SAP Business Objects Planning & Consolidations

BPC: Getting Started with Script Logic

Enterprise Performance Management
BPC Enablement:
August 2009

Business Objects
an SAP® company

THE BEST-RUN BUSINESSES RUN SAP™

SAP®

Legal Disclaimer

This presentation outlines our general product direction and should not be relied on in making a purchase decision. This presentation is not subject to your license agreement or any other agreement with SAP. SAP has no obligation to pursue any course of business outlined in this presentation or to develop or release any functionality mentioned in this presentation. This presentation and SAP's strategy and possible future developments are subject to change and may be changed by SAP at any time for any reason without notice. This document is provided without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP assumes no responsibility for errors or omissions in this document, except if such damages were caused by SAP intentionally or grossly negligent.

Agenda

1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic: Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

Agenda

1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic: Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

Calculations in BPC

- Logic allows you to define formulas that perform calculations on SAP Business Planning and Consolidation members and data.
- You can have three different type of calculations:
  - Front-End
    - Worksheet logic (Excel & VBA)
  - On-the-Fly
    - Dimension Member Formulas (MDX)
  - Stored output
    - Script logic (SQL or MDX syntax)
    - Business Rules (Table Driven)
    - Badi (ABAP)
    - Allocation Engine
- Each type has advantages and disadvantages.
- You typically use some combination of all four types to achieve the desired results.

Worksheet Logic Overview

- Uses SAP Business Planning and Consolidation and standard Excel mathematical operators and functions to produce results.
- Is typically used for simple calculations and events, such as data retrievals (EvGTS), sends (EvSND), and variance calculations (EvBET).
- Produces calculated values that do not need to be stored in the database.
- Usage of standard Excel Formulas to perform calculations
- Custom VBA code can be triggered for specific events (like refresh, expand, send)
- Custom VBA coding can also be created for custom selection boxes, validations, allocations, …

Worksheet Logics
VBA Events

- The following events cab be used to trigger custom VBA coding:
  - BEFORE_CHANGECVW
  - AFTER_CHANGECVW
  - BEFORE_REFRESH
  - AFTER_REFRESH
  - BEFORE_SEND
  - AFTER_SEND
  - BEFORE_EXPAND
  - AFTER_EXPAND
- Sample code (need to be created in an “Application Module”)

```
Function BEFORE_CHANGECVW(Argument As String)
    MsgBox Argument
    BEFORE_CHANGECVW = True
End Function
```

Worksheet Logic
Pros and Cons

Pros

- Speed - the formulas reside in the worksheet, so calculations are fast and have minimal impact on performance.
- Flexibility - you can define many formulas in the worksheet.
- No server load – worksheet logic uses the client machine to perform calculations, so it allows the server to run more efficiently.

Cons

- Static - the formulas are only available in the worksheet in which they are written, and need to be rewritten for each worksheet.
- Cannot be applied as a process - worksheet formulas cannot be executed via the Data Manager (i.e. the calculations are not performed until the sheet is opened and the data refreshed).
- Not stored - results are not stored in the database.
- Risk of incompatibilities between different version of MS Excel
- Usage of VBA can cause performance issues in Citrix environments
- Usage of VBA can increase maintenance work on reports / input schedules

Agenda

1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

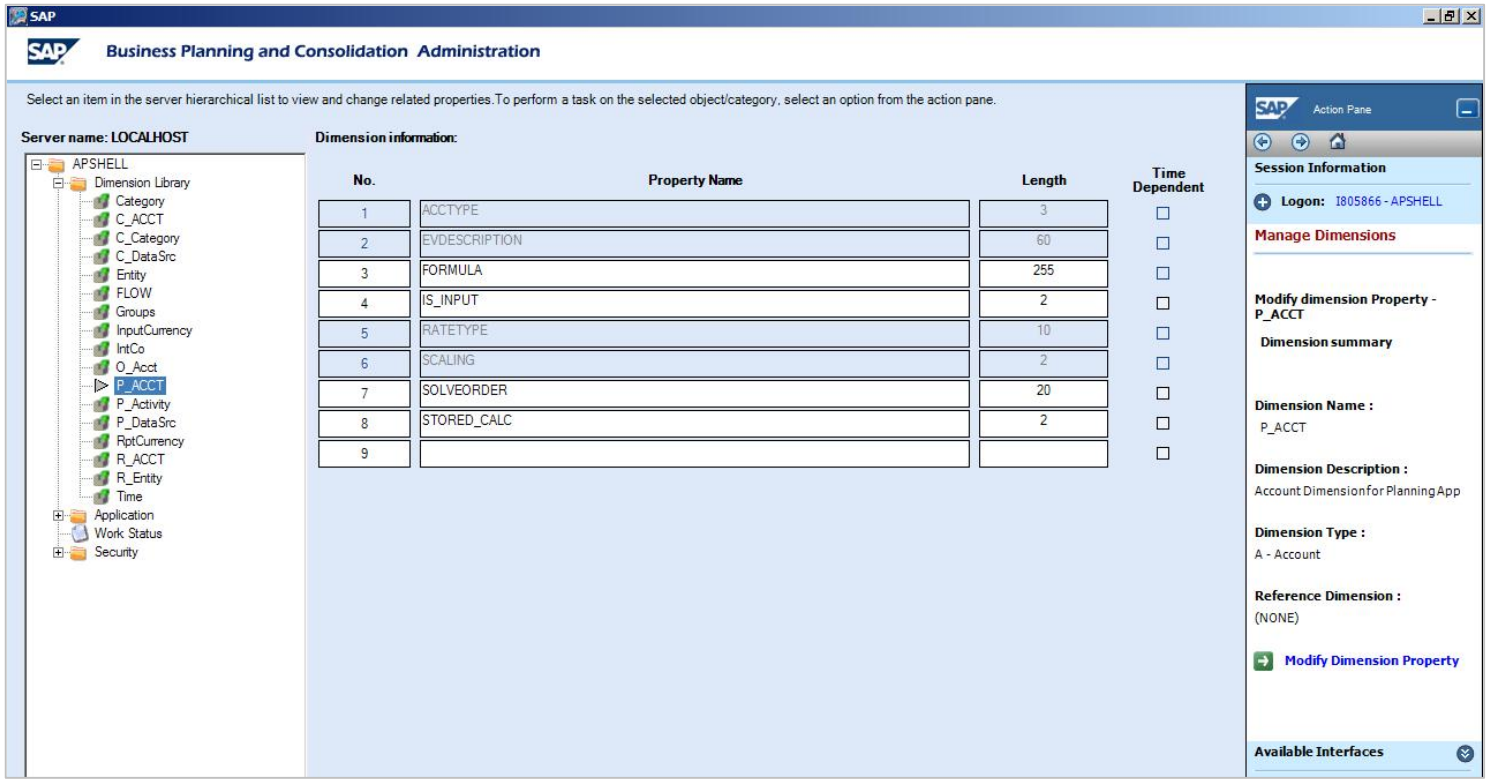
Dimension Member Formulas Overview

Dimension Member Formulas – are on-the-fly calculations defined on dimension members based on other member values.

- They are commonly used for calculated key figures, such as costs per unit, percent of sales, growth rates and other ratios.
- Dimension formulas should only be used for formulas that need to be calculated after aggregations (e.g. ratios).
- Dimension formulas should not be used on members that need to aggregate. We recommend that you use hierarchies, rather than formulas, to define aggregate (or sub-total) dimension member levels.
- Familiarity with the Multidimensional Expressions (MDX) language may be helpful when creating more complex dimension member formulas.
- However, fewer MDX keywords are supported in SAP NetWeaver Business Intelligence (BI) than were available in SAP Business Planning and Consolidation 5.x using SQL Server.

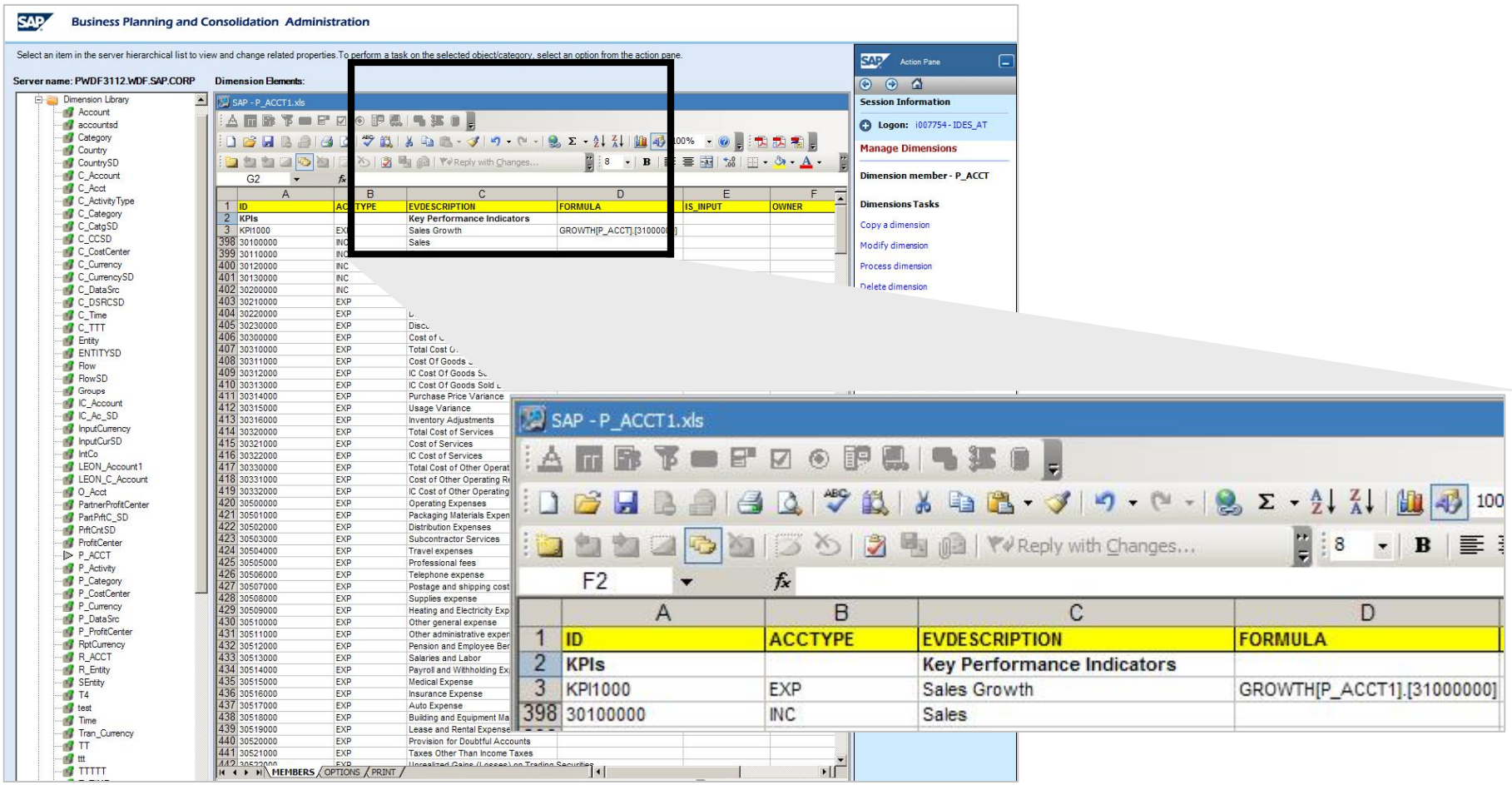
Dimension Member Formulas Prerequisites

- In order to use formulas in a dimension you must add a property called 'Formula' to the member sheet using the Manage Dimensions task in the SAP Business Planning and Consolidation Admin module.
- 'Formula' is a user-defined property, so you must define the length of the field. Enter a field length that is greater than 60 characters, and at least equal to the length of the longest expected formula.



Dimension Member Formulas Maintaining

Accessed from SAP Business Planning and Consolidation Administration in the Dimension Library subdirectory in the hierarchy of a particular Application Set → Maintain Dimension Members.



Dimension Member Formulas Syntax

Except for the Account dimension, the dimension name must always be specified.

Correct formulas:

[ACCOUNT].[Account1] / [ACCOUNT].[Account2]

Or

Account1 / Account2

And

[PRODUCT].[Product1] + [PRODUCT].[Product2]

Or (with use of optional Solve_Order property)

[PRODUCT].[Product1] + [PRODUCT].[Product2];Solve_Order=100

Tip → Make sure the dimension and dimension member case are correct!

Dimension Member Formulas
Solve_Order

Use this property to define the order in which calculated members are solved in the case of intersection with other calculated members. Zero is the highest priority

	A	B	C
1	**ID**	**EVDESCRIPTION**	**FORMULA**
2	Account1	Account 1	
3	Account2	Account 2	
4	Account3	Account 3	[ACCOUNT].[Account1]/[ACCOUNT].[Account2];SOLVE_ORDER=5
5			

In the event of a formula collision between the above members, the Account3 formula should take precedence over the Product3 formula because solve_order = 5 has higher priority than solve_order = 100.

	A	B	C
1	**ID**	**EVDESCRIPTION**	**FORMULA**
2	Product1	Product 1	
3	Product2	Product 2	
4	Product3	Product 3	[PRODUCT].[Product1]+[PRODUCT].[Product2];SOLVE_ORDER=100
5			

Dimension Member Formulas
Solve_Order Example

When Solve_Order is not used or is defined incorrectly:

A3= A1/A2 Then P3=P 1+P2

	Product1	Product2	Product3
Account1	120	130	250
Account2	5	5	10
Account3	24	26	50

1

2

24 + 26 = 50 but
250 / 10 <> 50
Incorrect result because account ratio is calculated prior to and over-ridden by the product addition formula.

When Solve_Order is used correctly:

P3= P1+P 2 Then A3= A1/A2

	Product1	Product2	Product3
Account1	120	130	250
Account2	5	5	10
Account3	24	26	25

2

1

250 / 10 = 25 but
24 + 26 <> 25
Correct result because account ratio is calculated after the product addition formula.

