Retail passion. Innovative solutions.

Retek® Predictive Application Server™ 11.1

Calculation Engine Guide

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403
USA

888.61.RETEK (toll free US)
Switchboard:
+1 612 587 5000

Fax:
+1 612 587 5100

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46

Fax:
+44 (0)20 7563 46 10

Customer Support

Customer Support hours

Customer Support is available 7x24x365 via email, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance. Retek customers on active maintenance agreements may contact a global Customer Support representative in accordance with contract terms in one of the following ways.

Contact Method	Contact Information
E-mail	support@retek.com
Internet (ROCS)	rocs.retek.com Retek's secure client Web site to update and view issues
Phone	+1 612 587 5800

Toll free alternatives are also available in various regions of the world:

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step-by-step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Contents

Chapter 1 – Introduction

Overview

The RPAS calculation engine is an engine that is built to support OLAP type calculations against a multi-dimensional model. The engine is very powerful and flexible, and, at first sight, very complex. However, when the ‘building blocks’ of the calculation engine are properly understood, much of this apparent complexity goes away. This overview of the calculation engine processes will therefore start by describing the three fundamental processes of aggregation, spreading and expression evaluation, before explaining how the various processes integrate into a comprehensive whole.

RPAS supports an OLAP type model. In this model, individual pieces of data, named ‘cells’, apply to a single position in one or more hierarchies or dimensions. These will typically include a ‘measures’ dimension, a calendar or time hierarchy, and other hierarchies such as for products and locations. The measures dimension is fundamentally different to the other hierarchies, since measures (which in other systems may be referred to by terms such as facts, performance indicators or variables) represent the fundamental events or measurements that are being recorded, whereas the positions in the other hierarchies provide a context for the measurement: where, when, what, etc. Measures relate to one another through rules and expressions: positions in all the other hierarchies relate to each other through hierarchical relationships.

RPAS supports two different forms of relationships between cells: hierarchical relationships that require ‘aggregation’ and ‘spreading’, and measure relationships that require rules and expressions. Hierarchical relationships, such as weeks rolling up to months, or stores rolling up to regions, require the aggregation of data values from lower levels in a hierarchy to higher levels, using a variety of methods, as appropriate to the measure. To enable such data to be manipulated at higher levels, RPAS supports ‘spreading’ the changes, also using a variety of methods. Aggregation and spreading are basic capabilities of the engine that require no coding by the implementer, other than the selection of aggregation and spreading types to use for a measure.

The inherent relationships between measures can be modeled through a rich rule and expression syntax. Most of the effort in configuring an application model is in modeling these relationships.

The RPAS calculation engine is designed to be robust, and extensible, but in complete control of the calculation process. It enforces integrity of the data by ensuring that, where possible, all known ‘relationships’ between cells, for whatever reason, are always enforced. Much of the logic of the processing of rules and rule groups depends on this basic principal.

Chapter 2 – Measure Definition and Base Intersections

Certain characteristics of a measure determine how the calculation engine should handle it with regard to calculation, aggregation and spreading, and the dimensions in the hierarchies at which the measure is calculated. Since this information applies across all rules and rule groups, it is set up as part of the definition of a measure.

Data Types

RPAS supports the following data types:

- Real
  Floating point numeric values. Most measures are of this type.
- Integer
  Numeric integer values. There are no special ‘spreading’ algorithms for integer measures, which should normally be used only for measures that are calculated ‘bottoms up’.
- Date
  Date and time. Can easily be converted to position names by standard functions.
- String
  Variable length strings, typically used for notes and names.
- Boolean
  True or false values, typically used for flags and indicators.

Base Intersection

The base intersection for a measure is a list of dimensions (such as Class/Store/Week), one per appropriate hierarchy, which defines the lowest level at which data is held for the measure. Data is assumed to apply to the ‘All’ position in any hierarchy not explicitly referenced in the base intersection (see Chapter 9 – Non-Conforming Expressions for more information). Through aggregation, data will logically exist (though there may not be a value) for all levels higher than the base intersection up all alternative rollups.

Aggregation and Spreading Types

The aggregation type defines the aggregation method to be used for the measure (see Chapter 3 – Aggregation) to produce values at higher levels from values at the base intersection. There is a ‘normal’ spreading method associated with an aggregation type, which defines the method to be used to spread changes from higher levels (see Chapter 4 - Spreading) to the base intersection. Depending upon the desired characteristics of the measure, there may be several valid allowed spreading types.

Chapter 3 – Aggregation

Overview

An OLAP type model has, by definition, hierarchical relationships between positions in hierarchies. The values of measures above their base intersections for these hierarchical relationships are automatically maintained through a process referred to as aggregation.

Different types of measures need to be aggregated in different ways. Many measures, such as sales, receipts and markdowns, record the events that actually occurred or are planned to occur during a period of time. Simple totaling can produce aggregate values for these: the value for a region is the sum of the stores in the region; the value for a month is the sum of the weeks in the month, and so on. But this technique does not work for all types of measures. With stock, for example, the values record a ‘snapshot’ at a point in time, not a total of events over a period of time. The value of stock for a region is the sum of the stock in the stores in the region, but the value of stock for a month is certainly not the sum of the stocks for the weeks in the month (it is usually either the value for the first week or the last week in the month). Similarly, there are measures where the appropriate aggregation type may be to calculate an average, or a minimum, and so on. For some calculation purposes, only cells that are ‘populated’ (that is, have a value other than their default value, which is typically zero) should participate in aggregations. RPAS supports a wide variety of aggregation types to support all these requirements.

There is also another class of measures where no aggregation technique would produce the correct result. These measures are typically prices, ratios, variances, and similar performance indicators. The average price of sales for a class cannot, of course, be calculated by summing the prices of items in the class. Averaging the prices of items in the class produces a better result, but it is still not accurate, as it fails to take account of the weighting of the sales of the items in the class: one item with a very large volume of sales at a low price, would pull down the average price attained for the class as a whole, but this would not be reflected in an average aggregation. The way to get a correct result is to redo the price calculation at the required level. By dividing the sales value for the class by the sales units for the class (both of which will have been aggregated by summing), a correctly weighted result will be produced. The type of measure that requires this type of ‘aggregation’ is referred to as a ‘recalc’ measure, as ‘aggregation’ is by recalculation of the expression used to calculate the measure. In planning applications it is not unusual for 40% or more of the measures to be of recalc type.

