How-to Guide
SAP EPM

How To Manage Allocation Logic Scripts in BPC for Microsoft Platform

Version 1.01 – December 2008

Applicable Releases:
SAP BPC 5.X, 7.X,

Introduction

The logic engine includes support for a structured way to define allocations. This method tries to simplify the job of the administrator by providing a simple and more intuitive syntax to represent the elements of an allocation. This approach is the first step to building a tabular storage format and user interface to support allocation processes in BPC. The current framework for implementing Allocations is based on script language and statements that are stored in .LGF files in the Script logic administrative console by application.

The basics

An allocation is essentially always made up of the following components:

- WHAT needs to be allocated
- WHERE the results of the allocation must be written
- What driver should be USED to perform the allocation
- How the allocation driver should be used, i.e. what FACTOR must be applied to the source amounts during the calculation

For example it might be that the RENTAL expense, as incurred by the ADMIN department, must be allocated to each operating unit using a predefined PERCENTAGE assigned to each of them.

The definition of such allocation could be as simple as this:

FACTOR:	USING/100

DIMENSION	WHAT	WHERE	USING
ENTITY	ADMIN	<>ADMIN	<>ADMIN
ACCOUNT	RENTAL	RENTAL	PERCENTAGE

The above instructions should be interpreted as follows: read account RENTAL from entity ADMIN and allocate it on all other entities using the PERCENTAGE account as defined in those entities. The formula to apply, as per the FACTOR instruction, is:

(The amount of the WHERE region) = (the amount of the WHAT region) * (the amount of the USING region) / 100

All dimensions not specified in such instructions (CATEGORY, DATASRC, etc.) would simply rely on a run-time selection from the user, to decide on what members the allocation should be performed. The above allocation, for example, could be executed on any data category, time period, data-source, etc., because its definitions do not specify anything for those dimensions. If no member is passed at run-time for any of these dimensions, the allocation would be executed for all of their (stored) members (The only exception is the CURRENCY dimension, which, if unspecified, would default to the LC member).

The data regions of the allocation

An allocation always needs to know what values to read and where to write the results. These two definitions correspond to two data regions we call the “WHAT” region and the “WHERE” region (if

you prefer, the source and the destination) that result from the intersection of the appropriate sets of members for all dimensions of the cube.

Two extra regions may need to be defined, based on the type of allocation being performed. There may be a region of data from which a driver should be extracted (the “USING” region), and, in case the driver is not an absolute value, but is relative to a total amount that should be accounted for in the factor, there may also be a “TOTAL” region. This region is in general equal to the USING region, but there may be exceptions to this, so its definition is also left to the administrator.

Script Logic Syntax

Our first pass to the script-based definition of an allocation uses a RUNALLOCATION / ENDALLOCATION structure defined as follows:

```
*RUNALLOCATION
        *FACTOR={expression}
        *NAME={allocation name}
        *APP [WHAT={app name};] [ WHERE={ app name };] [USING ={ app name }]
        *DIM {dim name} WHAT={set}; WHERE={set};[USING ={set};] [TOTAL={set}]
        *DIM …
*ENDALLOCATION
```

With this syntax, the example described in the previous paragraph will look as follows:

```
*RUNALLOCATION
        *FACTOR=USING/100
        *DIM ENTITY      WHAT=ADMIN;   WHERE<>ADMIN; USING<>ADMIN
        *DIM ACCOUNT WHAT=RENTAL; WHERE=RENTAL; USING=PERCENTAGE
*ENDALLOCATION
```

Below is a detailed description of the various instructions:

*FACTOR

This instruction can be used to define any arithmetic expression (written in the {expression} parameter) and may contain operands, parentheses, constants and one or both of the keywords USING and TOTAL, representing respectively the amount coming from the “USING” region (i.e. the amount of the driver) and the amount coming from the “TOTAL” region (i.e. the sum of the drivers). For example:

```
        *FACTOR=USING/TOTAL
```

Another keyword supported by this parameter is COUNT, which represents the number of members into which one amount must be allocated. For example, when allocating evenly a yearly value into all months of a year, the administrator may just use the COUNT keyword.

```
        *FACTOR=1/COUNT
```

In this case COUNT will automatically contain the value 12 (This keyword will obviously be more helpful in cases where the number of members is not predictable. This is just an example)

If omitted, the factor will always default to 1

*NAME

This is the name of the allocation. This parameter is optional when used in conjunction with the RUNALLOCATION / ENDALLOCATION structure as described above. It is however required by the RUN_TBL_ALLOCATION instruction and by the single-line RUN_ALLOCATION instruction (see below for the meaning of such instructions). For Example:

```
*RUNALLOCATION
        *NAME=RENTALLOCATION
        *FACTOR=USING/100
        *DIM ENTITY     WHAT=ADMIN;   WHERE<>ADMIN; USING<>ADMIN
        *DIM ACCOUNT WHAT=RENTAL; WHERE=RENTAL; USING=PERCENTAGE
*ENDALLOCATION
```

*DESCRIPTION

This optional string may be used to describe some detail of the allocation. Currently it is only used for display in the log file. For example:

```
*RUNALLOCATION
        *DESCRIPTION=Rent allocation Number1
        *NAME=RENTALLOCATION
        *FACTOR=USING/100
        *DIM ENTITY     WHAT=ADMIN;   WHERE<>ADMIN; USING<>ADMIN
        *DIM ACCOUNT WHAT=RENTAL; WHERE=RENTAL; USING=PERCENTAGE
*ENDALLOCATION
```

