BI Reporting user Exit Implementation Through Dynamic Class with Interface

Applies to:

SAP BI 7.0. For more information, visit the EDW homepage

Summary

This document provides a object oriented approach for modularizing the reporting user exit using dynamic class call and interfaces.

Author: Ashok Vijayasundaram

Company: Cognizant Technology Solutions

Created on: 09 July 2010

Author Bio

Working as a SAP BI Consultant for projects related to manufacturing and banking domain since 2002. His expertise includes SAP-XI and SAP-APO.

Table of Contents

Introduction

Nowadays creation of customer exit variables becomes mandatory to meet complex reporting requirements of BI. As many variables are created, the no of lines of code in user exit increases and at one stage becomes unmanageable. This becomes worse in case of a multi-team environment with issues like objects locked in some other transport requests, reimplementation of same variable by different team, regression test and so on.

Modularization approach

The answer to the problem would be modularization. There are many modularization approaches and some of them are listed below

a) Dynamic perform

b) Dynamic method

c) Dynamic class with Interface

In this document, I would focus on the third approach c) Dynamic class with Interface

Dynamic class with Interface

In this approach, one class per variable with one mandatory method is created. The variable specific implementation is done in the mandatory method. The design would be a 'generic exit class' which inherits an interface whose methods are used in the specific variable class will be instantiated in the user exit. This generic exit class has its own method to call the variable classes dynamically based on the variable name from the user exit.

Advantages

Dynamic class with interface is object oriented and hence all OOPs techniques like inheritance, polymorphism etc can be done on the variable specific class to make it more and more modular in future.

As the variable specific implementation is independent of the other variable implementation, at point of time the transport dependent issues will not occur , the same variable cannot be re-implemented by another team and hence no regression required.

Workbench Objects

Below are the ABAP workbench objects required for this approach.

Generic exit Interface

The generic exit interface will have all the methods that is required to access the variable information like name, step, etc and methods to set the variable values like currency key, unit etc..

Generic exit class

The generic exit class will inherit and has method implementation of the generic exit interface. It has a constructor which accepts the variable information as parameters from the user exit and additional method of its own to call the specific variable classes dynamically and assigns the variable values and passes back to the user exit .

Generic variable interface

This interface has one method with name “perform” which accepts the generic exit class type ref to generic exit interface and returns the variable values.

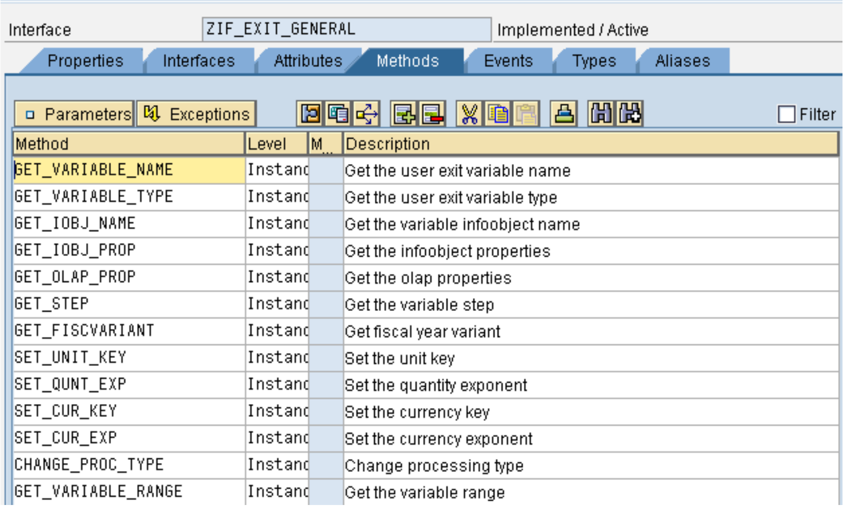
Variable class

This class is created for each user exit variable. This class inherits the generic variable interface and does the implementation of method “perform” to determine the variable values.

Implementation

Generic exit Interface

The generic interface ZIF_EXIT_GENERAL has 13 instance method declarations one for each parameter from the user exit include FM EXIT_SAPLRRS0_001 (includes import, export and changing) except for the export parameter E_T_RANGE (Variables value table).

