Whitepaper:

Script Logic for SAP BusinessObjects Planning and Consolidation version for Netweaver

Sheldon Edelstein
SAP BusinessObjects Enterprise Performance Management Regional Implementation Group

Applies to:
SAP BusinessObjects Planning and Consolidation version for Netweaver 7.x

Summary:
This whitepaper covers the basic as well as advanced uses of Script Logic design within a SAP BusinessObjects Planning and Consolidation version for Netweaver application. Topics addressed include: scoping, record manipulation, Boolean logic, conditional logic, and integration with Data Manager.

Version:
May 2011
Table of Contents

1. Script Logic Basics .......................................................................................................................... 4
   Purpose of Script Logic ...................................................................................................................... 4
   How Script Logic works ...................................................................................................................... 4
   Structure of Script Logic .................................................................................................................... 4
   Running Script Logic .......................................................................................................................... 5
   Saving a Script Logic File ................................................................................................................... 6

2. Script Logic Scoping .......................................................................................................................... 8
   *XDIM_MEMBERSET .............................................................................................................................. 8
   *XDIM_ADDMEMBERSET .................................................................................................................... 8
   *XDIM_FILTER .................................................................................................................................... 8

3. Record Manipulation ........................................................................................................................ 9
   SQL based commands .......................................................................................................................... 9
   *REC ................................................................................................................................................ 9
   MDX based commands ....................................................................................................................... 10
   Tuple Expressions ............................................................................................................................... 10

4. Filters, Loops and Variables .......................................................................................................... 11
   *WHEN/*IS/*ENDIFWHEN ................................................................................................................. 11
   *WHEN_REF_DATA ............................................................................................................................. 11
   *FOR /*NEXT .................................................................................................................................. 13
   *SELECT .......................................................................................................................................... 14
   *ADD /*ENDADD ............................................................................................................................... 14
   System Variables ............................................................................................................................... 14
   Special Variables/Keywords ............................................................................................................. 15
   %{DimName}_SET% ............................................................................................................................ 15
   %{DimType}_DIM% ............................................................................................................................. 15
   ${DataManagerVariable}$ .................................................................................................................. 16

5. Writing to the Database ................................................................................................................... 17
   *COMMIT ......................................................................................................................................... 17

6. Cross Application Logic ................................................................................................................... 18
   *LOOKUP ......................................................................................................................................... 18
   *DESTINATION_APP .......................................................................................................................... 20

7. Processing Control ........................................................................................................................... 22
   *XDIM_MAXMEMBERS ...................................................................................................................... 22
   *BEGIN/*END ................................................................................................................................. 22
   COMMENTS ..................................................................................................................................... 22
   *INCLUDE ...................................................................................................................................... 23

8. Special Keywords ............................................................................................................................. 24
TMVL ........................................................................................................... 24
9. Advance Programming Techniques ................................................................ 25
   Boolean Expressions .................................................................................. 25
   Conditional Logic ...................................................................................... 25
   New Key Symbols ...................................................................................... 25
   *IIF .......................................................................................................... 26
10. Delivered Functions .................................................................................. 27
    Allocation.lgf ............................................................................................ 27
    Calcaccount.lgf ....................................................................................... 28
    Consolidation.lgf .................................................................................... 28
    Copy_Opening.lgf ................................................................................... 28
    FX_Trans.lgf ............................................................................................ 28
    IC_DATA.LGF .......................................................................................... 29
    ICBooking.lgf .......................................................................................... 29
    ICElim.lgf ............................................................................................... 29
    Validation.lgf ........................................................................................... 29
    Additional delivered files: ........................................................................ 30
    MDXlib.lgf ............................................................................................... 30
    System_Constants.lgf .............................................................................. 31
    System_Library.lgf ................................................................................. 31
11. Business Add-Ins (BAdIs) ........................................................................ 32
    *CALL_CUSTOM_LOGIC .......................................................................... 33
    *START_BADI / *END_BADI .................................................................. 33
12. Performance Considerations ..................................................................... 34
    UJK_SCRIPT_LOGIC_TESTER ................................................................. 35
13. Resources and Help .................................................................................. 38
    Copyright ................................................................................................. 39
1. Script Logic Basics

Purpose of Script Logic

SAP BusinessObjects Planning and Consolidation, version for Netweaver, recognizes that each business has their own unique view of what is important to their enterprise. Script logic allows the development of unique calculations and data movements to occur that can be tailored made for the specific business requirement. Advance programming skills are not a prerequisite, and once the basics are understood programming in script logic can be intuitive and easily accomplished.

How Script Logic works

When the logic module is invoked, the system will:

- Read a specific data selection from the application.
- Applies to the data just read a set of user-defined formulas (stored in a logic file).
- Calculate any new/modifed records and their values.
- Results are written directly to the application database as base member data

Structure of Script Logic

Logic can be broken down into three main components

1. Scoping
   On what records am I going to be interested in when I execute my script logic file?
2. Body/Code
   What do I want to do with the scoped records?
3. Writing the record

.Scoping

*DIM_MEMBERSET P_ACCT = CE0004220
*DIM_MEMBERSET TIME = 2007.JAN

.Body/Code

*WHEN P_ACCT
*IS CE0004220
*REC (EXPRESSION = 901)
*ENDWHEN

.Writing

*COMMIT
Running Script Logic

There are two ways to execute script logic: 1) script logic can be automatically invoked each time data is sent to the database via an Input Schedule, or 2) script logic can be executed directly via a Data Manager package.

Automatic Execution:

Logic contained in a special application script logic file, called “Default logic”, is executed immediately after data is sent by an input schedule. The results of the script logic execution can be seen immediately after refreshing an open report/schedule.

Every BPC application has a default logic file named: DEFAULT.LGF.
Saving a Script Logic File

Script logic writers have the ability to save, validate, save and validate, and delete code:

- Validate Only
- Validate and Save
- Delete Logic
- Save Logic

There are a few potential situations that can occur that the user should be prepared for:

**Saving a blank DEFAULT.LGF file:**

The following error can occur when attempting to save data from an input schedule with a blank DEFAULT.LGF file:

**Error Message**

- Error running default logic ()
- Rejected record list
- Error converting records: The root element is missing.

