DEBUGGING BUSINESS RULES IN SAP BUSINESSOBJECTS PLANNING AND CONSOLIDATION

Auric IT Consulting Services

Applicable Releases: All versions of SAP BusinessObjects Planning and Consolidation for Netweaver

Version 1.0

Table of Contents

Introduction

SAP Business Objects Planning and Consolidation for Netweaver ("BPC") has many ways to perform calculations on transaction data used in reporting. One such feature used mainly in consolidation scenarios is known as "Business Rules". Business rules are parameter driven functions for calculating and posting amounts in support of common accounting activities such as currency translations or intercompany eliminations. They are sometimes thought of as table driven logic because the business rules tables are enabled at the model level and provide an alternative to writing complex logic scripts for calculating financial data. One merely fills in key parameters into a business rule table and a small logic script calls a standard delivered ABAP program that takes the parameters, performs the appropriate calculations, and then writes the financial data to the database just like a logic script. While business rules are meant to be a simpler approach versus writing complex logic scripts, at times it is not clear how the business rules process data. This causes issues when it is not understood either how the results are produced or whether the particular business rule can use certain parameters to restrict the data. The purpose of this paper is to document a simple example of how debugging a business rule can help.

Business Scenario

One of the business rules found in BPC is "US Eliminations". This business rule addresses the posting of intercompany eliminations at group levels in scenarios where a full legal consolidation design involving a type "G" group dimension and ownership values are not required. This scenario is more common to financial models involving simple eliminations since a consolidation model would use "automatic adjustments" and consider ownership data to produce a proper result. As intercompany elimination entries should be posted only in groups in which both the entity and the partner entity are included, US Eliminations uses a concept known as posting at the first common parent to record the eliminations. Under this concept, the results of the eliminations are posted to an elimination entity found immediately under the first common parent as defined in a hierarchy of the entity dimension. This is significantly different from a true legal consolidation involving ownership and a group dimension.

Since dimension members can be used in multiple hierarchies, it is possible that someone may wish to report eliminations using several hierarchies (e.g., parent/child, geographical, etc.). Regardless of the number of hierarchies found in the entity dimension, the default US Eliminations logic does its searches only using the first hierarchy. According to the online help documentation, this can be adjusted to have the elimination calculate on any hierarchy in the entity dimension. However, enabling this is not documented anywhere. In this example scenario, a user wishes to perform this function and therefore will need to debug the business rule to determine whether this is possible and if so how to parameterize the business rule.

Debugging Business Rules

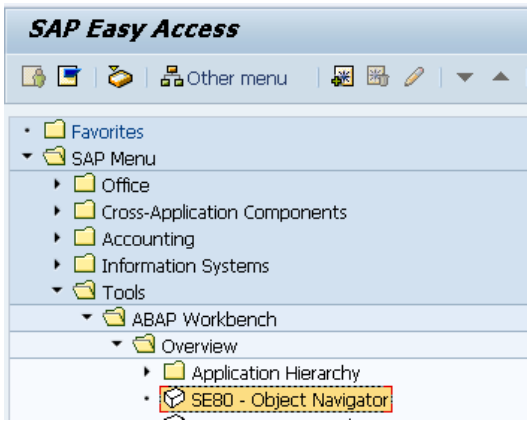
Every business rule requires a small logic script to call the appropriate program and pass parameters from the data manager package. According to the online help documents for US Eliminations, the logic script for calling this business rule is as follows:.

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

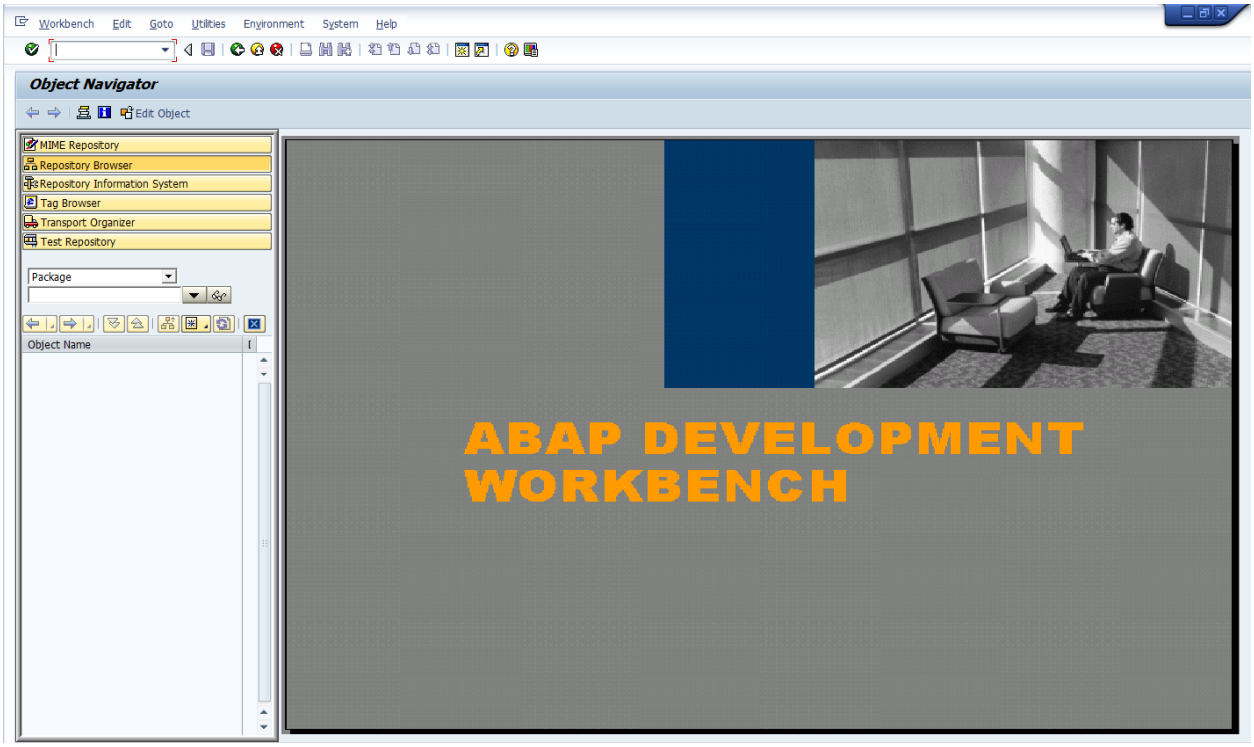
Based only on this logic script, the parameters passed to application include category, group (or currency), time, and "other" dimensions. There is no obvious parameter for the use of a particular entity hierarchy in the logic script. Therefore, a further examination of the actual program will be needed to see if US Eliminations can be adjusted to calculate using any entity dimension hierarchy.

