3-3. Technical Information about Macro

Purpose

- Explain some important technical points of DP macro. (Can also be applied to SNP macro.)
- Common trouble shooting guides of DP macro (Can also be applied to SNP macro.)

Prerequisite

You know basic knowledge of macros, for example, how to create a macro in macro builder, and common functions in macro book.

Contents

- Purpose
- Prerequisite
  - Contents
  - <Important Settings of Macros>
    - Execution Level
      - Example
      - Comment
      - Auxiliary Key Figure
      - Change Mode
      - Data Source
    - <Macros being executed on planning book>
    - <Debug a macro>
      - Enter point to debug a macro
      - Input/Output parameters for the forms (both for macro and macro step)
      - General check points
    - <Background run and Interactive run>
    - <Some other information>
  - Related Content
  - Related Documents
    - Related SAP Notes/KBAs

<Important Settings of Macros>

- Execution Level

In macro builder (/sapapo/advm), create a new macro or double click a macro, you can set the execution level of the macro.

<table>
<thead>
<tr>
<th>Settings for Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Initialize Aux Table</td>
</tr>
<tr>
<td>Details Only</td>
</tr>
<tr>
<td>All Planning Objects</td>
</tr>
<tr>
<td>Certain Level(s)</td>
</tr>
<tr>
<td>Do Not Display Init. Values</td>
</tr>
<tr>
<td>DB Alerts in Display Mode</td>
</tr>
</tbody>
</table>

There are three settings: Details Only, All Planning Objects and Certain Level(s). However 'Details Only' does not mean the macro will be executed at the most detailed level (CVC level). Here which level is the 'Details' level depends on the current drill down situation on the planning book. The 'Details' level always refers to the most detailed level currently on planning book.

Example
For example, you have three characteristics in your MPOS: PROD, LOC, CUSTOMER. And you have several CVCs, like:

<table>
<thead>
<tr>
<th>Product</th>
<th>Location</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>L1</td>
<td>C1</td>
</tr>
<tr>
<td>P1</td>
<td>L2</td>
<td>C1</td>
</tr>
<tr>
<td>P2</td>
<td>L1</td>
<td>C2</td>
</tr>
</tbody>
</table>

When you select in the shuffler to show 'PROD'. You'll get two objects 'P1' and 'P2' in the shuffler. Load both the two entries. Now there's only one level on the planning book, which is the aggregated level for all 3 CVCs, represented by planning object 'TEMP1'. This is the current details level, and macro will be executed on this level only.

Now you drill down by 'PROD'. Then there're two levels on the planning book.
- LEVEL 0: TEMP1 (The aggregated planning object for all 3 CVCs)
- LEVEL 1: Aggregated level for P1 (TEMP2 - first 2 CVCs) and for P2 (TEMP3 - 3rd CVC)

Now LEVEL 1 becomes to 'Details' level, and LEVEL 0 is the aggregation level 1.

If macro is set to be executed at 'Details Only' level, the macro will be executed for TEMP2 and TEMP3, but not for TEMP1. If macro is set to be executed at 'ALL Planning Objects', macro will be executed for TEMP1, TEMP2 and TEMP3.

If you continue to drill down by 'LOC', you'll have 3 levels on the planning book.
- LEVEL 0: TEMP1 (The aggregated planning object for all 3 CVCs)
- LEVEL 1: Aggregated level for P1 (TEMP2 - first 2 CVCs) and for P2 (TEMP3 - 3rd CVC)
- LEVEL 2: P1/L1 (1st CVC), P1/L2 (2nd CVC) and P2/L1 (3rd CVC)

In the debugger, planning objects on aggregated level are assigned with a temporary GUID like 'TMP0000000000000000002', while planning objects at CVC level will have the real GUID, which corresponds to the internal GUID in /SAPAPO/MC62. Below shows the situation in the debugger. In the MPOS, there're four characteristics. If CHAR_VALUES contains less than four entries, the planning object is on aggregated level, otherwise it's on details level. (Real details to CVC level, instead of the details level of the planning book.)

