Basic Rates Package - Introduction

Table of Contents

Basic Rates Package - Introduction

1. Purpose of Document

The purpose of this document is to introduce the 'Basic Rates' package in the elements object model. This document will provide sufficient background to the business problem to explain and motivate the subsequent description of the model. It will also give an overview of the model, outlining the 'mental picture' that drove the development of the object model, and then summarize the key features of the model, as documented in UML.

The readers of these documents will be:
- Anyone assessing the usefulness of the elements object model for their purpose.
- People preparing to study the associated package in detail.
- Anyone who wants an outline picture of a particular package because it is used in another package.

2. The Business Problems

The following businesses issues are addressed by this package:

2.1. What are Rates?

Rates are quantities that vary over time and are used in the valuation of assets. The most typical kinds of rates are used to convert an asset into another, different, asset. Examples of this type of rate include:
- Interest rates, which convert an amount of money at a given time into another amount of money at a different time (but in the same currency).
- Exchange rates, which convert an amount of money in one currency into an amount in another currency (at the same time).
- Share prices, which convert a share into an amount of money.

These are not the only types of rates. Another example of a rate is the volatility of a share price. This is a measure of the variability of the share price. Volatilities are used in option pricing.

A rate is identified by the set of characteristics that determine the purpose for which it can be used. For example, a rate may be identified as an exchange rate between USD and DEM, applying on 1 June 2001.

Rates have two further complications: Firstly, rates will often be quoted by a 'market maker' who is willing to either buy or sell a commodity, at an appropriate price. Market makers will invariably be willing to sell at a higher price than they will buy. Furthermore, market rate quotations often include some historical information: Highest price for the day, lowest price for the day, opening price, closing price, etc. The rate model will thus need to allow for rates with multiple components, including both buy (or 'bid') and sell (or 'ask') components.

The second complication is that we will often want to think of a rate as being composed of several independent pieces. For example, a forward FX rate can be thought of as the sum of the spot FX rate and a forward margin. Another example is yields, which may be thought of the sum of a risk-free yield (such as a US T-bill yield) and a 'credit margin'. The rate model will need to be rich enough to allow rates to be constructed from a set of such pieces.

2.2. Rate Sources

Rates can come from a variety of sources. They can be taken directly from the market, via some rate feed (such as a Reuters), or they can be calculated from other rates, using an appropriate financial model. Calculated rates would include FX cross rates and implied yields, where a yield in one currency is implied from the yield in another currency together with the appropriate exchange rate curve.

In general, a given rate may be available from several different sources. For example, the 3-month DEM yield may be available directly from a rate feed, as well as being calculated as an implied yield. Thus, we need to cater for multiple ‘versions’ of the same rate. These versions are distinguished by their ‘source’. In the case of calculated rates the source will need to specify both the source of raw data and the calculation method.

2.3. Rate Quotation Methods

Rates can be quoted in a variety of ways. For example, interest rates can be quoted as a yields or discounts; bond prices can be quoted as price per 100 face value or yield to maturity. We need to model the various ways that rates are quoted, and provide methods to convert between quotation methods.

2.4. Rate Curves

Typically, a rate will apply at a particular point in time. For example, the USD/DEM exchange rate at 3 June 2001 can only be used to convert cashflows at that date. In principle there is a USD/DEM exchange rate corresponding to each point in time. Such a set of exchange rates is referred to as an ‘exchange rate curve’.

Analogous curves arise in many situations. The parameter of the curve is not always time, and the curve is not always 1-dimensional. For example, volatilities used in option pricing will vary with both time and the strike price of the option, thus forming a volatility *surface*. In general, we want to allow for such rate ‘curves’ to have an arbitrary number of dimensions.

In practice, rate curves will often be constructed based on a limited set of rates received from an external rate source. In this case the curve will need to be constructed by using some *interpolation* method to derive values between known data-points, and an *extrapolation* method to derive values beyond all known points.

3. Overview of the Conceptual Model

The ‘Basic Rates’ package provides a set of very general classes and interfaces to support the modeling of rates. In practice, any concrete implementation of rates will require significant detail that is specific to the relevant business area. For example, an implementation of FX rates would inherit from classes in the Basic Rates package, but would contain details only relevant to FX. We have opted to keep all such business area-specific detail isolated in separate packages.