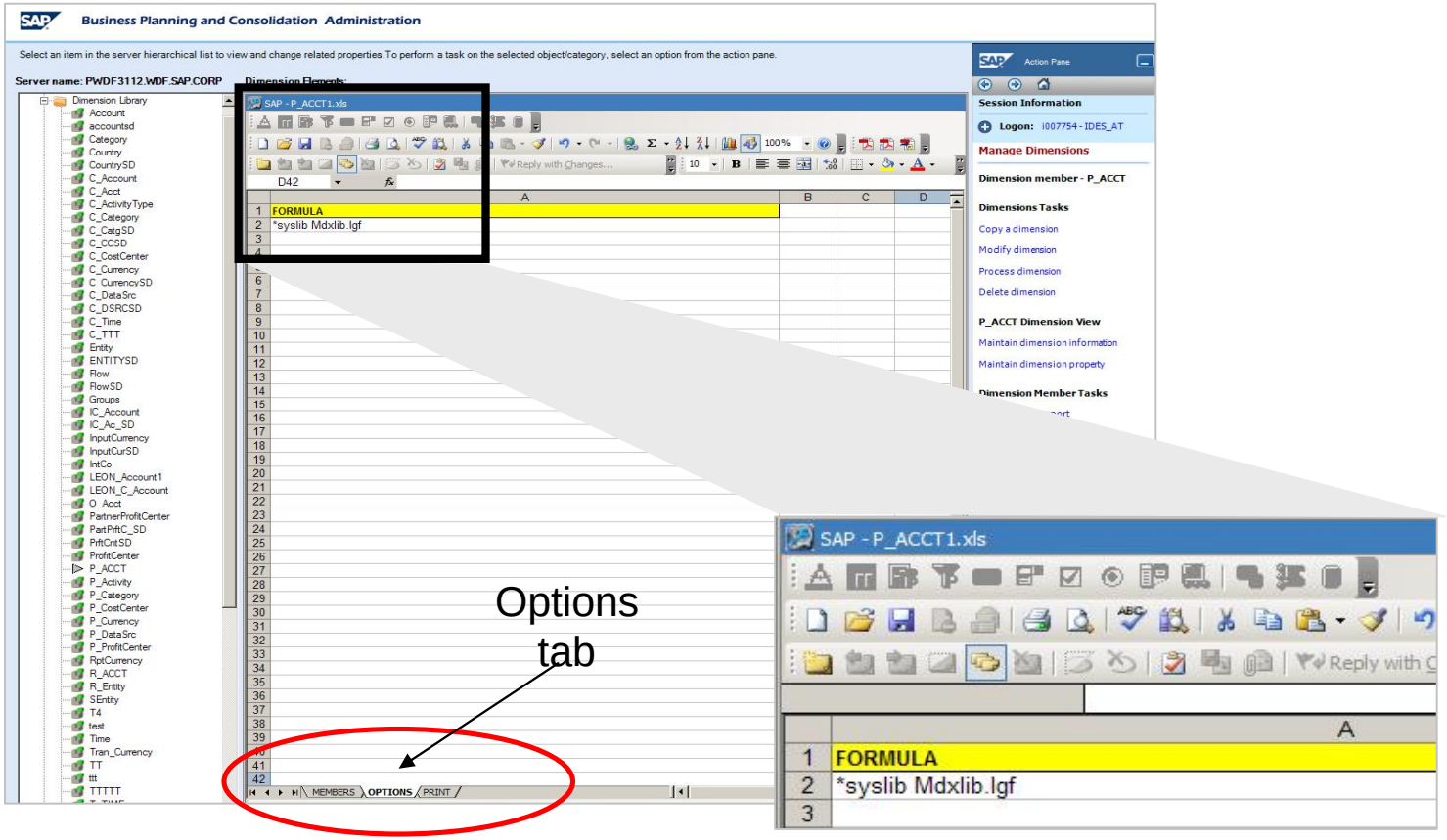
Dimension Member Formulas
The Library File

- SAP Business Planning and Consolidation provides a Library File with standard functions that you can use in your dimension member formulas.
- You can include the library file for use in dimension member formulas specified on the Options sheet of the Account dimension.
- Then you can define dimension formulas using the standard functions available from the library file.
- Syntax example:

	A	B	C
1	**ID**	**EVDESCRIPTION**	**FORMULA**
2	KPI1000	Sales Growth	GROWTH[ACCOUNT].[SALES]
3			

Dimension Member Formulas
Assigning the Library File

Assigned from SAP Business Planning and Consolidation Administration in the Dimension Library subdirectory in the hierarchy of a particular Application Set → Maintain Dimension Members.



Dimension Member Formulas Functions in the Library File

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.

The following SAP Business Planning and Consolidation 5.x standard functions are not included in SAP Business Planning and Consolidation 7.0, version for SAP NetWeaver:

- Profitability Ratios – Return on Assets, Return on Equity, Return on Common Equity, Cost of Goods Sold to Sales, Net Profit Margin, Gross Profit Margin, SG&A to Sales Ratio.
- Efficiency Analysis Ratios – Asset Turnover Ratio, A/R Turnover Ratio, Average Collection Period, A/P Turnover Ratio, Inventory Turnover Ratio, Average Age of Inventory, Sales to Total Assets Ratio, Days in Receivables, Days in Payables, Days in Inventory.
- Capital Structure Analysis Ratios – Debt Ratio, Debt to Equity Ratio, Interest Coverage Ratio, Debt Coverage Ratio.
- Capital Market Analysis Ratios – Earnings per Share, Price Earnings Ratio, Book Value per Share, Market to Book Ratio, Dividend Yield, Dividend Payout.

Dimension Member Formulas Pro and Cons

Pros

- Consistency - the same logic is applied to all members in all applications that use the dimension.

Cons

- Lack of flexibility - dimension logic applies to all levels in the dimension and in all applications using the dimension.
- If you want to apply a formula only to base level members or want a formula to be used in only one application you should use advanced logic.
- Creates load on the server.

Agenda

1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic: Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

Script Logic Overview

Script Logic – enables calculations on base-level cells that result in data stored within an application. The results are aggregated up the dimensional hierarchy intact, without being re-calculated at upper levels.

Some example of when to use a Script Logic formulas are: unit times price calculations, foreign currency translation, allocations, and others.

Script logic can be run in any of three ways:

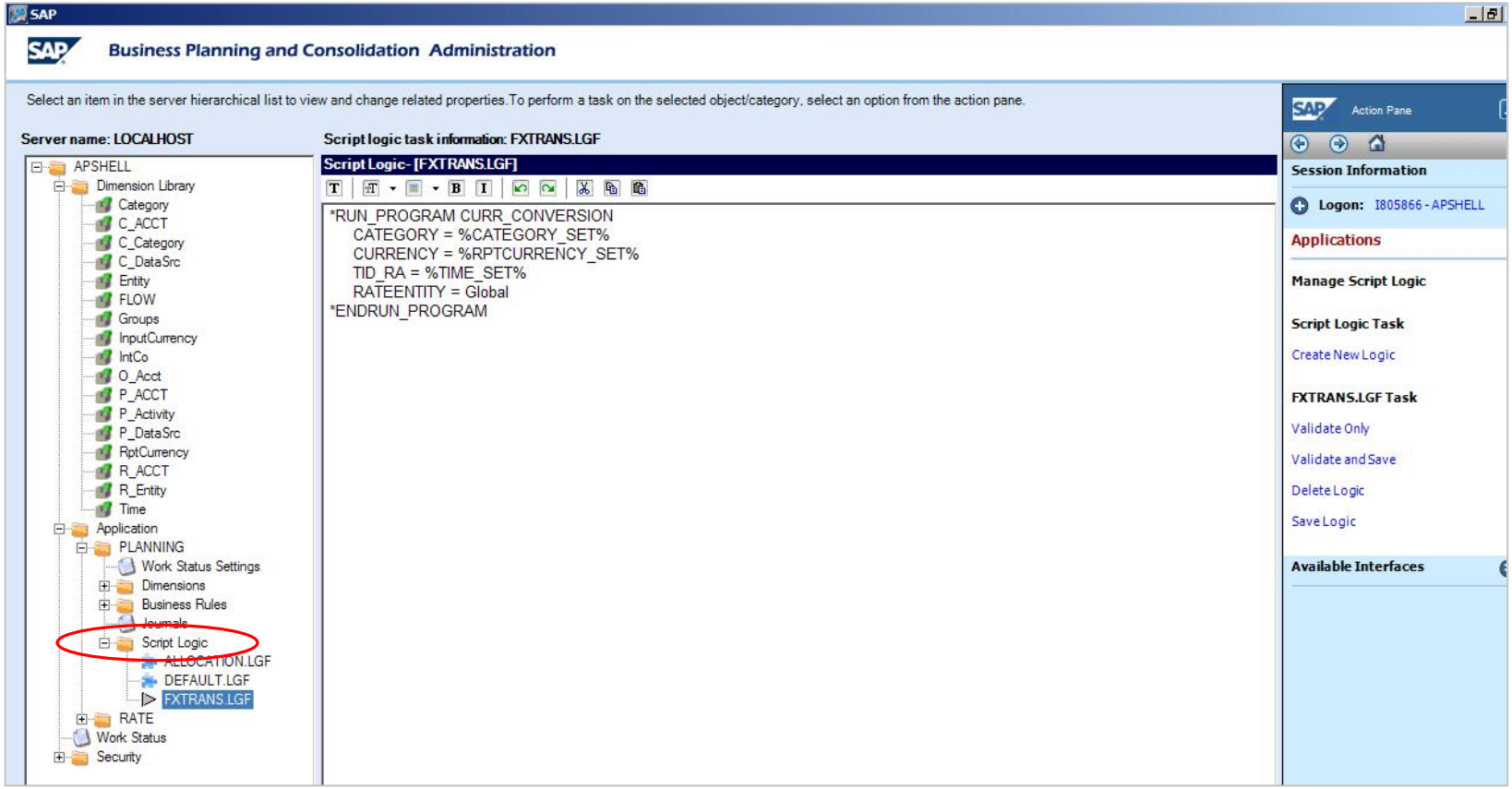
- Automatically after data is sent to the database from BPC for Excel using the Default.lgf file.
- After Journal data sends using either Default.lgf, or if present, Journal.lgf files.
- From Data Manager as a batch processing event to call Script Logic formulas.

Script Logic
How it works

- The Logic module:
  - Reads a specific data selection from the application.
  - Applies to it a set of user-defined formulas (stored in a logic file).
  - Derives the values.
  - Writes those values back to the application.
- Results are written directly to the application as base data, and are aggregated up the dimensional hierarchy as they are, without being recalculated at upper levels.
  - All dimension logic formulas are applied to these results.
  - This process allows you to perform such calculations as units times price because the formulas are applied only to those members specified in the logic file.

Script Logic Maintaining

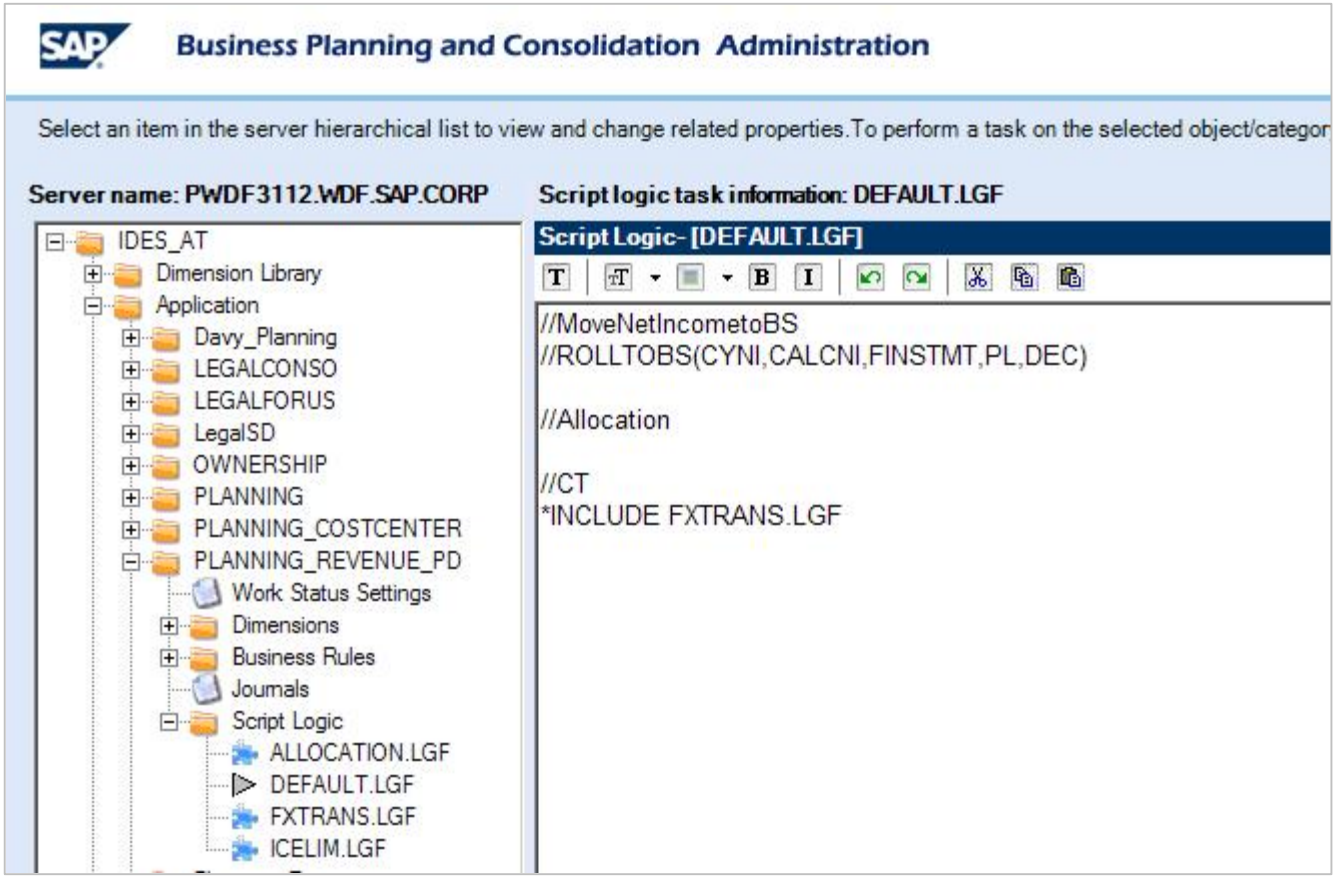
Accessed from SAP Business Planning and Consolidation Administration in the Script Logic subdirectory in the hierarchy of a particular Application.



Script Logic
Logic Files

Default Logic gets executed on every write-back to an application.

Use the *INCLUDE command to have one logic file call another logic file.