Aggregation Types

The RPAS calculation engine supports the following aggregation types:

- recalc
  (For any measure type) The measure is not aggregated, but is recalculated at all aggregated levels, through a recalc expression.
- none
  (For any measure type) The measure is not aggregated. The value for all cells above the base intersection is the naval.
- total
  (for numeric measures only) The measure is aggregated by taking the total (numeric sum) of all child values at the base intersection.
- average
  (for numeric measures only) The measure is aggregated by taking the numeric average of all child values at the base intersection.
- min
  (for numeric and date measures only) The measure is aggregated by taking the minimum of all child values at the base intersection. Note that for most purposes, the min_pop aggregation type will be more appropriate, as the minimum value of all child values will typically be the naval, which is usually zero.
- max
  (for numeric and date measures only) The measure is aggregated by taking the maximum of all child values at the base intersection.
- median
  (for numeric measures only) The measure is aggregated as the median value (the middle value when sorted from lowest to highest) of all child values
- pst [period start total]
  (for numeric measures only) For cells at the base intersection in the time hierarchy, the measure is aggregated by taking the total (numeric sum) of all child values. For cells at aggregated levels in the time hierarchy, the measure is aggregated by taking the value of the first child time period.
- pet [period end total]
  (for numeric measures only) For cells at the base intersection in the time hierarchy, the measure is aggregated by taking the total (numeric sum) of all child values. For cells at aggregated levels in the time hierarchy, the measure is aggregated by taking the value of the last child time period.
- and
  (for Boolean measures only) The measure is aggregated by performing a Boolean and of all child values.
- or
  (for Boolean measures only) The measure is aggregated by performing a Boolean or of all child values.

- ambig
  (for string type measures only) The measure is aggregated by considering the values of all child cells; if all child cells have the same value, the aggregated value is the same as the child cells, otherwise it is ambig.
- popcount
  (For any measure type) The measure is aggregated by counting the number of child cells which are populated (that is, have a value different to the naval for the measure).

There are also ‘pop’ (that is, “populated”) versions of several aggregation types. These aggregate in the same manner as the aggregation type, above, but only consider cells that are populated; that is, have a value different to the naval for the measure, which may not necessarily mean a value that an end-user thinks of as being “populated”). These aggregation types are:

- ambig_pop
  ambig of all populated values
- average_pop
  average of populated values
- min_pop
  minimum of populated values
- max_pop
  maximum of populated values
- median_pop
  median of populated values
- total_pop
  total of populated values

Chapter 4 – Spreading

Introduction

An OLAP type model has, by definition, hierarchical relationships between positions in hierarchies. Measures are calculated in dimensions above the base intersection by aggregation, using the parent-child relationships between the positions. RPAS allows such measures to be manipulated not only at the bottom levels, but also at aggregated levels. In order to preserve the integrity of the data with such a change, RPAS needs to change the underlying data values at the base intersection for the measure, so that when they are aggregated again, they result in the changed value at the aggregated level. The method of changing the base intersection values to achieve this is known as spreading.

Spreading always applies to cells at the base intersection for the measure. At all aggregated levels above the base intersection, the effect of any change is applied by considering all cells at the base intersection that are descended from (children, grandchildren, etc.) the changed cell. These calls are described as ‘child cells’ in this description. Spreading *does not* operate from level to level to level down a hierarchical roll-up, which would not only be less efficient, but would also generate different (and generally less acceptable) results when there are changes or locks at levels between the change being spread and the base intersection.

The RPAS engine allows changes to be made to a measure for positions at multiple levels, and the effect of all such changes are actioned in a single calculation step. The basic technique for managing this spreading is the same for all spreading methods, and is described in “

Multi-Level Spreading”.

For calculation purposes, a lock to a cell for a spreadable measure is treated as a change to that cell that re-imposes the previous value. If none of the child cells of the locked cell have changed, the lock has no effect, and all child cell values remain unchanged.

Locks, and spreading around locked and changed cells

Other than in the special case where there are no cells that are free to be changed, spreading only affects cells that are free to be changed. All child cells are free to be changed except those that are elapsed (see Chapter 8), locked by the user, that the user has explicitly changed, or that have already been recalculated as the result of spreading another (lower level) change. Spreading, therefore, always attempts to spread around locked or changed cells, without changing their values. Where none of the child cells are free to be changed, spreading applies to all child cells that are not elapsed, using the changed or recalculated values as the base values to spread upon. Thus for spreading purposes, when something has to give, elapsed cells are considered to be 'more important' than locked or changed cells.

Locked cells for recalc type measures are treated in an analogous manner: the mapping expression (see The Spreading of Recalc Type Measures) is reimposed (using recalculated values of other measures on the right hand side of the mapping expression if necessary) to recalculate the mapped measure. This is then spread normally.

Note that the effect of spreading where there are no child cells free to be changed, is that the result for some lower level locked or changed cells will be different to the locked value or change made. Effectively, higher level locks or changes are deemed to be 'more important' than lower level ones. Deliberately causing the circumstance where there are no free child cells can be a very useful technique especially when initializing data. For example, in a single calculation, a 'shape' can be applied to child cells, and then a 'total' to the parent cell, and the result (because the higher level change takes precedence) is that the parent total is spread across the children using the appropriate spreading technique, but according to the supplied shape.

Spreading methods

Just as different types of measures require different aggregation techniques, different types of measures require different spreading techniques. Measures that cannot be aggregated (that is, are of “recalc” type) are not usually spread at all (see The Spreading of Recalc Type Measures), though they may employ the replicate spreading technique. The default spreading method for a measure is set up as part of the definition of the measure. This is the spreading technique used for all changes to the measure, unless explicitly overridden on edit by the user.

The spreading methods that are supported by RPAS are listed here and described in the following sections:

- Proportional Spreading
- Replicate Spreading
- Even Spreading
- Delta Spreading
- PET and PST Spreading

Proportional Spreading

Proportional spreading is the most commonly used spreading technique once data has been initialized, and will be the default spreading method for most spreadable measures. In proportional spreading, all ‘children’ that are free to be changed are changed in the same proportion, so that their existing ratios to each other are maintained, and the required value for the parent is achieved. If proportional spreading is used for a measure that is not initialized (that is, its children all have the “naval”**)**, the children are assumed to all have the same weight, so the effect of the spreading is the same as the even spreading method.

Example:

Starting values: ChildA 10, ChildB 20, ChildC 30, ChildD 40, Parent 100.

Changes: Parent changed to 145, ChildA changed to 20, ChildB locked.

Resulting values: ChildA 20, ChildB 20, ChildC 45, ChildD 60, Parent 145.

Spreading Process: ChildA and ChildB are not free to be changed by spreading, since ChildA was explicitly changed and ChildB was locked. The required parent value of 145 must include 40 from ChildA and ChildB, and thus ChildC and ChildD must total 105. The previous total for ChildC and ChildD was 70, so their values must be changed by applying the multiplier of 105/70. Thus the new values for ChildC is 45, and for ChildD is 60. After aggregation, the result is that the parent has the value 145, as required, ChildA has the required 20, ChildB did not change, and the ratio of ChildC being 75% of ChildD is maintained.

This spreading method is not legal for measures with a recalc aggregation type.