*APP

This optional parameter can be used to explicitly define the application from which the various regions of data (the WHAT, the WHERE and the USING) must be retrieved. If omitted, the application will be the one from which the allocation is executed, but it can be used to specify a different application for EACH of the possible regions. In an extreme case the allocation could read the source values from one application and write the results in a second application, while the driver might be coming from a third application. And all this could be done irrespective of the application from which the allocation is launched. For Example:

```
*RUNALLOCATION
        *DESCRIPTION=Rent allocation Number1
        *NAME=RENTALLOCATION
        *FACTOR=USING/100
        *APP            WHAT=OPS;   WHERE=FINANCE USING=OPS
        *DIM ENTITY     WHAT=ADMIN;   WHERE<>ADMIN; USING<>ADMIN
        *DIM ACCOUNT WHAT=RENTAL; WHERE=RENTAL; USING=PERCENTAGE
*ENDALLOCATION
```

*DIM

With this keyword the administrator can define the set of members that each dimension should read for each specific region of the allocation (the WHAT, the WHERE, the USING and the TOTAL regions).

The syntax is:

*DIM {dimension name}

… and must be followed by the definition of the set of members to use for one or more of the possible data regions (see below).

The {dimension name} can be hard coded (like ACCOUNT) or can be expressed with the dimension type keywords ACCOUNTDIM, ENTITYDIM, etc. These do not need to be defined as FUNCTIONS in a library file anymore (as done for the rest of the logic syntax), as they will be automatically interpreted by the logic engine. If this creates conflicts with the rest of the logic, this other keyword can also be used:

DOT({Type})

DOT stands for “Dimension Of Type”. {Type} can be A, C, E, T, I or R (for Account, Category, Entity, Time, Inter-company and currency respectively).

A special case of dimension name is the keyword “**AMOUNT**”. This can be used to set a filter for the amounts to be allocated or to be used as drivers. See the example no.8 shown below, where the allocation is only performed on members where the amount of sales is greater than zero. Note that the amount is read as UNSIGNED. In other words, values are handled as they appear in BPC for Excel and NOT with the sign used in the SIGNEDDATA field of the FACT tables.

{set}

The set of members to use in the allocation for each of the dimensions specified in the allocation can be defined with any combination of the following formats:

- A comma delimited list of members. For example:

  *DIM ACCOUNT WHAT=MarketingExpense, AdvertisingExpense

- A set of members filtered using one or more properties. For example:

  *DIM ACCOUNT WHAT= [GROUP] = ”Profit & Loss” AND [ACCTYPE]=”EXP”

  The name of the property may need to be enclosed in [brackets] and the values must be enclosed in either double quotes or single quotes. The expression can be written with any SQL-supported syntax, as it will be passed as-is to the SQL query engine

- The base-level descendants of a selected parent, with the syntax BAS({parent name}). For example:

  BAS(SALESEUROPE)

- The “<<<” keyword, meaning “use the same definition specified to the left of this instruction”

- The “>>>” keyword, meaning “use the same definition specified to the right of this instruction”

  With these two keywords, the initial example could be written as follows:

```
*DIM ENTITY     WHAT=ADMIN;WHERE<>ADMIN;    USING=<<<
*DIM ACCOUNT    WHAT=>>>;    WHERE=RENTAL;  USING=PERCENTAGE
```

Important remarks:

Remark 1:

Not all regions must be specified to the right of a *DIM statement. The administrator must however remember that any unspecified region will take its set of members from the region passed to the logic by the user or defined by some other logic instruction like *XDIM_MEMBERSET and the like.

In the following example the member set of the ENTITY dimension for the (unspecified) WHERE region will be SALESITALY:

```
*XDIM_MEMBERSET ENTITY=SALESITALY

*RUNALLOCATION
        *DIM ENTITY WHAT=ADMIN; USING<>ADMIN // here the WHERE is omitted
*ENDALLOCATION
```

Remark 2:

Each set of members MUST be separated by a semicolon (“;”). This delimiter has been chosen to try and simplify the readability (as well as the parsing) of each logic line.

Remark 3:

The member set, however defined, can be preceded by the “<>” sign, as opposed to the “=” sign, to indicate that the set must include all members EXCEPT those specified in the list.

Example:

```
WHERE<>ADMIN
```

The keyword %YEAR%

Anywhere in the time dimension definition, the keyword %YEAR% can be used to dynamically identify the year to process. The run-time value of the %YEAR% keyword will be retrieved form the **YEAR PROPERTY** of the FIRST member in the passed set of members of the time dimension. In the following example the %YEAR% keyword is 2005.

```
*XDIM_MEMBERSET TIME = 2005.DEC, 2006.DEC, 2007.DEC
```

*RUNALLOCATION
	*FACTOR=1/COUNT
	*DIM TIME WHAT**=%YEAR%**.INPUT; WHERE=BAS(**%YEAR%**.TOTAL)
*ENDALLOCATION

Shifting the %YEAR% keyword

The time dimension can be shifted relative to the value of the %YEAR% keyword, using the syntax:

%YEAR%(n)

… Where n can be any positive or negative integer like in the following example:

*DIM TIME WHAT=%YEAR%_INPUT; WHERE=BAS(%YEAR%.TOTAL); USING=BAS(%YEAR%(-1).TOTAL)

The keyword EACH( )

The member set defined in a dimension of the “TOTAL” region is in general automatically added up into a TOTAL value, to generate the total amount to use as the denominator in the calculation of the allocated amounts. In some special situation, however, this is not desired, and yet the definition of the region generates an ambiguity that cannot be easily recognized by the allocation engine. In such situations the user may enforce the processing of the members in the set on an individual basis, by using the keyword:

EACH ({*member set}*)

… where *{member set}* is the set of members to process.