In the Netweaver version of BPC, debugging business rules requires a little bit of ABAP knowledge (or access to a friendly colleague that understands ABAP and will debug it with you). ABAP is a high-level programming language used by SAP in building business applications. A BPC configuration expert does not necessarily have to understand ABAP code to properly configure the system, but it certainly helps to have a basic understanding if you intend to debug any business rules on your own (Note: If you wish to learn more about ABAP programming, please check your local bookstore. This paper will not cover ABAP coding in much detail.).

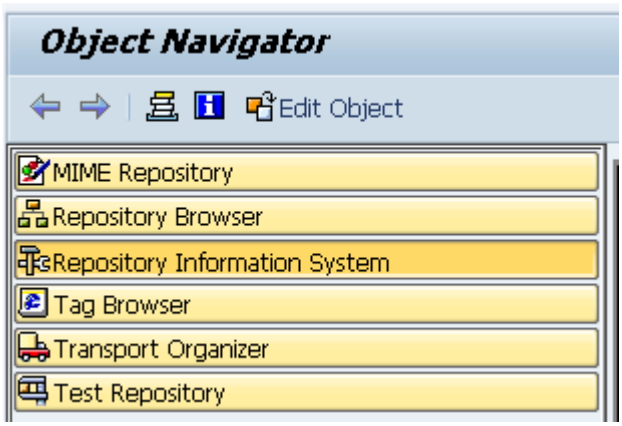
Once you are ready to start, you must first access the ABAP workbench on the SAP application server. This is not something you will find using any of the various BPC user interfaces. From the SAP easy access menu, follow the menu path Tools → ABAP Workbench → Overview → Object Navigator. Alternatively, you can enter transaction code SE80 in the t-code field to go directly to the Object Navigator without having to use the menu paths.

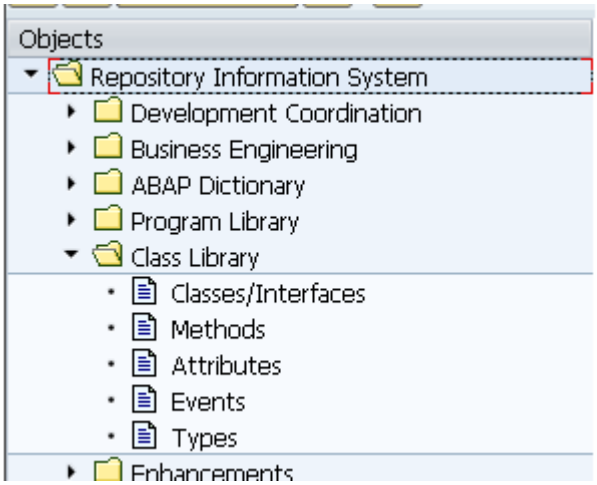


You should then be taken to the object navigator.