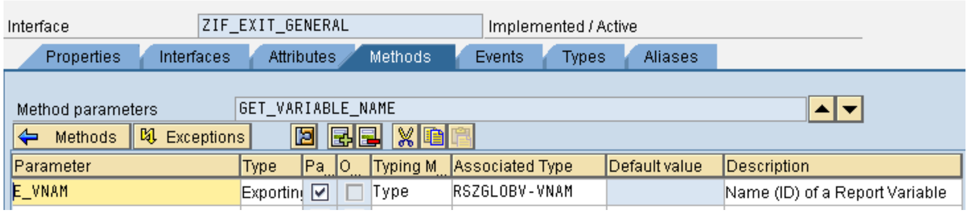


Interface ZIF_EXIT_GENERAL Implemented / Active

□ Filter

Method	Level	M...	Description
GET_VARIABLE_NAME	Instanc		Get the user exit variable name
GET_VARIABLE_TYPE	Instanc		Get the user exit variable type
GET_IOBJ_NAME	Instanc		Get the variable infoobject name
GET_IOBJ_PROP	Instanc		Get the infoobject properties
GET_OLAP_PROP	Instanc		Get the olap properties
GET_STEP	Instanc		Get the variable step
GET_FISCVARIANT	Instanc		Get fiscal year variant
SET_UNIT_KEY	Instanc		Set the unit key
SET_QUNT_EXP	Instanc		Set the quantity exponent
SET_CUR_KEY	Instanc		Set the currency key
SET_CUR_EXP	Instanc		Set the currency exponent
CHANGE_PROC_TYPE	Instanc		Change processing type
GET_VARIABLE_RANGE	Instanc		Get the variable range

Each method will hold one export/import/change parameter passed as value as below.



Interface ZIF_EXIT_GENERAL Implemented / Active

Method parameters GET_VARIABLE_NAME

Parameter	Type	Pa...	O...	Typing M...	Associated Type	Default value	Description
E_VNAM	Exportin	☑	□	Type	RSZGLOBV-VNAM		Name (ID) of a Report Variable

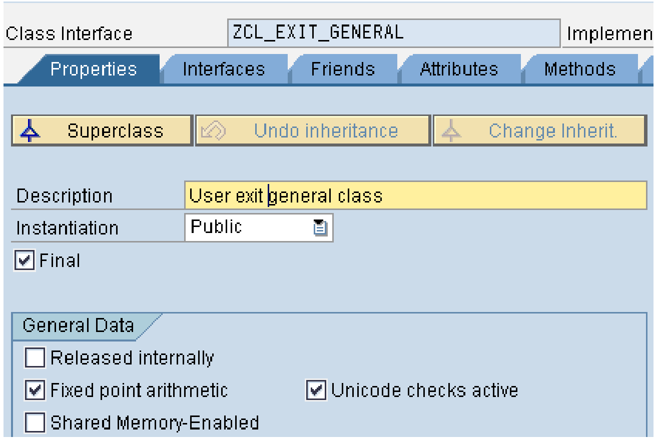
Below is the table with the method and its corresponding parameter with type.

Method name	**Parameter name**	**Type**	**Value/Reference**	**Typing method**	**Associated type**
GET_VARIABLE_NAME	E_VNAM	Export	Value	Type	RSZGLOBV-VNAM
GET_VARIABLE_TYPE	E_VARTYP	Export	Value	Type	RSZGLOBV-VARTYP
GET_IOBJ_NAME	E_IOBJNM	Export	Value	Type	RSZGLOBV-IOBJNM
GET_IOBJ_PROP	E_S_COB_PRO	Export	Value	Type	RSD_S_COB_PRO
GET_OLAP_PROP	E_S_RKB1D	Export	Value	Type	RSR_S_RKB1D
GET_STEP	E_STEP	Export	Value	Type	I
GET_FISCVARIANT	E_PERIV	Export	Value	Type	RRO01_S_RKB1F-PERIV
SET_UNIT_KEY	I_MEEHT	Import	Value	Type	RSUNIT
SET_QUNT_EXP	I_MEFAC	Import	Value	Type	RRMEFAC
SET_CUR_KEY	I_WAERS	Import	Value	Type	RSCURRENCY
SET_CUR_EXP	I_WHFAC	Import	Value	Type	RRWHFAC
CHANGE_PROC_TYPE	C_S_CUSTOMER	Change	Value	Type	RRO04_S_CUSTOMER
GET_VARIABLE_RANGE	E_T_VAR_RANGE	Export	Value	Type	RRS0_T_VAR_RANGE