Following is a real example where such keyword comes at the rescue (split in multiple lines for readability):

*DIM TIME
WHAT = BAS(%YEAR%.TOTAL);
WHERE = BAS(%YEAR%.TOTAL);
USING = BAS(%YEAR%(-1).TOTAL);
TOTAL = EACH(BAS(%YEAR%(-1).TOTAL))

Building a library of allocations

An additional syntax is currently available to define and run allocations. The syntax is based on two different instructions:

*RUN_ALLOCATION {allocation name} // this instructions runs a given allocation

In conjunctions with:

// this structure defines the allocation
//---------------------------------------

```
*ALLOCATION {allocation name}
        // {allocation definitions}
        // {allocation definitions}
        // ….
*ENDALLOCATION
```

The allocation definition syntax also supports the format:

```
*ALLOCATION
        *NAME {allocation name}
        // {allocation definitions}
        // {allocation definitions}
        // ….
*ENDALLOCATION
```

…where the name is defined with a NAME instruction inside the structure

Important remarks:

While the RUN_ALLOCATION instruction is specific to a give COMMIT section, the ALLOCATION structure defining an allocation is available throughout the entire logic file, similarly to the logic structures FUNCTION / ENDFUNCTION or SUB / ENDSUB, and can also be isolated inside one or more library files which could be included in the current logic with the instruction *INCLUDE {file name} (or also *SYSLIB {file name}).

Example:

```
// Main logic file
//-------------------------
*RUN_ALLOCATION ALLOC 1
*INCLUDE ALLOCATIONS LIBRARY.LGL

// Content of ALLOCATIONS LIBRARY.LGL
//-----------------------------------------
*ALLOCATION ALLOC 1
        *DESCRIPTION=ALLOCATION NUMBER 1
        *FACTOR=USING/100
        *DIM ACCOUNTDIM     WHAT=RENT;          WHERE=<<<;  USING=PERCENT
        *DIM ENTITYDIM      WHAT=GLOBALOPS;  WHERE=>>>;  USING<>GLOBALOPS
        *DIM INTCODIM       WHAT=NON_INTERCO; WHERE=<<<; USING=<<<
*ENDALLOCATION

// more allocations…
// …….
```

How to run an allocation from a logic script

Multiple RUNALLOCATION structures can be defined in the same logic script. These instructions will be executed in the order in which they are found and there is no need to put them in separate COMMIT sections. **The following remarks, however, apply:**

Allocation instructions can be mixed with WHEN / ENDWHEN structures or MDX formulas within the same logic file. However they must be isolated inside their own COMMIT section. If a COMMIT instruction is not found, the SQL-logic instructions will take precedence and the Allocations will be ignored.

Note also that the allocation instructions CAN coexist with other general-purpose instructions within the same COMMIT section and this may be useful to adjust the region onto which the allocation must be executed. Another example is the instruction *WRITE_TO_FAC2, which can be used to make the allocation engine write the results into FACT2.

Some other instructions are however currently ignored by the allocation engine. The most notable are:

*CALCULATE_DIFFERENCE=0 //the difference is always calculated by the allocation engine
*XDIM_MAXMEMBERS {dim} = {n} // all selected members will always be processed at once
*DESTINATION {dim}={set} // the definition of the destination is always derived by the WHERE

The allocation engine

The allocation is performed on the values stored in the FACT tables of the application. The technique we use to perform the calculation is however substantially different from what we use in our SQL-based logic. In this case the engine automatically generates a series of SQL queries that will perform the calculation DIRECTLY in the SQL database using SQL code, eliminating the need to pull the records to process into an ADO record set. This is expected to improve the performance of the calculation, as compared to our traditional SQL-based logic.

Note also that the allocation engine will automatically take care of calculating the difference between the existing values and the new values, perform a clear of the destination region where no data should exist, and also write the final values directly in the Write Back table, after having verified the status (locked or not locked) of the destination region.

This mechanism, while much faster than the mechanism used in the other types of modeling logics, does not allow to suppress the calculate_difference option, which will always remain on, even if the user sets it off. The only disadvantage of this technique is that the log file is now unable to display the calculated amounts (this feature could easily be added back, but the performance of larger allocations would deteriorate quite a bit, and we decided to ignore it, for the time being).

The log file

When an allocation is performed, the log file will display all SQL queries that are dynamically generated at run time. These may be copy-pasted into SQL Server Management Studio to test the calculations. The log even contains some “select * from…” statement which can help the user display in the SQL Server Management Studio the effects of the individual queries by listing the content of the generated temporary tables.

If desired, the content of the log file could be adjusted and saved as a custom stored procedure to be invoked from a DTS task and run as a customized allocation directly in such format.

Appendix I shows an example of the content of a log file as generated by the execution of an allocation.

Allocation examples

The following examples show several ways to combine all possible syntaxes and keywords to generate various types of allocations. For readability purposes the definitions of the sets for each data region are presented in a tabular form. In the first example the corresponding script is also shown.