Comment

- The planning objects like "TMPXXXX" is only for DP scenario. In SNP, since planning objects to be processed always belong to one of the SNP standard aggregates, they always have the specific GUID from SNP master data table. For example, if the planning objects belong to aggregate "9AMALO", you'll find the corresponding GUID in SNP master data table /SAPAPO/MATLOC. For SNP master data table of other SNP aggregates, please refer to note 1331576.
- The above content is for interactive macro run. In case of background macro job, macro will be executed according to the "aggregation level" set in the DP mass processing job (t-code: /SAPAPO/MC8E), ignoring the "Execution Level" set in macro builder.

- **Auxiliary Key Figure**

You can add auxiliary key figure into macros by add 'auxiliary table' elements, for holding temporary data during macro calculation. Auxiliary key figures can be used as normal key figures in the macro, however their value is not saved in liveCache or database. It only
exists during run time.

The most noticeable problem about auxiliary key figure is that, auxiliary key figure is saved in a global internal table during run time, which is not specific to ANY planning object, which means if you run the macro for multiple planning objects, the result for previous planning object may affect the next planning object. (This point in KBA 1623462 with an easy example.) So if you have the problem about "Macro result is different between running on single planning object and running on multiple planning objects", and you have auxiliary key figures in your macro, you'll need to first consider this point.

In order to eliminate the unwanted result caused by auxiliary key figure value of previous planning object, sometimes you'll need to clear out the auxiliary key figure. There're several ways to do this. The first one is recommended, since it's under user's control.

1. Clear the auxiliary key figure explicitly in the macro, by setting all buckets to zero.
2. Unset "Do Not Initialize Auxiliary Table" flag in the attributes of macro. This will clear all auxiliary key figures before the macro run. If you have many auxiliary key figures in use in the macro, sometimes you may want to clear some of them but not all. In this situation, this flag is not good to use. (*Notice this setting can only be changed when you create a new macro, or no auxiliary key figure in use in the macro. For the existed macro with auxiliary key figure in use, if you want to change this setting, you have to create a new macro and copy all the content to it.)

3. Use macro function HELPTAB_CLEAR(). This function will clear the value in all auxiliary key figures, comparing to 2), at any places in a macro.

This problem is significant also in SNP, in that when you load for example multiple location/products into planning book, some key figures populated by macros, like "stock on hand", "safety stock", may have problems. In this case, it is not advisable to adjust macros as above for DP, since most macros in SNP are SAP standard delivered. Instead you'll need to use SAP standard delivered SNP planning book 9ASNP94_INTERACT (or its copy) to work with multiple planning objects in SNP. In this planning book, there're complicated drill down and drill up macros to make sure the result for multiple planning objects is also consistent. However the performance may get affected by those macros.

Another issue with auxiliary key figure is with copying/importing macro from another macro book. In this process, auxiliary key figures will be copied by index. So problem may happen if you already have auxiliary key figures in both source and target macro book, but they're actually different (in description). So you'll need to take care such situation by reconstructing the auxiliary key figure in the target macro book after importing. See KBA 1805579.

• Change Mode

When you create a target row (or column) in the macro, you'll need to pay attention to the 'Change Mode' setting of it. In most cases we use the default 'Value Change' setting, which means we want to set the quantity value in the cell, but in some cases, we'll need to use 'Attribute Change' for some functions, like setting background/foreground color, visibility or change mode of a cell/row/column. So when you try to set some 'attributes' instead of values of the planning book with provided macro functions and it does not work, first check whether change mode is set correctly or not.

Except for 'Value Change' and 'Attribute Change', there're also some other options for this field (some of them are only available in higher releases after SCM7.0). The last four of them are only for the key figures that are set as ‘Fixable’ instead of ‘Simple’ in the planning area settings.

