Paper 020-2009

Custom Calculations Are a Snap When Using MDX and OLAP

Brian Miles, Zencos, Cary, NC
Ben Zenick, Zencos, Cary, NC

ABSTRACT

This paper will address the use of MDX to develop custom calculations in OLAP that aren't easy to implement or apply in reports based on standard tables. Have you ever attempted to apply if/else logic directly into an OLAP cube that will allow the engine to make certain decisions? Have you had difficulty applying Advanced Time Period Manipulation in an OLAP structure; like Year to Date or Rolling Year to Dates? Can you apply different calculations to a measure based on a certain level in a hierarchy? How can you deliver Non-Additive calculations in OLAP? This paper will address these questions as well as the following information custom options for cube building, using external aggregation tables, advanced MDX topics, and Cube display options.

INTRODUCTION

With version 9 of SAS® came with a completely new OLAP framework that would give developers the ability to customize OLAP report data in ways never before possible from within SAS. Multidimensional Expression Language (MDX) is the foundation of v9 SAS cubes and is responsible for the increased flexibility and functionality. There are more ways to access, manipulate, and display cubes than previously possible. This paper will focus on these new concepts to SAS OLAP and will guide you in the utilization of these new tools.

The main driver behind the increased flexibility described above is the "Define Member" and "Define Set" statements. These statements, which are available through the new OLAP framework, enable calculations to be processed at runtime. These calculations are built as part of the Proc OLAP code (manually or via OLAP Cube Studio or DI Studio Calculated Members tool) and can be referenced by reporting tools in the same selectable manner as the rest of the measures in the cube. The ability to add custom runtime logic to a cube through Defined Members and Sets allows users the flexibility to use OLAP in ways that previously were limitations. This flexibility also enables the Members and Sets to be utilized in 3 different scopes within your OLAP and Reporting environment. These scopes are defined as follows:

Query Scope – The defined member/measure is accessible during the query only. These members are implemented by using the "WITH Member/Set" keyword during an MDX Query.

Session Scope – The defined member/measure is accessible for the session. A session is synonymous with a connection to the OLAP Server. Session scope members can be implemented using the "Create Session Member/Set" keywords during an MDX Query.

Global Scope – The defined member/measure is associated to the cube metadata and will exist until the associated cube is deleted. Using the "Define" keyword during Proc OLAP or the OLAP Cube Studio/DI Studio Calculated Members tool will utilize this method.

DEFINED MEMBERS

Defined Members were commonly referred to as computed columns in version 8 of SAS. The statements can be defined within the Proc OLAP code and can include some standard SAS options like the "format" option. All of the SAS and Custom OLAP client tools will treat defined members in the same fashion as all of the other measures that were defined by "Measure" statements.

Example of a basic defined member during cube creation is:

```
DEFINE MEMBER "[SUGI].[Measures].[Difference]" AS
'([Measures].[Actual] - [Measures].[Predict]), format_string="10.1"';
```

Example of a basic defined member during execution of a query is:

```
WITH MEMBER "[Measures].[Difference]" AS
'([Measures].[Actual] - [Measures].[Predict]), format_string="10.1"';
```

Defined Members can also use other defined members in their calculations. This example uses the Difference measure that was defined in the prior statement.

```
DEFINE MEMBER "[SUGI].[Measures].[Difference*2]" AS
'([Measures].[Difference]*2), format_string="10.1"';
```

Members can also be removed from metadata without rebuilding the cube by issuing an “Undefine” statement.

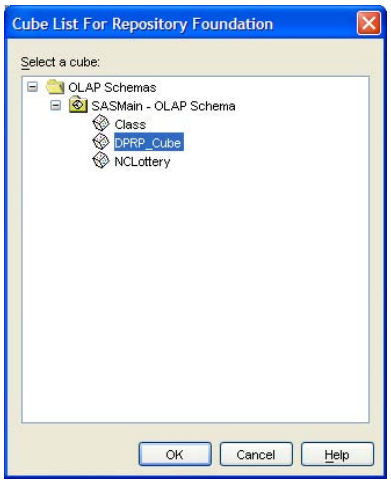
Below is an example of the MDX code that would be used to access the new measures.

```
SELECT
{[Measures].[Difference],[Measures].[Difference*2]} ON COLUMNS,
{[Products].[Product].Members} ON ROWS
FROM SUGI
```