Example 1

The account RENT in entered in entity GLOBALOPS, inter-company NON_INTERCO. This amount must be allocated using a percentage of allocation that is entered by the user in account PERCENT in the appropriate entities and for the desired members of the CATEGORY, TIME, DATASRC and RPTCURRENCY dimensions. This allocation demonstrates the following features:

- It uses the {dimensiontype}DIM keyword to identify the dimensions by type
- It uses the “<<<” and “>>>” keywords to reference the definitions used to the left or to the right
- It uses the “<>” operand to reference all members not equal to a given one

Factor: USING/100

	APP or {Dim} or VALUE	WHAT	WHERE	USING
	ACCOUNTDIM	RENT	<<<	PERCENT
	ENTITYDIM	GLOBALOPS	>>>	<>GLOBALOPS
	INTCODIM	NON_INTERCO	<<<	<<<

Here below is the corresponding script:

```
*RUNALLOCATION
        *FACTOR=USING/100
        *DIM ACCOUNTDIM     WHAT=RENT;          WHERE=<<<;      USING=PERCENT
        *DIM ENTITYDIM      WHAT=GLOBALOPS;     WHERE=>>>;      USING<>GLOBALOPS
        *DIM INTCODIM       WHAT=NON_INTERCO;   WHERE=<<<;      USING=<<<
*ENDALLOCATION
```

Example 2

The account RENT in entered in entity GLOBALOPS, inter-company NON_INTERCO. This amount must be allocated on the basis of the square meters of rented space used by all European entities. This allocation demonstrates the following features:

- It uses the sum of the square meters across the European entities in order to correctly allocate the entire rent expenditure.
- It uses the BAS( ) keyword to build a list of members

Factor: USING/TOTAL

APP or {Dim} or VALUE	WHAT	WHERE	USING	TOTAL
ACCOUNTDIM	RENT	<<<	SQMT	<<<
ENTITYDIM	GLOBALOPS	>>>	>>>	BAS(SALESEUROPE)
INTCODIM	NON_INTERCO	<<<	<<<	<<<

Example 3

The sum of all ADVERTISING expenses incurred by all European operations must be re-allocated to each European operation based on their external SALES. This allocation demonstrates the following features:

- It uses the DOT({type}) keyword to identify the dimensions by type
- It demonstrates the ability to perform aggregations in the WHAT region (SALESEUROPE and ALL_INTCO are parent members)
- It shows a many-to-one re-direction of a dimension (It reads the sum of the Inter-company members and writes it in the NON_INTERCO member of the INTCO dimension).
- It shows a one-to-one re-direction of a dimension (It reads the INPUT member and writes the results in the ALLOCATED member of the DATASRC dimension).

Factor: USING/TOTAL

APP or {Dim} or VALUE	WHAT	WHERE	USING	TOTAL
DOT(A)	ADVERTISING	<<<	EXTSALES	<<<
DOT(E)	SALESEUROPE	BAS(SALESEUROPE)	<<<	<<<
DOT(I)	ALL_INTERCO	NON_INTERCO	>>>	BAS(ALL_INTERCO)
DATASRC	INPUT	ALLOCATED	INPUT	<<<

Example 4

This allocation is the same allocation shown in prior example, with the only difference that the driver account (EXTSALES) is only valid if greater than zero. In other words any negative sales amounts will not be computed for the allocation process. Note that the sign of the amount is the sign as shown in WebExcel, and not the sign of the SIGNEDDATA field in the fact tables. Note also that the value in the amount field can be any value (like “>100”) and that other operands like “=” or “=>” or “<>” are accepted. Combined filters using the “AND” and the “OR” keywords are currently NOT supported.

Factor: USING / TOTAL

APP or {Dim} or VALUE	WHAT	WHERE	USING	TOTAL
ACCOUNT	ADVERTISING	<<<	EXTSALES	<<<
ENTITY	SALESEUROPE	BAS(SALESEUROPE)	<<<	<<<
INTCO	ALL_INTERCO	>>>	>>>	BAS(ALL_INTERCO)
DATASRC	INPUT	ALLOCATED	INPUT	<<<
AMOUNT			>0	>0

Example 5

The budgeted units sold are entered as a total amount for the entire year in the time member 2004_INPUT and must be evenly spread across all months on the year 2004. This allocation demonstrates the use of the COUNT keyword in the FACTOR instruction. In this example COUNT would automatically return 12 if the time dimension is monthly or 52 if the time dimension is weekly.

Factor: 1/COUNT

APP or {Dim} or VALUE	WHAT	WHERE
ACCOUNT	UNITS	<<<
TIME	2004_INPUT	BAS(2004.TOTAL)

Example 6

This allocation is similar to the one shown above, with the difference that the accounts to allocate are the entire list of profit and loss accounts, and the year to process is derived by the members of the time dimension selected by the user at run time. This allocation demonstrates the following features:

- It builds a list of accounts filtering them on a value of a property
- It uses the %YEAR% keyword to dynamically derive the year to process

Factor: 1/COUNT

APP or {Dim} or VALUE	WHAT	WHERE
ACCOUNT	[GROUP]="profit & loss"	<<<
TIME	%YEAR%_INPUT	BAS(%YEAR%.TOTAL)

Note that the COUNT keyword would work on any dimension. Its value is derived from the set of members defined in the first WHERE dimension which is the result of a one-to-many allocation (for this reason, in the above example, the ACCOUNT dimension is ignored and the TIME dimension is used).

Example 7

The values of the profit and loss are entered as a total yearly figure, but spread at will along all other dimensions (category, time, entity, intco, datasrc, etc.). These amounts must be spread of all months according to a factor (account DRIVER) entered in a GLOBALOPS entity, inter-company NON_INTERCO. This allocation demonstrates the support of blank fields to allow the user to provide a value for some dimensions at run time. In this case the ENTITY and INTCO members are those passed to the logic by the user.