This error has been associated with earlier support pack upgrades. To correct this issue, open the 'DEFAULT.LGF' script logic file and click 'Validate and Save' (even if it is empty default logic).

**Validation of code containing a Data Manager variable:**

Data Manager variables use the format $<variable_name>$. During the execution of code containing a data manager variable, the system will, at runtime, replace this variable with any user defined value. During script logic validation (or validation and save), the data manager variable is not fully recognized and the following error message is displayed:

```
In this situation, even though the validation fails, the user should still save the script logic code (selecting “save” only (no validation)).
```
Data Manager Execution of Script Logic

Script logic can also be executed from within a Data Manager package; this is typically used for batch processing of formulas.

Using Data Manager to execute Logic module formulas is useful for calculations that do not need to be executed immediately.

For example, an administrator may decide to wait until all the data has been entered in the local currency before executing script logic that will generate translated amounts in a series of multiple reporting currencies.

Default logic can also be invoked during the execution of a variety of data manager activities, such as record imports. Typically the Data Manager package asks the user to select whether or not he/she wishes to invoke default logic during the execution of the package:
2. Script Logic Scoping

Scoping identifies the specific records that will be subject to processing by the main body of the script logic program. Initial scope refers to the set of all records passed to the script logic engine prior to any code execution. The executable scope consists of the initial collection of records that may be modified by the application of special key words in script logic that limit or extend the initial scope of records that will be processed by script logic code. In other words, the executable scope can be defined as the final collection of all records that will be subject to the script logic body code.

There are two different methods available to initially scope the data:

Calling Script Logic from an Input Schedule Data Send (Default logic script)

This scope consists of all records created or modified by the Input Schedule. A list of all the unique members in the sent data is used to derive the scope. An exception is the account dimension, where all accounts are always part of the initial scope. Scope can be overridden / modified / filtered with the various (*XDIM) keywords as described in the executable scope section.

Running Script Logic from within a Data Manager (DM) Package

Scope is derived from the user prompt for the dimensions that are part of the DM package prompts. All non-calculated members are selected for the other dimensions. Independent from which method is used to define the initial scope, the executable scope can be overridden / modified / filtered with various keywords (i.e.: *XDIM commands).

*XDIM_MEMBERSET

This command allows the restriction of the initial scope to one or more specific values. For example, if the initial scope contains data including categories “PLAN” and “ACTUAL”, and the command *XDIM_MEMBERSET Category = PLAN is entered, the execution scope is restricted to only records that have a category equal to “PLAN”.

Data Manager Variables (see section 4) can be used as the subject of *XDIM statements, for example: *XDIM_MEMBERSET ENTITY=$SOURCE$

*XDIM_ADDMEMBERSET

This command allows additional dimension values to be added over and above the values included in the initial scope. For example, if the initial scope contains data including categories “PLAN” and “ACTUAL”, and the command *XDIM_ADDMEMBERSET Category = FORECAST is entered, the execution scope includes PLAN, ACTUAL, and FORECAST for the dimension Category.

*XDIM_FILTER

The XDIM_FILTER command allows the determination of execution scope using MDX filtering. Typically this command is used to deliver a set of dimension members that have a specific property (attribute) associated with the dimension definition. For example the command:

*XDIM_FILTER TIME = [TIME].PROPERTIES("MONTHNUM") = "2" will restrict the execution scope to only records containing a TIME dimension for February (month number 2).
3. Record Manipulation

The power of script logic is in its ability to create and modify records. There are several commands and techniques used to accomplish this action: in general, the techniques can be divided into two main categories: SQL based commands and MDX based commands.

SQL based commands

SQL based script logic commands start with an asterisk (“*”) in the first column of the code line.

*REC

The *REC instruction is used to generate a new database record, and/or it can be used to modify an existing record. The *REC statement is typically used to perform a calculation on the current value of the record being processed (using the keywords “EXPRESSION” and “FACTOR”). The *REC statement can also write calculation results to a different record that has one or more different dimension members (when compared to the original record).

Examples:
The following syntax multiplies the value of the record by 2 and writes the result back to the category BUDGET (note: all other dimensions remain the same):
*REC(FACTOR=2,CATEGORY="BUDGET")
The following statement adds 500 to the original value (represented by the variable %VALUE%) of the record being processed:
*REC(EXPRESSION=%VALUE% + 500)

The *REC command must be enclosed within a loop that selects specific records to be processed. This loop is established by the *WHEN/*IS/*ENDWHEN commands.

Example:
The following syntax loops through the execution scope and selects only those records that have the dimension P_ACCT equal to the value “CE0004220”. When this specific P_ACCT value is found, the record is processed by the *REC command (the original value of the record is multiplied by two and the result is stored with a new CATEGORY dimension, BUDGET. The original record is not modified.

*WHEN P_ACCT
*IS CE0004220
*REC(FACTOR=2,CATEGORY="BUDGET")
*ENDWHEN

Concatenation

You can use BPC variables to accomplish concatenation within a REC statement. In the following example, two script variables (%PCN% and %CCN%) are defined prior to execution of the REC statement. During the execution of the REC statement, the system will replace the variables with their current values. The net result is the concatenation of the two variables into the dimension “PCNCCN” during the execution of the *REC statement:

*FOR %PCN% = 40100
*FOR %CCN% = CC1000050
*WHEN P_ACCT
*IS CE0004220
*REC(FACTOR=1,PCNCCN=%PCN%-%CCN%)
*ENDWHEN
*NEXT
*NEXT
Restrictions in using *REC statement on the NW platform:

1. You cannot use other MDX keywords (such as PARENT and DESCENDANTS) in FACTOR or EXPRESSION instructions. The only permitted operations are addition (+), subtraction (-), multiplication (*), and division (/), combinations of these operators, and parenthesis for tuple and priorities of the operations.
2. You cannot use the GET() function to refer to another source value. The MDX tuple (see section “MDX based commands”) format acts as a GET function.
3. You cannot use NOADD or FLD functions within *REC.
4. Dynamic properties such as HLEVEL, PARENTHn are not supported in the *REC statement.
5. *REC always needs to be accompanied by WHEN / IS / ENDWHEN. Stand-alone *REC statements do not have any effect.
6. You cannot use SIGNEDDATA or any measure name in a WHEN statement to write a condition on a measure value.
7. WHEN / IS / *REC / ENDWHEN cannot be used in SELECTCASE / ENDSELECT.