Replicate Spreading

Replicate spreading is sometimes used when initializing data, especially for recalc type measures, and for measures with aggregation type such as average, minimum and maximum. It is unusual for it to be the default spreading method for any measure, but may be used by overriding the spread method on data entry. In replicate spreading, all child cells that are free to be changed are changed to the value of the parent cell. With replicate spreading there is no guarantee that after aggregation the value of the parent cell will be the value that was replicated (in fact it usually will not be): replicate spreading should simply be considered to be an indirect way of entering the same value into multiple child cells.

Example:

Starting values: ChildA 10, ChildB 20, ChildC 30, ChildD 40, Parent 100.

Changes: Parent changed to 145, ChildA changed to 20, ChildB locked.

Resulting values: ChildA 20, ChildB 20, ChildC 145, ChildD 145, Parent 330.

Spreading Process: ChildA and ChildB are not free to be changed by spreading, since ChildA was explicitly changed and ChildB was locked. The parent value of 145 is replicated to ChildC and ChildD. After aggregation, the result is that the parent has the value 330.

This spreading method is legal for measures with a recalc aggregation type.

Even Spreading

Even spreading is sometimes used when initializing data. It is unusual for it to be the default spreading method for any measure, but may be used by overriding the spread method on data entry. In even spreading, all child cells that are free to be changed are changed to the same value, which is the total for the parent cell for the free child cells, divided by the number of free child cells.

Example:

Starting values: ChildA 10, ChildB 20, ChildC 30, ChildD 40, Parent 100.

Changes: Parent changed to 145, ChildA changed to 20, ChildB locked.

Resulting values: ChildA 20, ChildB 20, ChildC 52.5, ChildD 52.5, Parent 145.

Spreading Process: ChildA and ChildB are not free to be changed by spreading, since ChildA was explicitly changed and ChildB was locked. The required parent value of 145 must include 40 from ChildA and ChildB, and thus ChildC and ChildD must total 105. This is spread evenly, thus the new values for ChildC and ChildD are both 52.5. After aggregation, the result is that the parent has the value 145, as required, ChildA has the required 20, ChildB did not change, and the remainder has been spread to ChildC and ChildD evenly.

This spreading method is not legal for measures with a recalc aggregation type.

Delta Spreading

Delta spreading is sometimes used when data is fully initialized (if it is used when the measure is not initialized, the effect will be the same as even spreading). It is unusual for it to be the default spreading method for any measure, but may be used by overriding the spread method on data entry. In delta spreading, all child cells that are free to be changed are changed such that the delta to the parent cell is spread evenly across those child cells.

Example:

Starting values: ChildA 10, ChildB 20, ChildC 30, ChildD 40, Parent 100.

Changes: Parent changed to 145, ChildA changed to 20, ChildB locked.

Resulting values: ChildA 20, ChildB 20, ChildC 47.5, ChildD 57.5, Parent 145.

Spreading Process: ChildA and ChildB are not free to be changed by spreading, since ChildA was explicitly changed and ChildB was locked. The required parent value of 145 must include 40 from ChildA and ChildB, and thus ChildC and ChildD must total 105. The previous total for ChildC and ChildD was 70, so the delta to the parent is 35. This delta is spread evenly across the children, so ChildC and ChildD are both increased by 17.5. Thus the new values for ChildC is 47.5, and for ChildD is 57.5. After aggregation, the result is that the parent has the value 145, as required, ChildA has the required 20, ChildB did not change, and the increase to the parent has been evenly divided between ChildC and ChildD.

This spreading method is not legal for measures with a recalc aggregation type.

PET and PST Spreading

PET (Period end total) and PST (period start total) are special spreading types to support measures with the PET or PST aggregation types, where the values of cells represent ‘snapshots’ at a period of time, rather than a total of events. Opening and closing stock (inventory) are typical examples of such measures, where the value for a month will be the value for the first (opening stock) or last (closing stock) week in the month, but values up non-time hierarchies will be produced by total aggregation.

PET and PST measures require special spreading. We anticipate a future enhancement to support spreading changes to such measures at aggregated time positions by spreading the effect of the change across all children of the time period. At present the PET and PST spread types change the first or last child only. Thus, at present, a change to closing stock for a month has exactly the same effect as a change to closing stock for the last week in the month.

Multi-Level Spreading

The RPAS engine allows changes to be made to a measure at multiple levels, all of which are dealt with in a single calculation. Because spreading requires parent-child relationships, and spreading is effected between the intersection that is changed and the base intersection for the measure, there is a requirement that all changes to be effected by a single calculation must fall on a single hierarchical roll-up. This is controlled by Hierarchical Protection Processing, which is described in the next section.

When there are changes at multiple levels, the spreading process fundamentally works 'bottoms-up', that is, lower level changes are actioned before higher level changes. The spreading algorithm starts with the lowest level in the hierarchical roll-up that has changes, and spreads each change at that level in turn.

The result of this process is that every child cell of a changed cell is no longer free to be changed, since if it was previously free to be changed it has now been recalculated by spreading. When all changes at a level have been actioned, the algorithm moves on to the next lowest level in the hierarchical roll-up that has changes, and continues in this manner until all changes have been actioned. If a higher level change overlaps a lower level change, the lower level changes are unaffected, because all child cells of the lower level change will not be free to be changed.

Example (using proportional spreading):

Starting values: jan 10, feb 15, mar 20, apr 25, may 30, jun 35, jul 40, aug 45, sep 50, oct 55, nov 60, dec 65. firsthalf 135, secondhalf 315, year 450.

Changes: year changed to 500, firsthalf changed to 150, jan changed to 15, feb changed to 20, mar locked, jul and aug changed to 50, sep locked.

Resulting values: jan 15, feb 20, mar 20, apr 26.39, may 31.67, jun 36.94, jul 50, aug 50, sep 50, oct 61.11, nov 66.67, dec 72.22. firsthalf 150, secondhalf 350, year 500.

Spreading process: The first change to be spread is the change to the first half to be 150. jan, feb and mar now total 55, so apr, may and jun must total 95. By proportional spreading the results are 26.39, 31.67 and 36.94. The second change to be spread is the 500 for the year. Only the months oct-dec are now free to be changed. The other months total 300, so oct-dec must total 200. By proportional spreading, the results are 61.11, 66.67, 72.22.