Factor: USING/TOTAL

APP or {Dim} or VALUE	WHAT	WHERE	USING	TOTAL
ACCOUNT	[GROUP]="balance sheet"	<<<	DRIVER	<<<
TIME	%YEAR%_INPUT	BAS(%YEAR%.TOTAL)	<<<	<<<
ENTITY			GLOBALOPS	<<<
INTCO			NON_INTERCO	<<<

Example 8

All values of this year BUDGET are entered as a total yearly figure, and must be spread across all months according to their distribution in LAST YEAR ACTUALS. This allocation (while a bit unrealistic, as it expects a perfect match of all members in the un-specified dimensions, like intco, datasrc. etc.) demonstrates the use of a time shift applied to the %YEAR% keyword.

Factor: USING/TOTAL

APP or {Dim} or VALUE	WHAT	WHERE	USING	TOTAL
TIME	%YEAR%_INPUT	BAS(%YEAR%.TOTAL)	BAS(%YEAR%(-1).TOTAL)	<<<
CATEGORY	BUDGET	<<<	ACTUAL	<<<

Example 9

All accounts of the profit and loss of ACTUAL category are copied into the corresponding region of the BUDGET category and increased by a percentage defined in the PERCENT account of entity GLOBALOPS, inter-company NON_INTERCO. This allocation demonstrates the ability to define more complex arithmetic expressions in the FACTOR instruction.

Factor: (1+ USING / 100)

APP or {Dim} or VALUE	WHAT	WHERE	USING
APP	FINANCE	FINANCE	FINANCE
ACCOUNT	[GROUP] = "profit & loss"	<<<	PERCENT
CATEGORY	ACTUAL	BUDGET	<<<
ENTITY			GLOBALOPS
INTCO			NON_INTERCO

Example 10

All accounts in the profit and loss of category ACTUAL, for the three entities SALESITALY, SALESFRANCE and SALESUK, are copied into the corresponding accounts of the entity GLOBALOPS for category BUDGET. This allocation is basically an example of a simple copy action which does not use any FACTOR at all. In this example the engine performs a one-to-one copy (ACTUAL into BUDGET) and a many-to-one copy (SALESITALY, SALESFRANCE and SALESUK are added up and copied into GLOBALOPS)

Factor:

APP or {Dim} or VALUE	WHAT	WHERE
ACCOUNT	[GROUP]="profit & loss"	<<<
CATEGORY	ACTUAL	BUDGET
ENTITY	SALESITALY,SALESFRANCE,SALESUK	GLOBALOPS
DOT(R)	LC	<<

Example 11

In this example the amounts to allocate are read from a different application (YEARLYFINANCE) which has a yearly time dimension (EARLYTIME) while the driver (EXTSALES) is read from the current application (FINANCE) and is used to spread to yearly ADVERTISING expense on all months of the selected year. This example demonstrates the ability to define a set of members for two different TIME dimensions, using two separate lines in the definitions of the allocation.

Factor: USING / TOTAL

APP or {Dim} or VALUE	WHAT	WHERE	USING	TOTAL
APP	YEARLYFINANCE	FINANCE	<<<	<<<
ACCOUNT	ADVERTISING	<<<	EXTSALES	<<<
ENTITY	GLOBALOPS			
INTCO	NON_INTERCO	<<<	<<<	<<<
YEARLYTIME	%YEAR%			
TIME		BAS(%YEAR%.TOTAL)	<<<	<<<

Example 12

In this example the amounts to allocate as well as the driver are read from a different application (YEARLYFINANCE) which has a yearly time dimension (EARLYTIME) while the destination is the current application (FINANCE). Here we demonstrate the ability to define a set of members for two different TIME dimensions, using one single line (referencing both dimensions by their common type “T”).

Factor: USING

APP or {Dim} or VALUE	WHAT	WHERE	USING
APP	YEARLYFINANCE	FINANCE	YEARLYFINANCE
ACCOUNT	ADVERTISING	<<<	DRIVER
ENTITY	GLOBALOPS		
INTCO	NON_INTERCO	<<<	<<<
TIMEDIM	%YEAR%	%YEAR%.JAN	%YEAR%

Example 13

The MARKETING expenses planned in this year’s BUDGET by entity GLOBALOPS need to be allocated on the basis of last year’s EXTSALES generated by the EUROPEAN operations. This needs to be automatically done on all budgeted months. This example demonstrates the use of the EACH( ) keyword in the TOTAL region, to enforce the processing of all periods one by one and make sure they do not get accumulated into a single weight factor.

Factor: USING/TOTAL

APP or {Dim} or VALUE	WHAT	WHERE	USING	TOTAL
TIME	BAS(%YEAR%.TOTAL)	<<<	BAS(%YEAR%[-1].TOTAL)	EACH(BAS(%YEAR%[-1].TOTAL)
CATEGORY	BUDGET	BUDGET	ACTUAL	<<<
ACCOUNT	MARKETING	<<<	EXTSALES	<<<
ENTITY	GLOBALOPS	BAS(SALESEUROPE)	<<<	<<<

Example 14

This example shows how to perform a bulk copy of all values of a given data CATEGORY into an entire set of categories sharing the same attribute value. In this case each destination category will contain the same values of the source category.

Important remark: this allocation would generate a HUGE query that might easily kill the server and is definitely NOT recommended. It is only shown to demonstrate the use of the <ALL> keyword to enforce the selection of all members in some dimension, and to show a situation of

possible memory issues. A better design of this allocation would be to combine it with some logic instruction that would break it in multiple smaller queries, like a PROCESS_EACH_MEMBER=TIME or the like.

Factor:

APP or {Dim} or VALUE	WHAT	WHERE
ACCOUNT	<ALL>	<<<
ENTITY	<ALL>	<<<
INTCO	<ALL>	<<<
TIME	<ALL>	<<<
DATASRC	<ALL>	<<<
RPTCURRENCY	LC	<<<
CATEGORY	ACTUAL	[GROUP]='FCST'

Table-based allocations

The allocation engine is capable of reading the definitions of the allocations from a couple of allocation tables stored in SQL database. There is NO USER interface for this OPTION, so please discuss and implement carefully at any customer. The tables DO NOT exist in the database in SQL and will need to be added, as instructed below.