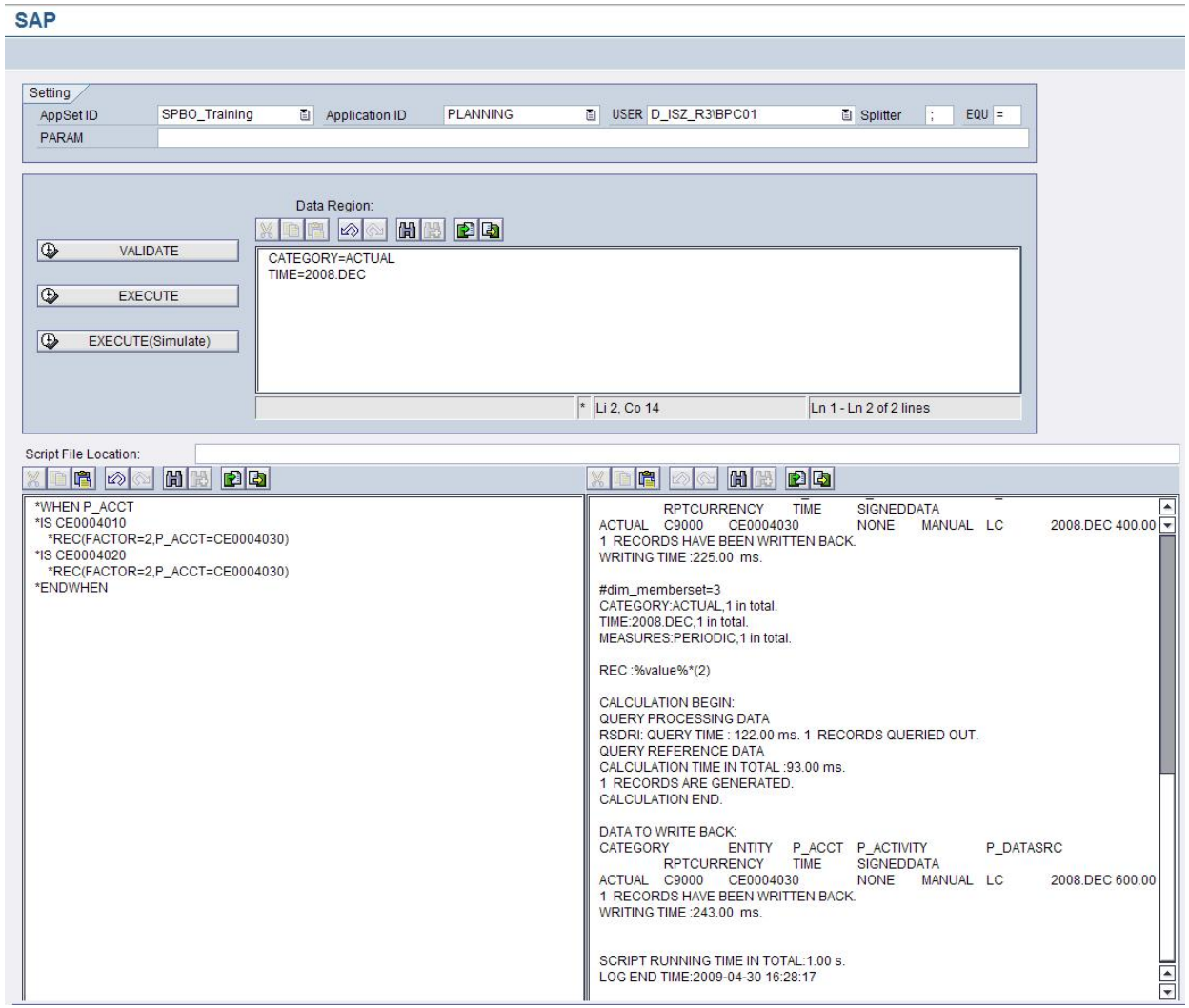
Creating a new logic script - Steps

Creating a new logic scripts involves the following 5 steps:

- Creating the logic script file (Admin Console)
- Compile the logic script (Admin Console)
- Create a Process Chain (Netweaver BI)
- Add the newly created package to the user interface (BPC for Excel)
- Set the run-time parameters of the Package (BPC for Excel), like the user prompt or the logic file that will be executed.

ABAP Program Tester

- UJK_SCRIPT_LOGIC_TESTER can be used to test and debug script logic statements



Script Logic Structure

Script Logic can be broken down into 3 components:

- Scoping- What am I running the data on?
  - Data manager scoping
  - Input schedule scoping
  - Scope control
- Body/Code – What do I want to do with the scoped records?
  - *IIf Statements
  - *REC
  - [#] based MDX statements
  - *WHEN / *IS/ *ENDWHEN
  - Allocation Logic
- Writing the record
  - *Commit

Script Logic
The Library File

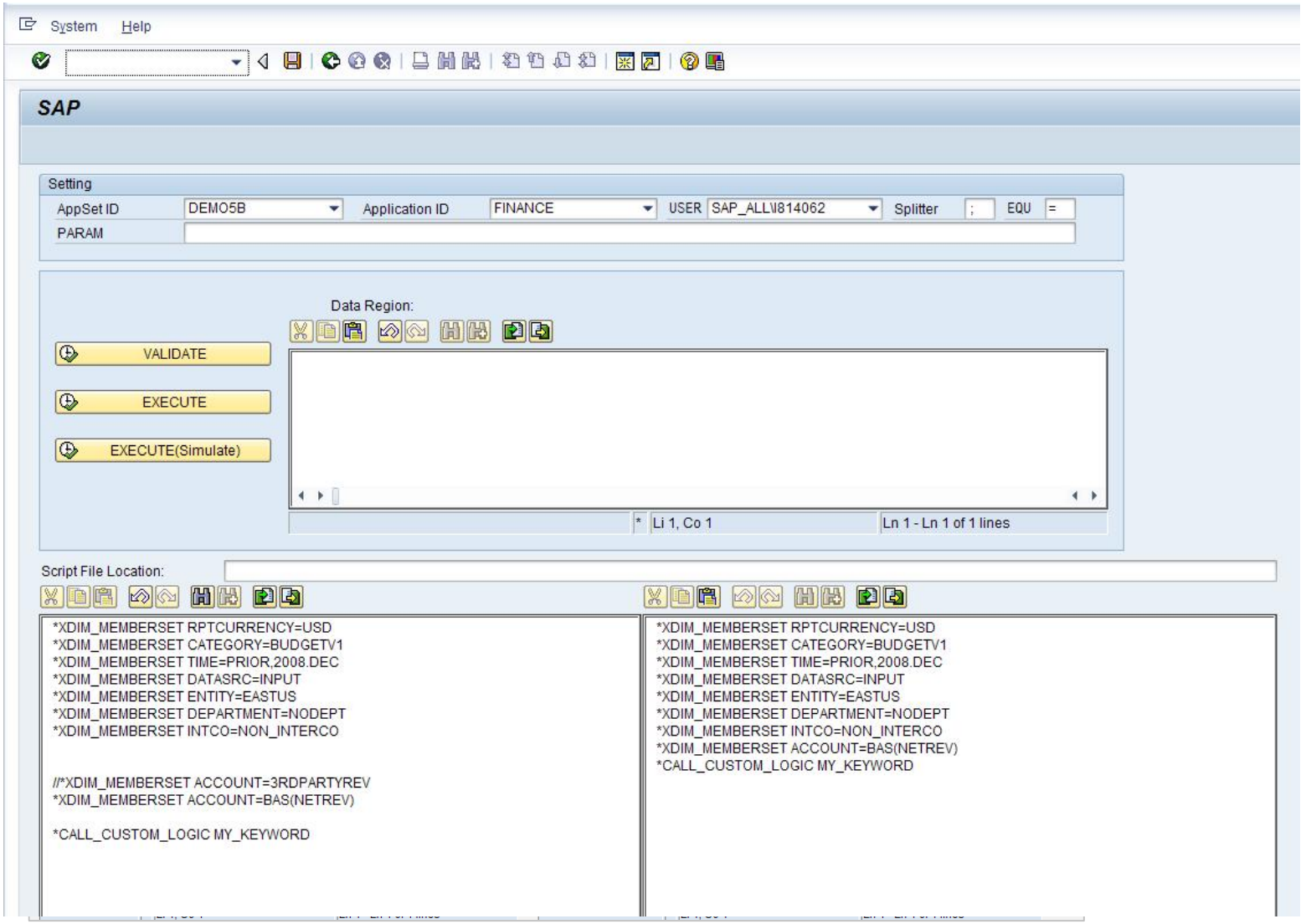
- SAP Business Planning and Consolidation provides a Library File with standard functions that you can use in your own Script Logic formulas.

- ApShell includes the default System_library.lgf which stores a library of standard functions.

- Can be called by using the INCLUDE function in your logic file.
  - The Logic module scans the library file for the appropriate formulas to use based on the information in the LGF file.

Script Logic
Script Logic Tester

Program UJK_SCRIPT_LOGIC_TESTER can be used to test and debug script logic.



Script Logic
Scope Source

Definition of the execution scope :

- Data Send (Default logic script)
  - Scope is derived by the sent data, A list of all the unique members in the sent data is used to derive the scope.
  - Scope can be overridden / modified / filtered with the *XDIM keywords
- Running a Data Manager Package
  - Scope is derived from the User Prompt for the dimensions that are part of the prompt. All non-calculated members are selected for the other dimensions.
  - Scope can be overridden / modified / filtered with the *XDIM keywords

Script Logic
Scope Source – Keywords

The following 3 keywords can be used to modify the scope:

*XDIM_MEMEBERSET

- Overwrites the scope for that dimension

*XDIM_ADDMEMBERSET

- Add members to the scope of that dimension

*XDIM_FILTER

- Filters the members of the scope of that dimension

*XDIM_MAXMEMBERS

- Specifies the maximum number of members that should be included in one query (per dimension)

*SELECT

- Assigns a list of members to a memory variable

Script Logic
*XDIM_MEMBERSET

*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
* DIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
*XDIM_MEMBERSET P_ACCT = CE0004220

*XDIM_MEMBERSET TIME = BAS(2007.TOTAL)
//*XDIM_MEMBERSET TIME = DEP(2007.Q3)
//*XDIM_MEMBERSET TIME = ALL(2007.TOTAL)

*WHEN P_ACCT
*IS CE0004220
*REC (EXPRESSION = 907)
*ENDWHEN
*COMMIT

Variant on XDIM_MEMBERSET
*XDIM_MEMBERSET {Dimension name} = BAS{Members Set}
*XDIM_MEMBERSET {Dimension name} = DEP{Members Set}
*XDIM_MEMBERSET {Dimension name} = ALL{Members Set}

Example:
*XDIM_MEMBERSET TIME = BAS(2007.TOTAL)
Selects all base level members of the 2007.TOTAL parent node

Script Logic
*XDIM_ADDMEMBERSET

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
*XDIM_MEMBERSET P_ACCT = CE0004220
*XDIM_MEMBERSET TIME = 2007.JAN
*XDIM_ADDMEMBERSET TIME = 2007.FEB

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

Adds to Member Selection
*XDIM_ADDMEMBERSET {dimension} = {members set}

Example:

User sends data only on Entity1 and the default logic contains:

*XDIM_ADDMEMBERSET ENTITY=Entity2

…Logic will be executed for Entity1 AND Entity2.

Script Logic
*XDIM_FILTER

*XDIM_FILTER

- This instruction does not replace the scope, but filters the existing scope.

Syntax

- *XDIM_FILTER {Dimension name} = {Members Set}

Example

- If the scope is France, Italy, USA, the instruction:
  - *XDIM_FILTER ENTITY= [ENTITY].Properties(“EUROPE”)=”Y”
  - …would limit the scope to the European Entities : France and Italy

Script Logic
*XDIM_MAXMEMBERS

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
*XDIM_MEMBERSET P_ACCT = CE0004220
//*XDIM_MEMBERSET TIME = 2007.JAN

*XDIM_MAXMEMBERS TIME = 4
// this forces the system to process the
//total number of records in
//smaller groups, in this case groups of four,
//rather than one big group

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

*COMMIT
```

Additional observations

- Significant performance enhancement by entering a large value for XDIM_MAXMEMBERS observed.
- ENITITY is automatically defaulted to XDIM_MAXMEMBERS ENTITY =1
- May want keep default XDIM_MAXMEMBERS ENTITY =1 when considering financial system implications.

Limits the maximum number of members in a query

***XDIM_MAXMEMBERS {dimension}= {max number of members}**

***XDIM_MAXMEMBERS Entity = 50**

Above instruction, if entities to process exceed the limit of 50 members, the logic will break the query into multiple queries of no more than 50 entities each.

PREFORMANCE RELATED

Script Logic
*SELECT

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1

*SELECT(%ENTSet%, "[ID]",  ENTITY, "[INTCO] ='I_C3000' ")
*XDIM_MEMBERSET Entity = %ENTSet%

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

*WHEN P_ACCT
*IS CE0004210
*REC (EXPRESSION = 10)
*IS CE0004220
*REC (EXPRESSION = 20)
*IS CE0004230
*REC (EXPRESSION = 30)
*ENDWHEN

*COMMIT
```

Dynamic Member Selection

*SELECT ({variable}, {What}, {From}, {Condition})

*SELECT(%ENTSet%, "[ID]", ENTITY, "[INTCO] ='I_C3000' ")