3.1. Rates

The starting points for the modeling of rates are the concepts of ‘Rate’ and ‘Price’. A ‘Rate’ is just capable of returning a ‘specifier’, that determines what it can be used for and how it is derived. Rate specifiers are described in more detail below. A ‘Price’ is an object capable of determining the purchase or sale cost of a given financial instrument. For example, a bond price would be capable of determining the amount of money that it would cost to buy or sell a specified bond.

Both Rate and Price are abstract interfaces, defining some very general behavior. The most important realization of these interfaces is ‘BasicPointRateModel’. A BasicPointRateModel will hold a rate specifier and a collection of named rate ‘quotes’. Individual quotes can be requested by name using the ‘quote()’ method.

Rate quotes allow specification of rates for multiple purposes: bid, ask, last, open, close etc. Rate quotes are identified by a ‘name’ attribute. Each rate quote consists of a set of rate ‘pieces’: A ‘base’ piece and a set of named ‘margins’. For example, a corporate FX rate may have the spot FX rate as its base piece, plus a forward margin and a corporate margin.

Each rate piece has a numerical value (the value of the rate), an ‘identifier’ string and a ‘quotation method’, which specifies how the rate is quoted.

To summarize: A Rate is identified by a *specifier*. It contains a collection of named *pieces* (bid, ask, etc). Each piece contains a set of name *components* (base rate, margins etc). Components are the bottom level of the structure, and contain the numerical values of a rate. Rate pieces are interpreted by adding together the values of their constituent components. A Rate is used by choosing one of its components.

3.2. Rate Specifiers

A rate is characterized by a ‘specifier’ that determines what it can be used for. A rate specifier has a string specifying its type, and holds parameters, which characterize the rate. These parameters are divided into two groups: Curve parameters and point parameters. The curve parameters specify what curve a rate falls on, and point parameters specify the point along this curve. For example, the 1-June-2000 USD/JPY exchange rate would have a type of ‘FX rate’, two curve parameters of type currency (‘JPY’ and ‘USD’) and a parameter of type date (1-June-2000).

Rate specifiers are actually used for three slightly different jobs:

1. The role already described. They are part of the internal data of a rate, specifying what the rate is (ie what it can be used to do), but not worrying about how it is quoted or where it comes from. For this purpose we use a ‘RateFunctionSpecifier’.

2. They can be used to interrogate the rate manager for a particular rate. For this purpose the user may or may not want to specify a derivation method (see the section on Rate Derivation Methods). If a derivation method is not specified, then the rate manager will return any rate of the right type, irrespective of how it is derived. For example, a user of rates may request the 6-month GBP interest rate, or they may request the 6-month GBP treasury rate, or they may request the 6-month GBP interest rate implied by the USD interest rate and the GBP/USD exchange rate. For this purpose we use a ‘RateDefinitionSpecifier’.

3. They can be used to specify a rate in reference data. Such rates may be *basic rates*, where the rate is derived from raw data from an external source, or they may be *implied rates*, where the rate is calculated from other rates in the system. Externally supplied (basic) rates come into the system as raw numbers, together with a string identifying the rate. The system has to be able to interpret these raw numbers. To do this, it needs to have information about how the rate is quoted (the ‘quotationMethod’) in addition to all the standard rate specifier information. Implied rates need to know what they are (their RateFunctionSpecifier), and how they are calculated, including the list of source rates from which they are calculated. For this purpose we also use a ‘RateDefinitionSpecifier’which has a derivationMethod (a RateDerivationSpecifier) in addition to its rate function information. If the rate is an external rate then the derivationMethod will be a BasicRateDerivationSpecifier, which specifies a rateSource, rateName and quotationMethod. If the rate is an implied rate then the derivationMethod will be an ImpliedRateDerivationSpecifier, which lists the source rates and construction method.

The ‘parameters’ of a RateSpecifier are modeled using two classes: ‘FormalParameters’ and ‘LogicalParameters’. FormalParameters identify the name and type of parameters. For example, a formal parameter may have name ‘startDate’ and type Date, or name ‘dealCurrency’ and type Currency. An ActualParameter will specify a formal parameter, together with the value of the parameter.

Note that rate specifier interfaces are defined in the BasicRates package. The realizations of these interfaces are defined in specific business rate packages.

