code between the braces. For example, the code {om} combines the option put/call code and the maturity month into a single symbol; a process used by Reuters ETO RIC codes.

The codes used are highly source- and instrument-specific. The coders and decoders for a particular source (see RateNameCoder §3.2 and RateNameDecoder §3.3) need to map the patterns to and from specific codes.

[5] This code table may be expanded by further instruments.

2.23.1 Relationships

	Class	Description	Notes
⇑	Validatable		
⇑	ValueSemantics		
⇑	Comparable		
↓	RateNameModel §4.19		

⇑:Inherits ↓:Realized by

2.23.2 Operations

String rateName() rateName

The rate name pattern. Return a string that contains the rate name pattern.

Reportable validate() validate

A rate name is valid if all codes within the {} braces are valid codes, as given in the code table.

Boolean equal(Comparable arg) equal

arg: Comparable The comparable to test for equality.

Equality test. Two rate names are equal if the rateName() operation returns the same string for both objects.

2.24 RatePiece

A rate piece expresses one part of a rate, either some full value or a margin on that value. Rate pieces obey ValueSemantics, making them suitable attributes.

2.24.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
⇑	ValueSemantics		
↓	RatePieceModel §4.20		

⇑:Inherits ↓:Realized by

2.24.2 Operations

QuotationMethod quotationMethod() quotation-Method

The quotation method. Return the quotation method that is used to express this rate.

Number value() value

The rate piece. Return the value for this rate.

RatePiece sum(RatePiece arg, String name, RateFunctionSpecifier specifier) sum

arg: RatePiece The rate piece to add to this value.
name: String The new identifier for the combined value.
specifier: RateFunctionSpecifier
Raises: RateQuotationException

The sum of two rate pieces. Return the rate piece that would be the sum of this rate piece and the arg rate piece. The resulting rate piece has an identifier of name. The two rate pieces must have compatible quotation methods; a RateQuotationException is raised if the two quotation methods are incompatible.

2.25 RateQuote

A rate quote consists of a single part of a rate: bid, ask, mid, last, etc. Quotes are identified by the part they represent, usually an all-lower-case name, interned, if possible.

Each quote is built from a base rate, which must have a full quotation method and a series of margins, which must have a margin quotation method. The sum of the base rate and all the margins makes the total rate. A quote may be *incomplete*, indicating that it does not have a base rate.

2.25.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
⇑	Validatable		
↓	RateQuoteModel §4.21		
↔	BasicPointRateModel §4.2	quotes 1..1	◊

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.25.2 Operations

RatePiece base() base

Return the base rate piece. Return the base rate onto which all other margin rates are added. If there is no base rate, return nil.

RatePiece total(RateFunctionSpecifier specifier) total
specifier: RateFunctionSpecifier

Return the total rate piece. Return the sum of the base rate and all margins.

RatePiece margin(String name) margin
name: String The name of the margin component.

Return the margin for a specific margin element. Return the value of the named margin, if one exists, return nil otherwise.

Boolean isComplete() isComplete

Is this quote completely specified? Return true if the quote has a base rate.

2.26 RateSource

A rate source describes a source of rate data. Rate sources are generally named and associated with some form of plug-in component that can interface with the outside source of data. Example rate sources would be a TCP/IP rate feed, a database table or a spreadsheet.

Rate sources can be either *stream* rate sources or *one-shot* rate sources. Stream rate sources feed a continuous stream of updates of a rate for the life of the rate. One-shot rate sources provide a single rate, semi-static in nature. One-shot rates are not intended to be completely static, just largely static. It is possible that new values of one-shot rates will need to be read, for example, when the processing date changes.

2.26.1 Relationships

	Class	Description	Notes
⇑	Comparable		
⇑	Identifiable		
↓	RateSourceModel §4.22		
↓	RateSourceReferenceData-Model §4.23		
↔	BasicRateDerivationSpecifier-Model §4.6	rate source 0..n	
↔	RateSourceReferenceData-Model §4.23	model 0..1	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.26.2 Operations

Boolean isStream() isStream

Is this a stream rate source? Return true if this rate source provides a continuous stream of updates, false otherwise.

RateNameCoder coder() coder

The coder to use when constructing rate names. Return the coder to use when converting parameterized rate names into keys acceptable to this external rate source.

RateNameDecoder decoder() decoder

The decoder to use when interpreting rate names. Return the decoder to use when converting rate names into logical rate specifiers.

Boolean equal(Comparable arg) equal

arg: Comparable The comparable to test for equality.

Equality test. Two rate sources are equal if the names of both sources are equal.

3 Service Interfaces

3.1 RateConstructor

A rate constructor takes a collection of rates and builds a new rate from the input

collection. Rate constructors can take many forms, and may be implemented as pluggable components.

3.1.1 Relationships

	Class	Description	Notes
↔	ImpliedRateCurveModel §4.8	constructor 0..n	

↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.1.2 Operations

Rate construct(OrderedCollection<Rate> sources) construct

sources: OrderedCollection<Rate> The rates used to build the new rate. The rates in this collection must have the same ordering as the specifiers given by the sources operation.

Raises: RateConstructorException

Build the rate. Take the supplied set of sources (which may be incomplete) and attempt to construct a result rate. If a rate cannot be constructed, raise a RateConstructorException.

OrderedCollection<RateDefinitionSpecifier> sources() sources

The sources used to build a rate. Return the collection of rate specifiers needed to build this rate. These rate specifiers are RateDefinitionSpecifiers, rather than RateFunctionSpecifiers, since this will allow us to specify the use of rates derived in a particular way. The derivationMethod does not have to be specified if this is irrelevant.

RateFunctionSpecifier result() result

The output rate specification. Return the logical specification for the rate that this constructor constructs.