• Redisaggregation
  This setting forces the re-disaggregation to detailed level (CVC level) after data is set to this key figure. If you only use 'Value Change' here, the data won't be disaggregated until it is saved to liveCache.
• Initialization
In case that you have "differentiate between zero and initial" functionality in the system (higher that SCM5.0), you may want to set a key figure to 'initial' instead of 'zero'. To achieve this, you use this functionality. SAP note 1223998 and note1068603 explain the usage in details.

- Fixing Relevant Settings
  These four options are used to control the fixing functionality in a key figure. The either fix or de-fix the value, or make adjustment at different levels according to fixing situation.
  - Value Change with Preceding Defixing
  - Value Change with Following Fixing
  - Adjustment of the level Fixing
  - Adjustment of the level Fixing with Subsequent Fixing

- Data Source

  Comparing to 'Change mode', 'Data source' is a setting for the source rows. For example, you have a macro that calculates KF1 = KF2, then 'Change mode' is setting for KF1, while 'Data source' is setting for KF2.

  Since the default setting is 'Value', you'll need to set it explicitly if you need 'Attribute' of a row or column. Generally 'Row Attribute' will return the row number while 'Column Attribute' will return the column number.

### Data Source: Value or Row/Column Attributes

<table>
<thead>
<tr>
<th>Data</th>
<th>Values</th>
</tr>
</thead>
</table>

<Macros being executed on planning book>

If you want to know during your operation on planning book, which macros are executed, you can use command 'MSDP_DBO', which you can input it into the command field at any time before the process you want to check. To use the command, you'll need to open any selection (load some data) before you activate this command, then you enter the command into the command field (where you enter transaction code).

Then press enter, you'll get a confirmation message in the status bar.

Macro debugging switched on

Now perform the process you want to check, for example, load some data, and you'll get pop-up window telling you which macro is to be executed.

Another command "MSDP_DBLIST" is more powerful. It can output calculation result for each macro being executed. Try it out if you're interested.

<Debug a macro>

- Enter point to debug a macro
1) Set Break Point in function module /SAPAPO/ADVX_MACRO_CALCULATION.

Every macro is called up in this function. There’s a statement at around line 500:

```plaintext
LOOP AT l_t_macro_sequence ASSIGNING <fs_s_macro_sequence>.
```

Coding before this line is to collect the macros to be executed into table l_t_macro_sequence[]. If you set breakpoint here, you’ll get to know what macros are to be executed. Then at around line 600, macro program is called up by statement:

```plaintext
PERFORM (<fs_s_macro_sequence>-gen_formname) IN PROGRAM (<fs_s_macro_sequence>-gen_prgname)
```

Here <FS_S_MACRO_SEQUENCE>-GEN_PRGNAME is the generated program name for the macro book, which you can review in transaction SE38, while <FS_S_MACRO_SEQUENCE>-GEN_FORMNAME is the routine which corresponds to the macro. It is the start point of the macro.

If you step into the routine, and check the comment lines above the routine, you’ll find:

```plaintext
*&---------------------------------------------------------------------*
*& Form 4LD09IE0HIWENBKIVQ6JN97VG ---------------------->This is <FS_S_MACRO_SEQUENCE>-GEN_FORMNAME*
* &---------------------------------------------------------------------*
```

Then you scroll down till the line with below comments:

```plaintext
* & #TR# Generated Form Routine for Macro
* Macro ID : 4LD09I6BYKAP4P12PW47D795O
* Macro Name: Forecast = Forecast * AddKF1 -------------->This is your macro name defined in macro builder
```

Below it you can see a series of ‘Perform’ statement, each of which corresponds to a step in the macro. If you step into the routine, and check the comment lines above the routine, you’ll find:

```plaintext
* &---------------------------------------------------------------------*
* & Form 4LD09L5VC0QMDGLCZL0Z9YR6K ------------->This is routine name
* &---------------------------------------------------------------------*
```

** Notice that displaying macro name and step name in the routine’s comment is only for higher versions (SCM 7.0 or higher)