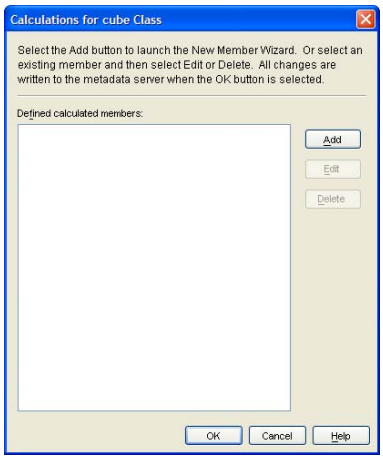
USING OLAP CUBE STUDIO/DI STUDIO CALCULATED MEMBERS TOOL TO DEFINE MEMBERS

The Calculated members tool in OLAP Cube Studio and DI Studio (Accessed by clicking Tools → Calculated Members) provides an easy to use interface for developing simple MDX calculations.

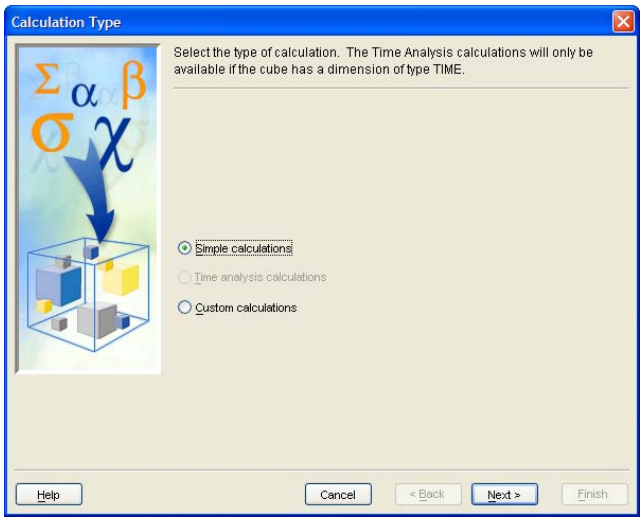
Once the interface opens, select a cube to develop the members for.



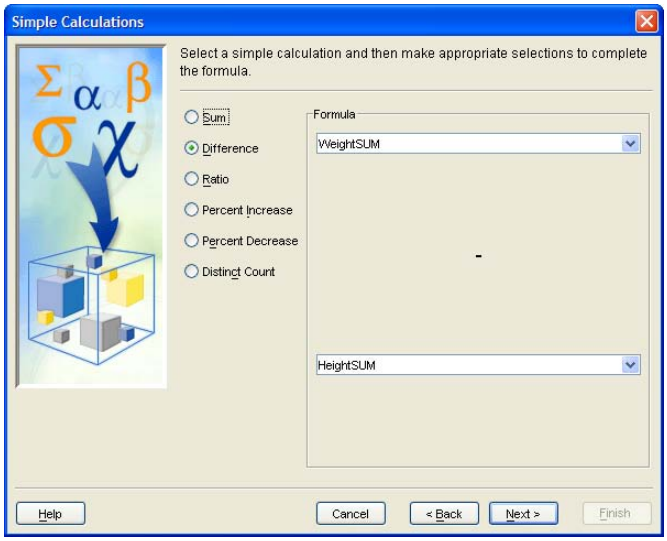
2) Click on the “Add” button to add a new member.