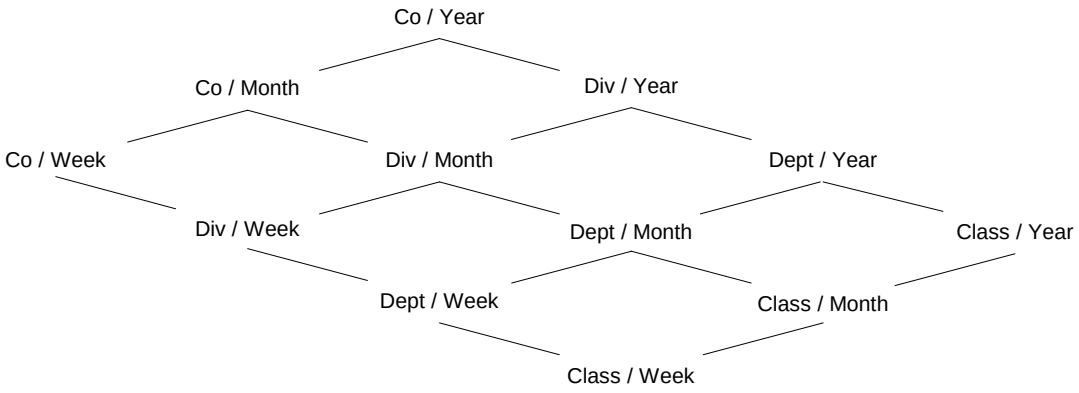
Hierarchical Protection Processing

Hierarchical Protection processing is a process that ensures that all changes made at aggregated levels fall on a single hierarchical roll-up, which is a prerequisite for the spreading process to function correctly. Hierarchical protection processing operates by protecting (that is, preventing direct manipulation) cells for intersections for combinations of dimensions that cannot lie on a single hierarchical roll-up with the changes already made.

In theory, since hierarchical protection processing is necessary to ensure the integrity of the spreading process, and each measure is individually spread, hierarchical protection processing could operate independently for each measure. Having the manipulatable measures varying from intersection to intersection would probably cause considerable confusion to the users, and would also make implementing a consistent methodology difficult, so for simplicity, hierarchical protection processing operates on all measures.

Since an OLAP type model has multiple hierarchies and spreading operates from the cell that has been changed to all child cells at the base intersection, hierarchical protection processing must operate across multiple hierarchies. A single hierarchy may have multiple roll-ups, and they are also considered. Whenever a change or a lock is made to an intersection for a new combination of dimensions, the calculation engine checks all other combinations of dimensions, and protects those that cannot be on the same hierarchical roll-up as changes already made. It does this by considering a 'cross multiplication' of hierarchical roll-ups across all the hierarchies.

A simple example will clarify the process. Consider the matrix of 'cross multiplied' dimension combinations that result when there is a 2-dimensional product by time model with the dimensions Co/Div/Dept/Class and Year/Month/Week, shown here schematically with parent-child relationships:



Other than at the top of a hierarchical roll-up, each combination of dimensions has two parent combinations: one per hierarchy with the next highest dimension, thus class/week has parents of class/month and department/week. Note that for the sake of simplicity the picture does display the roll-up of the "all" dimension (all products, all time periods). All spreading is from the changed level to the base intersection (class/week in this example). Consider a change at an intersection of Div/Month. We know that the spreading hierarchical roll-up must be a path from the top (Co/Year) to the bottom (Class/Week) that passes through Div/Month. There are six such paths, none of which go through the combinations Co/Week, Dept/Year or Class/Year, so those combinations of dimensions are all protected. If the next change is at an intersection of Class/Month, then Div/Week and Dept/Week are similarly protected.

Note that hierarchical protection processing always reflects the current set of locks and changes. RPAS allows cells that have been changed or locked to be unchanged or unlocked before the calculation is initiated. If an unchange or unlock removes the last change or lock for a combination of dimensions, some other combinations of dimensions that were previously protected could become unprotected. In our example, above, if the first change to a Div/Month were now unchanged, so that the only change outstanding is at Class/Month, then Dept/Year and Class/Year would now become manipulatable again (though Co/Week, Div/Week and Dept/Week would still be protected).

Further note that non-conforming measures (see Non-Conforming Expressions) may lead to hierarchical protection processing that may appear to be over protective. When considering hierarchical protection processing, all measures have their 'scope' expanded to include the 'all' level of all hierarchies that they are not dimensioned on. The implications of this are that, for example, a change to a measure with a base intersection of 'class' (which is interpreted as meaning 'class/all') would prevent the manipulation of a measure with a base intersection of 'year' (which is interpreted as 'all/year').

The Spreading of Recalc type Measures

Measures that are of recalc type are not usually spread by any spreading technique. Spreading techniques typically rely on the existing relationships between a parent and its children in the spreading process, and in a recalc measure, those relationships cannot be relied upon, as they are not 'weighted'. Spreading of changes to a recalc measure is therefore indirect, and applying what is referred to as a 'mapping rule' effects the change.

A mapping rule is a rule, with two or more expressions, that calculate a spreadable measure from the changed value of a recalc measure and other measures. The selected expression for the rule is evaluated at the level of the change to the recalc measure. It results in a changed value of a spreadable measure, and if this is above the base intersection for that measure it is spread normally, using its default spreading method. A change to a recalc measure should therefore be considered to be an indirect change to the spreadable measure that it is mapped to.

The only constraint on the manipulability of a normally spreadable measure is through protection processing, which prevents the manipulation of measures that will be calculated. For a recalc measure, in addition to this, the measure must have a mapping rule: without a mapping rule the measure cannot be manipulated.

Note that a recalc measures can only appear in a single rule in a rule group. The RPAS calculation engine therefore knows that rule contains the recalc expression for the recalc measure. If there are other expressions in the rule, they may be used as mapping expressions, thus allowing the recalc measure to be manipulatable. If there is just a single expression in the rule that calculates the recalc measures, the recalc measure is non-manipulatable through normal protection processing.

Chapter 5 – Expressions, Rules and Rule Groups

Introduction

Measures are related together through algorithmic relationships. Thus, for example, the sales value may be the sales units multiplied by the selling price. In RPAS, these relationships are specified through expressions, which are grouped for usage into rules and rule groups.

It is a fundamental principle in RPAS that the calculation engine maintains and guarantees the integrity of all the active relationships between cells at all times. Hierarchical relationships are maintained through the processes of spreading (see Chapter 4 - Spreading) and aggregation (see Chapter 3 - Aggregation). Relationships between measures are maintained by the evaluation of expressions. One of the great strengths of RPAS is that both of these types of relationships are automatically maintained in a non-procedural manner: the model configurer does not have to write code to determine what is calculated, or how, or in what sequence: all that is required is the definition of the relationships themselves, although, as we shall see, the configurer does provide prioritization information to guide the calculation engine when there is a choice of calculation paths.

In an RPAS model, all cell values at aggregated level can be determined by aggregation from the cells at the base intersection. Although the description that follows is a simplification (a more detailed and precise description is given in Chapter 6 – The Calculation Engine), a basic understanding of the working of the calculation process, and the importance of expressions, can be gained by understanding the interconnection between three fundamental processes: spreading, ‘bottom level’ expression evaluation and aggregation. Changes to measures at aggregated levels are spread down to their base intersections. Here, the calculation engine enforces all measure relationships that are no longer guaranteed to be true (because the cell values of one or more of the measures in the relationship have changed either directly, through spreading or by prior evaluation of an expression) by evaluating an expression. When base intersection calculation is complete, all measures at the base intersection that have been changed (directly, by spreading or by expression evaluation) are aggregated to re-impose cell integrity.