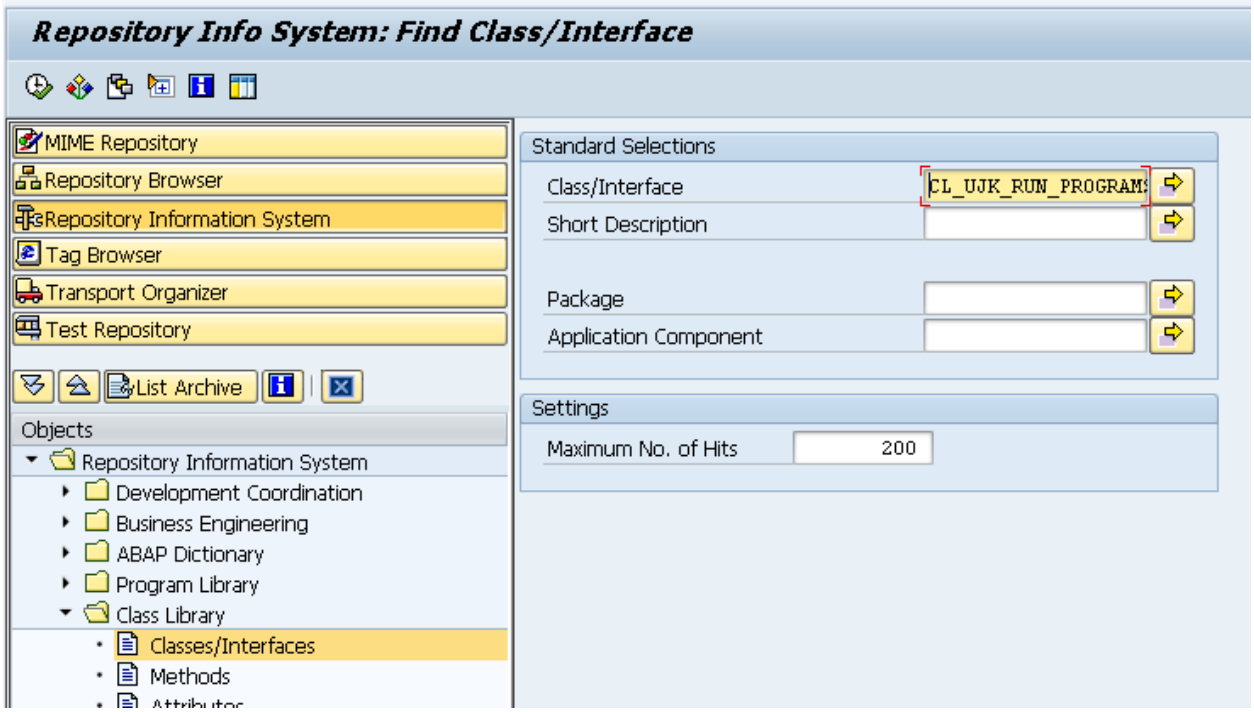


On the left side menu bar, double click on the icon for "Repository Information System". Next, under the folders that appear below the left side menu bar, open up the folder "Class Library" and double click on "Classes/Interfaces". This brings up the selection screen for searching various classes or interfaces which will help find the correct business rule program.

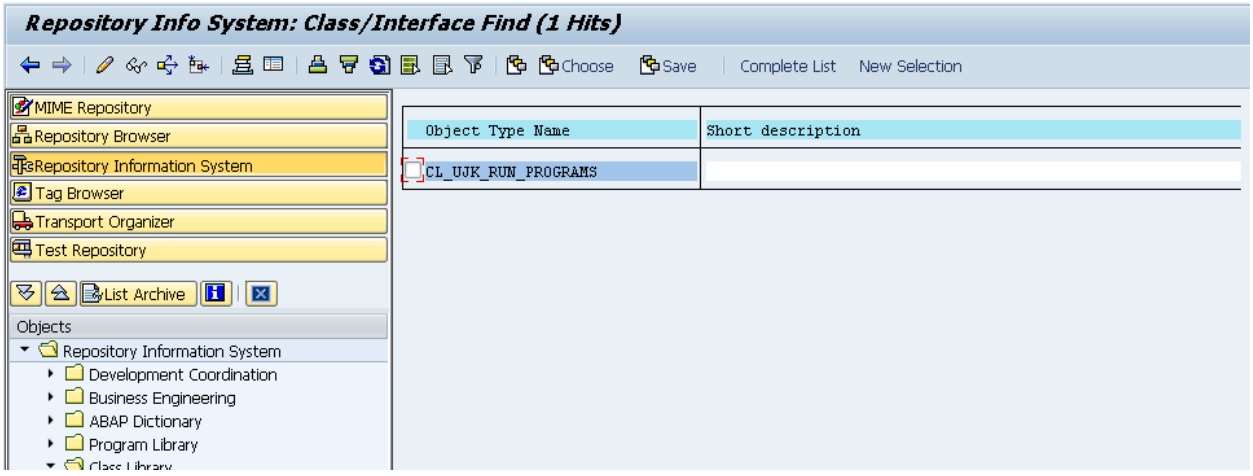




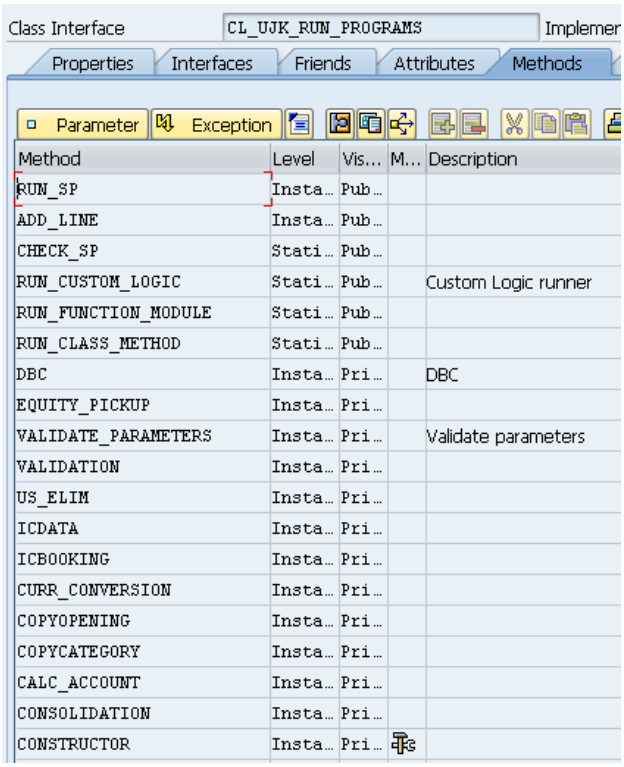
In the selection window, enter CL_UJK_RUN_PROGRAMS in the selection box for Class/Interface. A class is a collection of ABAP objects and this is the class that covers all of the business rule programs used in BPC. Click on the execute icon (or press F8) to begin the search.