Generic exit class

Definition

The generic exit class ZCL_EXIT_GENERAL is created with the below settings



Class Interface ZCL_EXIT_GENERAL Implemen

Description: User exit general class

Instantiation: Public

☑ Final

General Data

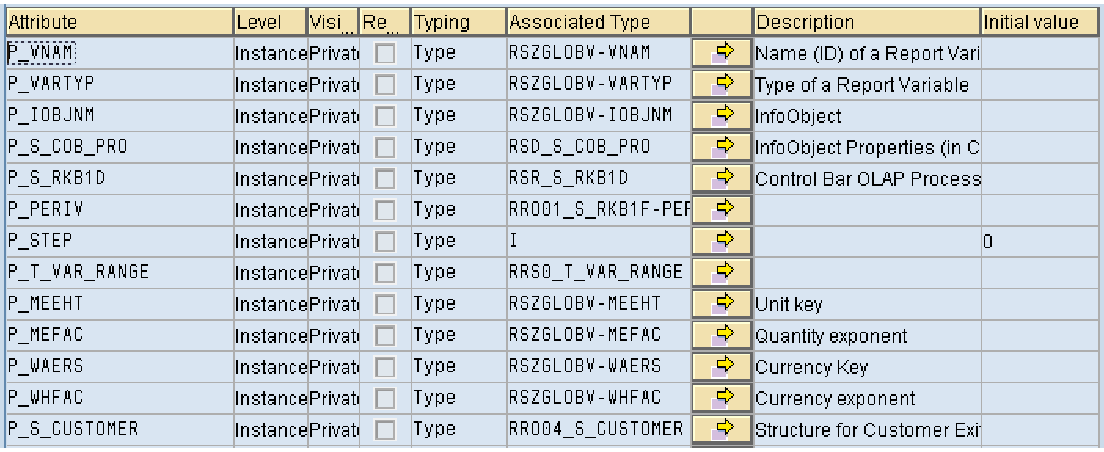
☐ Released internally

☑ Fixed point arithmetic ☑ Unicode checks active

☐ Shared Memory-Enabled

Attributes

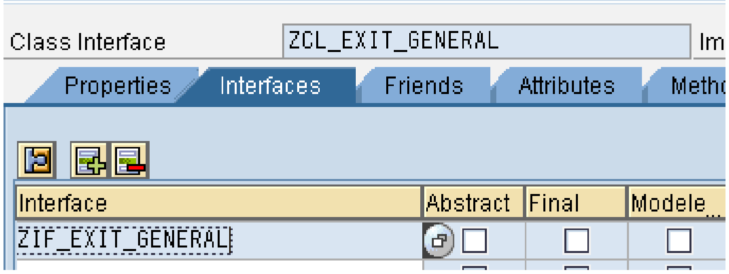
One private instance attribute per user exit parameter is created.



Attribute	Level	Visi...	Re...	Typing	Associated Type		Description	Initial value
P_VNAM	Instance	Privat	☐	Type	RSZGLOBV-VNAM	⇨	Name (ID) of a Report Vari	
P_VARTYP	Instance	Privat	☐	Type	RSZGLOBV-VARTYP	⇨	Type of a Report Variable	
P_IOBJNM	Instance	Privat	☐	Type	RSZGLOBV-IOBJNM	⇨	InfoObject	
P_S_COB_PRO	Instance	Privat	☐	Type	RSD_S_COB_PRO	⇨	InfoObject Properties (in C	
P_S_RKB1D	Instance	Privat	☐	Type	RSR_S_RKB1D	⇨	Control Bar OLAP Process	
P_PERIV	Instance	Privat	☐	Type	RR001_S_RKB1F-PEF	⇨		
P_STEP	Instance	Privat	☐	Type	I	⇨		0
P_T_VAR_RANGE	Instance	Privat	☐	Type	RRS0_T_VAR_RANGE	⇨		
P_MEEHT	Instance	Privat	☐	Type	RSZGLOBV-MEEHT	⇨	Unit key	
P_MEFAC	Instance	Privat	☐	Type	RSZGLOBV-MEFAC	⇨	Quantity exponent	
P_WAERS	Instance	Privat	☐	Type	RSZGLOBV-WAERS	⇨	Currency Key	
P_WHFAC	Instance	Privat	☐	Type	RSZGLOBV-WHFAC	⇨	Currency exponent	
P_S_CUSTOMER	Instance	Privat	☐	Type	RR004_S_CUSTOMER	⇨	Structure for Customer Exi	

Inheritance

The exit interface ZIF_EXIT_GENERAL is added to the exit class

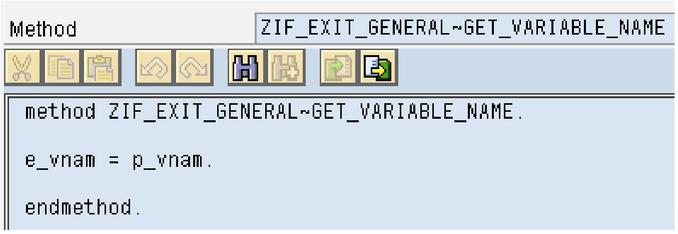


Class Interface ZCL_EXIT_GENERAL Im

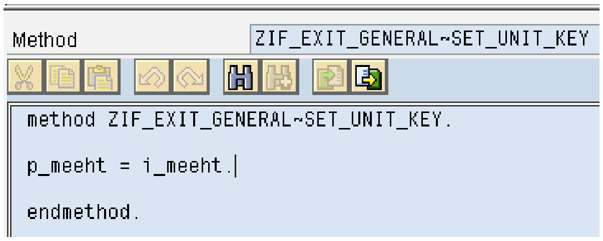
Interface	Abstract	Final	Modele...
ZIF_EXIT_GENERAL	☐	☐	☐