Script Logic
*SELECTCASE *CASE

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
*XDIM_MEMBERSET P_ACCT = CE0004220
*XDIM_MEMBERSET TIME = 2007.JAN

*SELECTCASE [P_acct].[CE0004210]
*CASE 5002
 [#CE0004220] = 9011
*CASEELSE
 [#CE0004230] = 9022
*ENDSELECT

*COMMIT
```

SELECTCASE does NOT allow nesting.

*SELECTCASE is a syntax friendly conditional statement similar to the IF>>Then>>Else statements used in other applications.

In this example: look at the record if the account is CE0004210, if the value of that record is 5002, then create a new records with account CE0004220 with a value of 9011, if the value of the record is NOT 5002, then create an alternative record with account CE0004230 with a value of 9022.

Script Logic
Calculations - *REC

The *REC instruction is used to generate a new database.

The *REC statement has two type of parameters:

- Calculation on the current value of the record (Using the keywords “EXPRESSION” and “FACTOR”
- Overwriting the members where the records should be generated (you usually do not want to overwrite your source record with the result, but write that result back to another member)

Example:

The following syntax multiplies the value of the record by 2 and writes-it back to the category budget

- *REC(FACTOR=2,CATEGORY=“BUDGET”)

The *REC statement can only be used within an *WHEN / *ENDWHEN structure

Script Logic
Calculations - *REC Factor / Expression

The amount to be posted on the new record can be derived from the existing value using ‘Factor’ and ‘Expression’ key words

- The ‘Factor’ key word multiplies the existing value by the specified factor
- The ‘Expression’ key word allows the input of a formula (can contain arithmetical operators, like ‘+’,’-’,’*’,’/’, fixed values of the value of the source value (%value%)
- A value from another account can be used with MDX Syntax, for example:
  EXPRESSION=%VALUE%/**[P_ACCT].[CE0004020]**
  EXPRESSION=%VALUE%/**([P_ACCT].[CE0652000],[P_ACTIVITY].[LABPRD])**
- [FACTOR | EXPRESSION = {Expression}]

Example

- *REC(FACTOR=1/2)
- … Current value is multiplied by 0.5
- *REC(EXPRESSION=%VALUE% + 500)
- … Adds 500 to the existing value

Script Logic
Calculations - *WHEN / *IS

The keywords *WHEN and *IS are used to filter the records that get processed

*WHEN Dimension

*IS “Value”,”Value2”

….

*ELSE

*ENDWHEN

A *When / *Endwhen structure can contain several *IS statements and one *ELSE statement

The *When keyword can use any property of a dimension, for example

“Entity.Currency”. The *”IS” keword can then filter on the values of that property

Script Logic
Calculations - *WHEN / *ENDWHEN Example

This example copies the values of the account sales from the category actual to budget and multiply the values by 2

```
*WHEN Category
*IS “Actual”
    *WHEN Account
    *IS “Sales”
        *REC(FACTOR = 2, Category=“Budget”)
    *ENDWHEN
*ENDWHEN
```

Script Logic
Calculations - Lookup values

For the “EXPRESSION” and “FACTOR” parameter, values of other members can also be used with the following two statements:

- *LOOKUP
  - Possibility to Lookup data in another application (for example for Currency Exchange Rates)
  - All values that are defined in the *LOOKUP are queried at the beginning of the logic execution and are then available throughout the logic execution
- *GET
  - Retrieves a value from memory, the value that is retrieved need to be part of the scope (no additional query is required to read the lookup values)
  - Only value from the same application can be retrieved

Script Logic
Calculations - *LOOKUP

*LOOKUP/*ENDLOOKUP

- *Lookup/*EndLookup - This set of instructions can be used in conjunction with a WHEN/ENDWHEN structure to retrieve (“lookup”) some other values that may be needed either in the calculation of the new value or to define some criteria to be evaluated.
- A relationship is defined between the current record being processed and the record to lookup
- Can be used in Factor\Expression or *WHEN
- Can lookup values in current application or different application in the same application set

Syntax

```
*LOOKUP {App}
        *DIM [{Lookup ID}:] {Dimension Name}=“Value”
                                                | {Calling Dimension Name} [. {Property}]
        [*DIM….]
*ENDLOOKUP
```

*Where:

- {App} is the name of the application from which the amounts are searched
- {DimensionName} is a dimension in the lookup app
- {CallingDimensionName} is a dimension in the current application
- {LookupID} is an optional identifier of the “looked-up” amount. This is only required when multiple values must be retrieved.

Script Logic
Calculations - *LOOKUP Example

Consider the currency calculation Value * Rate, where value is in the primary application and rate is in the supporting rate application

```
*LOOKUP RATE

*DIM RATESRC="RATECALC"

*DIM RATE=ACCOUNT.RATETYPE

*DIM SOURCECURR:INPUTCURRENCY=ENTITY.CURRENCY

*DIM USD:INPUTCURRENCY="USD"

*DIM EURO:INPUTCURRENCY="EURO"

*ENDLOOKUP
```

Script Logic
Calculations - Multi Application Logic

LOOKUP was used to read from another application previously. There is also a statement to write-back the result of a calculation to another application.

- *Destination_app = <<application name >>

The desination application needs to have the same structure than the source application, the keywords *Skip_dim, *Add_dim and *Rename_dim are not supported

Script Logic Calculations

There are two main ways to perform: A + B = C

- MDX Syntax:
  - [#C] = [A] + [B]
    - The calculated member is prefixed with a '#'
- SQL Syntax :
  - *WHEN ACCOUNT
  - *IS « A »
  - *REC(ACCOUNT=« C »)
  - *IS « B »
  - *REC(ACCOUNT=« C »)
  - *ENDWHEN
    - The calculated member is specified in the *REC()

The MDX syntax is straight forward, but may incur a performance penalty.

The SQL syntax involves the creation of two records: one with the value originally held by A and one with the value originally held by B. The two new records , assigned to account "C" by the REC statements, then aggregate up to the correct value.

Script Logic
Write-back – Overview

Once calculation have been performed, the results need to be sent back (or committed) to the database.

This operation is very resource intensive and should be limited to minimum number of commits.

The records are not written back directly to the write-back fact table, but are posted through the send governor. Depending on the send governor settings, a logic with several commit sections may take a long time to execute.

To avoid multiple write-backs, some calculations need to be rewritten to avoid dependency on other calculated accounts

Script Logic
Write-back - *COMMIT

- A « Commit » send the data back to the database
- A « Commit » may be required when there are several sections in a logic file and the second section need the output of the first section as input
- A « Commit » may also be required when the scope is different for different sections
- The number of « Commit » should be limited to a minimum, as this instruction has an important performance impact

Syntax

There are a variety of commit instructions that can be used:

*COMMIT

*COMMIT_EACH_MEMBER={dimname}

*COMMIT_EACH_LEVEL={dimname}

The behavior of the XDIM_MAXMEMBERS option for SQL logic is that the process will commit the generated result as they are generated by each single query.

When XDIM_MAXMEMBERS is used in MDX type logics, the logic query is broken in as many queries as required. However all resulting records are committed to the database in one lump at the end of the loop of queries.

- It may be preferable to perform a commit to the database after each individual query. This can be accomplished inserting the instruction:

*COMMIT_MAXMEMBERS

Script Logic Execution

- Advanced logic can be automatically invoked each time data is sent to the database.
  - Logic contained in the Default logic file is executed immediately after data is sent and the results can be seen in SAP Business Planning and Consolidation right away.
- Advanced logic can be run from Data Manager 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 generating the translated amounts in the reporting currencies.

Script Logic
Data Manager Package

These packages are executed on demand by the users and they will usually require some additional input by the users (like the period or the file they want to import).

The logic file cannot be executed by itself, it requires a Process Chain (Netweaver BI Process Chaine, transaction “RSPC”) to be setup first, that Process Chain contains a component that will then call a logic file. Each logic file can have it’s own Process Chain or a generic Process Chain can be created, where the “Logic Script” name is passed as parameter

Script Logic
Pros and Cons

Pros

- Real-time - allows for real-time calculations.
- Flexible - enables you to apply different formulas to different applications within an application set (dimension logic is applied to all applications using the dimension).
- Syntax options - you can use SQL or MDX syntax.

Agenda

1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic: Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

Scenario 1

- This works on a copy of Apshell.
  - The application has 7 dimensions: Category, entity, Account, Time, Datasrc, Rptcurrency, Activity
  - The code is as follows:

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
*XDIM_MEMBERSET TIME = 2007.JAN
*XDIM_MEMBERSET P_ACCT = CE0004220
*WHEN P_ACCT
*IS CE0004220
*REC (EXPRESSION = 22)
*ENDWHEN
*COMMIT
```

- What is the above code doing?
  - It is not reading what is currently the value is for the account
  - It is forcing the desired value

Scenario 2

- Let us change the code to:

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
*XDIM_MEMBERSET TIME = 2007.JAN
//*XDIM_MEMBERSET P_ACCT = CE0004220
*WHEN P_ACCT

*IS CE0004220
*REC (EXPRESSION = 33)

*IS CE0004230
*REC (EXPRESSION = 44)

*ENDWHEN
*COMMIT
```

- What is the above code doing?
- What would happen if we don't comment the account scoping?

Scenario 3

- Let us change the code to:

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
//*XDIM_MEMBERSET TIME = 2007.JAN
//*XDIM_MEMBERSET P_ACCT = CE0004220
*WHEN P_ACCT

*IS CE0004220
*REC (EXPRESSION = 33)

*IS CE0004230
*REC (EXPRESSION = 44)

*ENDWHEN
*COMMIT
```

- What does that change do?

Scenario 4

- Let us change the code to:

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
*XDIM_MEMBERSET TIME = 2007.JAN
//*XDIM_MEMBERSET P_ACCT = CE0004220
*WHEN P_ACCT

*IS CE0004220
*REC (FACTOR = 2)

*IS CE0004230
*REC (FACTOR = 3)

*ENDWHEN
*COMMIT
```

- What does that change do ?

Scenario 4

- Let us change the code to:

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
//*XDIM_MEMBERSET TIME = 2007.JAN
//*XDIM_MEMBERSET P_ACCT = CE0004220
*WHEN P_ACCT

*IS CE0004220
*REC (EXPRESSION = 10)

*IS CE0004220
*REC (FACTOR = 2)

*IS CE0004220
*REC (FACTOR = 3)

*ENDWHEN
*COMMIT
```

- What will be the end result?

Scenario 5

- Quiz:

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
//*XDIM_MEMBERSET TIME = 2007.JAN
//*XDIM_MEMBERSET P_ACCT = CE0004220
*WHEN P_ACCT

*IS CE0004220
*REC (FACTOR = 2)

*IS CE0004220
*REC (FACTOR = 3)

*IS CE0004220
*REC (EXPRESSION = 10)

*ENDWHEN
*COMMIT
```

- What will be the end result?

Scenario 6

- Let us change the code to:

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
*XDIM_MEMBERSET TIME as %SET% = 2007.JAN
*XDIM_MEMBERSET TIME = %SET%
//*XDIM_MEMBERSET P_ACCT = CE0004220
[#CE0004220] = 11
*COMMIT
```

- What is the advantage of this?

Scenario 7

- Let us change the code to:

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
//*XDIM_MEMBERSET TIME = 2007.JAN
//*XDIM_MEMBERSET TIME as %SET% = 2007.JAN
//*XDIM_MEMBERSET TIME = %SET%
*SELECT (%SET%, [ID], TIME, "[YEAR]='2007' and [CALC] = 'N'")
*XDIM_MEMBERSET TIME = %SET%

[#CE0004220] = 22

*COMMIT
```

- What is the advantage of this? How many records will this write?
  - Can you relate this to FOREACH?

Scenario 8

- Let us change the code to:

```
*XDIM_MEMBERSET Category = PLAN
*XDIM_MEMBERSET Entity = C3000
*XDIM_MEMBERSET P_DataSrc = MANUAL
*XDIM_MEMBERSET RptCurrency = LC
*XDIM_MEMBERSET P_Activity = EMPL1
//*XDIM_MEMBERSET TIME = 2007.JAN
//*XDIM_MEMBERSET TIME as %SET% = 2007.JAN
//*XDIM_MEMBERSET TIME = %SET%
*SELECT (%SET%, [ID], TIME, "[YEAR]='2007' and [CALC] = 'N'")
*XDIM_MEMBERSET TIME = %SET%

[#CE0004220] = ([CE0004210] * [CE0004230] ) * (3+4) * (-2)

*COMMIT
```

- What is the advantage of this? How many records will this write?

Scenario: Lookup

```
//*******************************************************************************
      //Example 'LOOKUP' data from the current or another application to perform common maths
      //*******************************************************************************************
      *XDIM_MEMBERSET RENTALACT=EXACTDAYSRATIO
      *XDIM_MEMBERSET MEASURES=PERIODIC
      *XDIM_MEMBERSET TIME=BAS(2008.Total)
      *XDIM_ADDMEMBERSET TIME=XXXX.MINP


      *LOOKUP PLANNING              // the Application you are looking up
      *DIM AR:RENTALACT="AREA"      // AR is the information being looked up
      *DIM AR:TIME="XXXX.MINP"      // everything after the : is the dimension members where the data are stored



      *DIM ST:RENTALACT="STARPM2PA"
      *DIM ST:TIME="XXXX.MINP"
      *DIM ST:RENTALOBJ="NOREBOBJ"
      *DIM ST:CONTRA="NOCONTRA"



      *DIM MEASURES="PERIODIC"
      *ENDLOOKUP


      *WHEN RENTALACT
      *IS EXACTDAYSRATIO
      *REC(EXPRESSION=%VALUE%*(LOOKUP(ST)*LOOKUP(AR)),RENTALACT="STATCHAR")



      *ENDWHEN

      *COMMIT
```

Scenario: Get Data

```
**************************************************************************

//Example 'getting' data from the current application to perform common maths

*XDIM_MEMBERSET ACCOUNT=UNITS,SALESPRICE,UNITCOGS
*XDIM_MEMBERSET MEASURES=PERIODIC
*XDIM_ADDMEMBERSET CUSTOMER=NOCUST


*WHEN ACCOUNT
    *IS UNITS
    *REC(EXPRESSION=%VALUE%*([ACCOUNT].[SALESPRICE],[CUSTOMER].[NOCUST]),
    ACCOUNT="REVENUE")
    *REC(EXPRESSION=%VALUE%*([ACCOUNT].[UNITCOGS],[CUSTOMER].[NOCUST]),A
    CCOUNT="COGS")
    *ENDWHEN


*COMMIT
```

Agenda

1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic: Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

Need for ‘VARV’ in Script Logic

- VARV is a very popular keyword from FOX in BPS/BI-IP
- VARV allows you to enter the values of dimensions (characteristics in BPS world) or numbers at run time
- This enables us to configure the function without hard coding master data values
- There is no similar functionality in BPC to allow entering such values at run time

Numeric type variable

- The simplest type of variable is the numeric type variable
- An example of this may be where the user wants to increase or decrease the plan value by a percentage
  - For instance, the user wants to increase the plan number by 10%
  - In this case, the user wants to enter ’10’ at run time and not hard code anywhere

- We can use the variables in data manager package to accomplish this
  - We can have a variable $PERCENT$ in the script logic
  - Then replace the value of that variable by a prompt %PERCENT% in the data manager package
  - The task that does the above step is: REPLACEPARAM,PERCT%EQU%%PERCT%

Variable for dimension members

- We can even use the same technique for variables for dimension members
- You can consider this to be similar to the characteristic type variables in BPS/BI-IP
- The data manager prompt can ask the end user to enter the dimension member.
  - For example, [P_ACCT].[#CE0004020] = [P_ACCT].[DIM_INP] in script logic
  - And %DIM_INP% in DM package will prompt user to enter the account dimension member at run time.
  - This variable can be used in conjunction with other variables in the same script logic

Variables for scoping

- We can use the same technique in the scoping section of script logic also
- We can even have dynamic scoping for script logic if we have variables in say *XDIM statements
  - For example *XDIM_MEMBERSET TIME = DIM_INP in script logic and a prompt %DIM_INP% in DM package will prompt the user to enter a time dimension member while running that script logic
  - If the user enters 2009.JAN, the package will run only for January 2009. If he/she enters BAS(2009.Q1), the package will run for Jan, Feb, Mar of 2009
  - Please note that the data manager package can have its own prompts for dimensions in %SELECTION% (entity, time, category)
  - In the above example if you enter 2009.JAN in the %DIM_INP% prompt and enter 2008.JAN in %SELECTION% prompt, then 2009.JAN will prevail

Automating this logic

- If for any reason, you want to automate this process where you don't want user to enter these values, you can create variants with the answer prompt
- If the dynamic script file is as follows:

```
PROMPT(SELECTINPUT,,,,"%ENTITY_DIM%,%CATEGORY_DIM%,%CURRENCY_DIM%,%TIME_DIM%")
PROMPT(TEXT,%SCRIPT_FILE%,"Choose Script Logic File",)
PROMPT(TEXT,%PERCT%,"Input W/S Percent in decimals",)
PROMPT(TEXT,%DIM_INP%,"Input Dimension Member for the script",)
```

- The answer prompt file can be:

```
%SELECTION%        ENTITY    CATEGORY        CURRENCY        TIME
%ENTITY_DIM%       <all>
%CATEGORY_DIM%     ACTUAL
%CURRENCY_DIM%     <all>
%TIME_DIM%  <all>
%SCRIPT_FILE%      DIMENSION_INPUT.LGF
%PERCT%    20
%DIM_INP%  CE0004010
```

- You can use this answer prompt in ujd_test_package to automate the process and/or debug the process

Agenda

1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic: Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

Custom BADIs in Script Logic Overview

Category
- Technical

Operation Type
- Control Flow

Usage:
- *CALL_CUSTOM_LOGIC

Description
- This instruction is used to call any custom ABAP programming you have written

Syntax

*CALL_CUSTOM_LOGIC <BADI name>

Examples:

*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

Custom BADIs in Script Logic Overview

Category
- Technical

Operation Type
- Control Flow

Usage:
- *START_BADI

Description
- This instruction is used to call any custom ABAP programming you have written *WITH PARAMETERS

Syntax

*START_BADI
    <key1> = <value1>
    <key2> = <value2>
*END_BADI

Examples:

*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

Optional Parameters

You can use the following optional parameters within a *START_BADI / *END_BADI instruction:

Query - Performs the default query. Valid values are On and Off; the default is On. Set Query to Off if you want to perform your own query.

Write - Automatically writes back the data. Valid values are On and Off; the default is On.

Example