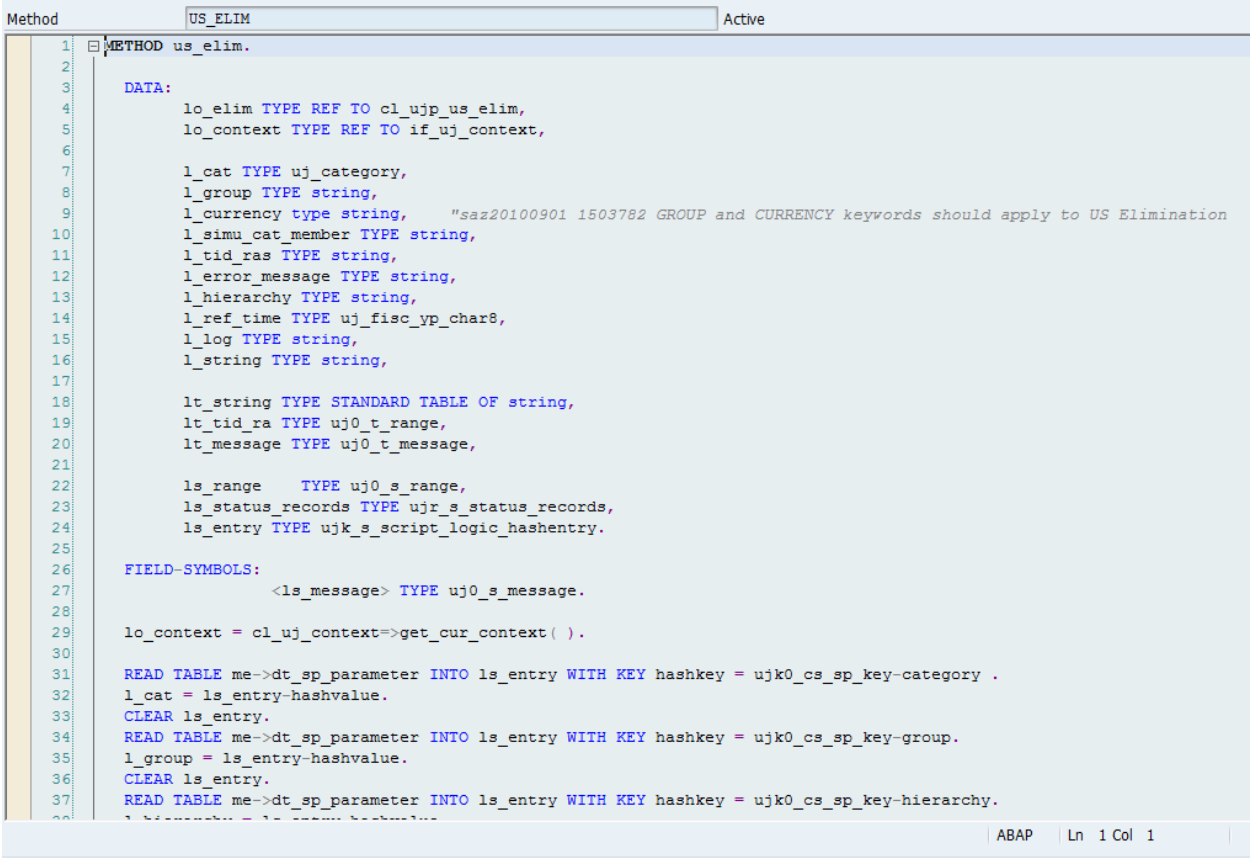
When the results screen appears, double click on the object name. This will take you to the class.



The class screen has many tabs and options. However, these will not be discussed in the context of this paper. Under the "Methods" tab, you will see various methods some of which correspond to the *RUN_PROGRAM calls found in logic scripts for running a business rule (note: the complete listing may differ based on your BPC release and patch level). These are the ABAP programs called by the logic scripts. In this case, we are interested in the method titled "US_ELIM" as that is what calculates US Eliminations. Double click on the method to see the ABAP code.