2) Use MSDP_DBO command, and drag ‘Debug’ shortcut or ‘Debug’ script onto the popup window to start debugger

Push ‘Generate a shortcut’ button which is on the general took bar of SAP GUI, enter below parameters and push ‘Finish’ to create a debug shortcut.
Or enter below text into a text file to create a 'Debug' script.

```
[FUNCTION]
Command=/H
Title=Debugger
Type=SystemCommand
```

Then when you get the popup window as below, just drag the shortcut or script directly on the window, and press the green tick to continue, you'll be led into the debugger. Press F5 once, and you'll be led to the macro relevant coding.

3) Find the program of your macro and step directly.

In the macro builder, if you select the menu Edit -> Book Information, you'll get a screen that shows some useful information of your macro book. There is a field called 'Generated Macro Program'. Here you can get the program name corresponds to your macro book. (The <FS_S_MACRO_SEQUENCE>-GEN_FORMNAME in 1). Open this program (should be quite long) in SE38, and look for your macro name or step name with 'Find' function, you'll directly find the macro or step you want to debug.

** Notice that displaying macro name and step name in the routine's comment is only for higher versions (SCM 7.0 or higher)

- **Input/Output parameters for the forms (both for macro and macro step)**

  - I_T_LINES (input)
    This table contains all lines in the data view. Here you'll get to know the line number (field 'LINE') corresponds to each key figure (field 'FELDH').
  - I_T_COLS (input)
    Corresponding to I_T_LINES, this table contains all columns in the data view. You'll get to know the periodicity (daily, weekly, monthly, etc.) for which macro is executed, as well as column number (field 'COLUMN' corresponding to each bucket (field 'PERDY').
  - I_T_ADV_PLOB_VALUES (input)
    As introduced in the 'Marco execution level' part, this table contains the planning objects on which macro is executed.
  - C_T_TAB (input and output)
    This table contains the value/data in each cell. The routine changes this table when data is changed in a cell. Each cell is identified by its row number (field 'Z') and column number (field 'C'), which correspond to the value in table I_T_LINES and I_T_COLS. The value of the cell is saved in field 'V'.
  - C_T_TAB_OLD (input and output)
    This table saves the old value of the changed cells. Whenever a change is made in table C_T_TAB (value change
or new value), a corresponding entry will be written to C_T_TAB_OLD with field 'V' filled with the old value. This table is important because when save data to liveCache, a delta save is performed, which means only the changed data are saved to liveCache. If a cell is not logged into C_T_TAB_OLD, system will not consider it as 'changed' and thus will not be saved to liveCache.

It is especially important to understand this when you create BADI macro. It is a general case that C_T_TAB is filled by BADI macro but C_T_TAB_OLD is not filled, which results data change in C_T_TAB could not be saved into liveCache.

**General check points**

Programs for macros are often quite long. Most of the time we can only focus on the part of coding which represents our designed logic. For example, in my macro "Forecast = Forecast + AddKF1", it does a simple multiplication. In macro program, it is done via below steps:

1 - Get Forecast value  
2 - Get AddKF1 value  
3 - Multiplies the two value  
4 - Set 3's result to Forecast

In most macro program, we can see calls of get_value() method and set_value() method, which are used to retrieve the value from a cell, or set a value into a cell. They look like:

```plaintext
i_o_advx_tab_access->get_value(
    exporting
    i_row = l_calc_line
    i_column = act_column
    i_column_offset = 0
    i_border = 'R'
    i_cols_left_border = l_advx_cols_left_border
    i_cols_right_border = l_advx_cols_right_border
    i_step_missing_values = 0
    i_t_tab = c_t_tab
    importing
    e_value = 4LDLRYOZ3TY351PV238SUUXI4
    changing
    c_flg_cancel_calculation = l_flg_cancel_calculation
).
```

