SAP BI Global Report Variable user exit modularization

Applies to:
SAP BI 7

Summary
When designing a report, some requirements have certain complexity that lead to the creation of custom exit code global report variables. For example, a user enters a month and the report requires an interval created for the ‘quarter to date’ of the entered month: the user entered May 2008, the report requires a variable that reflects the interval Apr-May 2008 (Q2-2008). This type of variable requires a 'custom exit' code. SAP has traditionally provided for custom code enhancements in the form of user exits and recently in the form of BADI (Business Add-In). Basically a user exit is include in a function module that can be modified by the customer. The user exit FM for Global Report Variables EXIT_SAPLRRS0_001 tends to grow in a single piece of code and its maintenance become cumbersome.

In an SAP BI Project, along with new reporting needs, new requirements for Global Report Variables of the type customer exit are generated. It is typical then for the respective user exit code to grow to a few thousand lines, resulting in code that is hard to manage, debug, and use in a multi-team environment. Code Modularization is a helpful tool for performance, documentation and multi-tasking

Author: Alfonzo Vega
Company: Vega Business Consulting
Created on: 05 March 2009

Author Bio
Alfonzo Vega is an independent Senior SAP BI Consultant. Alfonzo started working in the SAP technologies as a ECC MM Consultant in 1995, joined SAP Andina y del Caribe in Venezuela in 1996 and has worked on multiple SAP BI implementations since 1999. Alfonzo currently works in the US Midwest as an independent consultant
# Table of Contents

Modularization ....................................................................................................................................................3  
Dynamic Method Call..........................................................................................................................................3  
Useful Function Modules ....................................................................................................................................7  
   Where used list ...............................................................................................................................................7  
   Variable test ....................................................................................................................................................7  
   Code for standard ‘SAP’ exit variables ............................................................................................................7  
‘Refresher’ on I_STEP ........................................................................................................................................7  
Related Content..................................................................................................................................................9  
Disclaimer and Liability Notice..........................................................................................................................10
Modularization

Modularization has advantages:

- The code is easier to read. Instead of scrolling through pages and pages of code, just go to a specific method in a class.
- If the code modules are independent, the local use of variables reduces the interference between developers. Also, developers can be working on a specific module without interfering with each other. The standard approach based on multiple ‘case’ statements and ‘if’ statements based on the variable name and the specific step makes it hard to ‘know’ if you are inserting the code in the right place.
- The current approach goes through all the code to find the variable. A new approach could call the specific module directly.

Figure 1 Code approach: go through IF..THEN..ELSEIF or CASE versus a dynamic method call

Dynamic Method Call

The basis for this modularization approach is a dynamic method call. Based on the variable, it calls a method with a name made of a standard prefix and the variable name, for example, ZBW_<vnam>. In a dynamic call, the ABAP/4 code will fail unless there is a ‘catch’ command. The idea is to ‘migrate’: start moving variables to the dynamic method call step by step. A variable E_VAR_PROCESSED is set in the method to ‘tell’ that the calculation was made and no need to continue through the ‘old’ code. In case there is no method for the specific variable, it will continue through the old code. Define a custom class in transaction SE24 and create one method per custom-exit global report variable. The methods will have exactly the same interface as the user exit function module plus a exporting parameter 'E_VAR_PROCESSED' as mentioned above. In this example, the class name is ZBW_XRSR. See images for the class definition and a sample method below.

⚠️ Bear in mind that in the step 3 no variable name gets passed to the user exit, so you need to write more code for this specific step, if needed.
In the sample method code below, based on a fiscal period entered by the user, the code selects the same period but on the prior year.

Code for the dynamic method call is attached after the method code below.
METHOD zbw_zp_currfp_prior_year.

  *---------------------------------------------------------------------*
  * 2008-08-25                           D22K915560     Rel5       *
  *        Based on fiscal period ZP_CURRFP derive the same fiscal   *
  *      period for the previous year                                   *
  *---------------------------------------------------------------------*

  CONSTANTS:
    c_eq(2) TYPE c VALUE 'EQ',
    c_fp(9) TYPE c VALUE 'ZP_CURRFP',
    c_i(1)  TYPE c VALUE 'I',
    c_x(1)  TYPE c VALUE 'X',
    c_0     TYPE i VALUE 0,
    c_2     TYPE i VALUE 2.

  DATA:
    lv_fiscper   TYPE poper,
    lv_fiscyr    TYPE bdatj,
    s_var_range TYPE rrrangeexit,
    s_range     TYPE rsr_s_rangesid.

  IF i_step = c_2.
    READ TABLE i_t_var_range INTO s_var_range
      WITH KEY vnam = c_fp.
    IF sy-subrc = c_0.
      CLEAR s_range.
      lv_fiscyr = s_var_range-low(4).
      lv_fiscyr = lv_fiscyr - 1.
      lv_fiscper = s_var_range-low+4(3).
      CONCATENATE lv_fiscyr lv_fiscper INTO s_range-low.
      s_range-sign = c_i.
      s_range-opt  = c_eq.
      APPEND s_range TO e_t_range.
    ENDIF.
  ENDIF.

  e_var_processed = c_x.
ENDMETHOD.