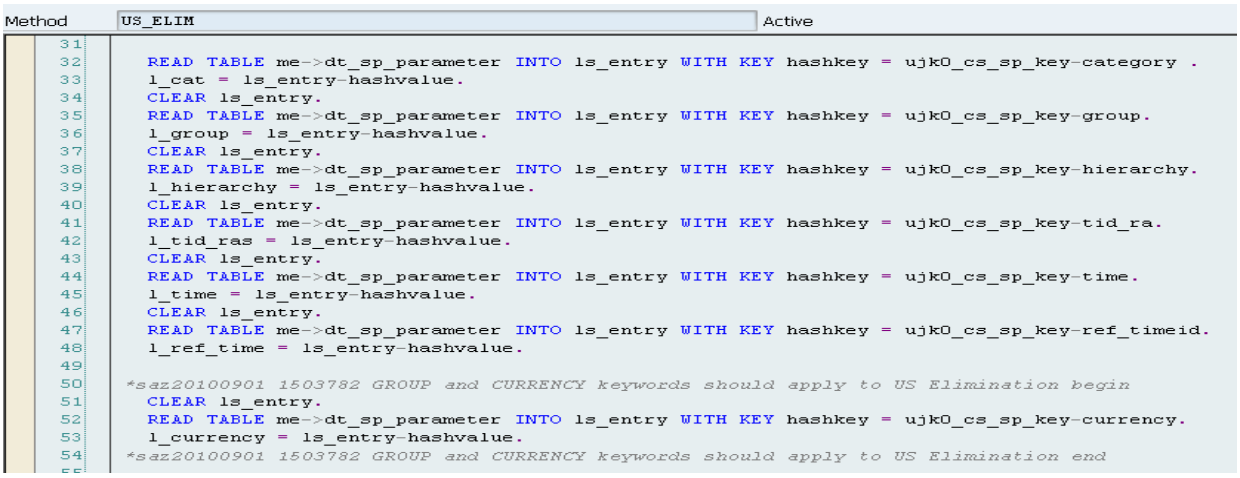


The ABAP code represents the program instructions that the system follows to calculate and post US Elimination amounts. Since we are interested in whether we can control parameters for an alternate hierarchy selection, we need to scroll down to see what input this particular method accepts.




```
1  METHOD us_elim.
2
3    DATA:
4          lo_elim TYPE REF TO cl_ujp_us_elim,
5          lo_context TYPE REF TO if_uj_context,
6
7          l_cat TYPE uj_category,
8          l_group TYPE string,
9          l_currency type string,    "saz20100901 1503782 GROUP and CURRENCY keywords should apply to US Elimination
10         l_simu_cat_member TYPE string,
11         l_tid_ras TYPE string,
12         l_error_message TYPE string,
13         l_hierarchy TYPE string,
14         l_ref_time TYPE uj_fisc_yp_char8,
15         l_log TYPE string,
16         l_string TYPE string,
17
18         lt_string TYPE STANDARD TABLE OF string,
19         lt_tid_ra TYPE uj0_t_range,
20         lt_message TYPE uj0_t_message,
21
22         ls_range    TYPE uj0_s_range,
23         ls_status_records TYPE ujr_s_status_records,
24         ls_entry TYPE ujk_s_script_logic_hashentry.
25
26   FIELD-SYMBOLS:
27                 <ls_message> TYPE uj0_s_message.
28
29   lo_context = cl_uj_context=>get_cur_context( ).
30
31   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-category .
32   l_cat = ls_entry-hashvalue.
33   CLEAR ls_entry.
34   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-group.
35   l_group = ls_entry-hashvalue.
36   CLEAR ls_entry.
37   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-hierarchy.
```

Scrolling down through the code can give you clues as to the behavior of the business rules. In this case, we will scroll to the block where several READ TABLE statements begin (line 32 in this example).




```
31
32   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-category .
33   l_cat = ls_entry-hashvalue.
34   CLEAR ls_entry.
35   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-group.
36   l_group = ls_entry-hashvalue.
37   CLEAR ls_entry.
38   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-hierarchy.
39   l_hierarchy = ls_entry-hashvalue.
40   CLEAR ls_entry.
41   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-tid_ra.
42   l_tid_ras = ls_entry-hashvalue.
43   CLEAR ls_entry.
44   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-time.
45   l_time = ls_entry-hashvalue.
46   CLEAR ls_entry.
47   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-ref_timeid.
48   l_ref_time = ls_entry-hashvalue.
49
50 *saz20100901 1503782 GROUP and CURRENCY keywords should apply to US Elimination begin
51   CLEAR ls_entry.
52   READ TABLE me->dt_sp_parameter INTO ls_entry WITH KEY hashkey = ujk0_cs_sp_key-currency.
53   l_currency = ls_entry-hashvalue.
54 *saz20100901 1503782 GROUP and CURRENCY keywords should apply to US Elimination end
```


Looking at this section of code, we notice that several parameters can be passed from the logic script to the program; among these are category, group, hierarchy, time, and currency. This is important because these are the only parameters that will be accepted barring a change from SAP Development. It is equally important to notice what is not there as well. For example, the standard logic script for US Eliminations implies that "other" dimension restrictions can be used and the delivered data manager package has an "entity" selection field. In looking at the code, we see that "other" dimensions cannot be restricted in running US Eliminations as there is no program line for such input. So any entity restriction made in the data manager package is ignored. Knowing what restrictions are possible helps to avoid potential issues later on in the process. Do not assume that default documentation is completely accurate as things change from release to release and patch to patch.

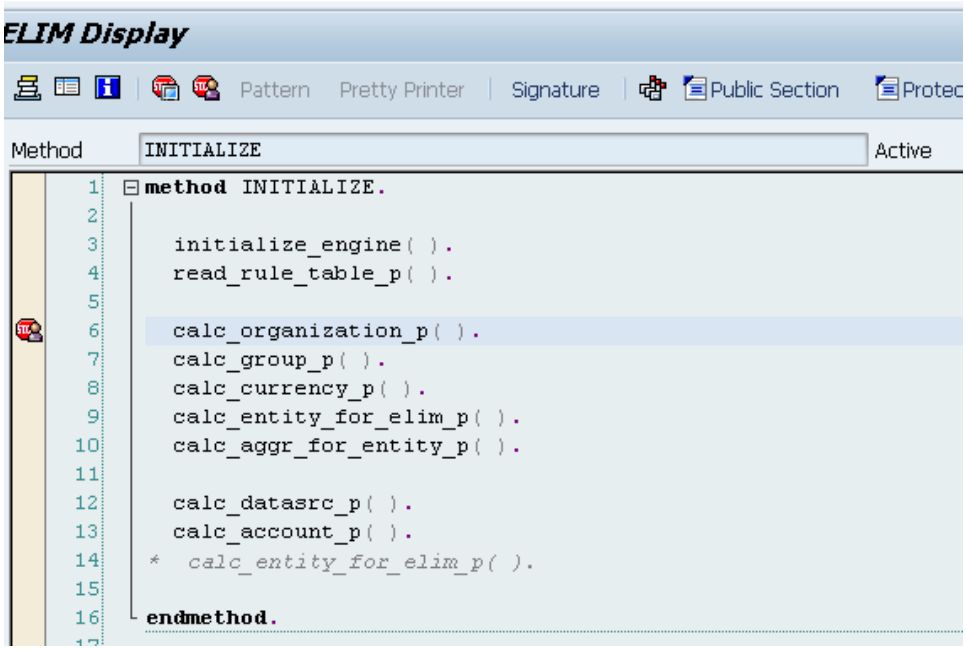
Reviewing the code answers the question as to whether US Eliminations could be restricted to run on any entity hierarchy even though the logic script and data manager packages do not show this. In the various hashkeys, the words on the right are the keywords from the logic script. By restricting on the keyword "HIERARCHY" in the logic script, it will pass a parameter value for the entity hierarchy. However, we do not know what value(s) to use. This is where we need to debug further than just looking at the code. To determine what value the hierarchy parameter accepts, we will need to trace through an execution of the program which can be done by setting a breakpoint in the code and running the script logic debugger.