Expressions

Expressions are the basis of all calculations of the relationships between measures, and are evaluated by the calculation engine during a calculation. Expressions are written in a syntax that allows for the calculation of one or more measures from other measures, constants and parameters, using standard arithmetical functions and a rich set of mathematical, technical and business functions. Expressions are therefore an algorithmic statement of a relationship between measures. Details of the allowable syntax for expressions are given in a separate document.

Rules

An expression describes the relationship between measures in a way that causes a measure to be calculated through the expression. An expression may be said to ‘solve’ the relationship for the measure that is calculated through the expression. In some cases, there may be business methodology reasons for wanting more than one of the measures in a relationship to be calculable or solvable through that relationship. To support this requirement, RPAS has the concept of a rule, which consists of one or more expressions that describe the same relationship between measures, but that solve for different measures. All the expressions in a rule should use the same measures, and must have a different target measure (that is, the measure on the LHS of the expression that is calculated by the expression).

Where a rule has multiple expressions, those expressions are given a priority sequence, to help the calculation engine to select a calculation path that follows business priorities. Consider the rule that relates together sales value, sales units and sales price. Let us assume that there are three expressions in this rule, that is, that each of the measures involved in the rule may be ‘solved’ through the rule. If a user makes a change to, say, sales value, it should be clear that the calculation engine could enforce the mutual integrity of all the cells by holding the sales price constant and recalculating a new sales units. It could also achieve the same end by keeping the sales units constant, and recalculating the sales price. Both approaches are mathematically valid, and produce a consistent result, with complete data integrity. However, it is likely that one approach makes more ‘business sense’ than the other (in this case, most businesses in most circumstances would want the price to remain constant and have the units recalculated), and the prioritization of the expressions in the rule provides this information to the calculation engine. Considerable care should be taken in the design of models to ensure that appropriate expression priorities are established.

In essence, the calculation engine, when given a choice, will always select the highest priority expression in the rule that is available to be selected. In this example, therefore, the expression that calculates sales units would have a higher priority than the expression that calculates sales price. Similar consideration of the desired effect of a change to sales units will probably lead to a conclusion that the expression that calculates sales value would also have a higher priority than the expression that calculates sales price. But what of the relative priority of the expressions to calculate sales value and sales units? What is the ‘business priority’ for those expressions? That may vary from implementation to implementation. Indeed, it may even vary from one type of plan to another in the same implementation: for a financial merchandise plan the preferred behavior may be that a change to just sales price causes a recalculation of the sales units, whereas in a unit-oriented lower level plan the preferred behavior may be that a change to just sales price causes a recalculation of sales value.

The same measure may appear in multiple rules. Indeed, this will frequently be necessary, since the same measure can be involved in a number of different relationships with other measures. For example, there may be a relationship between sales value, sales units and sales price. Sales value may also be involved in another relationship with closing stock and a cover value, and yet another with opening stock, receipts, markdowns and closing stock.

Rule Groups

It is most unusual for a model to only require a single rule. In most cases, there will be a collection of relationships between measures that must be maintained. In RPAS, a Rule Group is a collection of rules that are treated as a unit by the calculation engine, with the integrity of all the rules in the rule group being maintained together. The calculation engine always has one (and only one) active rule group. Even if all that is required is a single expression, that single expression will be in a rule, and that single rule will be in a rule group. The process by which the integrity of all the rules in a rule group is maintained is quite complex, and is described in detail in Chapter 6 – The Calculation Cycle.

Rules within a rule group are given a priority. The calculation engine uses this to select a calculation path that follows business priorities, by using rule priorities to determine which rule to enforce when there is a choice to be made. This is described in more detail in Chapter 6 – The Calculation Engine.

Within a system as a whole, there may be many rules defined. The validation of rules is performed in isolation, but rules within a rule group are also validated in the context of all the other rules in the rule group. This can mean that a rule that is perfectly valid syntactically, is not valid within a particular rule group. Rule group validations include:

Each rule in a rule group must represent a completely different measure relationship. Therefore no two rules in a rule group may use exactly the same collection of measures, and neither may one rule group use a collection of measures that is a sub-set of the collection of measures in another rule.

There must be an expression that calculates each recalc measure

Any measure that is on the LHS of the only expression in a rule may not be on the LHS of any other expression.

Although there may only be one active rule group at any time, RPAS allows for the definition of multiple rule groups to satisfy different calculation requirements. Rule groups may be one of four different types: load, calculate, refresh and commit. The RPAS application automatically uses the load rule group when loading data into the workbook, the refresh rule group to refresh data and the commit rule group when committing data to the domain. These rule groups are perfectly ‘normal’, so although they will typically include many rules that use the master modifier to load or commit data, they may also have other rules. It is perfectly possible, for example, to commit data to the domain for a measure that does not exist in the workbook, merely by including the appropriate rule to calculate the measure (with the master modifier) in the commit rule group. Similarly, a measure may be loaded into a workbook that does not exist in the domain, by including an appropriate rule to calculate the measure in the load rule group.

RPAS supports multiple calculation rule groups. Menu options may be configured to allow the user to select a different calculation rule group. RPAS ensures a smooth transition from one calc rule group to another.

Rule Group Transitions

Although only a single rule group may be active at any time, RPAS supports the transition from one rule group to another. The calculation engine ensures the integrity of measure relationships at all times, so this process is not merely a case of switching from one rule group to another, as there is no guarantee that the integrity of the rules in the rule group being transitioned into have been maintained.

RPAS makes a worst case assumption when transitioning rule groups. Any rule that is in both the old and new rule groups is assumed to have its integrity maintained. Any other rule is assumed to be potentially wrong, and so is flagged as 'affected'. A normal calculation is then initiated, with expressions to be evaluated determined by the usual process (see Chapter 6 – The Calculation Cycle). All affected rules will therefore have their integrity imposed by the evaluation of an expression, and 'knock-on' effects may cause some rules that occur in both the old and new rule groups to also be evaluated. Since all base intersections must be calculated during rule group transition, a large or complex rule group transition is likely to take longer than a normal calculate.

Note that there are circumstances when automatic rule group transitions occur:

- On data loading.
  Data is loaded using the load rule group. This will typically load measures by calculating them from the data values held on the domain using the master modifier, but may also calculate other measures that are not explicitly loaded. When the load is complete, the system will automatically transition to the calculate rule group
- On data refreshing.
  Data refreshing causes some measures to be updated from values held on the domain. Refreshing uses the refresh rule group, but there is no real transition: the measures that are affected by the refreshed measures are treated as affected in the calculate rule group, and a normal calculate of that rule group follows. Effectively, data refreshing causes a calculation using the calculate rule group as if the cells that were refreshed were directly changed by the user.
- On data committing.
  There is a normal transition from the current calculate rule group to the commit rule group. This will typically commit measures by calculating them on the domain by using the master modifier. When transitioning back from the commit rule group to the calculate rule group, there is an assumption that only measures with a "master" modifier have changed and therefore no transition is required.

Chapter 6 – The Calculation Cycle

Introduction

The calculation cycle always uses the current active rule group. It is a comprehensive process that uses non-procedural hierarchical cell relationships and expression-driven measure relationships from the rule group, together with details of the locks and changes to individual cells to determine and then execute the required actions to apply the effect of the changes and locks. This section describes how the calculation engine determines what to calculate, how to calculate it, and in what order to perform the calculations. Details of processes such as spreading, aggregation and the evaluation of expressions are given elsewhere.

There are 4 distinct stages of the calculation cycle to consider. The first stage, protection processing occurs whilst the user is making changes to cell values, and protects those measures that the user cannot change, either because they are never changeable, or because changes already made force them to be calculated. In the second stage, the engine decides what expressions will be evaluated, and the third stage determines the sequence of calculation. The final stage is the physical process of doing the calculation.

Note that the calculation cycle can operate in one of two modes: “full” and “incremental”. In “full” mode, all cells for the measures being evaluated are assumed to need to be calculated. This mode is used when calculating in batch, and in all rule group transitions. “Incremental” mode is used when manipulating cells in an online session, and only those cells that are directly or indirectly affected by user edits are calculated.

Protection processing

Protection Processing Overview

The calculation engine guarantees the integrity of all relationships and, other than in exceptional circumstances, ensures that the value for a cell changed by a user, after calculation, is the value entered by the user. In order to ensure this, it must prevent the user from making changes to any cells where it would be unable to guarantee that integrity. The process that achieves this is called protection processing.

A measure may only be manipulated when the calculation engine is able to change other cells, by spreading and/or evaluation of an expression to enforce the integrity of relationships. A measure that is not used in any rules may only be manipulated if it has a spreading technique other than recalc.

It is a basic principle of the calculation engine that a measure that is changed (or locked) cannot also be recalculated by evaluating an expression (it will, of course, be aggregated, which in the case of a recalc measure does involve the evaluation of an expression). A measure that is to be evaluated can only be evaluated using one expression, as there is no guarantee that the same result would be produced from two expressions that, by definition, represent different measure relationships. It is also a basic principle that any measure relationship (that is, rule) must be evaluated when one or more of the measures in that relationship have been changed, since this is the only way to enforce the integrity of the rule relationship. Therefore a rule where there is just a single expression means that the measure calculated by that expression cannot be changed by the user, since there is no expression to evaluate to effect that change for that measure relationship. Such measures can never be manipulated in any rule group that uses the rule, and are protected.

Where a rule has two expressions, the two measures that are calculated by those expressions are both available to be manipulated. However, as soon as one measure is manipulated by the user, we know that the expression that calculates the other measure must be evaluated, as one of the expressions in the rule has to be evaluated, and we cannot evaluate the expression that calculates the measure that was changed. The expression that must be calculated is said to be *forced*, and the measure that it calculates is protected, to prevent the user from changing it. Of course, that measure may be involved in more than one rule, and in the other rules in which it is used it must be treated as if the user changed it. This so-called *knock-on* effect may force further measures to be forced and protected. Evaluating these effects is the basic technique of protection processing.

Protection processing occurs continuously while the user is editing cells. Each time the ‘changed state’ of a measure changes (that is, the measure goes from not having changes or locks to having them, or vice versa) protection processing evaluates the measures that should now be protected. Protection processing always reflects the current set of locks and changes. RPAS allows cells that have been changed or locked to be unchanged or unlocked before the calculation is initiated. If an unchange or unlock removes the last change or lock for the measure, so that the measure is no longer affected, then protection processing is quite likely to find that other measures which were previously forced are no longer forced, and thus are free to be manipulated, and must be unprotected.

Protection Processing Details

The following terms are used in this description:

An *affected measure* is a measure that has been changed by the user, is locked by the user, or is forced.

An *affected rule* is a rule that contains one or more affected measures.

A *free measure* is a measure that is not affected.

A *free expression* is an expression for an affected rule that calculates a free measure.

A *forced rule* is an affected rule that has only one free expression.

A *forced measure* is the measure calculated by the free expression in a forced rule

Any measure that is the measure on the LHS of the only expression in a rule is protected.

Protection processing considers each affected rule in turn. Each affected rule will be in one of three conditions:

Affected rules that have previously been forced are ignored, as they are already fully dealt with.

If the affected rule has two or more free expressions it is ignored, as nothing is forced.

If the affected rule has just a single free expression, then it becomes a forced rule, and the measure calculated by the free expression is forced and becomes an affected measure. The forced measure is protected. All rules that use the forced measure become affected.

When a new measure becomes forced, checking of affected rules begins again. When all affected rules have been considered without any further measures becoming forced, the first stage of protection processing is complete.

The second stage of protection processing is to perform ‘look ahead’ protection processing. Look ahead protection processing ensures that all measures that are visible in windows and still unprotected can indeed be manipulated. It does this by performing the protection processing that would occur if the measure were changed, including ensuring that there is a solution to the processes of determining what to calculate, and ordering the calculation. If these processes fail to find a solution, the process that determines what to calculate will repeatedly back up the decision tree and select a different expression looking for a solution. If there is no such solution, the measure that was being checked is protected. In this manner, the calculation engine assures that there will always be a method to calculate the effects of all changes that it allows the user to make.

Note that this is a somewhat simplified description of protection processing, as it ignores the implications of ‘cycle groups’ (see Cycle Groups) and ‘synchronized measures’ (see Chapter 7 – Synchronized Measures).

Determining what to calculate

The protection processing process has established what measures are forced given the current set of changes and locks. When a calculate is issued, those forced measures will be calculated (using the forced expressions). However, there may be affected rules that are not forced. For those affected rules, we know that an expression must be evaluated (otherwise the integrity of the rule is compromised), and the calculation engine must select one of the expressions.

When there are one or more affected rules that are not forced, the highest priority affected rule is selected. From this selected rule, the highest priority free expression is selected, and will be evaluated. These are the only uses to which the rule and expression priorities are put. The measure that is calculated by the selected expression is then treated as forced, and *knock-on* effects considered, which are likely to cause other rules and measures to become forced. If at the end of this process there are still affected rules that are not forced, the process is repeated until there are no affected rules that are not forced. At this point, any rule that is not affected does not need to be evaluated, and an expression has been forced or selected for all rules that need to be evaluated to ensure the integrity of all measures.

Determining the calculation sequence

The previous section has established what expressions to evaluate, but not the sequence in which they are evaluated. The sequence of evaluation of expressions is driven by the status of the RHS measures. All normally spreadable (that is, not of recalc type) measures that are changed can be spread and then aggregated at the start of the calculation cycle. Normally spreadable measures that have been changed, and those measures that will not change at all during the calculate are therefore considered to be ‘complete’. Any expression whose RHS measures are all complete may be evaluated. If the expression is a mapping rule for a recalc measure, the changed values for the mapped spreadable measure will be calculated for all changed cells. That measure may then be spread and aggregated normally. If the expression is for normal ‘base intersection’ evaluation, the measure will be calculated, and may then be aggregated. In either case, the calculated measure is now ‘complete’ which may make further expressions available to be evaluated. The process continues until all expressions have been sequenced.

When determining the sequence of calculation, the evaluation of expressions is intermingled with spreading and aggregation. In very trivial cases, where all changed measures are spreadable, there will be a phase where a number of measures are spread, a second phase where a number of measures are calculated at the base intersection, and then a final phase where a number of measures are aggregated. However, if any recalc measures have been changed at aggregated levels, the ‘mapping rule’ cannot be applied until any affected measures on the RHS of the expression have been spread or calculated, and then aggregated.

Note that this is a somewhat simplified description of the calculation sequence, since, for efficiency purposes, groups of measures that must be spread, aggregated or evaluated are batched together, so that an individual measure is not necessarily spread, aggregated or evaluated as soon as it is available for that action. However, it is always spread, aggregated or evaluated before the results of that action are required for another step. Also, expressions are not evaluated for all cells, but only for those cells where one or more of the measures on the RHS of the expression have changed. There are similar efficiencies in aggregation, to avoid the redundant re-aggregation of cells that will not have changed.

Cycle Groups

Introduction

This section describes the cycle group feature of the RPAS calculation engine. This feature enables relationships between measures that have cyclic dependencies from the measure perspective (that is, there appears to be a 'deadly embrace' where each measure depends upon the other), but are actually acyclic when the time dimension of these measures is considered. Without this feature such relationships could not be set up, as the calculation engine would be unable to find a calculation sequence that enabled both measures to be calculated.

A common application of cycle groups can be found in inventory calculations that involve measures such as beginning of period (BOP) and end of period (EOP). It is typical that EOP is calculated in some way from BOP for the same period, and, other than in the very first period, that the BOP of a period is equal to the EOP of the previous period. Since BOP is dependent on EOP and EOP is dependent on BOP, a cycle exists from a measure perspective. However, when the time dimension is considered, calculations can be performed in an acyclic fashion. In this example, if EOP for the first period is calculated first, then BOP for the second period can be calculated. This in turn allows EOP for the second period to be calculated, and so on.

Cycle Breaking Functions

Some of the functions supported by the calculation engine have special cycle breaking logic associated with them. These include functions that reference previous time periods and functions that reference future time periods. When these functions are used, the calculation engine automatically determines when measure dependencies that appear to be cyclic are in fact acyclic when the calculations are performed one period at a time. The lag and lead functions are examples of cycle breaking functions.

Cycle Group Evaluation

A cycle group is a group of expressions that the calculation engine must calculate together to avoid cyclic dependencies. If the apparent cycle is broken by a function that looks backwards in the time dimension, such as lag, then calculation proceeds with the first time period of each expression in sequence, followed by the second time period of each expression in sequence, and continues until all time periods have been calculated. If the apparent cycle is broken by a function that looks forwards in the time dimension, such as lead, then calculation proceeds in reverse order starting with the last time period.

Cycle Group Example

Consider the following measures:

BOP: beginning of period inventory

EOP: end of period inventory

OS: opening stock (that is, the opening inventory for the first period in the plan horizon)

SLS: sales

RCP: receipts

And consider the following rules:

R1: BOP = if(current == first, OS, lag(EOP))

R2: EOP = BOP – SLS + RCP

RCP = EOP – BOP + SLS

When the measure RCP is edited, R2 is affected and the EOP expression in this rule is forced. Then rule R1 is affected and the BOP expression in this rule is forced.

Since the calculation of EOP requires BOP and the calculation of BOP requires EOP, a cycle is detected which contains both of the selected expressions. This is a valid cycle group since the calculation of BOP is dependent on the lag of EOP. Thus the cycle can be broken and the intra-cycle ordering results in the BOP expression being evaluated first and EOP expression second.

The evaluation of this cycle group involves the calculation of the first time period of BOP, followed by the first time period of EOP, followed by the second time period of BOP, followed by the second time period of EOP, and continues until all time periods have been calculated.

Chapter 7 – Synchronized Measures

Synchronized Measures Overview

Measure synchronization is an RPAS user interface and calculation engine feature. It enables measures that are very closely related to be represented in the UI in a more intuitive manner. It can give the appearance of a cell edit or lock affecting two different measures. However, from a calculation perspective, the cell edit or lock is only applied to one of these measures. A common application of synchronized measures is to allow BOP and EOP to be synchronized. From a business logic perspective, the BOP in one period and the EOP in the previous period are the same thing, and measure synchronization means that even before calculation an edit or lock of BOP in one period also appears on the UI as an edit or lock of EOP for the previous period, and vice versa.

To accomplish measure synchronization, a measure is defined with a synchronized view type and a list of synchronized source measures. The measure defined with these attributes is called the synchronized target measure. Synchronized target measures may be edited, but any such edits are actually treated as edits to the underlying synchronized source measures. Protection processing is performed on the synchronized source measures. The protection state of the target measure is then derived from that of the source measures. An edit to one of the source measures is also reflected in the display of the target measure.

The synchronized view types that can be used to define synchronized target measures are as follows:

1 sync_first_lag: The first period of the target measure is synchronized with the first source measure, and periods 2..N of the target measure are synchronized with periods 1..N-1 of the second source measure, where N represents the last period. The first source measure will not have a time dimension. This view type is particularly useful for defining BOP target measures. Here the first source measure would be an opening inventory, and the second source measure would be the EOP.

2 sync_lead_last: Periods 1..N-1 of the target measure are synchronized with periods 2..N of the first source measure and period N of the target measure is synchronized with the second source measure, where N represents the last period. The second source measure will not have a time dimension. This view type is particularly useful for defining EOP target measures. Here the first source measure would be BOP, and the second source measure would be a closing inventory.

3 sync_first: The target measure is synchronized with the first period of source measure. The target measure will not have a time dimension. This view type is particularly useful when defining OS target measures.

4 sync_ last: The target measure is synchronized with last period of the source measure. The target measure will not have a time dimension. This view type is particularly useful when defining CS target measures.

Synchronized Inventory Examples

Consider the following measures:

BOP: beginning of period inventory

EOP: end of period inventory

OS: opening stock (that is, the opening inventory for the first period in the plan horizon)

CS: closing stock (that is, the closing inventory for the last period in the plan horizon)

SLS: sales

RCP: receipts

And consider the following rules:

R1: BOP = if(current == first, OS, lag(EOP))

R2: EOP = BOP – SLS + RCP

RCP = EOP – BOP + SLS

BOP can be defined as a synchronized measure constructed from the OS and EOP measures, with the *sync_first_lag* type. Only one expression in the rule group may have BOP on the LHS. This expression is used to construct views of BOP and is merged with expressions that require BOP on the RHS.

When edits or locks are made to BOP, it is the underlying values of OS or EOP that are actually changed or locked. Thus, even though rule R1 has only one expression and this expression calculates BOP, the BOP measure is not protected by protection processing, because of the measure synchronization. The BOP measure is only protected when the underlying OS or EOP measures are protected, so the first period is protected when OS is protected and the remaining periods are protected when EOP is protected.

In this example, a CS measure is not required for calculation purposes, but it may be desired for viewing and editing purposes. For example, a window containing only OS and CS but not BOP nor EOP may be wanted. In this case, the CS measure should be defined as a synchronized measure with type *sync_last* and the synchronized source measure would be EOP. As a result, an edit to CS becomes an edit to the last period of EOP.

Chapter 8 – Elapsed Periods

Many RPAS models will cover a time horizon where some of the time periods are in the past, and other periods are in the future. There is special logic for handling such models, in particular the manner in which time periods that are in the past (referred to as elapsed periods) affect spreading and the manipulability of measures. The bottom level time period that is the last elapsed period is set through a rule.

RPAS assumes that time periods that are elapsed contain ‘actuals’, and that these actuals should not be manipulatable. Therefore, all measures are non-manipulatable during elapsed periods. For positions at aggregated levels in a time hierarchy, the position is only elapsed when the last bottom level time period descended from it is elapsed.

Measures that represent BOP data have special handling. From a business perspective, the BOP in a period is the same as the EOP in the previous period. Therefore, when an EOP value is elapsed, the following BOP value must also be elapsed. In RPAS, all measures with a default spread method of PST (see Spreading Methods) are assumed to be ‘BOP type’ measures, and are protected for all elapsed periods, and for the first non-elapsed period. There is also special handling of these measures for aggregated time positions. These are treated as elapsed, and are therefore protected, when the first bottom level time period descended from it is elapsed.

Elapsed periods and Spreading

The user may make changes to measures at high levels of aggregation whose scope covers elapsed periods. When such changes are spread, values for elapsed periods are never changed: they therefore may be considered to be of higher priority than normal cell locks.

Chapter 9 – Non-Conforming Expressions

Introduction

One of the strengths of the RPAS calculation engine is that a workbook may contain measures with different ‘scopes’: the size and shape of the ‘multidimensional cube’ of data may vary by measure. Any two given measures in a workbook may have scopes that align exactly (for example, both measures have a base intersection of SKU/Store/Week), or where one is a subset of the other (e.g. one has a base intersection of SKU/Store/Week and the other is at Class/Week). There can also be circumstances where each measure includes a hierarchy in its base intersection that the other dimension does not use (for example, one has a base intersection of Class/Week and the other is Store/Week). In extreme circumstances, the scopes of two measures may have no point of overlap at all (for example, one has a base intersection of Class and the other Store).

It is the scope of the measure on the left hand side of an expression (referred to as the LHS measure) that determines the cells that must be calculated by the expression, though that scope may be changed by the use of a modifier such as level. Where one or more measures on the right hand side of an expression (referred to as RHS measures) have a scope that is different (in any way) to the scope of the LHS measure, the expression is deemed to be ‘non-conforming’. There is special logic to handle the calculation of non-conforming expressions, which depends on the type of nonconformity.

Although not explicitly declared, there is a single logical ‘All’ position at the top of every hierarchy. When considering non-conformity, any measure that is not explicitly dimensioned on a hierarchy, is implicitly assumed to be dimensioned on the ‘All’ dimension of that hierarchy, and thus all data values are assumed to be for the ‘All’ position. This concept is key to the understanding of the handling of non-conforming expressions.

Handling of Non-conforming Expressions

When the concept of the ‘All’ position is understood, all expressions can be considered to contain measures that use exactly the same hierarchies, the only potential differences between them are the ‘bottom levels’ (dimensions in the base intersection). Thus for handling non-conformity, only three cases need to be considered, for each hierarchy:

- RHS same:
  In this case the RHS measure has the same bottom level as the LHS measure. The RHS measure is ‘conforming’ for that hierarchy, and values for the RHS measure are taken from the same position as the position being calculated for the LHS measure.
- RHS higher:
  In this case the RHS measure has a higher bottom level than the LHS measure. The RHS measure is ‘non-conforming’ for that hierarchy. The values for the RHS measure for the position being calculated are assumed to be the same as the value of the RHS measure for the position in its bottom dimension that is the parent (ancestor) of the position being calculated. Effectively, it can be considered that the value of the measure has been ‘replicated’ down the hierarchy to the required level.
- RHS lower:
  In this case the RHS measure has a lower bottom level than the LHS measure. The RHS measure is ‘non-conforming’ for that hierarchy, but because the scope of the RHS measure includes the bottom level for the LHS measure, values for the RHS measure are taken from the same position as the position being calculated for the LHS measure.

(The conceptual case where the measures have scopes that do not overlap, because they have base intersections in a hierarchy that are for dimensions that are up different ‘branches’ of the hierarchy, fails rule validation.)

Examples

These examples all use the simple expression a = b + c

Example 1

a has a base intersection of SKU/Store/Week

b has a base intersection of SKU/Week

c has a base intersection of SKU/Region/Week

For each SKU/Store/Week, a is calculated from the value of b at SKU/Week (that is, it is assumed that the value of b is the same for all positions in the location hierarchy) and the value of c at SKU/Region/Week, for the Region the Store belongs in. If 'replication' of c from the Region level is not appropriate, the rule writer can simulate other 'spreading' techniques by the use of functions and modifiers such as **count** and **level**. For example, the **count** function may be used to determine the number of Stores in the Region, and so dividing the measure c by that count will simulate 'even' spreading.

Example 2

a has a base intersection of SKU/Store/Week

b has a base intersection of SKU/Week

c has a base intersection of SKU

For each SKU/Store/Week, a is calculated from the value of b at SKU/Week (that is, it is assumed that the value of b is the same for all positions in the location hierarchy) and the value of c at SKU (that is, it is assumed that the value of b is the same for all positions in the location hierarchy and time hierarchy). Note that an alternative approach, if required, would be to use a **level** modifier on the measure a, so that it is calculated at, say, SKU/Week, and then spread down to SKU/Store/Week, using the existing store participations to the measure a.

Example 3

a has a base intersection of SKU/Week

b has a base intersection of SKU/Store/Week

c has a base intersection of SKU/Region/Week

For each SKU/Week, a is calculated from the value of b and c at SKU/'All'/Week.