MDX based commands

MDX based commands typically use square brackets ("["]) to identify specific records for processing. MDX commands allow the programmer to specify a specific record by indicating its dimension value, for example:

\[P\_ACCT].[CE0004020] this MDX structure points to only one value of the dimension P\_ACCT, that value is CE0004020. Note that this structure consist of two parts, the dimension [P\_ACCT] and the specific dimension member [CE0004020], a decimal point ties the two components together to uniquely specify a dimension and specific value. Note: the ACCOUNT dimension is an exception in that it is not necessary to specify the dimension name. In other words, [CE0004020] will have the same effect as [P\_ACCT].[CE0004020] as long as P\_ACCT is an account dimension.

The hash sign, #, identifies a receiving record for a calculated result. For example in:
\[#CE0004020\] = \[CE0004050\]+\[CE0004060\], the calculated member is prefixed with a ‘#’.

Tuple Expressions

MDX commands can be strung together to uniquely specify a specific record(s) by using a MDX tuple expression:

\([\{ACCOUNT\}.[CE0001020],[\{TIME\}.[2009.JAN]\}]\) specifies two specific dimension values: the ACCOUNT dimension with the member value equal to CE0001020, and the TIME dimension equal to 2009.JAN.

An example of a MDX statement using a tuple specifying more than one dimension:
\([\{ACCOUNT\}.[\#CE0001010]\] = \([\{ACCOUNT\}.[CE0001020],[\{TIME\}.[2009.JAN]\}]\)

Tip: Multiple tuple combinations can only be used on the right side of the equal sign.
4. Filters, Loops and Variables

*WHEN/*IS/*ENDWHEN

Tip: As of BPC75NW SP04 and BPC70NW SP09 *WHEN/*IS/*ENDWHEN statements supports looping through transactional data. Please see SAP Note 1474115 for complete details.

The *WHEN/*IS/*ENDWHEN statements allow the script logic engine to determine if the record currently being processed, meets specific conditions prior to executing a specified set of additional instructions. The *WHEN statement allows the identification of the dimension that will be used in the *IS condition. The *IS statement specifies the condition that must be found. Once the condition is met, the subsequent code is executed. If the condition is not met, the subsequent code is skipped and the logic cycles back to the *WHEN command to check the next record. This process continues until all records in the execution scope have been processed.

In this example, the logic will look at each record’s P_ACCT dimension. If that dimension is equal to CE0004220, the following *REC command will be executed (the current record’s value will be multiplied by a factor of 2 and the result will be stored in a new record with all the same dimension values except the CATEGORY dimension will now be BUDGET).

```
*WHEN P_ACCT
*IS CE0004220
*REC(FACTOR=2,CATEGORY="BUDGET")
*ENDWHEN
```

*WHEN_REF_DATA

As of BPC75NW SP04 and BPC70NW SP09, a new command key word has been provided: WHEN_REF_DATA. This key word allows control over the records that are looped through (based on Master Data values or Transactional Data values) in a WHEN/ENDWHEN loop. The WHEN_REF_DATA code line is used to switch the calculation mode of subsequent WHEN/ENDWHEN loops. For example:

```
*WHEN_REF_DATA=MASTER_DATA will enable master data mode.
*WHEN_REF_DATA=TRANS_DATA will enable transaction data mode
```

In the following example, all accounts existing in master data will be looped through the WHEN/ENDWHEN loop.

```
*WHEN_REF_DATA=MASTER_DATA
*WHEN P_ACCT
*IS *
*REC(FACTOR=2,CATEGORY="BUDGET")
*ENDWHEN
```
Tip: the location of the `WHEN_REF_DATA` will determine how subsequent WHEN/ENDWHEN will behave.

Using the `WHEN_REF_DATA=TRANS_DATA` option will limit execution to only those values contained in the scoped data. Using the `WHEN_REF_DATA=MASTER_DATA` option will allow the looping of the logic through all master data values regardless if the specific values are included in the scoped transactional data. Obviously, appropriate utilization of the `WHEN_REF_DATA=TRANS_DATA` option can greatly impact total record processing and significantly impact overall performance and total runtime.

Tip: the transaction data mode is the default mode for all script logic if there are no `WHEN_REF_DATA` statements specified.
*FOR / *NEXT

Script logic variables can consist of one or more dimension member values in a reference list. In the following example, the variable %Q1% is defined in the *FOR statement as consisting of the three dimension members: 2006.JAN, 2006.FEB, 2006.MAR:

*FOR %Q1% = 2006.JAN, 2006.FEB, 2006.MAR

The *FOR command is followed by a *NEXT command to establish a process loop. The intent when using a *FOR/*NEXT is to define a variable with one or more values, then cycle through each member of the variable list one at a time.

In the following example the logic will execute the specified allocation function three times, once for each value of the %Q1% variable defined within the *FOR statement:

*FOR %Q1% = 2006.JAN, 2006.FEB, 2006.MAR

*RUNALLOCATION
*FACTOR=1/3
*DIM TIME WHAT = 2009.JAN; WHERE = %Q1%;
*DIM CATEGORY WHAT=ACTUAL; WHERE=<<<;
*ENDALLOCATION

*NEXT
*SELECT
The *SELECT statement allows the dynamic programming of a variable typically based upon properties of the dimension. The resulting variable definition can be used to dynamically define scoping commands (such as *XDIM_MEMBERSET) and/or included in subsequent *FOR/*NEXT loops.

In the following example, the *SELECT statement is defining the variable “%ACCSET%” by populating it with dimension member IDs from the dimension P_ACCT where the property CALC has a value of “N”. The variable is then being used to dynamically define the execution scope of the dimension P_ACCT in the subsequent *XDIM_MEMBERSET command:

*SELECT(%ACCSET%,"[ID]",P_ACCT,"[CALC]='N'")
*XDIM_MEMBERSET P_ACCT = %ACCSET%

TIP: The *SELECT statement will be executed before any other statements regardless of its actual position in the script logic file.