First, set an external breakpoint in an appropriate part of the code. Scrolling further down in the method, we notice the statement where the US Eliminations calculations are initialized.

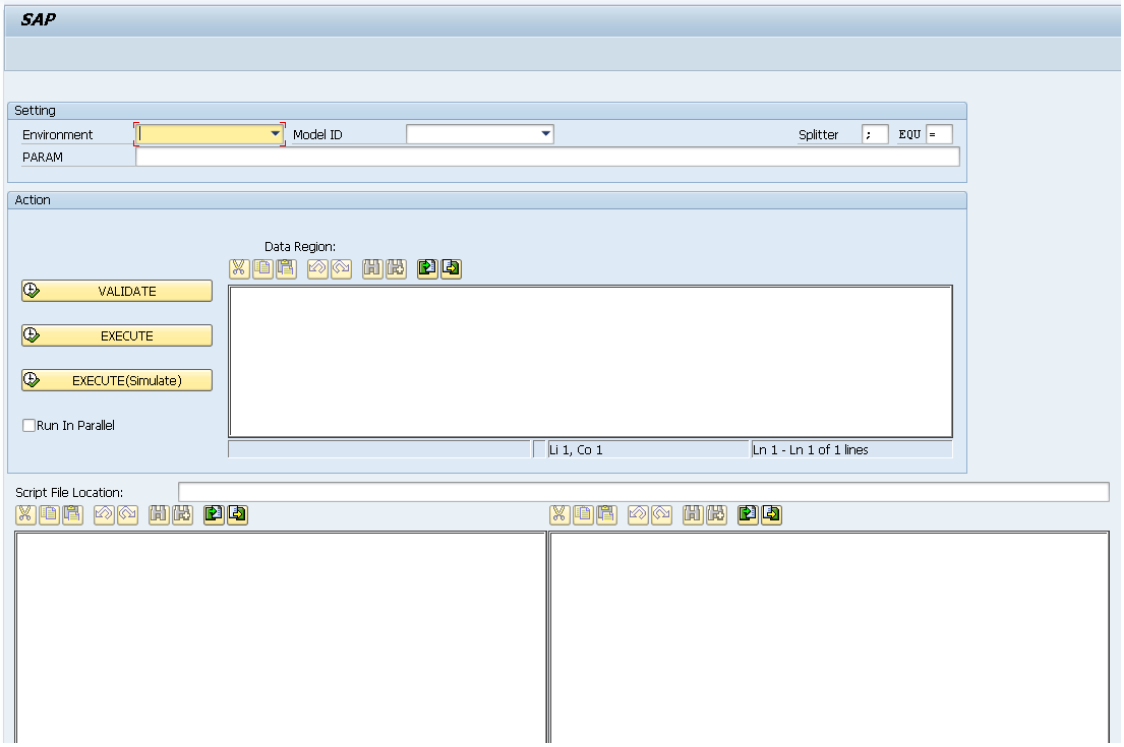
```
IF me->d_run_mode NE ujk0_cs_run_mode-execute.
  RETURN.
ENDIF.
lo_elim->initialize( ).
cl_ujk_logger=>log( 'RUN US ELIMINATION' ).

CALL METHOD lo_elim->run_intco_elimination
  IMPORTING
    es_status_records = ls_status_records
    et_error_records  = et_error_records
    et_message        = lt_message.
```

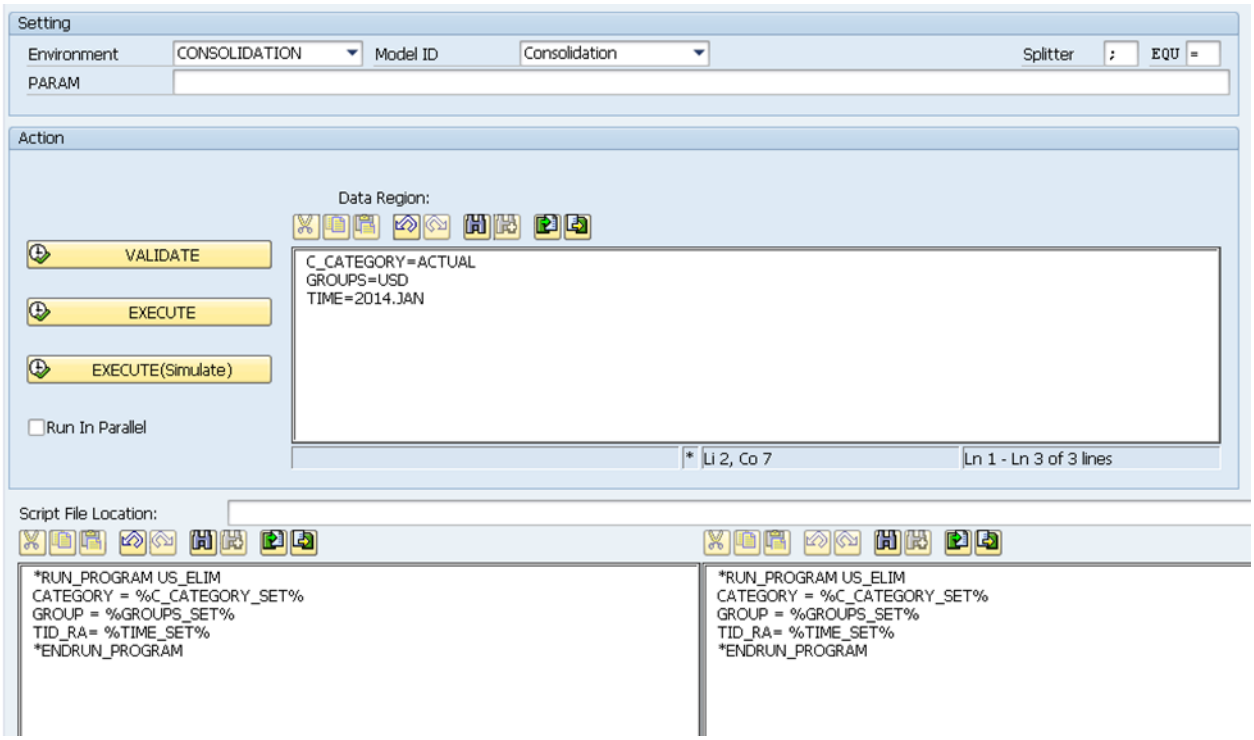
By double clicking on that line, we are taken to the section of the code where the system determines various metadata used in calculating the eliminations. In this case since we wish to know what organizational hierarchy is used, we will set the breakpoint there. Do this by selecting the appropriate line of code and clicking on the set external breakpoint icon (stop sign with person icon) found in the menu bar.