The allocation tables are two specific tables utilized for all applications in the application set: One table, called **clcAllocH**, is the “header table”, and contains the name of the allocation, the application from which it can be run, a description, and the formula to apply in the FACTOR. A second table, called **clcAlloc**, is the “details table”, containing all the definitions for the various regions of data of the allocation.

Here is how the header of the allocations may look in a tabular format:

APP	ID	DESCRIPTION	FACTOR
	ALLOC01	allocate RENT using PERCENTAGE	(-USING/100)
	ALLOC02	Allocate RAWMATERIALS on SQMT	(USING/TOTAL)
	ALLOC03	Allocate ADVERTISING on EXTSALES	(USING/TOTAL)
	ALLOC04	Allocate UNITS from 2004_INPUT to months	1/COUNT
	ALLOC05	Allocate P&L from CURRENT YEAR_INPUT to months	1/COUNT
	ALLOC06	Allocate P&L from SALESKOREA to SALESEUROPE	1/COUNT
	ALLOC07	Allocate P&L using a DRIVER for seasonality	(USING/TOTAL)
	ALLOC08	grow ACTUAL into BUDGET using PERCENT	(1-USING/100)
	ALLOC09	transform MANY to ONE	
	ALLOC10	similar to ALLOC06	1/COUNT
	ALLOC11	Allocate ADVERTISING on EXTSALES>0	(USING/TOTAL)
	ALLOC12	Allocate ADVERTISING on EXTSALES	(USING/TOTAL)
	ALLOC13	Allocate ADVERTISING from YEARLY APP on EXTSALES	(USING/TOTAL)
	ALLOC14	Allocate ADVERTISING on EUROPE across APPs	(-USING/100)
	BIG01	grow ACTUAL into BUDGET using PERCENT	1.2
	ACT_TO_FCST	Copy ACTUAL to all FCST	1
	ALLOC15	Allocate RENT using PERCENTAGE across APPs	(-USING/100)
	ALLOC16	Allocate ADVERTISING on AVG RATE	(USING/TOTAL)

… and these are the detail records associated to some of the above listed allocations, as stored in the table clcAlloc:

ID	DIM	WHAT	WHERE	USING	TOTAL
ALLOC01	ACCOUNT	RENT	RENT	PERCENT	
ALLOC01	CATEGORY				
ALLOC01	ENTITY	*GLOBALOPS*	*<>GLOBALOPS*	*<>GLOBALOPS*	
ALLOC01	INTCO	NON_INTERCO	NON_INTERCO	NON_INTERCO	
ALLOC01	RPTCURRENCY	LC	LC	LC	
ALLOC01	TIME				
ALLOC02	ACCOUNTDIM	RAWMATERIALS	<<<	SQMT	<<<
ALLOC02	ENTITYDIM	*GLOBALOPS*	*bas(SALESEUROPE)*	<<<	<<<
ALLOC02	INTCODIM	*NON_INTERCO*	*bas(all_interco)*	<<<	<<<
ALLOC03	DOT(A)	ADVERTISING	<<<	EXTSALES	<<<
ALLOC03	DOT(E)	*SALESEUROPE*	*bas(SALESEUROPE)*	<<<	<<<
ALLOC03	DOT(I)	*all_interco*	>>>	>>>	*bas(all_interco)*
ALLOC03	DATASRC	INPUT	ALLOCATED	INPUT	<<<
ALLOC04	ACCOUNT	UNITS	UNITS		
ALLOC04	TIME	*2004_INPUT*	*BAS(2004.TOTAL)*		
ALLOC05	ACCOUNT	[GROUP]="profit & loss"	<<<		
ALLOC05	TIME	*%YEAR%_INPUT*	*BAS(%YEAR%.TOTAL)*		
ALLOC06	entity	*saleskorea*	*bas(SALESEUROPE)*		
ALLOC06	account	cash,accrec	<<<		
ALLOC07	ACCOUNT	[GROUP]="balance sheet"	<<<	DRIVER	<<<
ALLOC07	TIME	*%YEAR%_INPUT*	*BAS(%YEAR%.TOTAL)*	*BAS(%YEAR%.TOTAL)*	<<<
ALLOC07	ENTITY			GLOBALOPS	<<<
ALLOC07	INTCO			NON_INTERCO	<<<
ALLOC08	ACCOUNT	[GROUP]="profit & loss"	<<<	PERCENT	
ALLOC08	CATEGORY	ACTUAL	BUDGET	BUDGET	
ALLOC08	ENTITY			GLOBALOPS	
ALLOC08	INTCO			NON_INTERCO	
ALLOC08	RPTCURRENCY	LC	LC	LC	
ALLOC08	TIME				

The allocations defined in these allocation tables can be run from the logic script with the instruction:

*RUN_TBL_ALLOCATION = {allocation name}

The tables needed to store the above information can be created in the SQL database running the following scripts from the SQL Server Management Studio, for the selected appset: (Please not – Verify that you are selecting the correct Database for the Object Browser.)