*ADD /*ENDADD
The *ADD command allows the specification of a variable list. This variable can be incorporated into a MDX statement and allow the execution of the MDX statement for all values of the define variable. In the following example, the *ADD statement defines a variable %ACC% which consists of three values. When the MDX statement is executed, the logic will loop through each value in the variable list and execute the MDX statement three times:

*ADD %ACC%=[CE0004010],[CE0004020],[CE0004030]
[CE06610000] = %ACC%/[CE06520000]
*ENDADD

The equivalent logic without the *ADD/*ENDADD loop:

[CE06610000] = [CE0004010]/[CE06520000]
[CE06610000] = [CE0004020]/[CE06520000]
[CE06610000] = [CE0004020]/[CE06520000]

TIP: Only one formula can be written inside the ADD/ENDADD structure

System Variables
In addition to program defined variables, the system has several predefined variables that are always available:

%USER% - Returns current Planning and Consolidation User
%APPSET% - Returns current Planning and Consolidation AppSet
%APPLICATION% - Returns current Planning and Consolidation Application
%YEAR% - Returns current calendar year
For example:

*XDIM_MEMBERSET P_MONTH = BAS(%YEAR%.TOTAL)

Tip:  %YEAR% will always return only the current calendar year, if the execution scope is for a year other than the current calendar year, %YEAR% will still return the current calendar year.

Special Variables/Keywords

%{DimName}_SET%
A system generated keyword is available for each dimension. This keyword holds the set of members passed to the logic engine for a given dimension (i.e. the initial scope). This keyword can be used as a replacement string anywhere in the logic.

The format of this keyword is: %{DimName}_SET% , where DimName is any valid dimension name in the application. For example: %P_ACCT_SET% for the dimension P_ACCT.

%{DimType}_DIM%
A system generated keyword is available for each non-user defined dimension. This keyword holds the actual name of a dimension of a given type, and can be used as a replacement string to use anywhere in the logic.

The format of this keyword is: %{DimType}_DIM% , where DimType is the technical type of the dimension.

Example:

If, in an application, the category dimension is called SCENARIO, the keyword %CATEGORY_DIM% returns the word SCENARIO.

Additional valid dimension types are: ACCOUNT, CATEGORY, TIME, ENTITY, INTCO, CURRENCY, and DATASRC.
Variables can be passed from Data Manager into script logic during runtime. The format of the DM variable is: `${DataManagerVariable}`. Please see the article titled “How to... pass dynamic parameters to script logic” at http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/d01ce779-f1b2-2b10-07ba-da3734013245 for details on the required Data Manager configuration to pass these variables into script logic.

In the following example, the DM variable `$INFLATION$` was passed from a DM package and is used to recalculate a new budget:

*WHEN P_ACCT
*IS CE0004220
*REC(FACTOR=$INFLATION$,CATEGORY="BUDGET")
*ENDWHEN

Tip:

When validating a script logic file with a DM variable, the current response in a BPC75NW SP04 system (and earlier) is to error on the line containing the `${DataManagerVariable}`. This occurs due to the fact that the DM variable is not really defined until runtime, and the system cannot determine if the variable contains a correct value during the validation process. Therefore the best practice is to save the data file without validation and test for correct operation.
5. Writing to the Database

*COMMIT

The *COMMIT statement instructs the saving of all new or changed records back to the database.

Example:
*WHEN P_ACCT
*IS CE0004220
*REC (EXPRESSION = 22)
*ENDWHEN
*COMMIT

In the above example, a record that contains the account number CE0004220 is modified by setting its value to 22. This record is then posted into the database by the COMMIT statement.

Tip: The *COMMIT statement will end any execution scope defined by any previous *XDIM scoping statements. If your script logic needs to continue after a *COMMIT, you will need to re-scoped by using additional *XDIM statements.
6. Cross Application Logic

*LOOKUP

The Lookup command allows access to records outside of the current application. The value of the record for the external application is made available in a variable that can be incorporated into *REC statements.

In the following example the LOOKUP command finds a record in the RATE application (corresponding to 2006, August, Actual, AVG, GLOBAL, EUR, and PERIODIC) and returns that value back to originating application in the form of the Lookup Id ("RATEEUR"). The Lookup Id is then used within a *REC statement to create a new record in the current application.

```
... 
*LOOKUP RATE
*DIM RATEEUR:TIME="2006.AUG"
*DIM RATEEUR:CATEGORY="ACTUAL"
*DIM RATEEUR:R_ACCT="AVG"
*DIM RATEEUR:R_ENTITY="GLOBAL"
*DIM RATEEUR:INPUTCURRENCY="EUR"
*DIM RATEEUR:MEASURES="PERIODIC"
*ENDLOOKUP
...
...
*WHEN P_ACCT
*IS "CE0004010"
*REC(EXPRESSION=%VALUE%/LOOKUP(RATEEUR), RPTCURRENCY="EUR")
*ENDWHEN
...
```

Tip: the LookUp ID only needs to be defined on one LookUp dimension.
Multiple lookups can be facilitated by incorporating a FOR/NEXT loop within the LOOK UP construct:

*SELECT(%CUR%, "[ID]", RPTCURRENCY, "[REPORTING]=Y")

*LOOKUP RATE
*DIM CATEGORY="ACTUAL"
*DIM R_ACCT="AVG"
*DIM R_ENTITY="GLOBAL"
*DIM TIME="2006.AUG"
*DIM MEASURES="PERIODIC"
*FOR %LOOP_CUR%=%CUR%
*DIM LOOKRATE:INPUTCURRENCY="%LOOP_CUR%"
*NEXT
*ENDLOOKUP
...
*WHEN P_ACCT
*IS "CE004010"
*FOR %LOOP_CUR%=%CUR%
*REC(EXPRESSION=%VALUE%/LOOKUP(LOOKRATE), RPTCURRENCY=%LOOP_CUR%)
*NEXT
*ENDWHEN
*DESTINATION_APP*

The DESTINATION_APP keyword allows you to write the results of a calculation to an application other than the current (source) application which is executing the logic.

Special keywords are provided to allow correct mapping if the target application has a different data model than the source application:

* **SKIP_DIM** - when the source application has dimensions that are not found in the target application
* **ADD_DIM** - if the targeted application has dimensions that do not exist in the source application
* **RENAME_DIM** - can be used when data is to be written into a targeted application that contains a targeted dimension that was named differently from the source dimension.