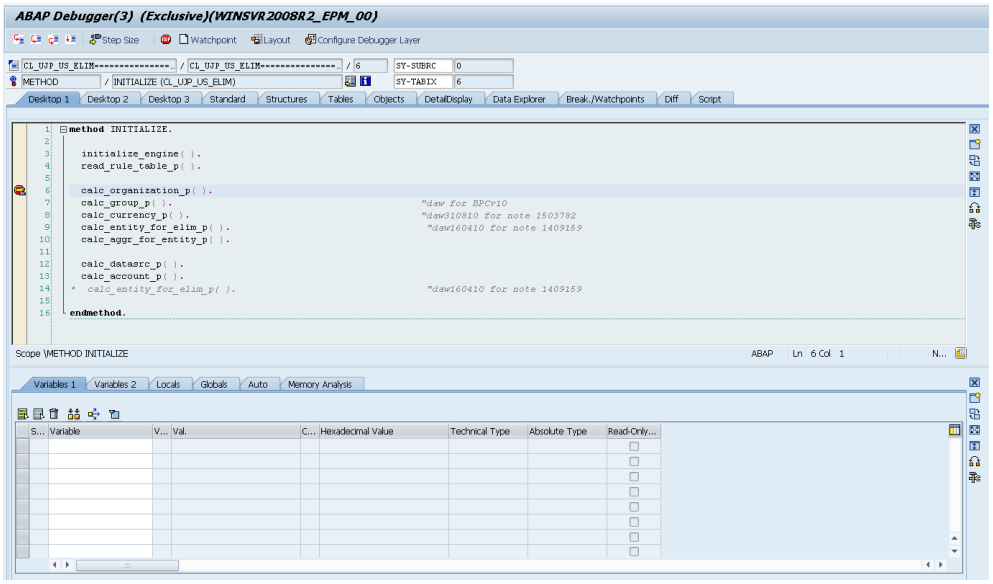
Now that we have set the breakpoint, we need to run the program in order to step through the code. Open up another SAP session and navigate to transaction code UJKT. This is the logic script tester which allows you to test/debug logic scripts without having to use data manager packages. The screen should look similar to the following as it can differ by release and patch level.



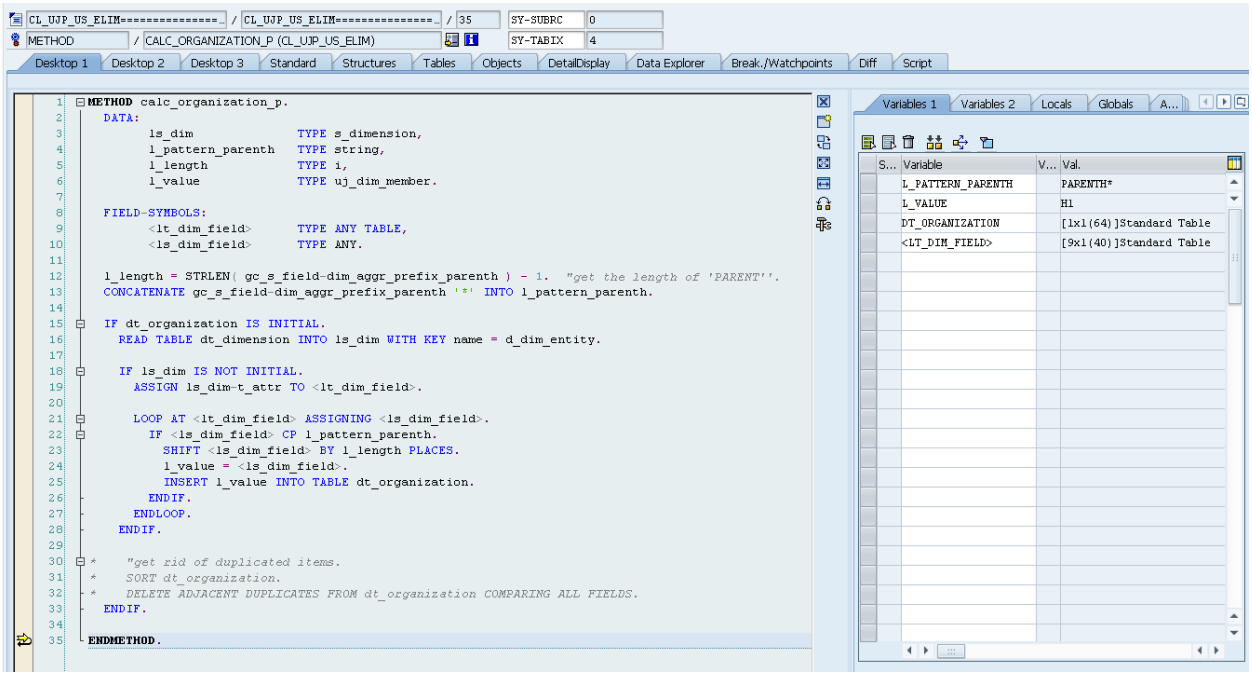
Enter all of the appropriate criteria to run the logic script. You must select an environment/model, enter the appropriate logic script, and define any data parameters which are normally provided by variables in the data manager package. In this example, the logic script for US Eliminations taken from the help documentation has been entered (the "other" parameter was left out since it was determined earlier to have no effect). Once you have entered all the information, click on "Validate" to make sure there are no syntax issues with the parameters/logic script. If the validated script shows no issues, click "Execute". This will run the program.