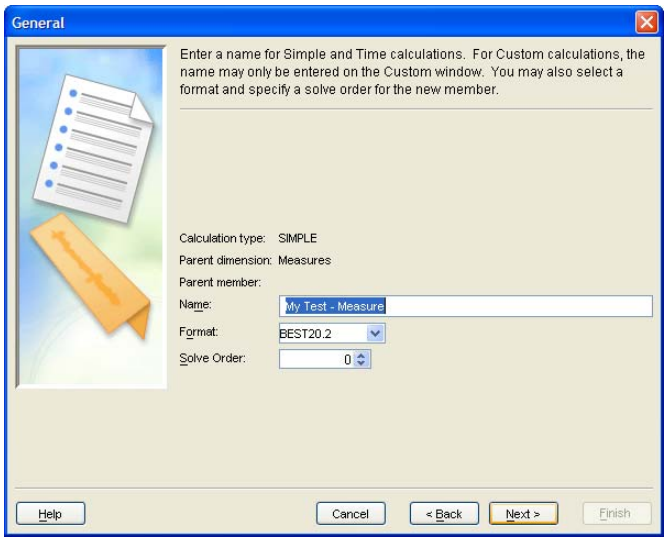
3) At this point, a user can select a simple calculation, or break out to custom code by selecting custom calculation. For custom and more advanced calculation, we suggest creating, editing, and testing the new measures in Enterprise Guide before placing it into the custom calculation interface.



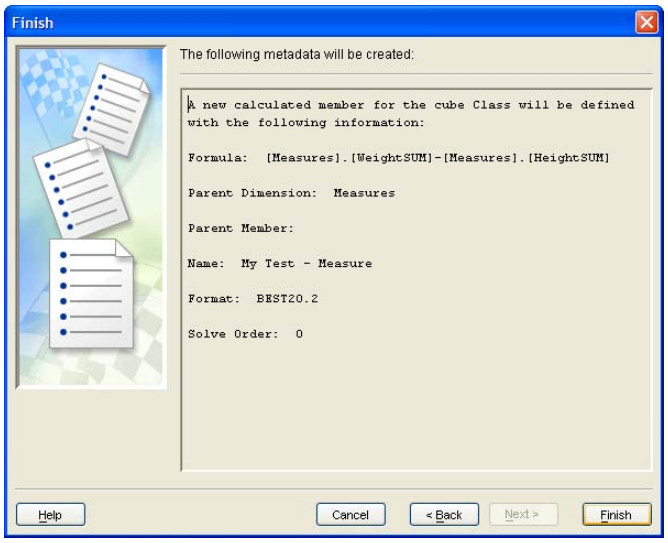
4) For a simple calculation, select the calculation type and the measures to use. Then click “Next”.



5) Provide the measure with a name (can include spaces and some special characters). Select a default SAS format and the solve order. The solve order will prioritize the order in how multiple calculated measures get resolved.



6) Click “Finish”.



Once the cube is refreshed, the new measure will be available for reporting.

SETS

A set is used to define a slice of the cube for calculated members to use in their calculations. A set consists of measures from existing dimensions or hierarchies in the cube. A set can also contain another set and use functions such as union and intersect to piece sets together. A Set must be defined in Proc OLAP or during runtime. Sets cannot be calculated through the Calculated Members tool with OLAP Cube Studio or DI Studio.

Example:

```
DEFINE SET '[SUGI].[EducationCanada]' as
'Crossjoin([Products].[All Products].[EDUCATION],
[Geo].[All Geo].[Canada])';
DEFINE SET '[SUGI].[ConsumerGermany]' as
'Crossjoin([Products].[All Products].[CONSUMER],
[Geo].[All Geo].[Germany])';
DEFINE MEMBER "[SUGI].[Measures].[Percent_CanEdu]" AS
'Aggregate([EducationCanada],[Measures].[Actual]), format_string="Percent10.1"';
DEFINE MEMBER "[SUGI].[Measures].[Percent_GerCons]" AS
'Aggregate([ConsumerGermany],[Measures].[Actual]), format_string="Percent10.1"';
```

CALCULATION EXAMPLES

IIF/ELSE

Conditional logic can be used to define calculations based off of the current level, the current classification, or the current value that is being displayed. An iif function has three arguments:

1. the condition
2. what to do if the condition is true
3. what to do if the condition is false

If there is more than one condition, multiple iif functions are used and nested within one another. Here is an example of conditional logic with three conditions defined by comparing measures to determine how to set a traffic light value:

```
DEFINE MEMBER "[SUGI].[Measures].[TLight]" AS
'iif([Measures].[Difference] > ([Measures].[Actual] * .10),
1,
    iif([Measures].[Difference] > (Measures].[Actual] * .30)),
    2,
    3
)
), format_string="1."';
```

The logic sets TLight equal to 1 if the difference is greater than 10% of actual. If the first condition is false and difference is greater than 30% of actual, then TLight is set to 2, else TLight is set to 3.