3.2 RateNameCoder

Certain rate names are in the form of patterns, giving a general description for a class of rate names, which need to be filled-out by the exact nature of the contract involved. These patterns use a RateNameCoder to convert the pattern, in conjunction with a RateFunctionSpecifier §2.21 into an acceptable rate name. (Similarly, a RateNameDecoder §3.3 converts supplied rate names into logical rate specifiers.)

The exact nature of a coder is dependent on the nature of the rate feed that is being handled and the type of rates being requested. As such, a coder needs to be supplied in the form of a pluggable component.

Eg.
For a Reuters feed, a Rate Name may be expressed as simple string such DEM= for the Deutchemark Spot rate or as a parameterized string such as ZBmy where:
ZB is the code for British Pounds on the Philadelphia Board of Trade,
m is the Month (See Reuters manuals for codes. eg. F = Jan .. Z = Dec for Futures contracts
y is the last digit of the year, eg. 2001 = 1
The values for m and y would be supplied by the RateFunctionSpecifier §2.21

3.2.1 Relationships

	Class	Description	Notes
↔	RateSourceModel §4.22	encoding 0..n	

↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.2.2 Operations

String encode(RateNameCoder rateName, RateFunctionSpecifier specifier) encode

rateName: RateNameCoder The rate name to encode.
specifier: RateFunctionSpecifier The logical rate specifier to use for parameter interpretation.
Raises: RateNameException

Convert a rate name into a suitable key string. Convert the supplied rate name into a completely specified key intelligible to the associated rate source. Any parameters used within the rate name are expanded using the supplied rate specifier.

3.3 RateNameDecoder

Certain rate names are in the form of patterns, giving a general description for a class of rate names. When a rate is received from a rate feed, it needs to be decoded so that the incoming rate can be matched to a suitable logical rate (See RateFunctionSpecifier §2.21).

A RateNameDecoder matches the supplied key against all the possible RateNames §2.23 and returns the matching logical rate specifier. (Similarly, a Rate-

NameCoder §3.2 converts logical rate specifiers and rate names into complete keys.)

The implementation of a RateNameDecoder is likely to be quite difficult. In theory, the decoder needs to know about all possible traded contracts which get information from the associated feed and implement some sort of matching algorithm. In addition, the decoder needs to be able to handle the pattern-based rate names that are used to specify families of contracts.

The exact nature of a decoder is dependent on the nature of the rate feed that is being handled and the type of rates being requested. As such, a decoder needs to be supplied in the form of a pluggable component.

Eg.
A value from the Reuters feed could be: ZBZ1 123.456 The decoder will have to recognize the ZBZ1 as the RIC code for the value being quoted and then link this back to a RateFunctionSpecifier §2.21 that cause the original rate request.

3.3.1 Relationships

	Class	Description	Notes
↔	RateSourceModel §4.22	decoding 0..n	

↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.3.2 Operations

RateFunctionSpecifier decode(String key) decode
key: String The string used to identify the rate.
Raises: RateNameException

Decode a rate key string. Convert the incoming rate string into a matching RateFunctionSpecifier §2.21.

4 Classes

4.1 ActualRateParameterModel

4.1.1 Relationships

	Class	Description	Notes
↑	ActualRateParameter §2.1		
↔	FormalRateParameter §2.5	formalParameter	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.1.2 Attributes

value: Object

4.2 BasicPointRateModel

A concrete implementation of the BasicPointRate interface where the various rate components are modeled as a collection of one or more components. Subclasses of this class provide specific information on commodities and transformations.

4.2.1 Relationships

	Class	Description	Notes
↑	BasicPointRate §2.11		
↔	RateQuote §2.25	quotes 1..n	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.2.2 Attributes

specifier: RateFunctionSpecifier The specifier for the rate.

quoteTime: Timestamp The time at which the rate was quoted

isYours: Boolean True if this rate is quoted from the external perspective.

expiry: Timestamp The expiry date and time. Can be null.

4.2.3 Operations

RateQuote quote(String quoteName) quote
quoteName: String

Get an arbitrary rate component. Search the associated list of components for a component with the same identifier as the supplied component argument. Return nil if not found.

RateQuote bid() bid

The bid component. Return quote identified by 'bid'

RateQuote ask() ask

The bid component. Return quote identified by 'bid'

4.3 BasicRateCurveModel

A concrete implementation of the BasicRateCurve interface. The model is constructed by building a curve tree which can be searched to build the appropriate point rate.

4.3.1 Relationships

	Class	Description	Notes
↑	BasicRateCurve §2.16		
↔	BasicRateCurveTree §2.2	constructor 1..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

4.3.2 Attributes

specifier: RateFunctionSpecifier The specifier for the rate.

isYours: Boolean True if this rate is quoted from the external perspective.

quoteTime: Timestamp

expiry: Timestamp

4.4 BasicRateCurveNodeModel

A concrete implementation of the BasicRateCurveNode interface. The elements of the interface are implemented as associations.

4.4.1 Relationships

	Class	Description	Notes
↑	BasicRateCurveNode §2.3		
↔	LogicalRateParameterRegion §2.7	region 1..1	→
↔	BasicRateCurveTree §2.2	inside 0..1	→
↔	BasicRateCurveTree §2.2	outside 0..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

4.4.2 Operations

LogicalRateParameterRegion region() region

The region for the split. When the curve tree is being constructed each Segment is tested against this region. If the segment is "inside" the region then it is added to this node's insideBranch otherwise it it added to the outsideBranch Return the associated region.

4.5 BasicRateCurveSegmentModel

A concrete implementation of the BasicRateCurveSegment interface. The base point and coefficient matrix are kept as dictionaries of parameters.