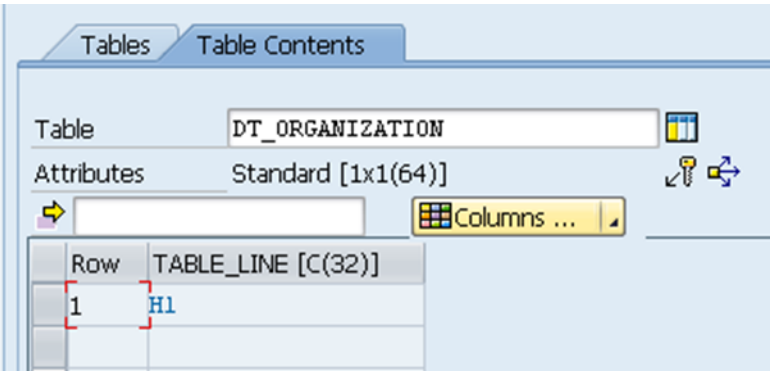
Once the code executes to where the external breakpoint is located, a debugging session will start. The program will pause at the breakpoint and you can now step through each line of code. If the program completes instead of going to a debugging session, that means the breakpoint was not encountered and you should determine why this happened or set another breakpoint.



In the variable area of the debugging screen, select field parameters to see their values as you step through the code. In this example, we have chosen some including L_VALUE and DT_ORGANIZATION (this is the results table into which values in L_VALUE are placed). Step through the code by pressing F5 until the system gets to where it will end the function. At that point, double click on DT_ORGANIZATION in the variable area and the view will change to display its contents.

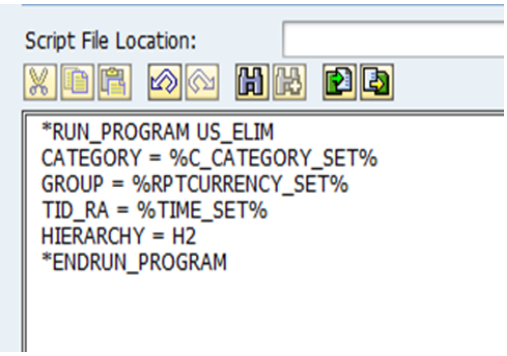


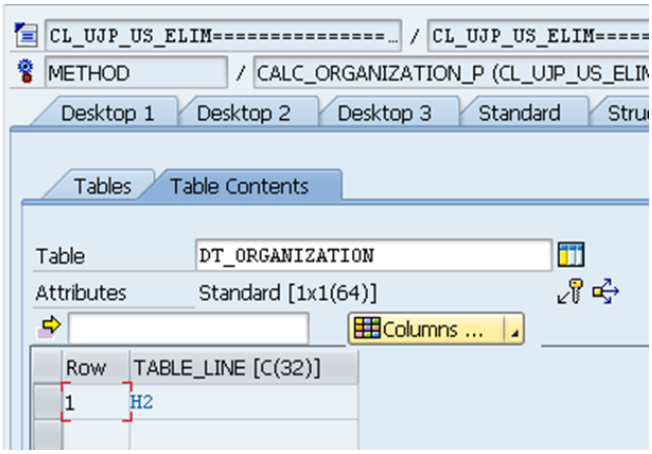
Now we know that the system defaulted the value "H1" for the hierarchy selection when no parameter value for hierarchy is passed to the program. So PARENTH1 is used for the calculations.



By debugging, we have now determined both the parameter name and the format for the value to be passed. Now we just need to make sure we can adjust the logic script and have it be recognized.

Going back to the logic script tester, we alter the logic script to include the line "HIERARCHY = H2". For proof of concept, this value is hardcoded in the logic script for testing. It can be altered later to a package variable if you wish to make the selection more dynamic. Now if we walk through the same steps to debug, we can see that the system does indeed recognize the "H2" value and that it would now use the PARENTH2 entity hierarchy to determine the organization to use for posting to elimination entities.





By using the debugging features of BPC for Netweaver, we were able to determine that the documentation on US Eliminations was correct about being able to use any hierarchy in the entity dimension and how to enable its use using standard BPC configuration.

Conclusion

The above is a simple example of how a BPC consultant can work with the debugging features in BPC to diagnose issues with business rules. Business rule issues can take on many forms (data problems, missing configuration, patches, etc.) and sometimes this requires determining what a business rule program does. While no BPC consultant necessarily needs to be expert in ABAP code, understanding how and when to use the debugging tools can be very helpful. When used correctly, they are an excellent feature to assist in resolving issues with business rules.