Process to add Table #1: clcAllocH

```
CREATE TABLE [clcAllocH] (
        [SEQ] [decimal](8, 0) NOT NULL ,
        [APP] [char] (30) NOT NULL ,
        [ID] [nvarchar] (30)  NOT NULL ,
        [DESCRIPTION] [nvarchar] (70) NULL ,
        [FACTOR] [nvarchar] (50) NOT NULL
        )
```

Process to add Table #2: clcAlloc

```
CREATE TABLE [clcAlloc] (
        [SEQ] [decimal](8, 0) NOT NULL ,
        [ID] [nvarchar] (30) NOT NULL ,
        [DIM] [nvarchar] (20) NOT NULL ,
        [WHAT] [nvarchar] (50) NOT NULL ,
        [WHERE] [nvarchar] (50) NOT NULL ,
        [USING] [nvarchar] (50) NOT NULL ,
        [TOTAL] [nvarchar] (50) NOT NULL
        )
```

The header table also contains an APP field which can be used to enforce the name of the application from which a given application can be invoked. If left blank, the allocation can be invoked from any application.

Appendix I: a sample of the log file

```
**********************************************************************************************************
Start time --->5:26:59 PM  -  Date:1/18/2006  (build code:215)
**********************************************************************************************************

User:pferreri
Appset:apshell2
App:finance
Logic mode:1
Logic by:
Scope by:
Data File:
Debug File:C:\Everest\WebFolders\apshell2\finance\PrivatePublications\pferreri\TempFiles\debuglogic.log
Logic File:ALLOCATION SCRIPT.LGF
Selection:
Run mode:1
Query size:0
Delim:,
Query type:0
Simulation:1
Calc diff.:0
Formula script:|
Max Members:
Test mode:0
Is Modelling:1

Number of logic calls:1
----------------------------------------------------------------------------------------------------
Call no. 1, logic:C:\Everest\WebFolders\apshell2\AdminApp\finance\ALLOCATION SCRIPT.LGF
----------------------------------------------------------------------------------------------------

-------------------------
Building sub-query 1
-------------------------
Query Type:0
Max members:

-----------------------------------------------------
Executing allocation MYFIRSTALLOCATION
-----------------------------------------------------
APP/DIM                         WHAT                            WHERE                       USING     TOTAL
-----------------------------------------------------------------------------------------------
ACCOUNTDIM -->  RAWMATERIALS    <<<                                         SQMT    <<<
ENTITYDIM           -->         GLOBALOPS                   >>>                                 >>>
            BAS(SALESEUROPE)
INTCODIM                        -->         NON_INTERCO                 <>NON_INTERCO   <<<
            <<<

select distinct [ID],[ISBASEMEM] from mbrENTITY where [PARENTH1] = 'SALESEUROPE' or [PARENTH2] =
'SALESEUROPE'

-- read WHAT
-----------------------------------------------------
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
into #349789
from tblFactFINANCE
where [ACCOUNT] = 'RAWMATERIALS' and [RPTCURRENCY] = 'LC' and [INTCO] = 'NON_INTERCO' and [ENTITY] =
'GLOBALOPS'
insert into #349789 ([ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA)
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
from tblFactWBFINANCE
where [ACCOUNT] = 'RAWMATERIALS' and [RPTCURRENCY] = 'LC' and [INTCO] = 'NON_INTERCO' and [ENTITY] =
'GLOBALOPS' and SOURCE = 0
insert into #349789 ([ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA)
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
from tblFac2FINANCE
where [ACCOUNT] = 'RAWMATERIALS' and [RPTCURRENCY] = 'LC' and [INTCO] = 'NON_INTERCO' and [ENTITY] =
'GLOBALOPS'
select
a.[ACCOUNT],a.[RPTCURRENCY],a.[TIMEID],a.[CATEGORY],a.[INTCO],a.[DATASRC],a.[ENTITY],sum(SIGNEDDATA)  as
amtWHAT
into #WHAT_349789 from #349789 a
group by a.[ACCOUNT],a.[RPTCURRENCY],a.[TIMEID],a.[CATEGORY],a.[INTCO],a.[DATASRC],a.[ENTITY]
```

```
drop table #349789

--Time to load WHAT:0.0 sec.
go
select * from #WHAT_349789


-- read destination
----------------------------------------------------
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
into #349789
from tblFactFINANCE
where [ACCOUNT] = 'RAWMATERIALS' and [RPTCURRENCY] = 'LC' and NOT [INTCO] = 'NON_INTERCO' and [ENTITY] in
('ESALESEUROPE','SALESFRANCE','SALESITALY','SALESSWITZER','SALESUK')
insert into #349789 ([ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA)
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
from tblFactWBFINANCE
where [ACCOUNT] = 'RAWMATERIALS' and [RPTCURRENCY] = 'LC' and NOT [INTCO] = 'NON_INTERCO' and [ENTITY] in
('ESALESEUROPE','SALESFRANCE','SALESITALY','SALESSWITZER','SALESUK') and SOURCE = 0
insert into #349789 ([ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA)
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
from tblFac2FINANCE
where [ACCOUNT] = 'RAWMATERIALS' and [RPTCURRENCY] = 'LC' and NOT [INTCO] = 'NON_INTERCO' and [ENTITY] in
('ESALESEUROPE','SALESFRANCE','SALESITALY','SALESSWITZER','SALESUK')