4.5.1 Relationships

	Class	Description	Notes
↑	BasicRateCurveSegment §2.4		

↑:Realizes

4.5.2 Attributes

basePoint: Collection<ActualRateParameter> The collection of base values. For a region that is based on the RectangularRegionModel classes this will be the point that is closest to -∞ coordinates. For example for a 1-D line defined by:

((3), (5))
it would be (3).

For a 2d region define by the rectangular region:
(2,3),
(2,6),
(4,3),
(4,6)
it would be the point: (2,3).

For a 3-D region defined by the cube with corners:

(-10,1,2), (-10,1,3), (-10,3,2), (-10,3,3),
(2,1,2), (2,1,3), (2,3,2), (2,3,3),

it would be the point: (-10,1,2).

For a rectangle based region model the method for finding the base point is to sort the coordinates in ascending order by each of their ordinates and then taking the first one in the list. (eg for a 3-D system, order by x,y,z and take the first one).

coefficients: Dictionary A dictionary that maps $c \times p \times i_1 \times \cdots \times i_n \rightarrow v$ where c is the component name, p is the piece name, i_j is the power of the j^{th} parameter and v is the coefficient value.

This dictionary is likely to be sparse.

4.6 BasicRateDerivationSpecifierModel

An implementation of the BasicRateDerivationSpecifier interface.

4.6.1 Relationships

	Class	Description	Notes
↑	BasicRateDerivationSpecifier §2.19		
↔	RateSource §2.26	rate source 1..1	→
↔	QuotationMethod §2.13	quotationMethod 1..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.6.2 Attributes

rateName: RateName The name of the rate.

4.6.3 Operations

RateSource rateSource() rateSource

The supplying rate source. Return the associated rate source.

4.7 FormalRateParameterModel

4.7.1 Relationships

	Class	Description	Notes
↑	FormalRateParameter §2.5		

↑:Realizes

4.7.2 Attributes

identifier: String

description: String

type: String

isContinuous: String

4.8 ImpliedRateCurveModel

A concrete implementation of the ImpliedRateCurve interface. Underlying an implied rate is the collection of rates that are used to construct the implied rate and a constructor that build the resulting rate.

4.8.1 Relationships

	Class	Description	Notes
↑	ImpliedRateCurve §2.17		
↔	Rate §2.14	sources 1..n	→
↔	RateConstructor §3.1	constructor 1..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.8.2 Attributes

specifier: RateFunctionSpecifier The specifier for the rate.

isYours: Boolean True if this rate is quoted from the external perspective.

4.8.3 Operations

Reportable validate() validate

Validate the object.

An implied rate curve must be consistent in the sense that the source rates can be queried in such a way as to give the correct point rates for the rate constructor.

4.9 IntervalModel

4.9.1 Relationships

	Class	Description	Notes
↑	Interval §2.8		
⇓	InfiniteInfiniteIntervalModel §4.14		
⇓	InfiniteClosedIntervalModel §4.13		
⇓	InfiniteOpenIntervalModel §4.15		
⇓	ClosedInfiniteIntervalModel §4.11		
⇓	OpenInfiniteIntervalModel §4.17		
⇓	OpenOpenIntervalModel §4.18		
⇓	OpenClosedIntervalModel §4.16		
⇓	ClosedClosedIntervalModel §4.10		
⇓	ClosedOpenIntervalModel §4.12		
↓	InfiniteInfiniteIntervalModel §4.14		
↓	InfiniteClosedIntervalModel §4.13		
↓	InfiniteOpenIntervalModel §4.15		
↓	ClosedInfiniteIntervalModel §4.11		
↓	OpenInfiniteIntervalModel §4.17		
↓	OpenOpenIntervalModel §4.18		
↓	OpenClosedIntervalModel §4.16		
↓	ClosedClosedIntervalModel §4.10		
↓	ClosedOpenIntervalModel §4.12		

⇓:Inherited by ↑:Realizes ↓:Realized by

4.10 ClosedClosedIntervalModel

4.10.1 Relationships

	Class	Description	Notes
⇑	IntervalModel §4.9		
↑	IntervalModel §4.9		

⇑:Inherits ↑:Realizes

4.10.2 Attributes

lowerBoundary: Comparable Lower boundary of the interval.

upperBoundary: Comparable Upper boundary of the interval.

4.10.3 Operations

Boolean inside(ActualRateParameter parameter) inside
parameter: ActualRateParameter

Return True if:
$lowerBoundary \leq parameter \leq upperBoundary$

4.11 ClosedInfiniteIntervalModel

4.11.1 Relationships

	Class	Description	Notes
⇑	IntervalModel §4.9		
↑	IntervalModel §4.9		

⇑:Inherits ↑:Realizes

4.11.2 Attributes

lowerBoundary: Comparable Lower boundary of the interval.

4.11.3 Operations

Boolean inside(ActualRateParameter parameter) inside
parameter: ActualRateParameter

Return True if:
$lowerBoundary \leq parameter$

4.12 ClosedOpenIntervalModel

4.12.1 Relationships

	Class	Description	Notes
⇑	IntervalModel §4.9		
↑	IntervalModel §4.9		

⇑:Inherits ↑:Realizes

4.12.2 Attributes

lowerBoundary: Comparable Lower boundary of the interval.

upperBoundary: Comparable Upper boundary of the interval.

4.12.3 Operations

Boolean inside(ActualRateParameter parameter) inside
parameter: ActualRateParameter

Return True if:
$lowerBoundary \leq parameter < upperBoundary$

4.13 InfiniteClosedIntervalModel

4.13.1 Relationships

	Class	Description	Notes
⇑	IntervalModel §4.9		
↑	IntervalModel §4.9		

⇑:Inherits ↑:Realizes

4.13.2 Attributes

upperBoundary: Comparable Upper boundary of the interval.

4.13.3 Operations

Boolean inside(ActualRateParameter parameter) inside
parameter: ActualRateParameter

Return true if:
$parameter \leq upperBoundary$

4.14 InfiniteInfiniteIntervalModel

4.14.1 Relationships

	Class	Description	Notes
⇑	IntervalModel §4.9		
↑	IntervalModel §4.9		

⇑:Inherits ↑:Realizes

4.14.2 Operations

Boolean inside(ActualRateParameter parameter) inside
parameter: ActualRateParameter

Return True.
Every point is inside the infinite-infinite boundary.

4.15 InfiniteOpenIntervalModel

4.15.1 Relationships

	Class	Description	Notes
⇑	IntervalModel §4.9		
↑	IntervalModel §4.9		

⇑:Inherits ↑:Realizes

4.15.2 Attributes

upperBoundary: Comparable Upper boundary of the interval.

4.15.3 Operations

Boolean inside(ActualRateParameter parameter) inside
parameter: ActualRateParameter

Return true if:
$parameter < upperBoundary$

4.16 OpenClosedIntervalModel

4.16.1 Relationships

	Class	Description	Notes
⇑	IntervalModel §4.9		
↑	IntervalModel §4.9		

⇑:Inherits ↑:Realizes

4.16.2 Attributes

lowerBoundary: Comparable Lower boundary of the interval.

upperBoundary: Comparable Upper boundary of the interval.

4.16.3 Operations

Boolean inside(ActualRateParameter parameter) inside
parameter: ActualRateParameter

Return true if:
$lowerBoundary < parameter \leq upperBoundary$

4.17 OpenInfiniteIntervalModel

4.17.1 Relationships

	Class	Description	Notes
⇑	IntervalModel §4.9		
↑	IntervalModel §4.9		

⇑:Inherits ↑:Realizes

4.17.2 Attributes

lowerBoundary: Comparable Lower boundary of the interval.

4.17.3 Operations

Boolean inside(ActualRateParameter parameter) inside
parameter: ActualRateParameter

Return True if:
$lowerBoundary < parameter$

4.18 OpenOpenIntervalModel

4.18.1 Relationships

	Class	Description	Notes
⇑	IntervalModel §4.9		
↑	IntervalModel §4.9		

⇑:Inherits ↑:Realizes

4.18.2 Attributes

lowerBoundary: Comparable Lower boundary of the interval.

upperBoundary: Comparable Upper boundary of the interval.

4.18.3 Operations

Boolean inside(ActualRateParameter parameter) inside
parameter: ActualRateParameter

Return True if:
$lowerBoundary < parameter < upperBoundary$

4.19 RateNameModel

A concrete implementation of the RateName interface.

4.19.1 Relationships

	Class	Description	Notes
↑	RateName §2.23		

↑:Realizes

4.19.2 Attributes

rateName: String The rate name.

4.20 RatePieceModel

A concrete implementation of the RatePiece interface.

4.20.1 Relationships

	Class	Description	Notes
↑	RatePiece §2.24		
↔	QuotationMethod §2.13	quotation method 1..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.20.2 Attributes

identifier: String The name of the rate piece.

value: Number The value of the rate piece.

4.21 RateQuoteModel

A basic implementation of the RateQuote interface. This model describes a rate in terms of a single base value and a series of margins over that base value.

4.21.1 Relationships

	Class	Description	Notes
↑	RateQuote §2.25		

↑:Realizes

4.21.2 Attributes

identifier: String The quote name.

base: RatePiece The base rate piece (may be nil).

margins: Collection<RatePiece> A collection of RatePieces. This attribute may be implemented as a dictionary-like collection.

4.21.3 Operations

RatePiece margin(String name) margin

name: String The name of the margin component.

Return the margin for a specific margin element. Return the matching value from the margins collection that has the same identifier as the margin argument.

Boolean validate() validate

An item is valid if:

- Each element of the margins collection has a different identifier and must be in margin form.
- No element of the margins collection may have the same identifier as the base value
- The base attribute value may not be in margin form.

.

4.22 RateSourceModel

A concrete implementation of the RateSource interface.

4.22.1 Relationships

	Class	Description	Notes
↑	RateSource §2.26		
↔	RateNameDecoder §3.3	decoding 1..1	→
↔	RateNameCoder §3.2	encoding 1..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.22.2 Attributes

identifier: String The unique identifier for the rate source.

stream: Boolean Is this a stream source?

4.23 RateSourceReferenceDataModel

An implementation of the RateSource interface that is a subclass of ReferenceDataModel, so that rate sources can be managed by the reference data systems. This class holds an instance of a RateSource and delegates all RateSource queries to the held model.

4.23.1 Relationships

	Class	Description	Notes
⇑	ReferenceDataModel		
↑	RateSource §2.26		
↔	RateSource §2.26	model 1..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.24 RectangularRegionModel

The region consists of rectangular regions. The use of the description 'rectangular' region is somewhat misleading. This is actually a type of hyper rectangles. In 1-D the larger region being defined is a curve and the RectangularRegionModel holds a collection of 1 Interval that defines the end points of a line. In 2-D the larger region is a surface and this model holds a pair of Interval objects that define the boundaries of a rectangle. In 3D the model holds an interval triplet that define a rectangular prism. In an n-D curve there will be n intervals that define an n-D rectangle .

4.24.1 Relationships

	Class	Description	Notes
↑	LogicalRateParameterRegion §2.7		
↔	Interval §2.8	intervals 1..n	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

5 Exceptions

5.1 RateConstructorException

An exception raised when a rate constructor is unable to construct a rate, either through some error in the supplied sources, through some missing information or through the sources being outside the domain of application.

5.1.1 Operations

RateConstructor constructor() constructor

The constructor. Returns the constructor that was attempting to build the rate.

Collection<Rate> sources() sources

The source rates. Returns the sources that were being used to build the rate.

String description() description

A description of the error. Returns a description of the error that has occurred.

5.2 RateConversionException

An exception that is raised when a price or rate cannot convert a supplied Instrument.

5.2.1 Operations

Instrument instrument() instrument

The instrument that could not be converted.

Price price() price

The rate or price that was attempting to perform the conversion.

5.3 RateNameException

An exception raised when a RateName §2.23 is being expanded with escape sequence information.

5.3.1 Operations

RateName rateName() rateName

The rate name. Returns the rate name that caused the exception to be raised.

RateNameCoder coder() coder

The coder for the rate name. Returns the coder that was attempting to expand the rate name.

Integer position() position

The position where the error occurred. Returns the position within the rate name string that caused the error to occur. If the error was caused by an attempt to expand an escape sequence (usually the case), then the position points to the opening brace ({) of the sequence.

5.4 RateQuotationException

An exception raised when a quotation method cannot be sensibly used, usually while converting to another quotation method.

5.4.1 Operations

QuotationMethod source() source

The source quotation method. Returns the quotation method that the rate was initially quoted in terms of. Returns nil if there is no source quotation method.

QuotationMethod target() target

The target quotation method. Returns the quotation method that the rate was begin converted to. Returns nil if there is no target quotation method.

Number rate() rate

The rate being converted. Returns the number that the quotation method could not convert.

5.5 RateSpecificationException

This exception is raised if it is not possible to derive some constrained parameter from rate specification and a set of parameters.

5.5.1 Operations

ActualRateParameter parameter() parameter

The parameter being derived. Returns the parameter that raised this exception.

RateFunctionSpecifier specifier() specifier

The specifier that could not be satisfied. Returns the specifier that could not be used to fix this parameter.

6 Associations

Table 2: Basic Rates— Associations

Association Role	Class	Card.	Notes
constructor			
constructor	BasicRateCurveTree §2.2	1..1	→
curve	BasicRateCurveModel §4.3	0..n	
region			
region	LogicalRateParameterRegion §2.7	1..1	→
tree	BasicRateCurveNodeModel §4.4	0..n	
sources			
source	Rate §2.14	1..n	→
requesting rate	ImpliedRateCurveModel §4.8	0..n	◊
quotation method			
quotation method	QuotationMethod §2.13	1..1	→
rate piece	RatePieceModel §4.20	0..n	
rate source			
source	RateSource §2.26	1..1	→
basic derivation	BasicRateDerivationSpecifier-Model §4.6	0..n	
decoding			

Table 2: . . . continued

Association Role	Class	Card.	Notes
decoder	RateNameDecoder §3.3	1..1	→
source	RateSourceModel §4.22	0..n	
encoding			
encoder	RateNameCoder §3.2	1..1	→
source	RateSourceModel §4.22	0..n	
model			
model	RateSource §2.26	1..1	→
wrapper	RateSourceReferenceData-Model §4.23	0..1	
constructor			
constructor	RateConstructor §3.1	1..1	→
implied rate	ImpliedRateCurveModel §4.8	0..n	
inside			
inside branch	BasicRateCurveTree §2.2	0..1	→
tree	BasicRateCurveNodeModel §4.4	0..n	
outside			
outside branch	BasicRateCurveTree §2.2	0..1	→
tree	BasicRateCurveNodeModel §4.4	0..n	
quotes			
quote	RateQuote §2.25	1..n	→
rate	BasicPointRateModel §4.2	1..1	◊
intervals			
	Interval §2.8	1..n	→
	LogicalRateParameterRegion §2.7	1..n	
intervals			
	Interval §2.8	1..n	→
	RectangularRegionModel §4.24		
formalParameter			
	FormalRateParameter §2.5		→
	ActualRateParameterModel §4.1		
quotationMethod			
	QuotationMethod §2.13	1..1	→
	BasicRateDerivationSpecifier-Model §4.6		

→:Navigable ◊:Aggregate ♦:Composite

6.1 constructor

Role: constructor *Navigable* BasicRateCurveTree, 1..1.
Role: curve BasicRateCurveModel, 0..n.
The tree that the curve uses to build a rate.

6.2 region

Role: region *Navigable* LogicalRateParameterRegion, 1..1.
Role: tree BasicRateCurveNodeModel, 0..n.
The region that the tree node uses to divide the curve into segments.

6.3 sources

Role: source *Navigable* Rate, 1..n.
Role: requesting rate *Aggregate* ImpliedRateCurveModel, 0..n.
The sources for the curve.

6.4 quotation method

Role: quotation method *Navigable* QuotationMethod, 1..1.
Role: rate piece RatePieceModel, 0..n.
The quotation method for this rate piece.

6.5 rate source

Role: source *Navigable* RateSource, 1..1.
Role: basic derivation BasicRateDerivationSpecifierModel, 0..n.
The source for the basic rate.

6.6 decoding

Role: decoder *Navigable* RateNameDecoder, 1..1.
Role: source RateSourceModel, 0..n.
The decoder for a rate source.

6.7 encoding

Role: encoder *Navigable* RateNameCoder, 1..1.
Role: source RateSourceModel, 0..n.
The encoder for a rate source.

6.8 model

Role: model *Navigable* RateSource, 1..1.
Role: wrapper RateSourceReferenceDataModel, 0..1.
The held RateSource model for the reference data implementation.