-------------------------------------

```plaintext
i_o_advx_tab_access->set_value(
    exporting
    i_row = l_calc_line
    i_column = act_column
    i_column_offset = 0
    i_value = l_value
    i_change_mode = 0
    i_border = 'R'
    i_cols_left_border = l_advx_cols_left_border
    i_cols_right_border = l_advx_cols_right_border
    i_no_override = ' '
    i_s_adv_line_index = l_s_adv_line_index
    CHANGING
    c_t_tab = c_t_tab
    c_t_tab_old = c_t_tab_old
).
```

In the two methods, by checking input parameter 'l_calc_line' and 'act_column', you'll get to know the current line/column being processed. From input tables I_T_LINES[] and I_T_LINES[], you'll find the corresponding key figure and bucket.

get_value() fills the value of the cell to a generated variant (4LDLRYOZ3TY351PV238SUUXI4 in the above sample), while set_value() will change C_T_TAB[] and C_T_TAB_OLD[].

For auxiliary key figures, as stated previously, they're saved in a global table G_T_TAB_AUX[] separately. Correspondingly, methods get_value_aux() and set_value_aux() are used for auxiliary key figures.

For the SAP provided macro functions, they're realized using corresponding function modules. A best practice would be debugging the 'Stock Balance' macro in standard SNP planning book 9ASNP94/SNP94(1). For example, in this macro, the first step is defined as:
In the corresponding program of this step, you'll see that it first get ACT_PRODUCT, ACT_LOCATION, ACT_VERSION with corresponding function module like /SAPAPO/ACT_PRODUCT, as well as bucket end date with function module /SAPAPO/BUCKET_EDATE, which has a quite similar name to the macro function.

For most of these function modules, parameters are imported via internal table L_T_VALUE_TAB[], so in a macro with macro functions in use, you'll always see the coding that builds up this table, which appends every input parameters in to this table. Then this table is passed to function module (e.g. /SAPAPO/BUCKET_EDATE, /SAPAPO/DEMAND_CALC).

I would not suggest to debug into these /SAPAPO/ function modules, though some of them are quite easy, since many of these function modules contains complex logic behind. These function modules are widely used by customers for a long time, so there's a small chance to have a problem inside them. Most macro relevant problems can be found only be checking the input and output parameters.

Documentation of all macro functions in online help is [here](#).

<Background run and Interactive run>

Why I get different result between background run and interactive run for the same macro? This is a question being asked quite often. It's hard to tell without actually checking a specific issue, but there're several general points to check before going into debugging.

- Are the macros run on the same level in background and foreground?
- If auxiliary key figure is involved, check whether the issue is because of multiple planning objects.
- Is the initial data the same in background and foreground? Sometimes the data in planning book does not represent the real data in liveCache due to some default macros have been executed when you load the data.

To debug a background job defined in /SAPAPO/MC8E, you can debug report /SAPAPO/TS_BATCH_RUN in SE38 directly. Just set the breakpoint as stated before, and run the job with report /SAPAPO/TS_BATCH_RUN -- everything is the same as debugging foreground run.

<Some other information>

You may wonder why there're many coding looks strange in macro program, e.g. the strange variant name. Macro programs are generated when you activate the macro or save the macro book. They're so called generated programs, because they're not the program written directly by our developer, instead generated by the system based on user definition, with the predefined template and generating program (generator). (Except for macros, some other programs in APO, like data extraction, are also generated programs.)

The template to generate macro program is /SAPAPO/ADVX_TEMPLATE, while generator is GP_MET_/SAPAPO/ADVX. You can view their coding in SE38 as other programs, but just looks a little different. Generally we do not need to understand them, but if you apply a note (for example note 1502964) which makes correction to these programs, you'll need to regenerate the macro book to reflect the changes to your own macro program.

Related Content

Related Documents

Online help for macros: Advanced Macros

Related SAP Notes/KBAs

674238 - MacroBuilder: Start, final, level change and default macros
Use this structure to help you compose your contributions for WIKI and at the same time will ensure spelling and grammar.