select                    a.[ACCOUNT],a.[RPTCURRENCY],a.[TIMEID],a.[CATEGORY],a.[INTCO],a.[DATASRC],a.[ENTITY],-
sum(SIGNEDDATA) as signeddata
into #DIFF_349789 from #349789 a
group by a.[ACCOUNT],a.[RPTCURRENCY],a.[TIMEID],a.[CATEGORY],a.[INTCO],a.[DATASRC],a.[ENTITY]
drop table #349789

--Time to load DIFF:0.0 sec.
go
select * from #DIFF_349789


-- read USING
----------------------------------------------------
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
into #349789
from tblFactFINANCE
where [ACCOUNT] = 'SQMT' and [RPTCURRENCY] = 'LC' and NOT [INTCO] = 'NON_INTERCO' and [ENTITY] in
('ESALESEUROPE','SALESFRANCE','SALESITALY','SALESSWITZER','SALESUK')
insert into #349789 ([ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA)
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
from tblFactWBFINANCE
where [ACCOUNT] = 'SQMT' and [RPTCURRENCY] = 'LC' and NOT [INTCO] = 'NON_INTERCO' and [ENTITY] in
('ESALESEUROPE','SALESFRANCE','SALESITALY','SALESSWITZER','SALESUK') and SOURCE = 0
insert into #349789 ([ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA)
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
from tblFac2FINANCE
where [ACCOUNT] = 'SQMT' and [RPTCURRENCY] = 'LC' and NOT [INTCO] = 'NON_INTERCO' and [ENTITY] in
('ESALESEUROPE','SALESFRANCE','SALESITALY','SALESSWITZER','SALESUK')
select
a.[ACCOUNT],a.[RPTCURRENCY],a.[TIMEID],a.[CATEGORY],a.[INTCO],a.[DATASRC],a.[ENTITY],sum(SIGNEDDATA)  as
amtUSING
into #USING_349789 from #349789 a
group by a.[ACCOUNT],a.[RPTCURRENCY],a.[TIMEID],a.[CATEGORY],a.[INTCO],a.[DATASRC],a.[ENTITY]
drop table #349789

--Time to load USING:0.0 sec.
go
select * from #USING_349789

-- Merge WHAT with USING into WHERE
----------------------------------------------------
select
w.ACCOUNT,w.RPTCURRENCY,w.TIMEID,w.CATEGORY,u.INTCO,w.DATASRC,u.ENTITY,amtWHAT,amtUSING,amtWHA
T*0 as amtWHERE
into #WHERE_349789 from #WHAT_349789 w,#USING_349789 u
where
w.RPTCURRENCY = u.RPTCURRENCY and w.TIMEID = u.TIMEID and w.CATEGORY = u.CATEGORY and w.DATASRC =
u.DATASRC

--Time to merge:0.0 sec.
go
```

```
select * from #WHERE_349789


-- read TOTAL
----------------------------------------------------
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
into #349789
from tblFactFINANCE
where [ACCOUNT] = 'SQMT' and [RPTCURRENCY] = 'LC' and NOT [INTCO] = 'NON_INTERCO' and [ENTITY] in ('ESALESEUROPE','SALESFRANCE','SALESITALY','SALESSWITZER','SALESUK')
insert into #349789 ([ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA)
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
from tblFactWBFINANCE
where [ACCOUNT] = 'SQMT' and [RPTCURRENCY] = 'LC' and NOT [INTCO] = 'NON_INTERCO' and [ENTITY] in ('ESALESEUROPE','SALESFRANCE','SALESITALY','SALESSWITZER','SALESUK') and SOURCE = 0
insert into #349789 ([ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA)
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],SIGNEDDATA
from tblFac2FINANCE
where [ACCOUNT] = 'SQMT' and [RPTCURRENCY] = 'LC' and NOT [INTCO] = 'NON_INTERCO' and [ENTITY] in ('ESALESEUROPE','SALESFRANCE','SALESITALY','SALESSWITZER','SALESUK')
select a.[RPTCURRENCY],a.[TIMEID],a.[CATEGORY],a.[DATASRC],sum(SIGNEDDATA) as amtTOTAL
into #TOTAL_349789 from #349789 a
group by a.[RPTCURRENCY],a.[TIMEID],a.[CATEGORY],a.[DATASRC]
drop table #349789

--Time to load TOTAL:0.0 sec.
go
select * from #TOTAL_349789

-- Merge WHERE with TOTAL into WHERE2
----------------------------------------------------
select
r.ACCOUNT,r.RPTCURRENCY,r.TIMEID,r.CATEGORY,r.INTCO,r.DATASRC,r.ENTITY,amtWHAT,amtUSING,amtWHERE,amtTOTAL
into #WHERE2_349789 from #WHERE_349789 r,#TOTAL_349789 t
where
r.RPTCURRENCY = t.RPTCURRENCY and r.TIMEID = t.TIMEID and r.CATEGORY = t.CATEGORY and r.DATASRC = t.DATASRC

--Time to merge:0.0 sec.
go
select * from #WHERE2_349789

-- Apply factor
----------------------------------------------------
update #WHERE2_349789 set amtWHERE = cast(amtWHAT as float) * cast(amtUSING as float)/cast(amtTOTAL as float)

--Time to apply factor:0.0 sec.
go
select * from #WHERE2_349789

-- Calculate difference
----------------------------------------------------
insert into #DIFF_349789 ([ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],signeddata)
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],amtWHERE as signeddata
from #WHERE2_349789
select [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY],sum(signeddata) as signeddata
into #RESULT_349789 from #DIFF_349789
group by [ACCOUNT],[RPTCURRENCY],[TIMEID],[CATEGORY],[INTCO],[DATASRC],[ENTITY]

--Time to calculate and count differences (0 found):0.0 sec.
go
select * from #RESULT_349789 where signeddata<>0

-- drop temp tables
----------------------------------------------------
drop table #WHAT_349789,#USING_349789,#TOTAL_349789,#WHERE_349789,#WHERE2_349789,#DIFF_349789,#RESULT_349789


Time to run Allocation:.1 sec.

SIMULATED call 1 completed in 0.2 sec.
```

SIMULATED Run completed in 0.3 sec.
****
End time --->5:26:59 PM - Date:1/18/2006
****