```
*START_BADI TEST
QUERY=ON
WRITE=OFF
*END_BADI
```

Custom BADIs in Script Logic Overview

- Call Custom Logic
  - Can be used to call ABAP programs and parameters can be passed from the BPC Script Logic (it's like defining new key words)
- Script Logic Debugging
  - Custom Logic can be debugged
- If the BADI has a filter, then in the script logic before calling the BADI, the customer needs to provide the filter value
  - NOTE: To pass parameters to the BADI, you can use the *START_BADI / *END_BADI command
  - NOTE: The BADI will run on data that was filtered through the Data Manager prompts
- All BPC Script Logic within BPC 7.0 Obeys Concurrency Locking
  - BW Transaction RSPLSE

Agenda

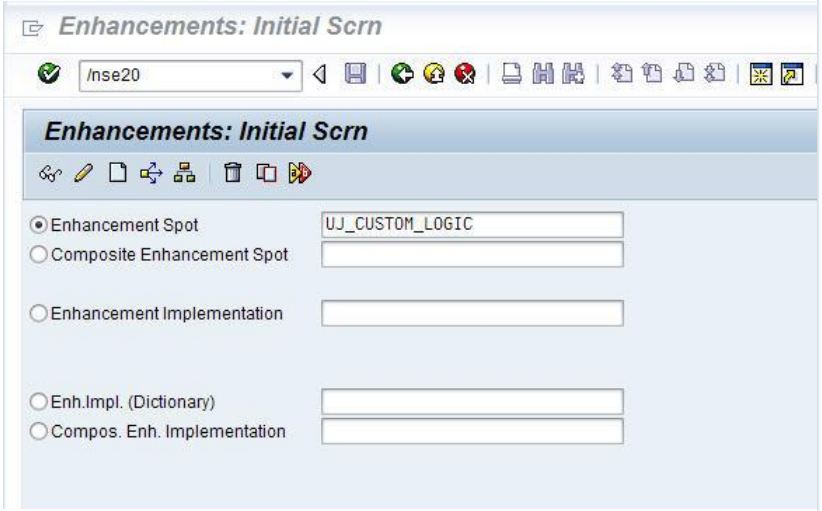
1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic: Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

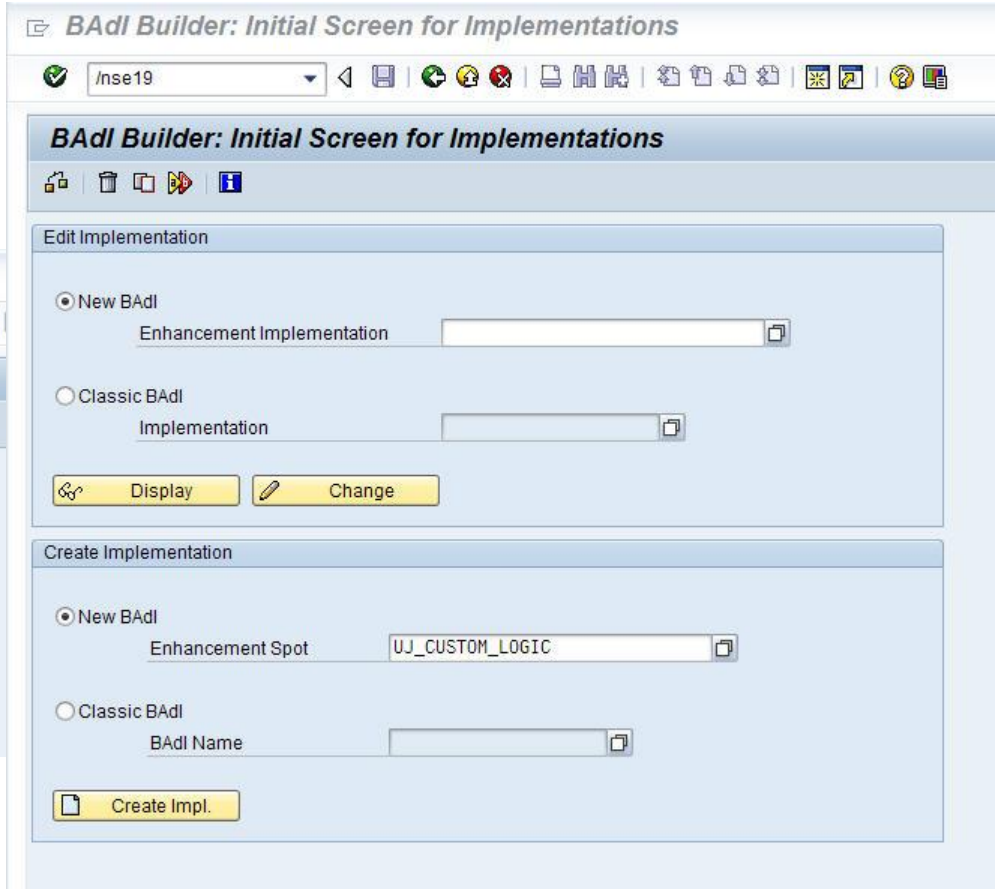
Create a Custom BADI

- To create a BPC Script Logic Custom BADI, you must first create an Enhancement Implementation of the Enhancement Spot: UJ_CUSTOM_LOGIC

TWO ENTRY POINTS:

(transactions SE19 and SE20)

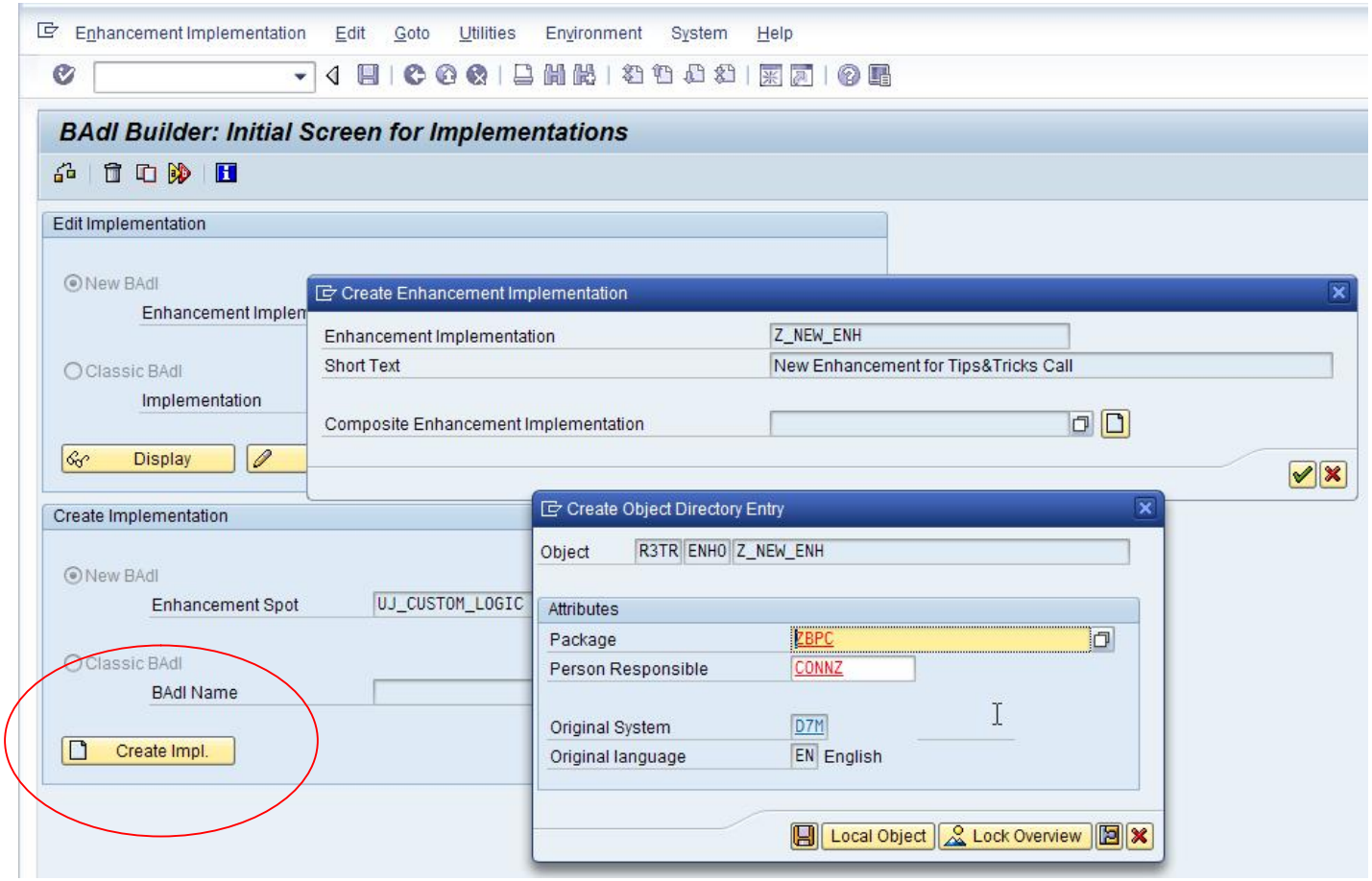




Create a Custom BADI

- Using Transaction SE19, you can create a new Enhancement Implementation like so:

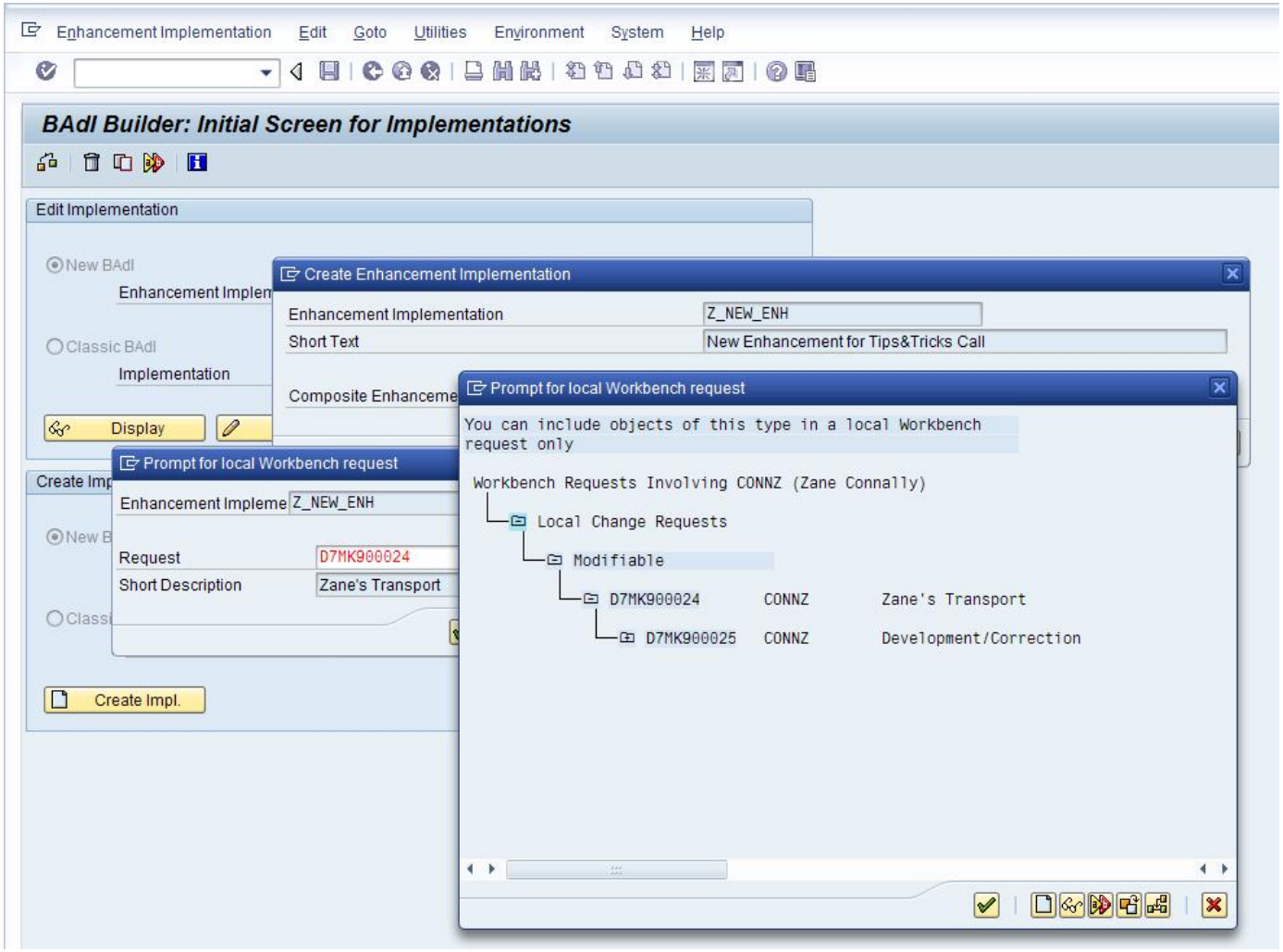
Name the new Enhancement Implementation and assign it to an ABAP Package:



Create a Custom BADI

- Using Transaction SE19, you can create a new Enhancement Implementation like so:

Assign the new Enhancement Implementation to a transport (Type Workbench Request) to move it through the QA/PROD landscape:

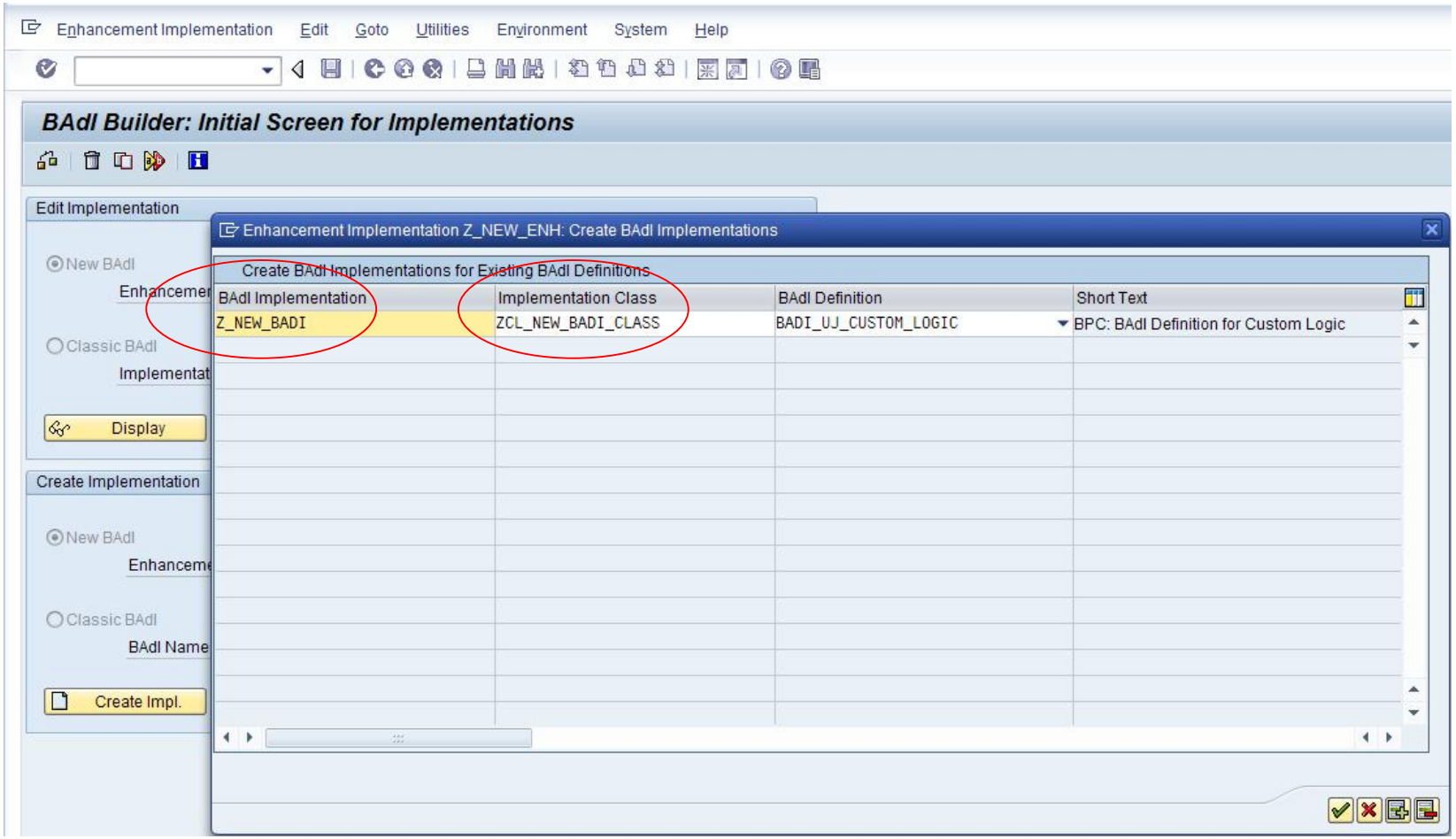


Create a Custom BADI

- Using Transaction SE19, the next step is to create the new BADI Implementation like so:

  Define the BADI Implementation Name and the ABAP Implementation Class Name behind the BADI

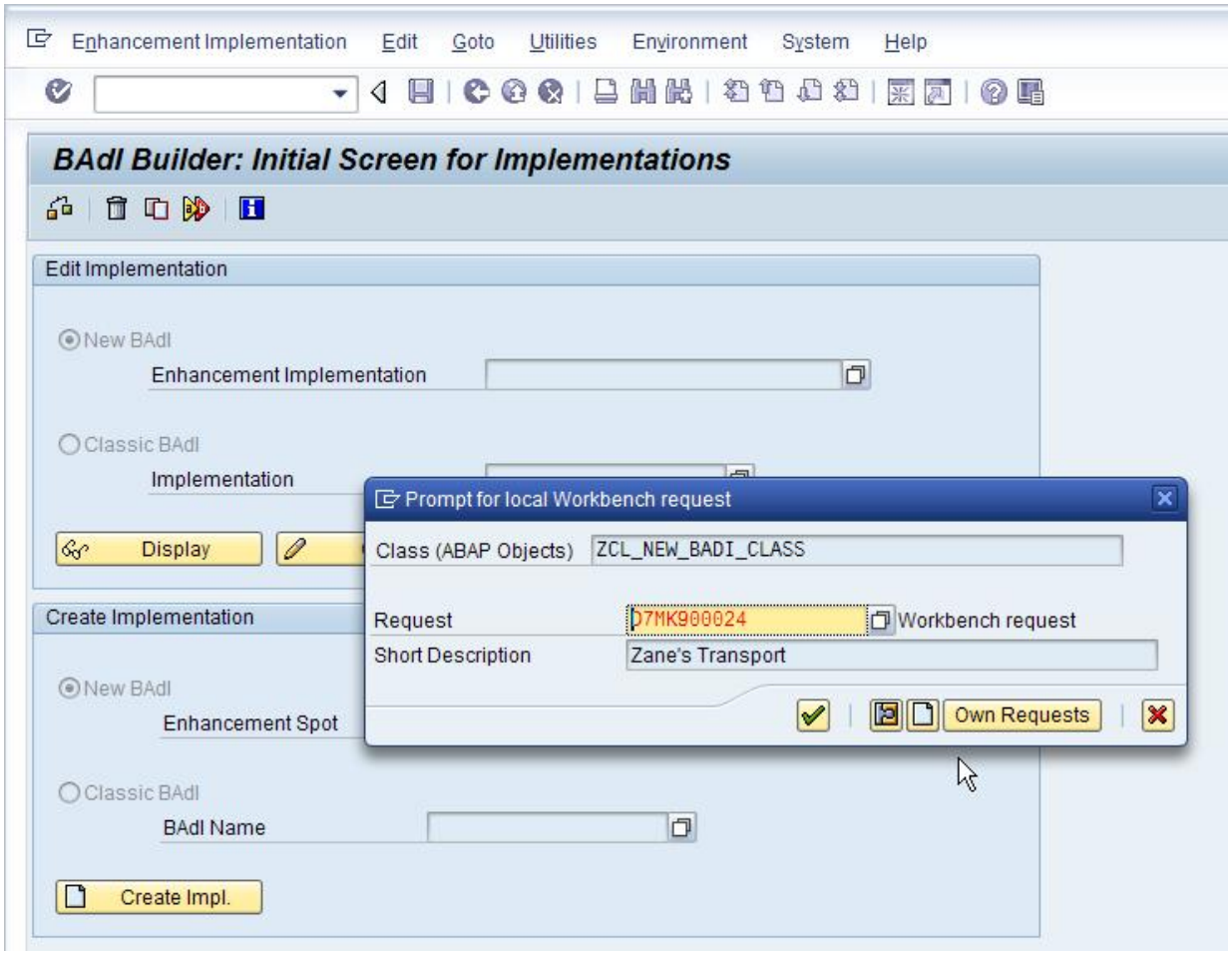
  (The Implementation Class is where you code your custom logic in ABAP)



Create a Custom BADI

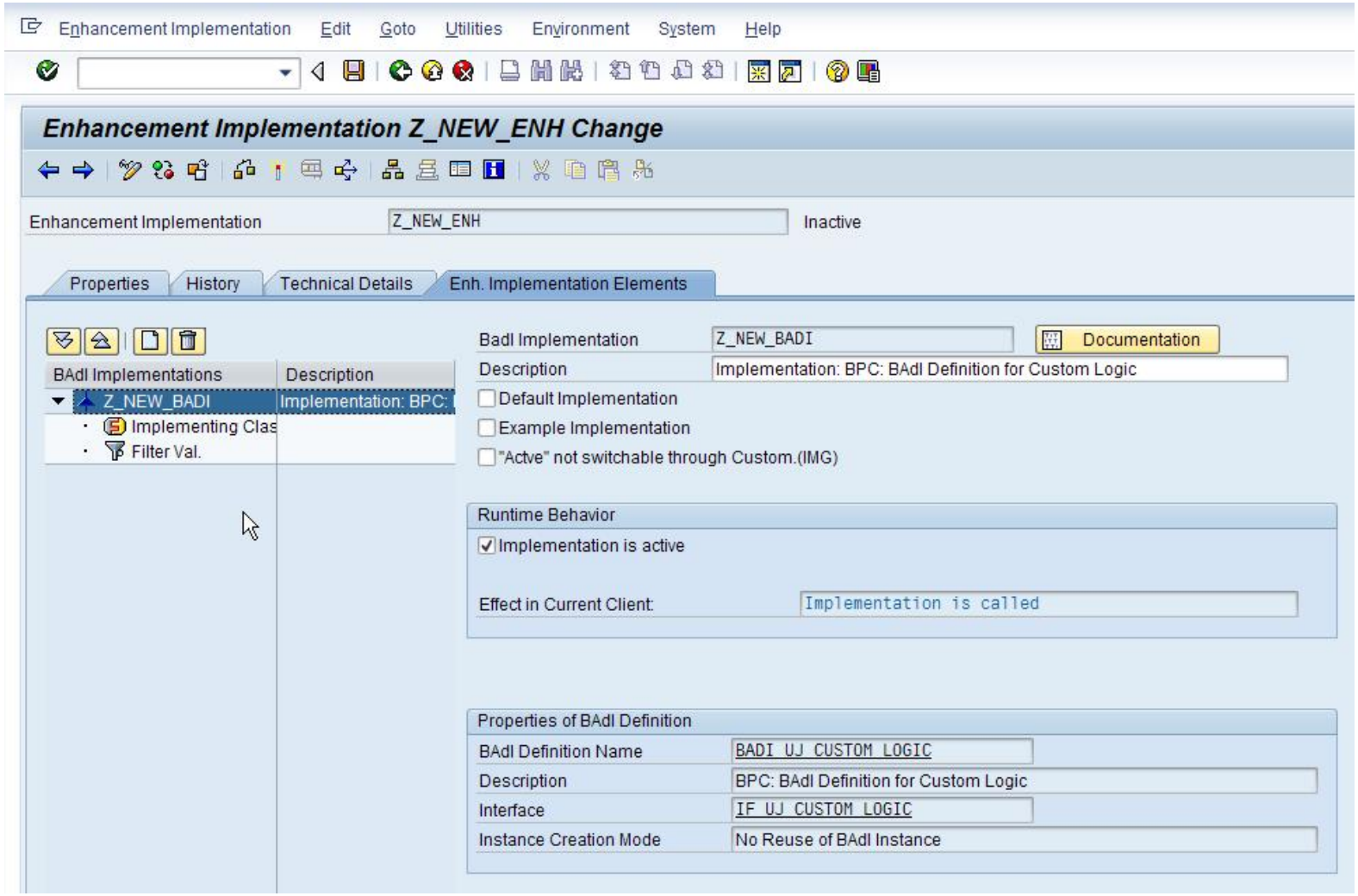
- Using Transaction SE19, the next step is to assign the ABAP Implementation Class to a transport:

Assign the new Implementation Class to a transport (Workbench Request) to move it through the QA/PROD landscape:



Create a Custom BADI

- Using Transaction SE19, you have just created the starting framework for your new BADI:
  - This screen allows you to modify your BADI configuration

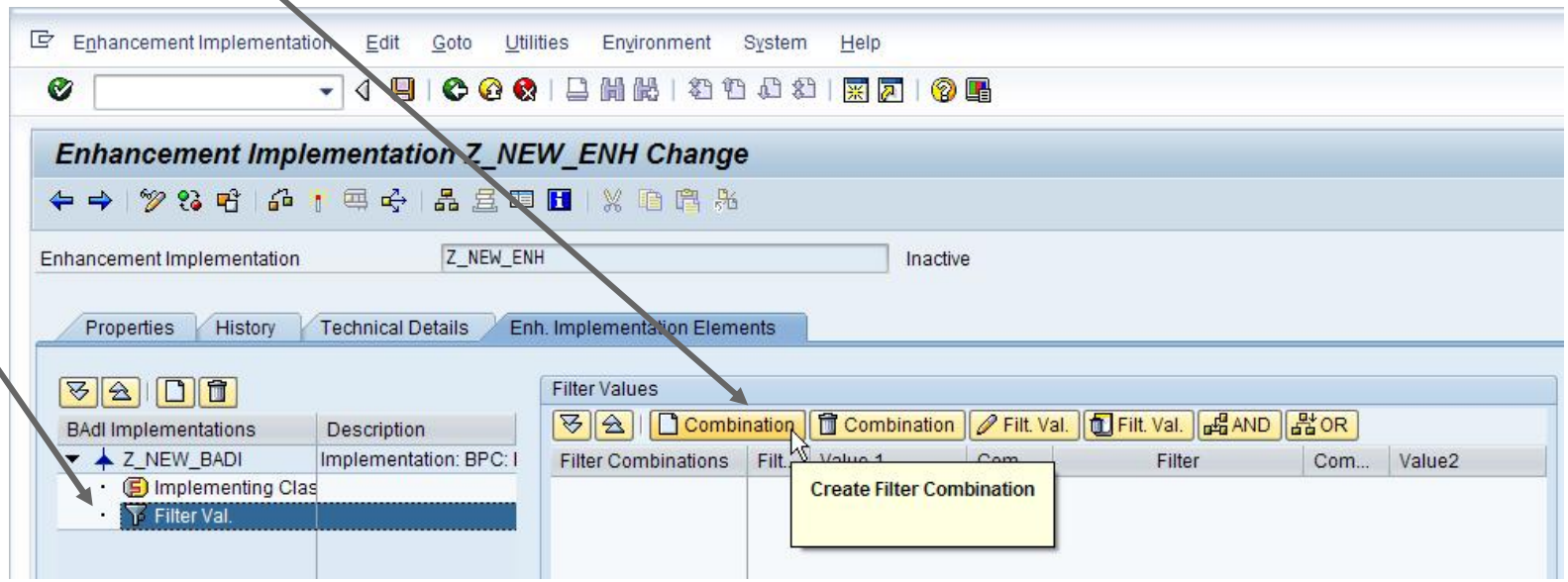


Create a Custom BADI

- Define a new KeyWord to call your newly created Script Logic Custom BADI:
  - The BADI Filter Val *is the same* as the Key Word you use inside BPC Script Logic. Using this KeyWord with “CALL_CUSTOM_LOGIC” or “START_BADI / END_BADI” will invoke your BADI code