3.3. Rate Derivation Methods

A given rate may be derived in several different ways: It may come from any one of several external feeds (Reuters, Telerate etc), or it may be calculated from other rates by some specified methodology (implied yields, implied FX rates, cross FX rates). ‘RateDerivationSpecifiers’ are used to specify where a rate comes from and how it is derived. There are two kinds of RateDerivationSpecifier: ‘BasicRateDerivationSpecifier’ and ‘ImpliedRateDerivationSpecifier’.

BasicRateDerivationSpecifiers are used for rates that are taken directly from some external feed. They have a ‘RateSource’ that identifies where the rate comes from, and how to translate between internal rate names and external identifiers. They also record the way in which the external rate is quoted (its quotationMethod).

ImpliedRateDerivationSpecifiers are used for calculated rates. They have a list of ‘source’ rates and a constructor. The constructor determines how to calculate the implied rate from its sources. The details of constructors are specific to particular rate calculations. ImpliedRateDerivationSpecifiers are also used to specify rate curves; a rate curve is a rate that is constructed from a set of other rates. If the rate curve is built directly from point rates, then the constructor will be an interpolation mechanism. If the rate curve is implied from other curves, then the constructor will be an implied rate calculation, such as crossing exchange rates.

3.4. Rate curves

As discussed above, we need to be able to model rate curves. That is, sets of rates that depend on one or more continuous parameters, with a different ‘point rate’ applying for each possible value of the parameters. Note that rate curves will be identified by a rate specifier (with only curve parameters), and will thus be ‘rates’ in their own right.

The Basic Rates package provides a very general model for describing a rate curves. It says nothing about how these curves are derived in the first place: This will be specific to individual types of curve, and the interpolation methods desired to be used.

The Basic Rates package assumes that all rate curves can be described as piecewise-polynomial functions. That is, that each curve can be broken down into a set of separate regions, in each of which the rate curve can be specified by a polynomial.

For convenience of interrogation, these regions are organized into a binary tree structure. Each node of the tree *(a BasicRateCurveNode)* has a region *(a LogicalRateParameterRegion)* defined that determines what parts of the curve are defined by each of the nodes branches; one branch is known as the *inside* branch, the other is the *outside*. If a parameter value falls inside the specified region then the segment information for that item lies down the *inside* branch, otherwise it’s down the *outside* branch. To find a desired point on a rate curve, you ask the top node for its value at that point. This node will determine whether the desired point lies on its inside or outside branch, and pass on the query accordingly. This process recurses down the tree until a leaf node *(a BasicRateCurveSegment)* is reached. Leaf nodes store a polynomial function, and are capable of calculating their value at any specified point.

4. Summary of the UML Model

The Basic Rates package introduces the following principal classes and interfaces:

4.1. BasicRateCurveTree

Root object class for construction of a Curve Tree. A curve tree is a *binary tree* that breaks a rate curve into a series of segments. The segment’s position on the curve tree is determined by the *region* that it occupies. The regions value is determined by its LogicalRateParameterRegion (4.4) and LogicalRateSpecifier (4.5).

4.2. BasicRateCurveNodeModel

A rate tree is made up of a series on nodes and segments. The nodes of a tree have a region (*LogicalRateParameterRegion)* definition that determines what data is on its branches, one branch is known as the *inside* branch, the other is the *outside*. If a parameter value falls inside the region then the segment information for that item lies down the *inside* branch, otherwise it's down the *outside* branch.

4.3. BasicRateCurveSegmentModel

A segment is the leaf object of the curve tree. This object contains information about the base point where the segment starts and a series of coefficients that form the polynomial that makes up the curve that the segment spans.

4.4. LogicalRateParameterRegion

This object contains data that defines the boundaries that the tree under a node spans. For example, if the tree under a node holds data that spans the date period 1/1/2000 to 31/1/2000 then that date interval is held in a LogicalRateParameterRegion object. The intervals that a region can hold are:

Model	Range
InfiniteInfiniteIntervalModel	$-\infty < x < \infty$
InfiniteClosedIntervalModel	$-\infty < x \le i_e$
InfiniteOpenIntervalModel	$-\infty < x < i_e$
ClosedInfiniteIntervalModel	$i_s \le x < \infty$
OpenInfiniteIntervalModel	$i_s < x < \infty$
OpenOpenIntervalModel	$i_s < x < i_e$
OpenClosedIntervalModel	$i_s < x \le i_e$
ClosedClosedIntervalModel	$i_s \le x \le i_e$
ClosedOpenIntervalModel	$i_s \le x < i_e$

The interval defines what is *inside* and *outside* of the region. Values that are within the interval are *inside* otherwise the values are *outside*. The concept of inside and outside are used when navigating a BasicRateCurve tree.