Figure 4 Sample code for 'same period – last year'
DATA:
  lv_class       TYPE string,
  lv_exc_ref     TYPE REF TO cx_sy_dyn_call_error,
  lv_exc_text    TYPE string,
  lv_meth        TYPE string,
  lv_var_proc(1) TYPE c.
CLEAR lv_var_proc.
lv_class = 'ZBW_XRSR'.
* Format called method name: prefix variable name with "ZBW_".
CONCATENATE 'ZBW_' i_vnam INTO lv_meth.
* Execute the assigned method for the DataSource.
TRY.
  CALL METHOD (lv_class)=>(lv_meth)
    EXPORTING
      i_vnam          = i_vnam
      i_vartyp        = i_vartyp
      i_iobjnm        = i_iobjnm
      i_s_cob_pro     = i_s_cob_pro
      i_s_rkb1d       = i_s_rkb1d
      i_periv         = i_periv
      i_t_var_range   = i_t_var_range
      i_step          = i_step
    IMPORTING
      e_t_range       = e_t_range
      e_meeht         = e_meeht
      e_mefac         = e_mefac
      e_waers         = e_waers
      e_whfac         = e_whfac
      e_var_processed = lv_var_proc
    changing
      c_s_customer    = c_s_customer.
*** Handle Exceptions.
  CATCH cx_sy_dyn_call_error INTO lv_exc_ref.
  lv_exc_text = lv_exc_ref->get_text( ).
* message exc_text type 'I'.
ENDTRY.
IF NOT lv_var_proc IS INITIAL.
  EXIT.
ENDIF.
*Change to 'modularize the user exit' - End

Figure 5 Dynamic method call code
Useful Function Modules

Where used list

Function module RSZ_X_WHERE_USED_LIST_GET finds where a report element is used. To get where a global report variable is used, enter the value from table RSZGLOBV, field VARUNIID as the import parameter I_ELTUID in the function module. (field VNAM contains the 'readable' variable technical name). The table E_T_WHERE_USED_LIST will contain the list of queries that use the variable.

Variable test

Function Module RRS_VAR_VALUES_EXIT_FILL executes a global report variable user exit. As some of the variables execute in a specific step (I_STEP) it may need some break points and data manipulation to have the variable executed.

Code for standard 'SAP' exit variables

Function Module RRREX_VARIABLE_EXIT has the code for SAP exit global report variables. If the variable is not 'solved' in this part of the code, it may execute at the end of this FM code as a RSVAREXIT_<varname>. For example RSVAREXIT_0I_CUFQU executes the code for variable 0I_CUFQU 'Current Quarter of Fiscal Year (SAP Exit)', resulting in a fiscal period interval for the current fiscal quarter.

'Refresher' on I_STEP

The global report variable user exit EXIT_SAPLRRS0_001 has I_STEP as an input parameter. You normally develop the custom code for a specific step, as explained below. It is useful to know the function modules involved when debugging, to set an appropriate breakpoint.

- I_STEP = 0. This step is not called during report execution but when a report variable is used in other instances such as InfoPackages and authorizations.
- I_STEP = 1 - Called prior to processing of variable pop-up and for each “customer exit” variables. Can be used to fill variable with default values. Function module EXIT_SAPLRRS0_001 is called with I_STEP = 1 in function module RRS_VAR_VALUES_EXIT_BEFORE

![Figure 6 Prior to User entry - I_STEP = 1](image-url)
- **I_STEP = 2** - Called after the processing of the variable pop-up. Only for those variables that are not marked as “ready for input” or are set to “mandatory variable entry”. Function module EXIT_SAPLRRS0_001 is called with I_STEP = 2 in function module RRS_VAR_VALUES_EXIT_AFTER.

![Function module RRS_VAR_VALUES_EXIT_AFTER](image1)

**Figure 7 After User entry - I_STEP = 2**

- **I_STEP = 3** - Called after all variable processing and gets called only once and not per variable. Here you can validate the user entries. Function module EXIT_SAPLRRS0_001 is called with I_STEP = 3 in function module RRS_VAR_VALUES_EXIT_CHECK.

![Function module RRS_VAR_VALUES_EXIT_CHECK](image2)

**Figure 8 After all variable processing - I_STEP = 3**
Related Content

Customer Exits (SAP Library - BI Suite: Business Explorer)

Dependencies for Variables of Type Customer Exit provides the explanation for the input variable I_STEP and a code sample

SAP Note 792779 Report ANALYZE_RSZ_TABLES helps in fixing underlying table inconsistencies and it is a very good introduction to the main tables for reporting elements.
Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.