**Scenario 1: Source application has a dimension not in the target application**

Source Application | Target Application
--- | ---
PLANNING | PLANNING3
Work Status Settings | Work Status Settings
Dimensions | Dimensions
Category | Category
Entity | Entity
P_ACCT | P_ACCT
**P_Activity** | P_DataSrc
P_DataSrc | P_DataSrc
RplCurrency | RplCurrency
Time | Time

Note the dimension P_Activity exists in the target application (PLANNING3), but not in the source application (PLANNING).

Use the keyword **SKIP_DIM** to specify which source dimension should be skipped, for example:

*DESTINATION_APP= PLANNING3
*SKIP_DIM INTOC=P_Activity
Scenario 2: Target application has an additional dimension

Source Application | Target Application
--- | ---
![Diagram](image1.png) | ![Diagram](image2.png)

Note the change in the `IntCo` dimension in the target application (PLANNING4). Use the keyword `*ADD_DIM` to specify a value for the new target dimension, for example:

*DESTINATION_APP= PLANNING5
*ADD_DIM INTOC=I_NONE
(Note: additional dimensions can be referenced using comma separators)

Scenario 3: Source and Target applications have the same dimensions, but one or more dimension names are different.

In this scenario one or more of the source dimensions have a different name in the target application:

Source Application | Target Application
--- | ---
![Diagram](image3.png) | ![Diagram](image4.png)

Note the change in the `P_ACCT` dimension in the source application (PLANNING) to the `P2_ACCT` dimension name in the target application (PLANNING2).

Use the keyword `RENAME_DIM` to identify the new target dimension, for example:

*DESTINATION_APP= PLANNING2
*RENAME_DIM P_ACCT=P2_ACCT
7. Processing Control

*XDIM_MAXMEMBERS*

At times, very large datasets can overwhelm the memory resources of a heavily used or underpowered system. As a result, it may be advantageous to divide up the work into smaller more manageable “chunks” for efficient processing. This can be accomplished through the use of the *XDIM_MAXMEMBERS command.

For example:

*XDIM_MAXMEMBERS ACCOUNT = 50*, this command will break up the execution scope into sub packets that will be processed at a maximum of fifty accounts at a time.

*BEGIN/*END

The *BEGIN/*END commands can be used to split long formulas across multiple lines to improve readability. This command does not change the behavior of the execution; it is only for organizing your code. For example:

*BEGIN

<<Long multi-line executable statement>>

*END

COMMENTS

The forward slash (“/”) indicates that all code occurring after this mark should be considered a comment.

For example:

*WHEN P_ACCT  //this a comment placed at the end of a command
*IS "CE0004010"
*REC(EXPR=\%VALUE\%/LOOKUP\(\text{RATEEUR}\), RPTCURRENCY=\"EUR\")
// This entire line is a comment and will not be executed
*ENDWHEN
*INCLUDE

The *INCLUDE statement allows one logic file to be logically incorporated into another logic file.

In the following example, the ALLOCATION.LGF logic file is embedded in the DEFAULT.LGF logic file:

The contents of the ALLOCATION.LGF file is copied into the DEFAULT logic file during the validation and/or runtime process. If subsequent changes to the ALLOCATION.LGF file are made, the DEFAULT.LGF file will automatically be updated with the changes during runtime.
8. Special Keywords

**TMVL**
The TMVL keyword allows the script logic engine to determine a time period based upon an offset. The syntax for this parameter is TMVL(offset, base_period). The offset can be either negative or positive, but only integers are allowed.

In the following example, records are created by the *REC command for the 2007.APR (12 periods into the future) time period:

```
*XDIM_MEMBERSET TIME = 2006.APR
*WHEN CATEGORY
*IS ACTUAL
*REC(FACTOR=1.1, TIME=TMVL(12,2006.APR))
*ENDWHEN
```

Limitations:
- Nested TMVL parameters such as TMVL(-1, TMVL(-3, 2009.JAN)) are not supported.
- The offset value can only be an integer.
- The first period of the TIME_SET is used as the base period for a negative offset and the last period of the TIME_SET is used as the base period for a positive offset.

You can use TMVL in the following situations:
- Within *REC statements including FACTOR/EXPRESSION
- Within FOR/NEXT loops
- As part of the IS conditions inside WHEN/ENDWHEN.
- Along with time variables, like %TIME_SET%
- Within scoping statements
9. Advance Programming Techniques

Boolean Expressions

Boolean (or yes/no logic) can be incorporated in script logic. By enclosing a Boolean expression in parenthesis the formula will be evaluated as a true or false condition, returning a value of 1 for the expression if TRUE, and 0 if FALSE.

Example:

```
*BEGIN
*REC (EXPRESSION=((%VALUE% > 10000) * 125) + ((%VALUE% < 10000) * (%VALUE% * LOOKUP(LR_RATE))),P_ACCT="TEMP")
*END
```

This *REC statement is evaluated as follows:

If the original record's value (%VALUE%) is greater than 10,000

(%VALUE% > 10000) is interpreted as a logical “1”, then the original value of the record is multiplied by 125 and the expression (%VALUE% < 10000) is interpreted as a logical “0”. A new record is created with the original record’s P_ACCT dimension value is replaced with a new member id: “TEMP”.

If the original record's value (%VALUE%) is less than 10,000.

(%VALUE% > 10000) is interpreted as a logical “0”: 0 multiplied by 125 = 0. (%VALUE% < 10000) is interpreted as a logical “1” and the original value of the record is multiplied by the Look Up rate “LR_RATE”. A new record is created with the original record’s P_ACCT dimension value is replaced with a new member id: “TEMP”.

Conditional Logic

New Key Symbols

Conditional logic can also be incorporated into a MDX statement using new key symbols. In the following example:

```
[ACCOUNT].[#CE0001010] = ([ACCOUNT].[CE0001020]>0? [ACCOUNT].[CE0001020] : 0 )
```

The “?” performs a conditional test, if the test is true then the left hand side of the equation is set equal to [ACCOUNT].[CE0001020]. If the test is false, then the left hand side of the equation is set equal to 0. The special character “:” separates the two TRUE/FALSE outcomes.
The IIF statement allows a conditional IF/Then/Else statement to be defined for MDX based syntax. The IIF statements cannot be include *REC commands, but can be used within MDX tuple expressions:

```
*XDIM_MEMBERSET P_DATASRC=MANUAL
[#CE0004010]=IIF([P_ACCT].[CE0004010]>100,[P_ACCT].[CE0004010],[P_ACCT].[CE0004510])
*COMMIT
```

The above example line executes as follows:
If the record associated with P_ACCT.CE0004010 is greater than 100, then assign the value of the record P_ACCT.CE0004010 to P_ACCT.CE0004010 (i.e.: no changes are made), but (“else”) if the value of P_ACCT.CE0004010 is less than or equal to 100, then assign the value of the record at P_ACCT.CE0004510 to P_ACCT.CE0004010.
10. Delivered Functions

Planning and Consolidation has a library of standard logic functions available for your use. Each file has the extension .LGF, which can be called at validation by using the *INCLUDE function in your logic file (see Section 7).