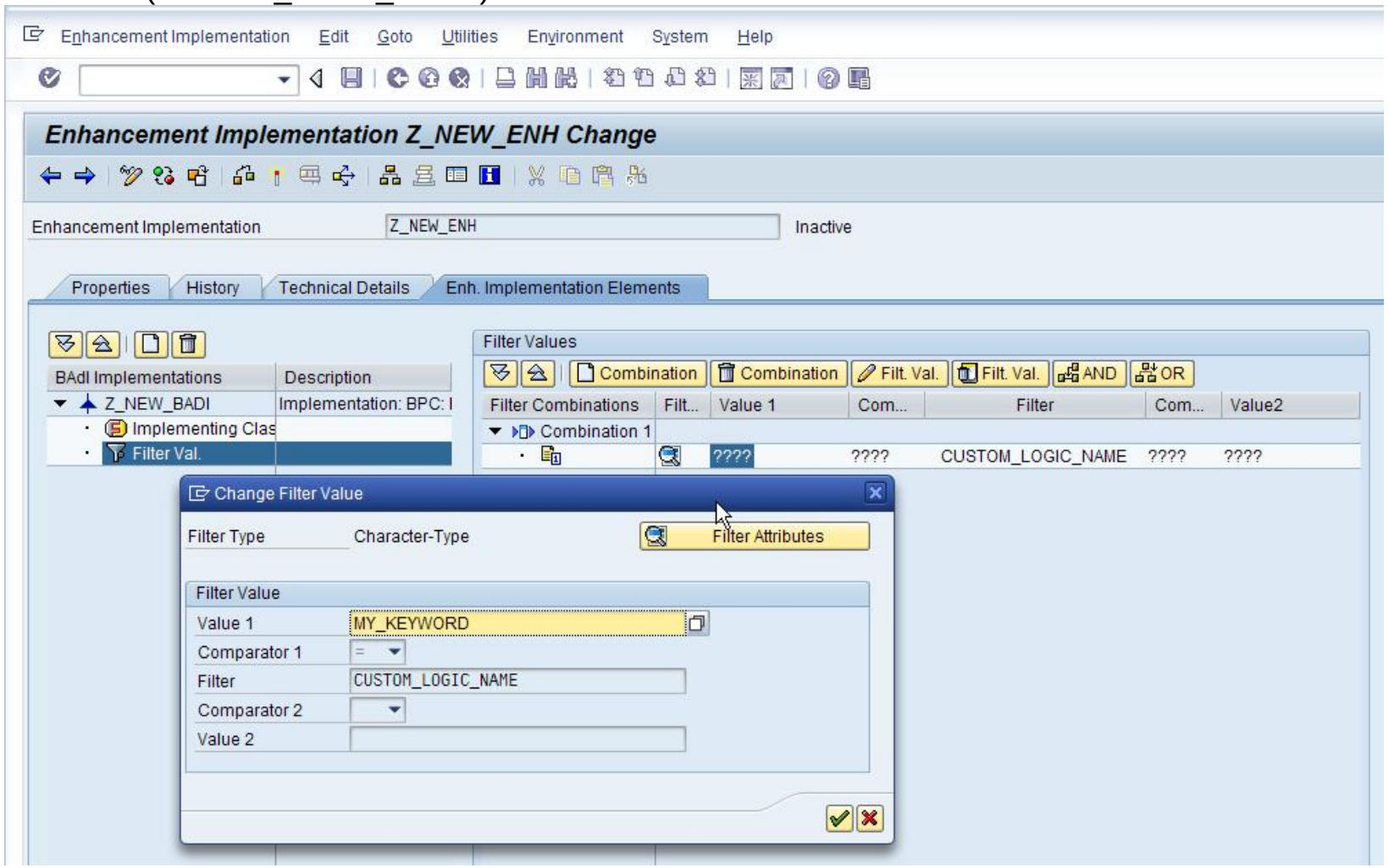
The BADI Filter Value *is* the Script Logic KeyWord

Click Here



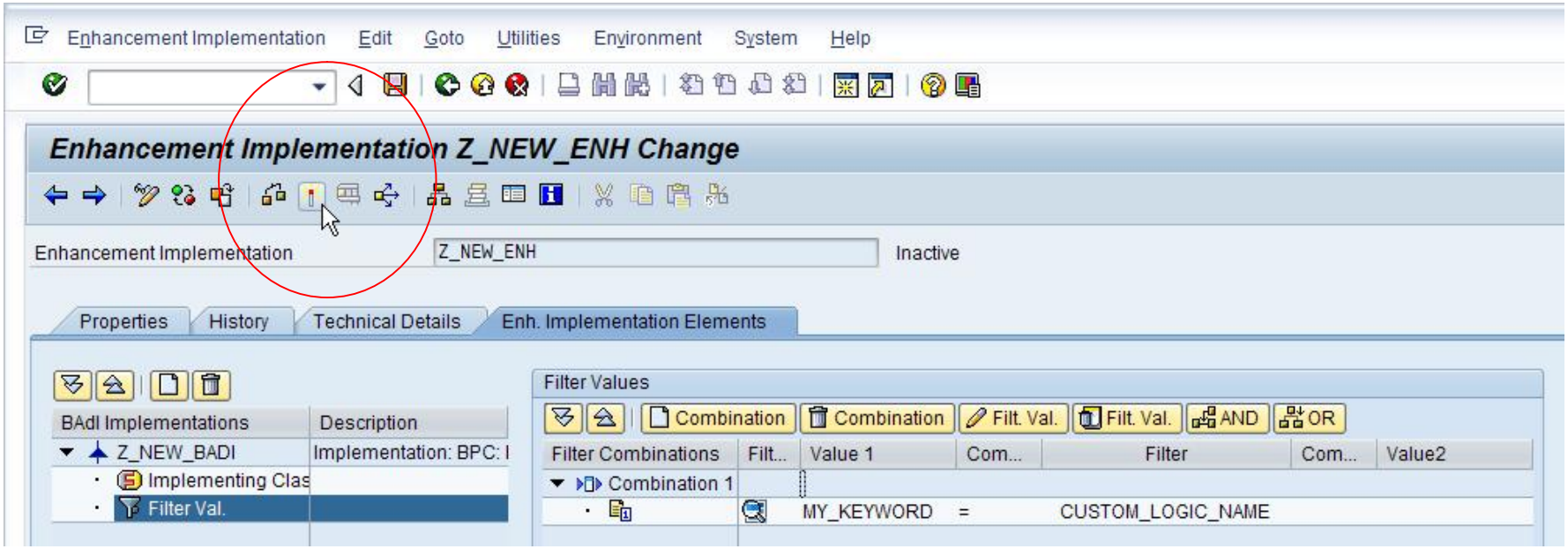
Create a Custom BADI

- Choose your KeyWord and set the value:
  - NOTE: The KeyWord should represent what the Script Logic Custom BADI actually does (ie CUST_CALC_ACCT)



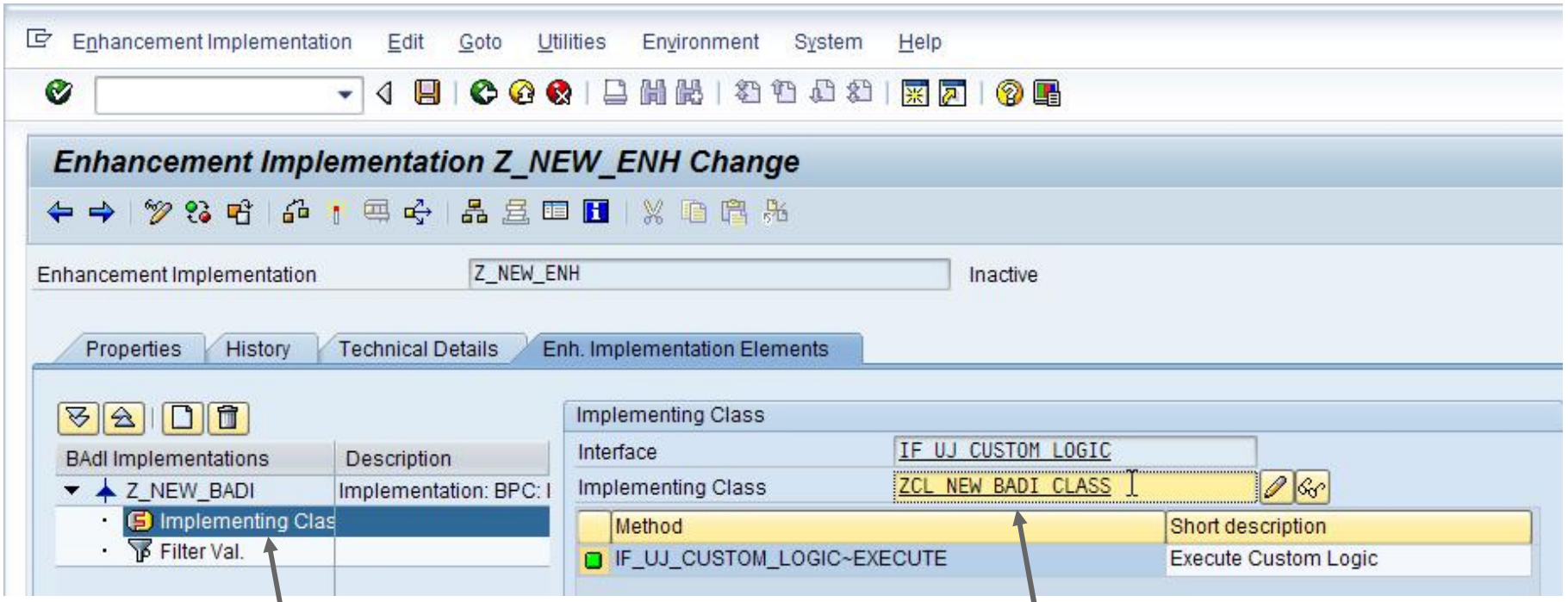
Create a Custom BADI

- ACTIVATE the BADI
  - NOTE: This activation is only for the BADI Enhancement Implementation – you must also activate your ABAP Implementing Class (see next few slides)



Create a Custom BADI

- ACTIVATE the Implementing Class
  - NOTE: Once you have activated your BADI you should thereafter only have to deal with your Implementing Class (unless you want to change the BADI properties – such as the Filter Value)



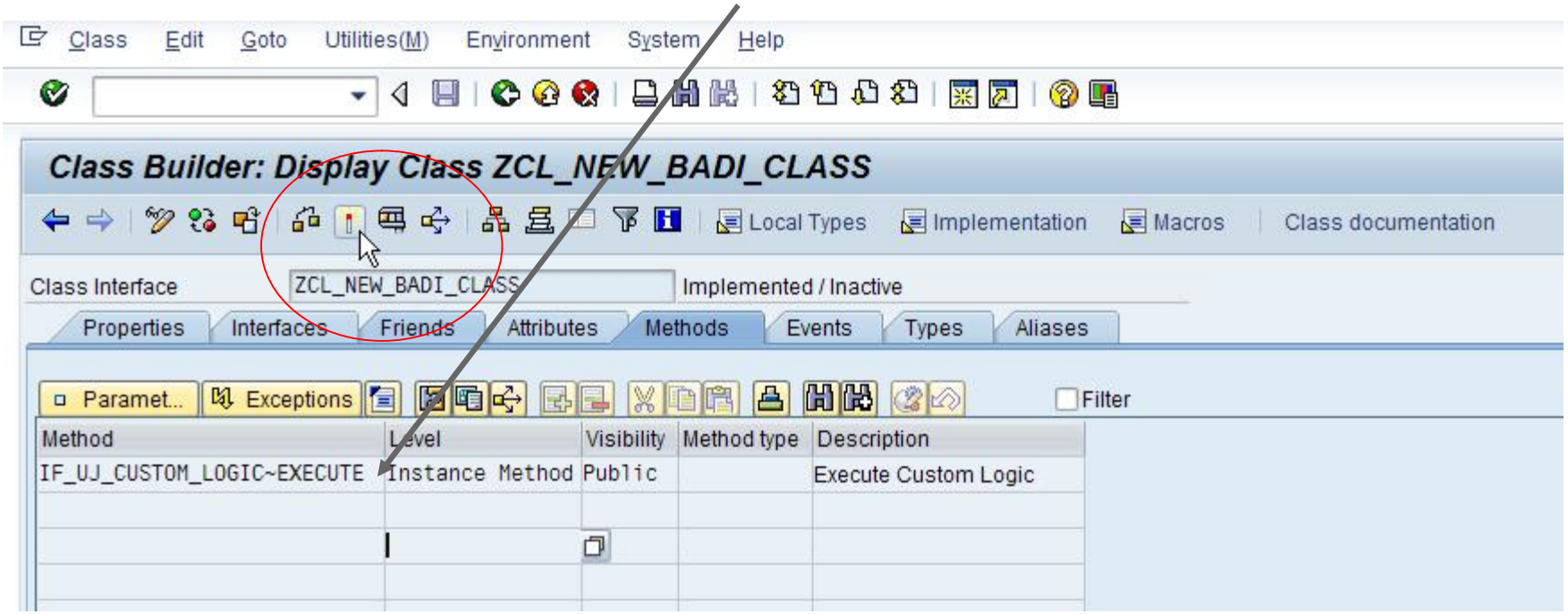
1st Click Here

Next, Double-Click the text (or go straight to Transaction SE24 and enter the name of your Implementing Class)

Create a Custom BADI

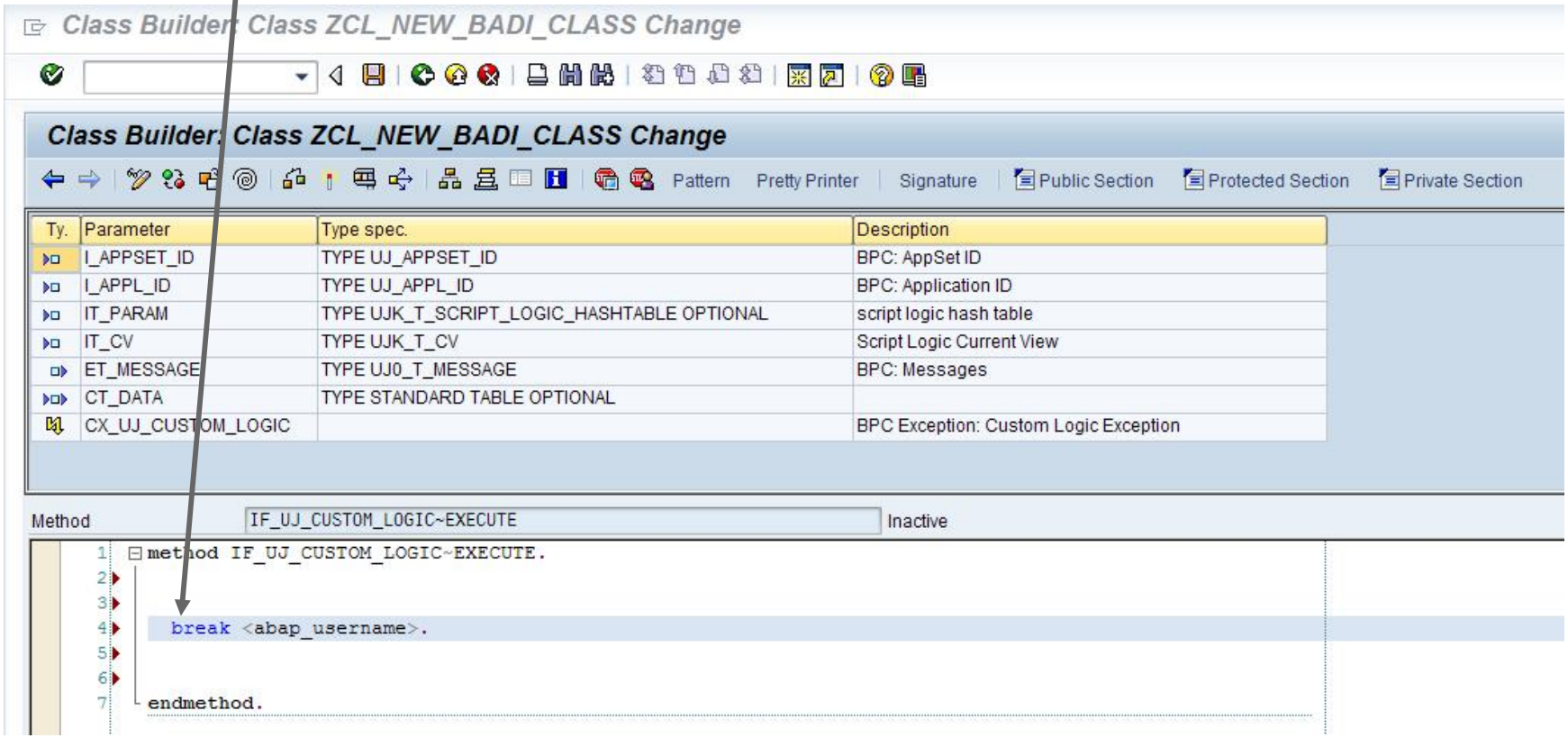
- ACTIVATE the Implementing Class
  - NOTE: You have all the options of ABAP Object-Oriented Class Methodologies (Inheritance, Encapsulation, Poly-Morphism, Public and Private Methods, Attributes and Types, etc)

Your Business-Specific Logic will go inside the Execute Method



Create a Custom BADI

- You are now ready to begin coding ABAP for your Script Logic Custom BADI:
  - NOTE: Beginners should set a break-point inside the execute method first and see what data will be passed to the BADI (filtering should be done in a careful and controlled manner).

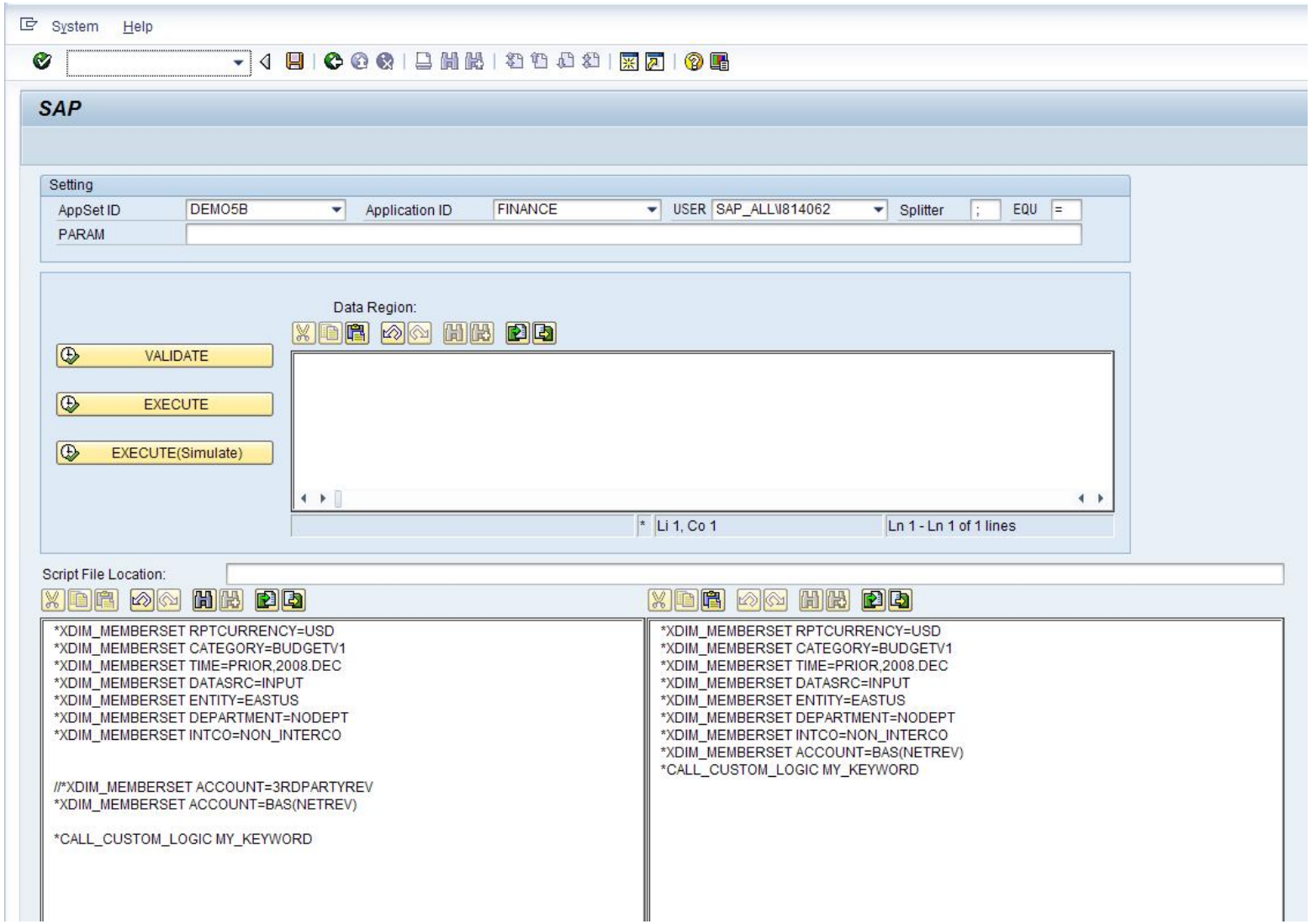


Agenda

1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic: Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

Testing and Debugging Custom BADIs

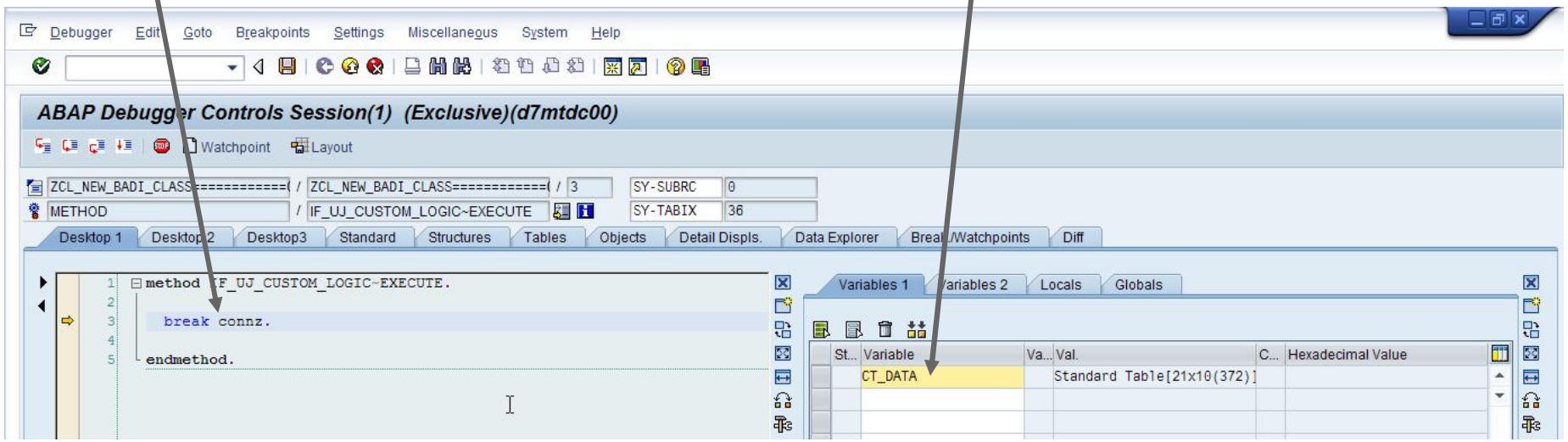
- Program UJK_SCRIPT_LOGIC_TESTER can be used to test and debug script logic statements



Testing and Debugging Custom BADIs

- Once you hit execute on the Script Logic Tester, the Debugger should pop up at the break point you set in the Implementing Class

CT_DATA is the table containing the retrieved data
NOTE: it is also where you insert new records to be written back to the Application



Agenda

1. What we are trying to do in these tips and tricks sessions
2. Introduction to different calculations methods
   - Worksheet Logic
   - Dimension Member Formulas
   - Script Logic Overview
3. Script Logic: Scoping Scenarios
4. User Defined Variables
5. Custom BADI Overview
6. Create a Custom BADI
7. Testing and Debugging Custom BADIs
8. Custom BADI Example

Custom BADI – Simple example to delete records