6.9 constructor

Role: constructor *Navigable* RateConstructor, 1..1.
Role: implied rate ImpliedRateCurveModel, 0..n.
The constructor used to build the implied rate. The constructor must take a set of point rates and build a resulting rate from the set of point rates.

6.10 inside

Role: inside branch *Navigable* BasicRateCurveTree, 0..1.
Role: tree BasicRateCurveNodeModel, 0..n.
The branch to follow when inside the region.

6.11 outside

Role: outside branch *Navigable* BasicRateCurveTree, 0..1.
Role: tree BasicRateCurveNodeModel, 0..n.
The branch to follow when outside the region.

6.12 quotes

Role: quote *Navigable* RateQuote, 1..n.
Role: rate *Aggregate* BasicPointRateModel, 1..1.
The set of quotes that make up the rate.

6.13 intervals

Role: *Navigable* Interval, 1..n.
Role: LogicalRateParameterRegion, 1..n.

6.14 intervals

Role: *Navigable* Interval, 1..n.
Role: RectangularRegionModel.

6.15 formalParameter

Role: *Navigable* FormalRateParameter.
Role: ActualRateParameterModel.

6.16 quotationMethod

Role: *Navigable* QuotationMethod, 1..1.
Role: BasicRateDerivationSpecifierModel.

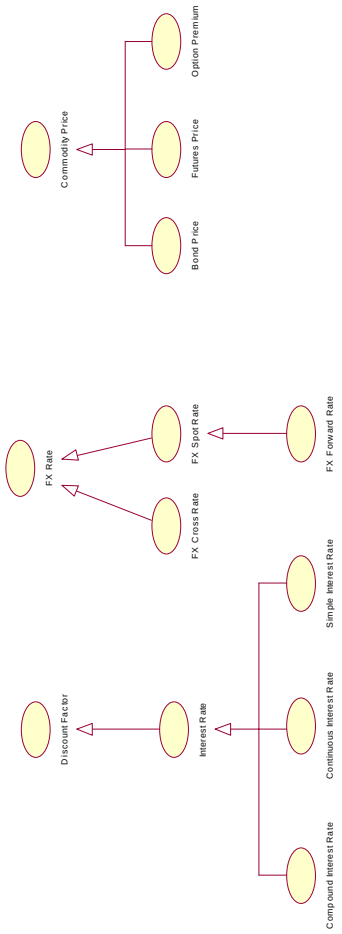


Figure 1: Class Diagram— Example Rates

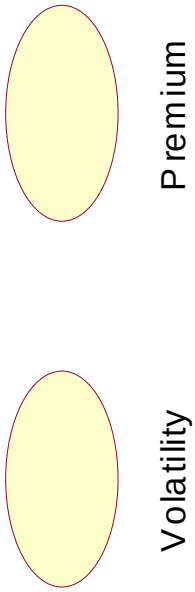


Figure 2: Class Diagram— Example Operator Rates

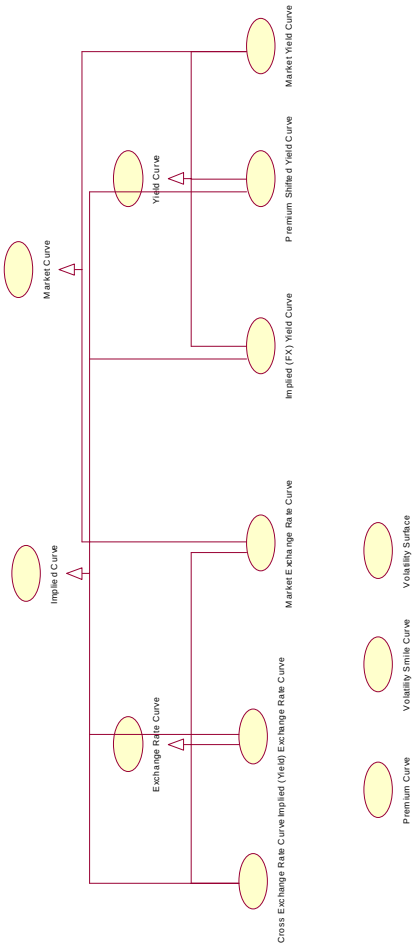


Figure 3: Class Diagram— Example Curves and Surfaces

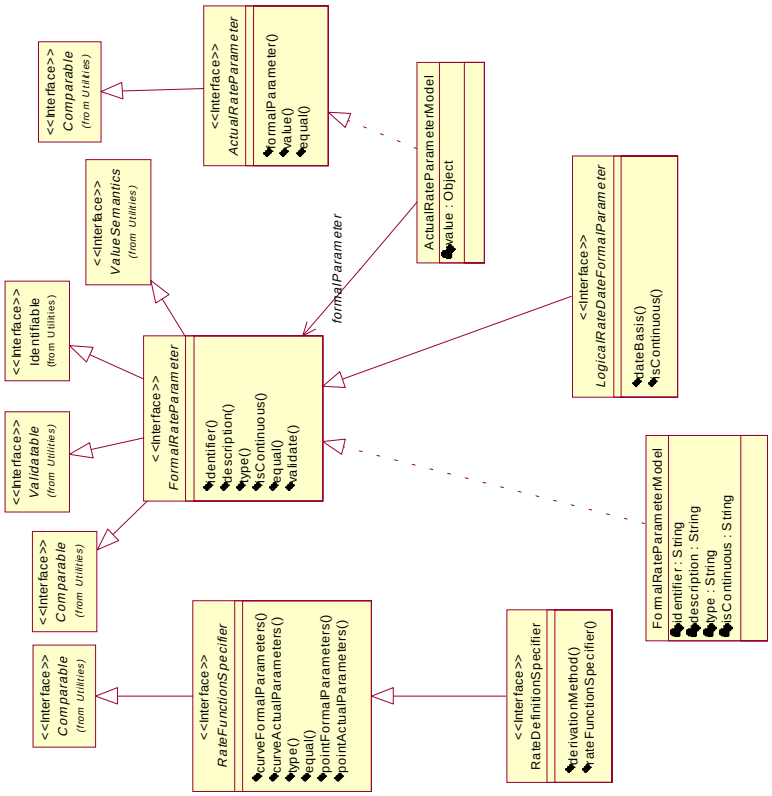


Figure 4: Class Diagram— Logical Rates

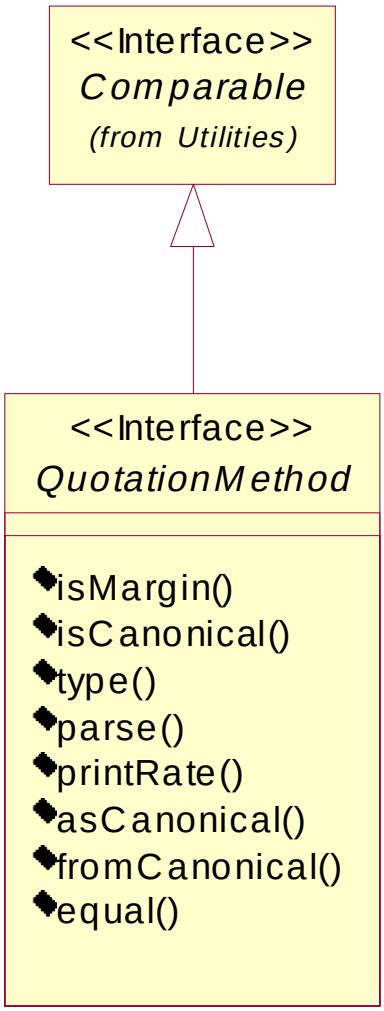


Figure 5: Class Diagram— Quotation Methods

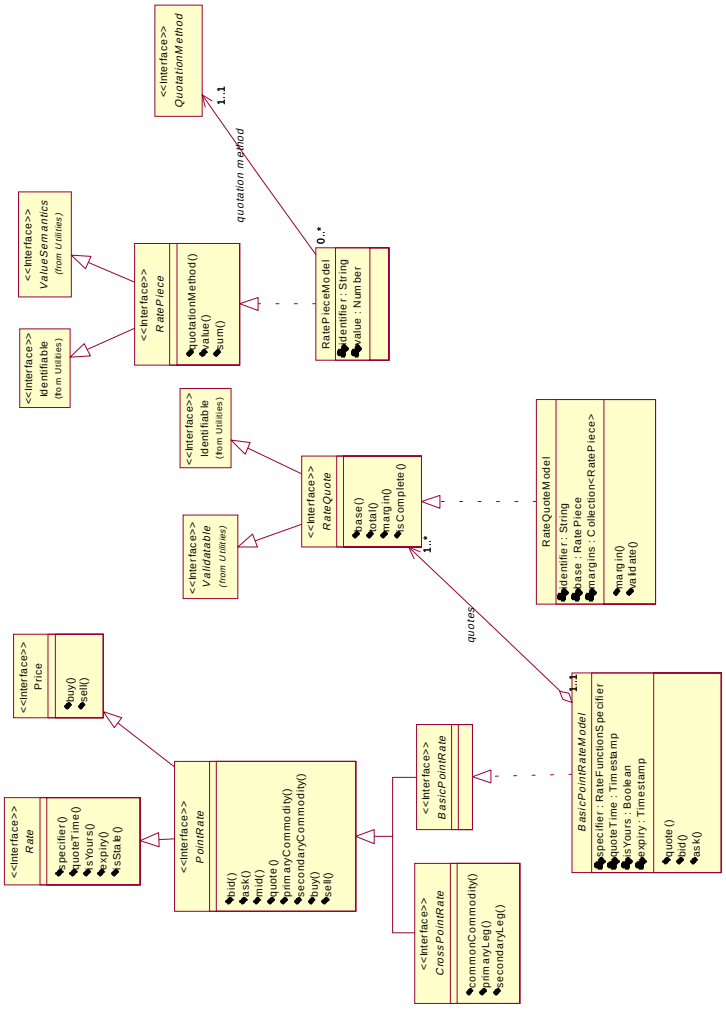