The following predefined logic is delivered with Planning and Consolidation:

**Allocation.lgf**

Allocation executes a distribution of values based upon a defined basis. This functionality does not require any additional business rules to function. There are two methods to call the allocation engine: *RUNALLOCATION and *RUN_ALLOCATION:

The *RUNALLOCATION format is commonly used when an allocation is to be performed once in the script logic file:

```
*RUNALLOCATION
*FACTOR=<driver>
*DIM P_ACCT WHAT=<source>; WHERE=<target>; USING=<distribution key>;
[TOTAL=<distribution key>]
*DIM <other dimensions>
*ENDALLOCAT
```

The *RUN_ALLOCATION format can be used when an allocation is to be performed multiple times in the same script logic file. In the following example, the allocation called DIST2 is first defined, and then the allocation is called multiple times later in the code by the RUN_ALLOCATION command:

```
*ALLOCATION DIST2
*FACTOR=USING/TOTAL
*DIM P_ACCT WHAT=CE0004010; WHERE=CE0004020; USING=CE0004030; TOTAL=<<<
*DIM ENTITY WHAT=A1000; WHERE=<<<; USING=<<<; TOTAL=<<<
*DIM TIME WHAT=2009.JAN; WHERE=<<<; USING=BAS(2009.TOTAL); TOTAL=<<<
*ENDALLOCATION
...

*RUN_ALLOCATION DIST2
...

*RUN_ALLOCATION DIST2
...
```

Calcaccount.lgf
Calcaccount is used to prepare Cash Flow and must be executed along with appropriately defined Account Calculation Business Rules.

*RUN_PROGRAM CALC_ACCOUNT
CATEGORY = %CATEGORY_SET%
CURRENCY = %CURRENCY_SET%
TID_RA = %TIME_SET%
CALC=A
//OTHER = [ENTITY=%ENTITY_SET%]// or OTHER=[ENTITY=C1000] or
//[ENTITY=%ENTITY_SET%;INTCO=%INTCO_SET%...]
*ENDRUN_PROGRAM

TIP:
The error message “Run logic: Business Rule is not available” is typically observed when the CALC= line is referencing a value that does not exist in the current application. If CALC = A is entered, the code refers to an ID line defined within the account transformation table business rule. For CALC=A to be valid, you must have created a valid rule line with the ID of “A”.

Consolidation.lgf
Consolidation is used to perform legal consolidation and must be executed along with appropriately defined Legal Consolidation Business Rules.

*RUN_PROGRAM CONSOLIDATION
CATEGORY = %C_CATEGORY_SET%
GROUP = %GROUPS_SET%
TID_RA = %TIME_SET%
*ENDRUN_PROGRAM

Copy_Opening.lgf
COPYOPENING is used to transfer ending balances from one time period to starting periods in a subsequent time period. COPYOPENING must be executed with the appropriate Balance Carry Forward business rules.

*RUN_PROGRAM COPYOPENING
CATEGORY = %C_CATEGORY_SET%
CURRENCY = %GROUPS_SET%
TID_RA = %TIME_SET%
*ENDRUN_PROGRAM

FX_Trans.lgf
Currency Conversion is used to translate existing records into alternative currency views. Currency Conversion must be executed with the appropriate Currency Conversion business rules.

*RUN_PROGRAM CURR_CONVERSION
CATEGORY = %C_Category_SET%
GROUP = %GROUPS_SET%
TID_RA = %TIME_SET%
OTHER = [ENTITY=%ENTITY_SET%]
RATEENTITY = Global
*ENDRUN_PROGRAM
ICDATA.LGF
ICDATA is used to perform intercompany reconciliation activities. ICDATA must be executed with the appropriate ICBOOKING business rules.

*RUN_PROGRAM ICBOOKING
CATEGORY = %CATEGORY_SET%
CURRENCY = %GROUPS_SET%
DATASRC = INPUT\ value of datasrc dimension member that represents the input datasrc
TID_RA = %TIME_SET%
ENTITY = %ENTITY_SET%
ACCOUNT = %ACCOUNT_SET%
FLOW = %FLOW_SET%
TYPE = 'I'.
*ENDRUN_PROGRAM

ICBooking.lgf
ICBooking is used to perform intercompany reconciliation activities with difference postings. ICBooking must be executed with the appropriate ICBOOKING business rules.

*RUN_PROGRAM ICBOOKING
CATEGORY = %CATEGORY_SET%
GROUP = %CATEGORY_SET%
TID_RA = %TIME_SET%
OTHER = [ENTITY=%ENTITY_SET%]
*ENDRUN_PROGRAM

ICElim.lgf
ICElim is used to perform intercompany reconciliation. ICElim must be executed with the appropriate ICElim business rules.

*RUN_PROGRAM US_ELIM
CATEGORY = %C_CATEGORY_SET%
GROUP = %GROUPS_SET%
TID_RA = %TIME_SET%
OTHER = [ENTITY=%ENTITY_SET%]
*ENDRUN_PROGRAM

Validation.lgf:
Validation is used to perform validation activities define within the validation business rules.