Interface methods

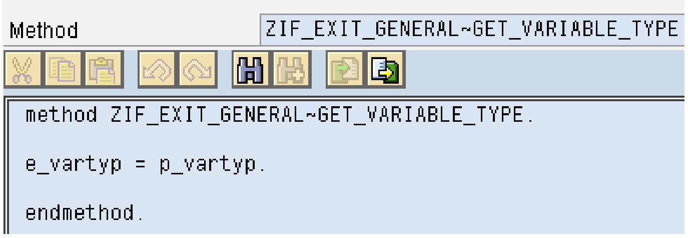
Interface method implementation in the exit class will look like the below



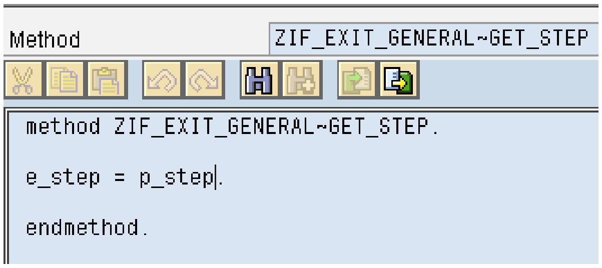
```
method ZIF_EXIT_GENERAL~GET_VARIABLE_NAME.

e_vnam = p_vnam.

endmethod.
```



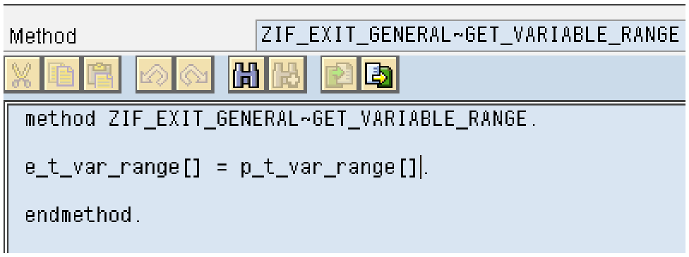
```
method ZIF_EXIT_GENERAL~SET_UNIT_KEY.

p_meeht = i_meeht.

endmethod.
```



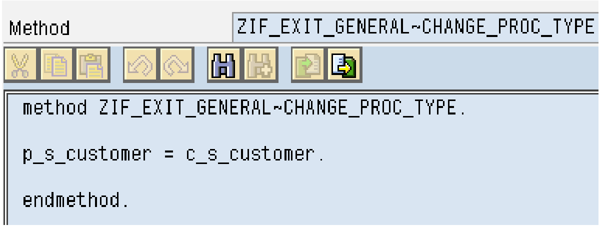
```
method ZIF_EXIT_GENERAL~GET_VARIABLE_TYPE.

e_vartyp = p_vartyp.

endmethod.
```



```
method ZIF_EXIT_GENERAL~GET_STEP.

e_step = p_step.

endmethod.
```



```
method ZIF_EXIT_GENERAL~GET_VARIABLE_RANGE.

e_t_var_range[] = p_t_var_range[].

endmethod.
```

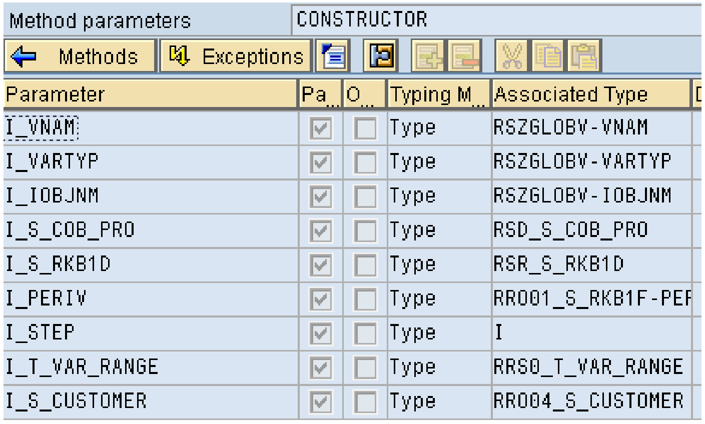


```
method ZIF_EXIT_GENERAL~CHANGE_PROC_TYPE.

p_s_customer = c_s_customer.

endmethod.
```

Note: The above screenshots are given for only important methods covering all the get, set and change methods. Similar method implementation has to be carried out for all the remaining interface methods.

Constructor

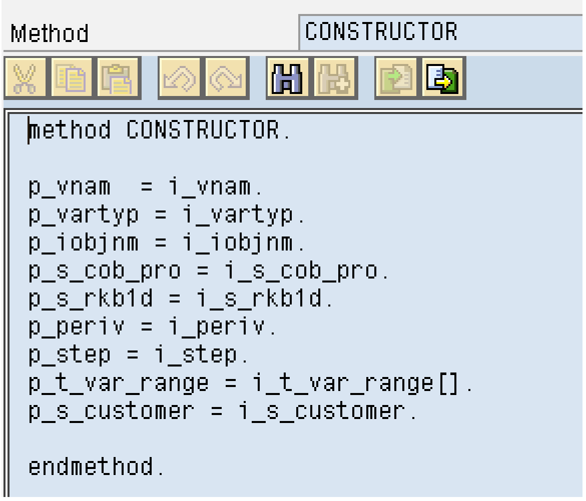
The constructor method of the exit class will have the parameters as shown below.