Figure 6: Class Diagram— Point Rates

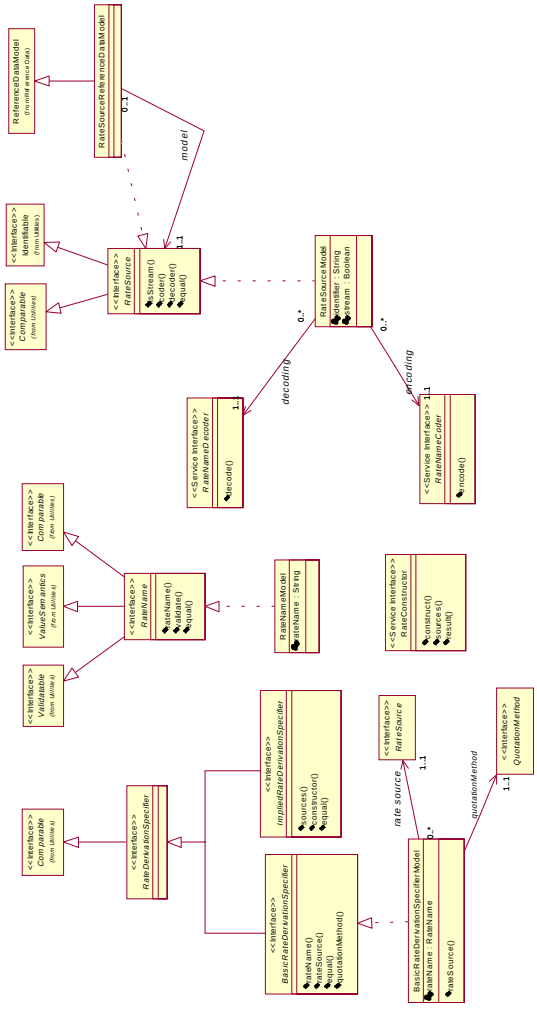


Figure 7: Class Diagram— Derivation Methods

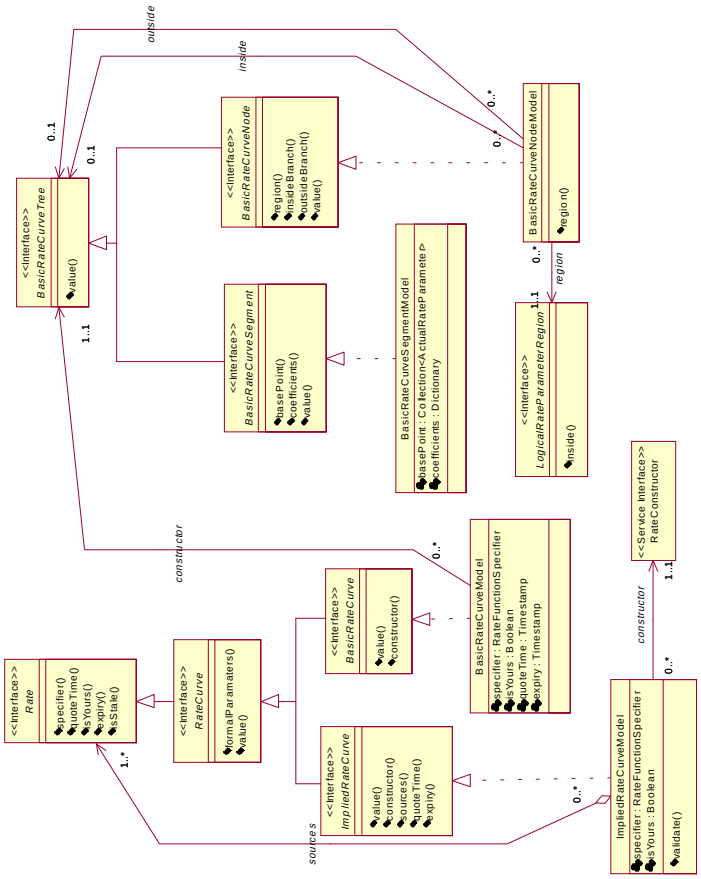


Figure 8: Class Diagram— Curves and Surfaces

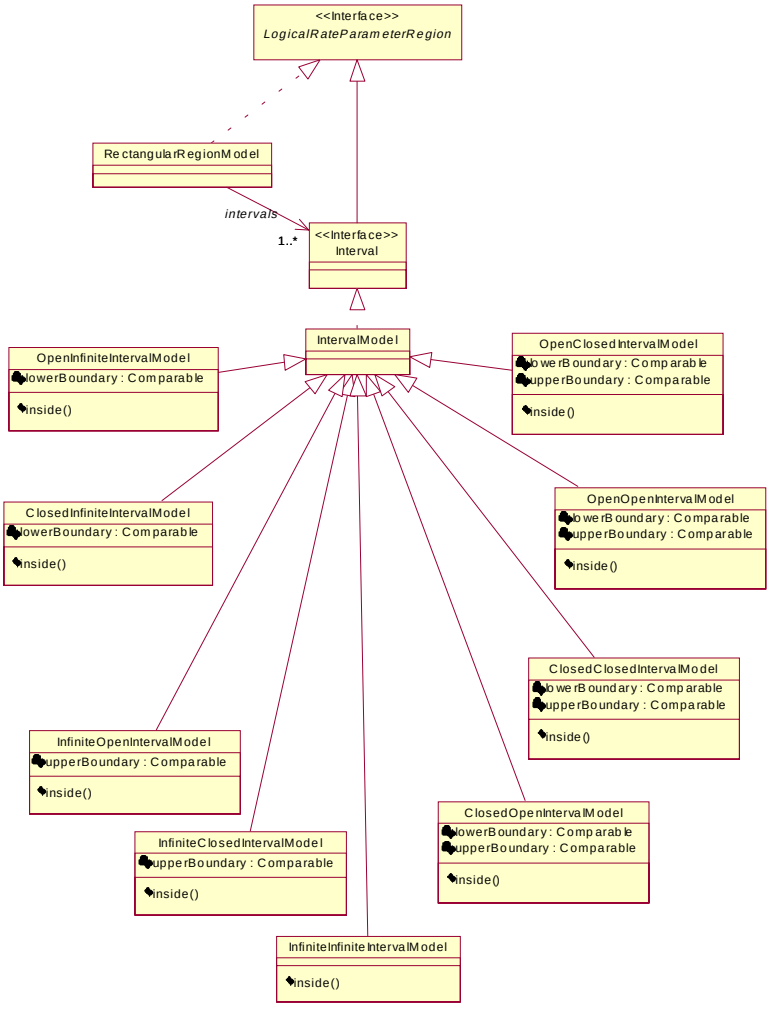


Figure 9: Class Diagram— Regions

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

[2] Michael Sherris. *Money and Captial Markets*. Allen and Unwin, 1991.

[3] Robert Steiner. *Mastering Financial Calculations*. Pitman Publishing, 1998.