AVERAGE REVENUE PER CUSTOMER

The complication with performing this calculation in version 8 stemmed from needing a distinct count of products at any given level to divide by. With the use of the MDX count function you can add a measure to perform this when building the cube or at query time. The same logic would be used for Average Revenue per Customer calculations in OLAP.

```
DEFINE MEMBER "[SUGI].[MEASURES].[Avg_Rev_Per_Product]" AS
'([Measures].[Actual] /
COUNT([Products].[ProdID].[All ProdID].Children))';
```

PERFORMING CALCULATIONS BASED OFF OF CURRENT LEVEL (NON ADDITIVE MEASURES)

Non-additive metrics usually mean that a particular calculation should behave differently based off of the slice of data that is being displayed. In version 8 OLAP this was very tough to accomplish and it usually required front end application changes and metabase extensions. One of the advantages of the new OLAP Framework and the enhancements MDX provides is that non-additive metrics can now be handled through OLAP code.

Below are two ways of defining the same calculation:

```
Define MEMBER "[SUGI].[Measures].[New_Actual]" AS
'iif(NOT([Time].currentmember.level is null) AND [Time].currentmember.level is
[Time].[Month],
Avg([Geo].[All Geo].[Germany],[Measures].[Actual]),
[Measures].[Actual])';
Define MEMBER "[SUGI].[Measures].[New_Actual2]" AS
'iif(NOT([Time].currentmember.level is null) AND [Time].currentmember.level.Name =
"Month",
Avg([Geo].[All Geo].[Germany],[Measures].[Actual]), Measures].[Actual])';
```

ROLLING 12 MONTHS

Creating rolling 12 month calculations from cubes has always presented challenges since multiple month values roll into the calculation for one month. The value for a month must be overwritten with the aggregation of the prior 12 months at a given level of a report. Here is a piece of code that generates a rolling 12 calculation in MDX. Notice that it will check to see if there are values for the prior 12 months and if not the calculation defaults to 0. If there are at least 12 prior members to the currentMember (for month), an aggregation of actual is taken over the last 12 months.

```
DEFINE MEMBER "[SUGI].[Measures].[Rolling_Actual]" AS
'iif([Time].[Months].CurrentMember.lag(12) is NULL ,
0,
Aggregate([Time].[Months].CurrentMember.lag(1):[Time].[Months].CurrentMember.lag(12)
,[Measures].[Actual]))';
```

YEAR TO DATE

Year to date formulas can be calculated in a variety of ways. The example below assumes that 1994 is the current year in the data and therefore sums up the year and places a new level to the time dimension/hierarchy named YTD.

If a dimension is set to a “date” type dimension, functions like closingPeriod(), lag(), lead(), and openingPeriod() can be used to dynamically perform year to date calculations.

```
with MEMBER [Time].[All Time].[YTD] AS
'Sum([TIME].[ALL TIME].[1994])'
MEMBER [Time].[All Time].[Difference] AS
'([TIME].[ALL TIME].[1994] - [TIME].[ALL TIME].[1993])'
/* My MDX Code goes here */
SELECT
{[Measures].[Actual],[Measures].[Predict]} ON COLUMNS,
{[Time].[Time].[Year].Members, [Time].[All Time].[Difference],
[Time].[All Time].[YTD]} ON ROWS
FROM SUGI
```

CONCLUSION

The ability to create complex calculations within a cube; or on the fly from any application truly opens up OLAP to be used for more than just typical Multidimensional Reporting. This same flexibility and adherence to industry standards allows developers to build OLAP cubes that can address more than your standard “OLAP” reporting of basic rows and columns. Being able to provide run-time logic dependent on how the end-user is interacting with a report allows OLAP applications the ability to truly integrate the Statistical and Analytical Power of SAS into one highly optimized data structure.

REFERENCES

SAS 9.1.3 OLAP Server MDX Guide
http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/olap_mdx_9317.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Brian Miles
E-mail: brian.miles@zencos.com
Web: www.zencos.com

Name: Ben Zenick
E-mail: ben.zenick@zencos.com
Web: www.zencos.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.