4.5. LogicalRateSpecifier

A LogicalRateSpecifier describes the components that define a rate. Taking the example of an F.X. Cross Rate, the LogicalRateSpecifier would define the Parameters of the rate (Commodity, Counter Currency, Forward Date) and Quotation Method (Direct Method).

A LogicalRateSpecifier is used to control navigation in a BasicRateCurveTree (4.1)

4.6. LogicalRateFormalParameter

Instances of this class are used to define the components that specify a rate. For example, if a rate is defined by a currency and spot date then the Formal Parameters for that rate will be:

Identifier	Description	Type	IsContinuous
Currency	Currency for Rate	Currency	yes
Spot Date	Date for rate	Date	yes

4.7. LogicalRateActualParameter

An object of this type is an instance of a rate parameter. For example, f the rate is defined by a currency and spot date then the Actual Parameter could be:

FormalParameter	Value
Currency	“USD
Spot Date	“01/01/2000”

5. Use of the Object Model

5.1. LogicalRateFormalParameter

The Formal Parameter for an F.X. Currency Cross Rate would be made up of a Commodity, a Counter Currency and a Forward Date. e.g.

Identifier	Description	Type	IsContinuous
Commodity	Currency being traded	Currency	yes
Counter Currency	Counter currency	Currency	yes
Forward Date	Date for rate	Date	yes

5.2. LogicalRateActualParameter

The Actual Parameter for the F.X. Currency Cross Rate for “USD/GBP, Forward = 01/01//2000” would be:

FormalParameter	Value
Commodity	“USD
Counter Currency	“GBP”
Forward Date	“01/01/2000”

5.3. Rate Curve Trees

A Yield Curve:

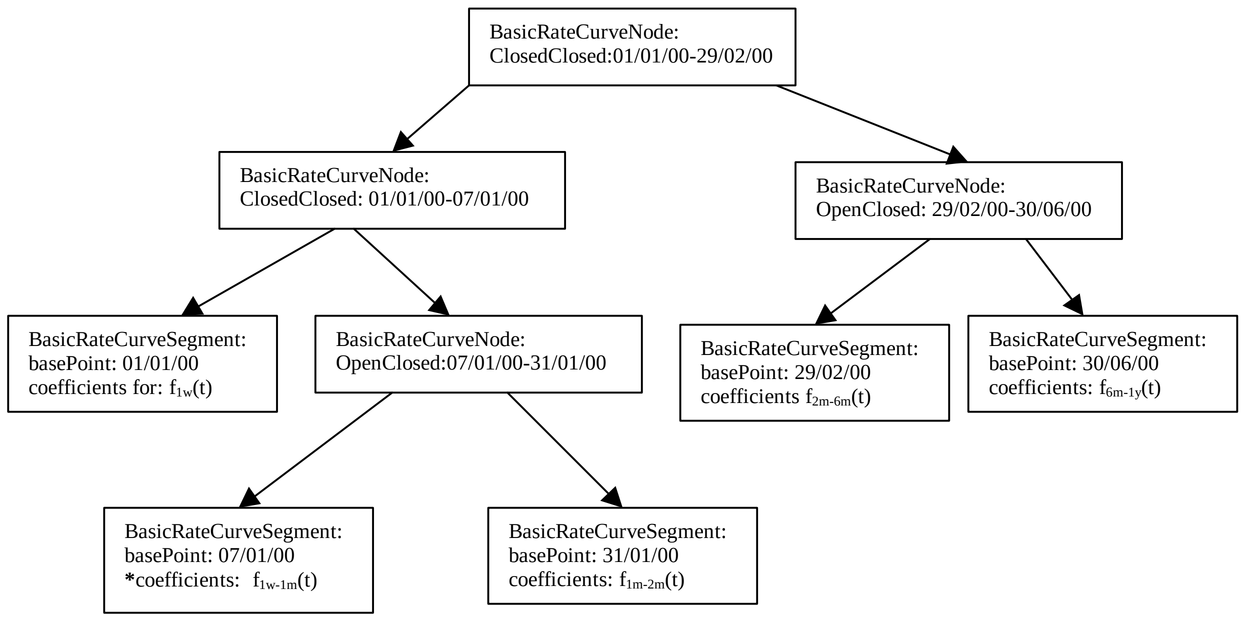
Date From 01/01/2000 for:	Yield
1 Week (to 07/01/2000)	2%
1 Month (to 31/01/2000)	3%
2 Months (to 29/02/2000)	4%
6 Months (to 30/06/2000)	4.5%
1Year (to 31/12/2000)	5%

Gives us the following intervals:

Date Range	Time Range ($t = d - \text{start date}$)	Function (Assuming linear interpolation*)
$01/01/2000 \le d \le 07/01/2000$	$0 \le t \le 7$	$f_{1w}(t) = 0.02$ (There is no point before 1 week, so we assume the curve is flat between 1 day and 1 week)
$07/01/2000 < d \le 31/01/2000$	$7 < t \le 31$	$f_{1w\text{-}1m}(t) = 0.02 + ((0.03 - 0.02) * (t - 7) / (31 - 7))$ $= 0.0170833 + (0.00041666 * t)$
$31/01/2000 < d \le 29/02/2000$	$31 < t \le 60$	$f_{1m\text{-}2m}(t) = 0.03 + ((0.04 - 0.03) * (t - 31) / (60 - 31))$ $= 0.01931034 + (0.00034483 * t)$
$29/02/2000 < d \le 30/06/2000$	$60 < t \le 182$	$f_{2m\text{-}6m}(t) = 0.04 + ((0.045 - 0.04) * (t - 60) / (182 - 60))$ $= 0.03754098 + (0.000040984 * t)$
$30/06/2000 < d \le 31/12/2000$	$182 < t \le 366$	$f_{6m\text{-}1y}(t) = 0.045 + (0.05 - 0.045) * (t - 182) / (366 - 182)$ $= 0.04005435 + (0.000027174 * t)$

* Note: From Mastering Financial Calculations, p24:
Interpolated annual rate is: $i = (i_1 + ((i_2 - i_1) * (t - t_1)/(t_2 - t_1)))$

These intervals can be organized in any one of several possible trees. For example:


BasicRateCurveNode: ClosedClosed:01/01/00-29/02/00
BasicRateCurveNode: ClosedClosed: 01/01/00-07/01/00
BasicRateCurveNode: OpenClosed: 29/02/00-30/06/00
BasicRateCurveSegment: basePoint: 01/01/00 coefficients for: $f_{1w}(t)$
BasicRateCurveNode: OpenClosed:07/01/00-31/01/00
BasicRateCurveSegment: basePoint: 29/02/00 coefficients $f_{2m-6m}(t)$
BasicRateCurveSegment: basePoint: 30/06/00 coefficients: $f_{6m-1y}(t)$
BasicRateCurveSegment: basePoint: 07/01/00 *coefficients: $f_{1w-1m}(t)$
BasicRateCurveSegment: basePoint: 31/01/00 coefficients: $f_{1m-2m}(t)$

* If $f(t) = a_0 + a_1t + a_2t^2$ then the coefficients would be (a_0, a_1, a_2)
e.g. if $f_{1w-1m}(t) = 0.01 + 0.001t + 0.0001t^2$ the coefficients are (0.01, 0.001, 0.0001)

For the example above, the $f_{1w-1m}(t) = 0.0170833 + (0.00041666 * t)$ so the coefficients are:
(0.0170833, 0.00041666)

To use the Rate Tree to determine the rate for a given date. eg. what is the rate for 02/02/00.

1. Navigate though the Rate Tree:
   1.1. 02/02/00 is **inside** *ClosedClosed:01/01/00-29/02/00* take the **inside** (left) branch.
   1.2. 02/02/00 is **outside** ClosedClosed: 01/01/00-07/01/00 take the **outside** (right) branch.
   1.3. 02/02/00 is **outside** OpenClosed:07/01/00-31/01/0000 take the **outside** (right) branch.
   1.4. Segment: BasicRateCurveSegment:
   basePoint: 31/01/00
   coefficients: $f_{1m-2m}(t)$
   is reached. This is used for the calculation.
2. $f_{1m-2m}(t) = 0.01931034 + (0.00034483 * t)$
   t = 32.
   interest = 0.030345