*RUN_PROGRAM VALIDATION
CATEGORY = %C_CATEGORY_SET%
CURRENCY = %GROUPS_SET%
OTHER = [ENTITY=%ENTITY_SET%]
TID_RA = %TIME_SET%
*ENDRUN_PROGRAM
MDXlib.lgf

The MDXlib file provides examples of pre-delivered MDX financial functions:

```plaintext
// MDXLIB.LGF includes the Explanation and Examples of BPC7 WW MDX.

// Dimension constants in BPC7 are no longer needed, as the pre-defined DIM KEYWORDS (for example: \$TIME_DIM\$ could be used for:

// To comply with SAP BW syntax, following rules need to be aware:
// 1. Member and dimension names must be written in correct Upper or Lower case.
// 2. All member IDs must be enclosed in [square brackets]
// 3. All member IDs must be preceded by the name of their related dimension (enclosed in square brackets), unless the dimension
// 4. All calculated members must have their ID preceded by the \$ (pound) sign.

//Example:
//\$[REVENUE]=-[UNITS]*\$[PRICE]
//[$ACCRC]=[$ACCRC],[\$TIME].[prevmember] +[\$REVENUE]
//[$BUD_INCOME]=($CATEGORY).[BUDGET],[\$NETINCOME])

// Additional delivered functions

// Basic Financial Function, (Miscellaneous)

//calculate Average Account for using below function min, max etc.

/*function AvgHis(\$ACCOUNT)*/
(closingperiod(LEVEL00, ancestor(\$Time_Dim\$.currentmember, LEVEL00).prevmember), \$ACCOUNT% ) + (\$Time_Dim\$.currentmember, 1
*endfunction

//calculate Account Growth used in Sales growth, Expense growth etc.
/*function Growth(\$ACCOUNT)*/
If (closingperiod(LEVEL00, 1, \$Time_Dim\$.currentmember), \$ACCOUNT% ) = 0,
null,

(\$Time_Dim\$.currentmember, \$ACCOUNT% )

(closingperiod(LEVEL00, 1, \$Time_Dim\$.currentmember), \$ACCOUNT% ) / (closingperiod(LEVEL00, 1, $TimeDim\$.currentmember), \$ACCOUNT% )
*endfunction
```
System_Constants.lgf

The system constants file is the logic file that maps your dimension names for an application to the standard (delivered) BPC logic. For example, the file maps the name of the category dimension to the dimension to be used as the category. BPC 7x NW internally generates and includes all system constants automatically. As a result this file does not have to specifically be added with an *INCLUDE statement in order to access the system constants in your coding.

Example:
If, in an application, the category dimension is called SCENARIO, the keyword %CATEGORY_DIM% returns the word SCENARIO. The mapping between SCENARIO and %CATEGORY_DIM% is defined by the System_Constants.lgf file. The keyword %CATEGORY_DIM% can be used in the script logic definition in place of writing out the dimension name “SCENARIO”.

The system constants file is located in the \root\Data\Webfolders\<AppSet>\AdminApp\<App> folder:

```
// application constants
-------------
*FUNCTION CATEGORYDIM=CATEGORY
*FUNCTION TIMEDI=TIME
*FUNCTION CURRENCYDIM=RPTCURRENCY
*FUNCTION ENTITYDIM=ENTITY
*FUNCTION ACCOUNTDIM=ACCOUNT
*FUNCTION INPUTDIM=INPUT

// This part is needed when a RATE cube
// is associated to the application
// (Fx = single or multi currency)
-------------
*FUNCTION THISAPP=FINANCE
*FUNCTION RATEAPP=RATE
*FUNCTION RATEENTITYDIM=RATESRC
*FUNCTION RATEACCOUNTDIM=RATE
*FUNCTION INPUTCURRENCYDIM=INPUTCURRENCY
*FUNCTION RATEENTITYDIM=RATEINPUT
*FUNCTION RATEINPUTDIM=RATECALC
*FUNCTION AVERAGEID=AVG
*FUNCTION ENDGORITHM=END
```

System_Library.lgf

The System Library includes basic examples of delivered keywords and functions. The following standard functions are available in SAP Business Planning and Consolidation:

- Basic Financial Formulas – Account Average and Growth Rate.
- Liquidity Analysis Ratios – Current Ratio, Quick Ratio, Networking Capital Ratio.
11. Business Add-Ins (BAdIs)

BAdIs consist of customizable and reusable ABAP code and objects. They can be inserted into a SAP system to accommodate user requirements too specific to be included in the standard delivery.

BAdIs can be called from within script logic to perform an endless variety of functions. The enhancement spot UJ_CUSTOM_LOGIC forms the initial framework for the development of the customization.

Transaction SE19:

Once a BAdI is coded and activated, the user can call the BAdI from within script logic using two methods. The first method just calls the BAdI and runs the customized ABAP code. The second method allows the script logic to pass parameters into the BAdI code.
*CALL_CUSTOM LOGIC

*CALL_CUSTOM_LOGIC is a script logic key word that allows the calling of a defined BAdI without passing any parameters:

*XDIM_MEMBERSET ACCOUNT=CASH
*XDIM_MEMBERSET RPTCURRENCY = LC
*XDIM_MEMBERSET TIME = 2006.JAN
*XDIM_MEMBERSET CATEGORY = ACTUAL
*XDIM_MEMBERSET INTCO=NON_INTERCO
*CALL_CUSTOM_LOGIC CUST_CALC_ACCT

In the example above, CUST_CALC_ACCT is the name of the BAdI.

*START_BADI / *END_BADI

*START_BADI / *END_BADI are script logic key words that allow the calling of a defined BAdI with the ability to pass parameters:

*XDIM_MEMBERSET ACCOUNT=CASH
*XDIM_MEMBERSET RPTCURRENCY = LC
*XDIM_MEMBERSET TIME = 2006.JAN
*XDIM_MEMBERSET CATEGORY = ACTUAL
*XDIM_MEMBERSET INTCO=NON_INTERCO
*START_BADI CUST_CALC_ACCT
    ACCT_FROM = 10000
    ACCT_TO = 10001
*END_BADI

In the example above, CUST_CALC_ACCT is the name of the BAdI, and two parameters are passed to the BAdI from script logic: ACCT_FROM and ACCT_TO.
12. Performance Considerations

Script performance is always a major consideration when performing calculations on large data sets. There are few basic rules to apply when optimizing any script logic:

1. Make sure your execution scope is properly sized. Excessive scoping can cause the system to cycle through many unnecessary records, wasting time, overusing memory capacity. *It cannot be overstated that you must make absolutely sure your logic ONLY operates based on your intended scope - meaning... that you must make sure you are not reading too many records (scoping too much) for your script execution.*

2. Use the *XDIM_MAXMEMBERS* command to divide the processing of large data sets (see section 2).

3. Don’t put too much logic in your default logic file. Remember that default logic is executed for all write backs from input schedules as well as from certain data manager packages.