Method parameters CONSTRUCTOR

Parameter	Pa..	O..	Typing M..	Associated Type
I_VNAM	☑	☐	Type	RSZGLOBV-VNAM
I_VARTYP	☑	☐	Type	RSZGLOBV-VARTYP
I_IOBJNM	☑	☐	Type	RSZGLOBV-IOBJNM
I_S_COB_PRO	☑	☐	Type	RSD_S_COB_PRO
I_S_RKB1D	☑	☐	Type	RSR_S_RKB1D
I_PERIV	☑	☐	Type	RRO01_S_RKB1F-PEF
I_STEP	☑	☐	Type	I
I_T_VAR_RANGE	☑	☐	Type	RRS0_T_VAR_RANGE
I_S_CUSTOMER	☑	☐	Type	RRO04_S_CUSTOMER

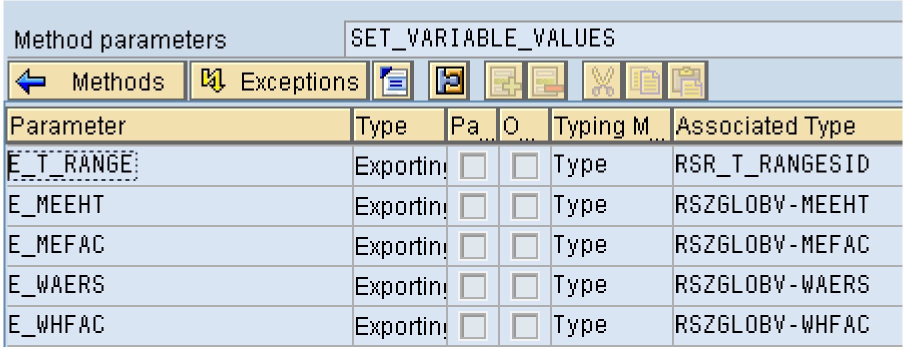
The constructor will populate the corresponding class attributes as below.



```
method CONSTRUCTOR.

p_vnam  = i_vnam.
p_vartyp = i_vartyp.
p_iobjnm = i_iobjnm.
p_s_cob_pro = i_s_cob_pro.
p_s_rkb1d = i_s_rkb1d.
p_periv = i_periv.
p_step = i_step.
p_t_var_range = i_t_var_range[].
p_s_customer = i_s_customer.

endmethod.
```

Dynamic class call

The dynamic class call happens in the SET_VARIABLE_VALUES method of the exit class . Below are the parameters for the method.



Parameter	Type	Pa...	O...	Typing M...	Associated Type
E_T_RANGE	Exportin	□	□	Type	RSR_T_RANGESID
E_MEEHT	Exportin	□	□	Type	RSZGLOBV-MEEHT
E_MEFAC	Exportin	□	□	Type	RSZGLOBV-MEFAC
E_WAERS	Exportin	□	□	Type	RSZGLOBV-WAERS
E_WHFAC	Exportin	□	□	Type	RSZGLOBV-WHFAC

Following is the source code for this method.

```
method SET_VARIABLE_VALUES.

  DATA:
      lv_class       TYPE string,
      ld_class       TYPE ref to object,
      lv_exc_ref     TYPE REF TO cx_sy_dyn_call_error,
      lv_exc_cre     TYPE REF TO cx_sy_create_error,
      lv_step        TYPE c,
      lv_method      TYPE string,
      lv_exc_text    TYPE string,
      ptab           TYPE abap_parmbind_tab,
      ptab_line      TYPE abap_parmbind,
      etab           TYPE abap_excpbind_tab,
      etab_line      TYPE abap_excpbind.

    move p_step to lv_step.
    CONCATENATE 'ZCL_' lv_step '_' p_vnam INTO lv_class.

  TRY.

    create object ld_class type (lv_class).

    refresh ptab[].
    refresh etab[].

    ptab_line-name = 'C_T_VAR_RANGE'.
    ptab_line-kind = cl_abap_objectdescr=>changing.
    GET REFERENCE OF e_t_range INTO ptab_line-value.
    INSERT ptab_line INTO TABLE ptab.

    ptab_line-name = 'I_EXIT_VARIABLE'.
    ptab_line-kind = cl_abap_objectdescr=>exporting.
    GET REFERENCE OF me INTO ptab_line-value.
    INSERT ptab_line INTO TABLE ptab.

    etab_line-name = 'OTHERS'.
    etab_line-value = 4.

    INSERT etab_line INTO TABLE etab.

     move ‘ZIF_EXIT_GENERAL~PERFORM’ to lv_method.

      CALL METHOD ld_class->(lv_method)
        PARAMETER-TABLE
          ptab
        EXCEPTION-TABLE
          etab.

      e_meeht = p_meeht.
      e_mefac = p_mefac.
      e_waers = p_waers.
      e_whfac = p_whfac.

*  Handle Exceptions.
```

```
      CATCH cx_sy_dyn_call_error INTO lv_exc_ref.
        lv_exc_text = lv_exc_ref->get_text( ).

      CATCH cx_sy_create_object_error into lv_exc_cre.
        lv_exc_text = lv_exc_cre->get_text( ).

    ENDTRY.
           endmethod.
```

This method determines the class name by concatenating the strings ZCL_, processing step and variable name. Hence for a variable which is processed under step 2 of name LASTYEARDEC will have its class name as ZCL_2_LASTYEARDEC.

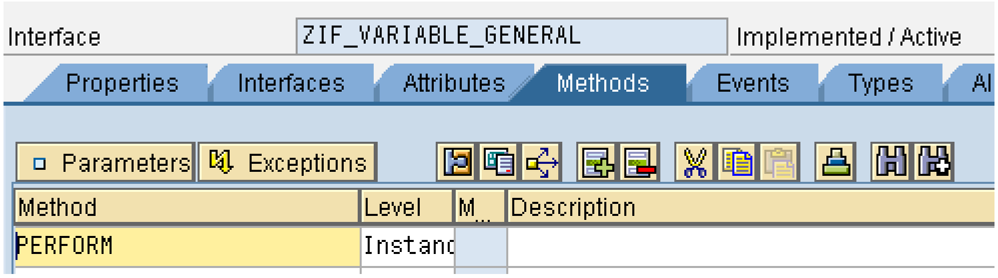
The class object is created and the perform method of the class is called dynamically. The reference of the current exit class is passed to the exporting parameter `I_EXIT_VARIABLE` and the reference of variables value table E_T_RANGE is passed to the changing parameter `C_T_VAR_RANGE` of the PERFORM method.

The dynamic call exceptions and object creation exceptions are caught to avoid dump in case if the class and method doesn't exists.

Since the reference of the exit class is passed to the exit interface parameter of the variable class (narrow casting) only the methods available in the interface is accessible to the variable class. This avoids the variable class calling the method SET_VARIABLE_VALUES in variable class and endless loop.

Generic variable Interface

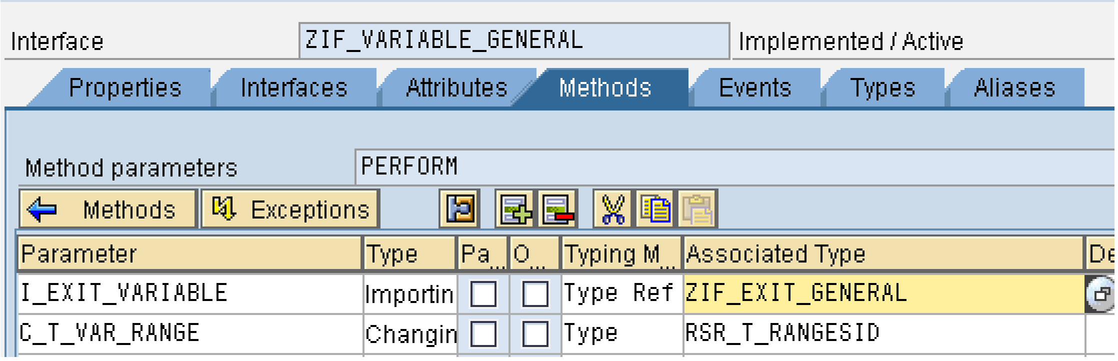
This interface has ZIF_VARIABLE_GENERAL has one instancing method PERFORM shown as below



Interface ZIF_VARIABLE_GENERAL Implemented / Active

Method	Level	M...	Description
PERFORM	Instanc		

The parameters for this method will be as below



Interface ZIF_VARIABLE_GENERAL Implemented / Active

Method parameters PERFORM

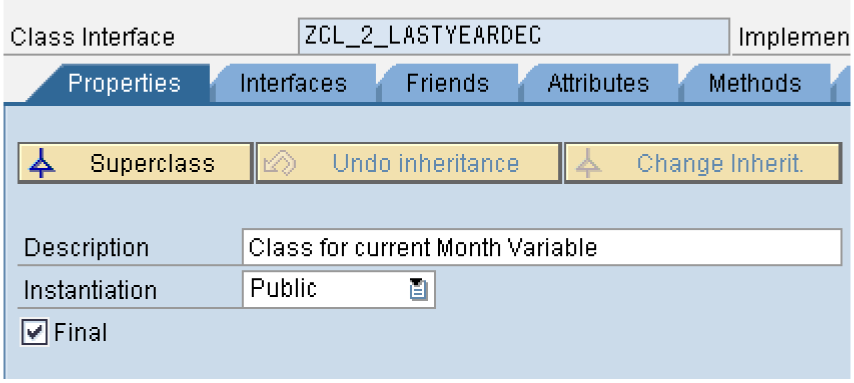
Parameter	Type	Pa...	O...	Typing M...	Associated Type	De
I_EXIT_VARIABLE	Importin	□	□	Type Ref	ZIF_EXIT_GENERAL	
C_T_VAR_RANGE	Changin	□	□	Type	RSR_T_RANGESID	

Variable class

The variable class in case of user exit variable LASTYEARDEC which gets processed under step 2 will have its name ZCL_2_LASTYEARDEC. (In my example, I would like to return the dec of a year based on multiprovider on which the query is executed)

Definition

The definition of the class would be as below



Class Interface ZCL_2_LASTYEARDEC Implemen

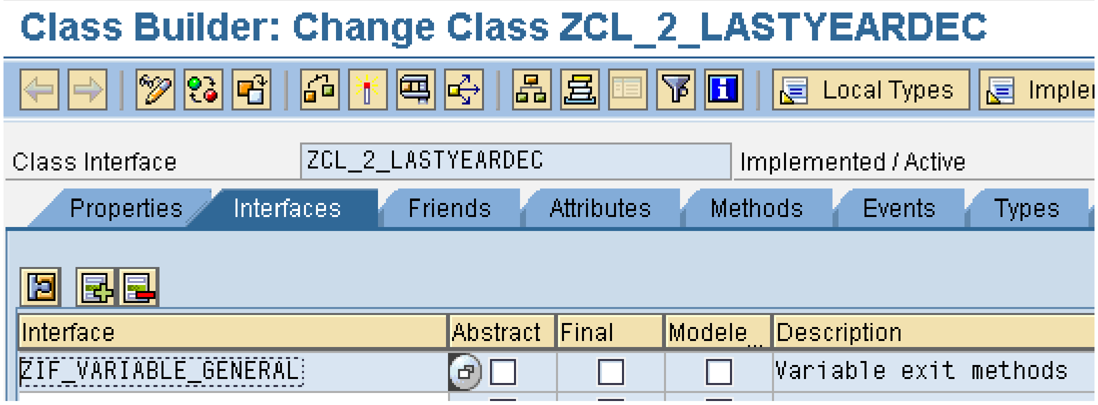
Description: Class for current Month Variable

Instantiation: Public

☑ Final

Inheritance

The interface YIF_VARIABLE_GENERAL is added to the class

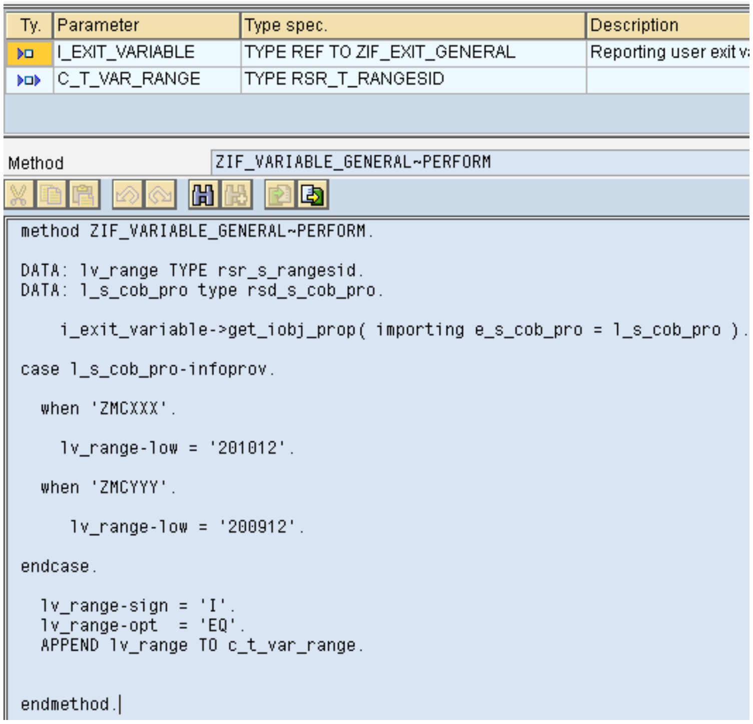


Class Interface ZCL_2_LASTYEARDEC Implemented / Active

Interface	Abstract	Final	Modele...	Description
ZIF_VARIABLE_GENERAL	□	□	□	Variable exit methods

Perform method

The example coding for the perform method is shown below. The method GET_IOBJ_PROP is accessed through the parameter I_EXIT_VARIABLE to get the info provider name and the resulting calendar month is passed to the changing variable C_T_VAR_RANGE.



Ty.	Parameter	Type spec.	Description
▶□	I_EXIT_VARIABLE	TYPE REF TO ZIF_EXIT_GENERAL	Reporting user exit v
▶□▶	C_T_VAR_RANGE	TYPE RSR_T_RANGESID	

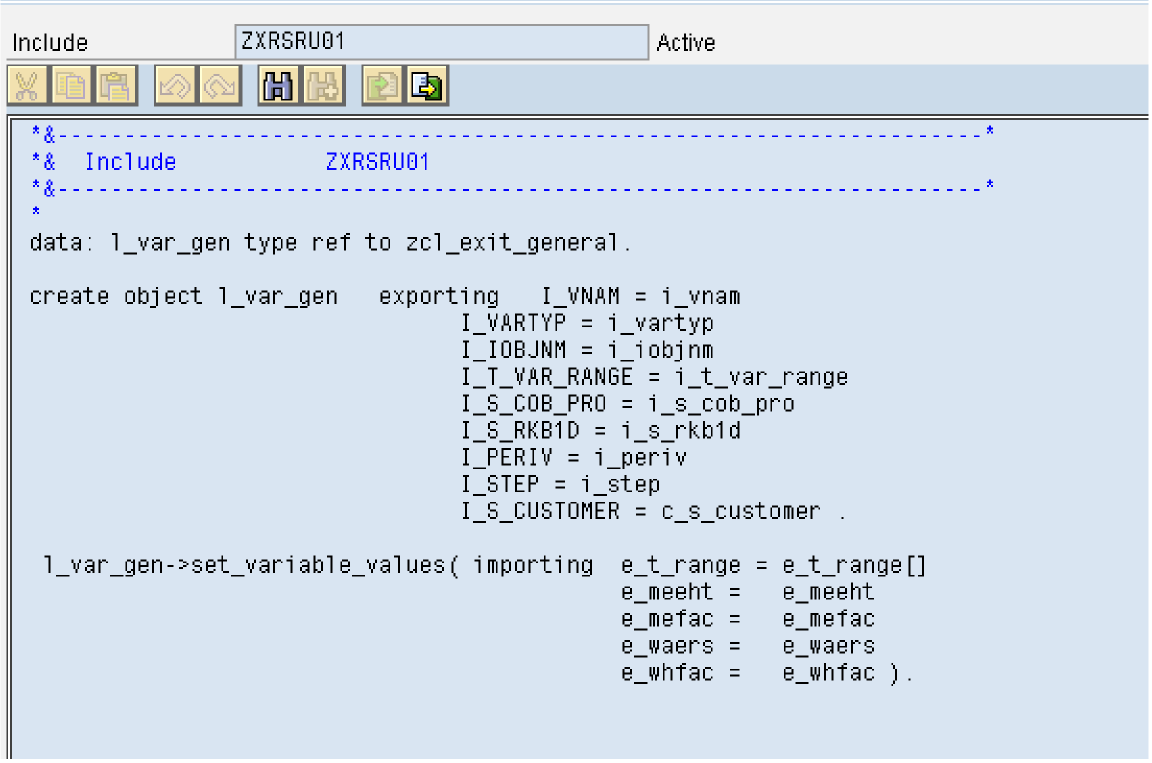
Method ZIF_VARIABLE_GENERAL~PERFORM

```
method ZIF_VARIABLE_GENERAL~PERFORM.

DATA: lv_range TYPE rsr_s_rangesid.
DATA: l_s_cob_pro type rsd_s_cob_pro.

    i_exit_variable->get_iobj_prop( importing e_s_cob_pro = l_s_cob_pro ).

case l_s_cob_pro-infoprov.

  when 'ZMCXXX'.

    lv_range-low = '201012'.

  when 'ZMCYYY'.

     lv_range-low = '200912'.

endcase.

  lv_range-sign = 'I'.
  lv_range-opt  = 'EQ'.
  APPEND lv_range TO c_t_var_range.


endmethod.
```

Note: The above screenshot is just an example for implementing the requirement in the perform method. The set methods of the user exit class can also be called through the variable I_EXIT_VARIABLE.

Exit Include ZXRSRU01

The exit class is instantiated and the SET_VARIABLE_VALUES method is called from the user exit include ZXRSRU01 as below.



Include ZXRSRU01 Active

```
*&---------------------------------------------------------------------*
*&  Include           ZXRSRU01
*&---------------------------------------------------------------------*
*
data: l_var_gen type ref to zcl_exit_general.

create object l_var_gen   exporting   I_VNAM = i_vnam
                                 I_VARTYP = i_vartyp
                                 I_IOBJNM = i_iobjnm
                                 I_T_VAR_RANGE = i_t_var_range
                                 I_S_COB_PRO = i_s_cob_pro
                                 I_S_RKB1D = i_s_rkb1d
                                 I_PERIV = i_periv
                                 I_STEP = i_step
                                 I_S_CUSTOMER = c_s_customer .

 l_var_gen->set_variable_values( importing  e_t_range = e_t_range[]
                                            e_meeht =   e_meeht
                                            e_mefac =   e_mefac
                                            e_waers =   e_waers
                                            e_whfac =   e_whfac ).
```

Transports

For the first time transports to deploy in other environments, the generic exit class, generic exit interface , generic variable interface and the user exit include ZXRSRU01 is transported. Subsequently only the specific variable class needs to be transported.

Related Content

For more information, visit the EDW homepage.

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.