```
method if_uj_custom_logic~execute.
  * Declarations
    constants:
      c_bus          type string value 'BUSINESS',
      c_acct         type string value 'ACCOUNT',
      c_ind          type string value 'INDUSTRY',
      c_data         type string value 'SIGNEDDATA'.

    data:
      l_success          type abap_bool,
      l_log_msg          type string,
      l_ref              type ref to data,
      lines              type i,
      clines             type string,
      lo_cx_uj_null_obj  type ref to cx_uj_null_obj_ref,
      lo_cx_uj_static    type ref to cx_uj_static_check.

    field-symbols:
      <lt_data>       type standard table,
      <l_acct>        type any,
      <l_srcacct>     type any,
      <l_ind>         type any,
      <l_data>        type any,
      <ls_tmpline>    type any,
      <ls_line>       type any,
      <l_bus>         type any.
  * End Declarations

    break connz.
    cl_ujk_logger=>log( 'Starting Delete function.' ).

    describe table ct_data lines lines.
    clines = lines.
    clear l_log_msg.
    concatenate 'Read' clines 'lines for deletion in Application' p_appl_id into l_log_msg separated by space.
    cl_ujk_logger=>log( l_log_msg ).

    loop at ct_data assigning <ls_line>.
      assign component c_data of structure <ls_line> to <l_data>.
      clear <l_data>.
    endloop.

    cl_ujk_logger=>log( 'Ending Delete function.' ).

  endmethod.
```

Your Turn!

References

- For instructions about implementing an SAP Business Add-In, see the ABAP online help at:
  - http://help.sap.com/saphelp_nw70/helpdata/en/32/a83942424dac04e10000000a1550b0/content.htm.
- See each of the BPC version for NetWeaver Support Package's Documentation Addendum for new supported Script Logic keywords
- SDN BLOGS
  - Running BPC process chains within non-BPC process chains: https://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/12929
  - How to Pass Parameters to Custom Logic BADI using START_BADI https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/20f4252d-98ca-2b10-e689-f85085ae2d12

Q & A

Business Objects
an SAP company

SAP

Any ???

Copyright 2009 SAP AG
All rights reserved