4. Use of BADIs can greatly improve runtimes of large, complex logic. Note: BADIs can also negatively impact performance if the ABAP coding is poorly written.

5. Refrain from using “pure” MDX syntax when possible. MDX based syntax requires the script logic engine to parse the MDX code using the MDX Parser and this module can impact performance negatively.

6. Try to keep the Script Logic simple. If you have very large scripts and multiple nested logic files, then chances are you have written logic which is too complex and will perform poorly. Try to evaluate the use of BADIs in this case.

7. Consider implementing the RUNLOGIC BAdI (the RUNLOGIC function allows parallel execution of script logic code on multiple data sets) as detailed in the How To Guide on SDN: “How To Implement the RUNLOGIC Keyword in SAP Business Objects Planning and Consolidation, version for NetWeaver”

http://www.sdn.sap.com/irj/scn/index?rid=/library/uuid/b0ad38cf-9e0e-2e10-9d9a-fbf57e69cd40
The Script Logic Tester allows the direct prototype testing of script logic from a BW backend application. The tester will capture and display log information and provide detail information on how the file was interpreted during execution.

The Script Logic Tester can be accessed by typing in the transaction code UJKT from a SAP GUI display:

The SETTING region of the display (identified as section 1 above) allows the definition of the appset, application as well the identification of the user id that will be executing the logic (this is handy for example, when evaluating security issues).

The DATA REGION section (section 2) allows definition of the initial scope. Use the format dimension=dimension_member, for example: CATEGORY=ACTUAL.
The prototyped script logic code is entered into the lower left hand section (section 3). By clicking on the VALIDATE button the entered code will be tested for syntax. Any errors that are discovered will be presented on the lower right hand screen (section 4). If the logic passes validation, the compiled logic file is displayed, for example:

<table>
<thead>
<tr>
<th>Script Logic Primer for SBOP Planning and Consolidation version for Netweaver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Once the code has been validated, you can execute it in simulation mode (results will not be posted back into the database):</td>
</tr>
</tbody>
</table>

```sql
/*SELECT (%ACT%, [BD], P_ACCT,[BD])='CE0004020')
 *RUNALLOCATION
 *FACTOR=USING/TOTAL
 *DIM P_ACCT WHAT='CE0004010' WHERE=%ACT% USING='CE0004030' TOTAL=<<
 *DIM ENTITY WHAT=A1000 WHERE=< Using TOTAL=<<
 *DIM TIME WHAT=2009,JAN WHERE=><> USING=BAS(2009,TOTAL); TOTAL=<<
 *ENDALLOCATION

/*RUNALLOCATION
 *FACTOR=USING/TOTAL
 *DIM P_ACCT WHAT='CE0004010' WHERE=%ACT% USING='CE0004030' TOTAL=<<
 *DIM ENTITY WHAT=A1000 WHERE=< Using TOTAL=<<
 *DIM TIME WHAT=2009,JAN WHERE=><> USING=BAS(2009,TOTAL); TOTAL=<<
 *ENDALLOCATION

LOG:
FILE:\root\webfolders\sheldon \adm\APP\PLANNING\TEST.LGF
USERS=APL0010010017
APPSET=SHELDON
APPLICATION=PLANNING
FACTOR=USING/TOTAL
ALLOCATION DATA REGION:
Category=ACTUAL
P_ACCT WHAT='CE0004010' WHERE='CE0004020' USING='CE0004030' TOTAL=<<
ENTITY WHAT=A1000 WHERE=< Using TOTAL=<<
TIME WHAT=2009,JAN WHERE=><> USING=BAS(2009,TOTAL); TOTAL=<<

SCRIPT RUNNING TIME IN TOTAL: 0.11 s.```
The script can be executed, and all results saved to the database, by clicking on the EXECUTE button. The resulting system log will display the compiled code, the number of records processed and the time to execute.

A sample log extract is as follows:

APPSET:SHELDON
APPLICATION:PLANNING
FACTOR:USING/TOTAL
ALLOCATION DATA REGION:
  Category: ACTUAL,
  P_ACCT: WHAT: CE0004010, WHERE: CE0004020, USING: CE0004030, TOTAL: <<<
  ENTITY: WHAT: A1000, WHERE: <<<, USING: <<<, TOTAL: <<<

-- Read WHAT region
[P_ACCT] = CE0004010
[TIME] = 2009.JAN
[Category] = ACTUAL
[ENTITY] = A1000
-- Time to load WHAT: 0.151957 second(s).

-- Read USING region
[P_ACCT] = CE0004030
-- Time to load USING: 0.144757 second(s).

-- Merge WHAT and USING
-- Time to merge WHAT and USING: 0.000442 second(s).

-- Apply factor
[P_ACCT] = CE0004030
WHERE = WHAT * USING/TOTAL
-- Time to apply factor: 0.13066 second(s).

-- Read destination and calculate difference
[P_ACCT] = CE0004020
-- Time to read destination and calculate difference: 0.142889 second(s).

-- Records succeeded to write back: 12
-- Records failed to write back: 0

-- Time to run Allocation: 0.596787 second(s).

SCRIPT RUNNING TIME IN TOTAL: 0.72 s.
13. Resources and Help

- EPM How-To Guides

- RKT Online Knowledge Product
  [http://service.sap.com/rkt](http://service.sap.com/rkt)
  On the left hand side, navigate to SAP Ramp-Up Knowledge Transfer -> SAP BusinessObjects EPM Solutions -> SAP BO PC 7.5, version for SAP NetWeaver

- For further details about SAP Business Add-Ins, see the ABAP online help at:
  [http://help.sap.com/saphelp_nw70/helpdata/en/32/a83942424dac04e10000000a1550b0/content.htm](http://help.sap.com/saphelp_nw70/helpdata/en/32/a83942424dac04e10000000a1550b0/content.htm)

- SDN Blog Wiki
  The following wiki compiles blogs specifically targeting BPC topics

- SAP BPC Help
  [http://help.sap.com](http://help.sap.com) has an extensive library of help documentation for all SAP products. Help documentation for BPC 75 NW can be found at the following location: