Creating ABAP Process type to use in Multiple Process chains that Reports Success (or) Failure

Applies to:

SAP BW 3.5, 7.0. For more information, visit the EDW homepage

Summary

In BW, there is an ABAP process type for use in Process Chains. This process type runs the ABAP program, but doesn't report whether the program was successful or not. A number of articles have been written that address this lack, but this could be more easy to implement and moreover could be implemented in more than one process chain which can be triggered at same time. This article explains how to create a new Process Type and reports the success or failure. It also demonstrates the business scenario where it can be implemented with sample code.

Author: Yokesh Kumar Velusamy

Company: Wipro Technologies

Created on: 11th April 2011

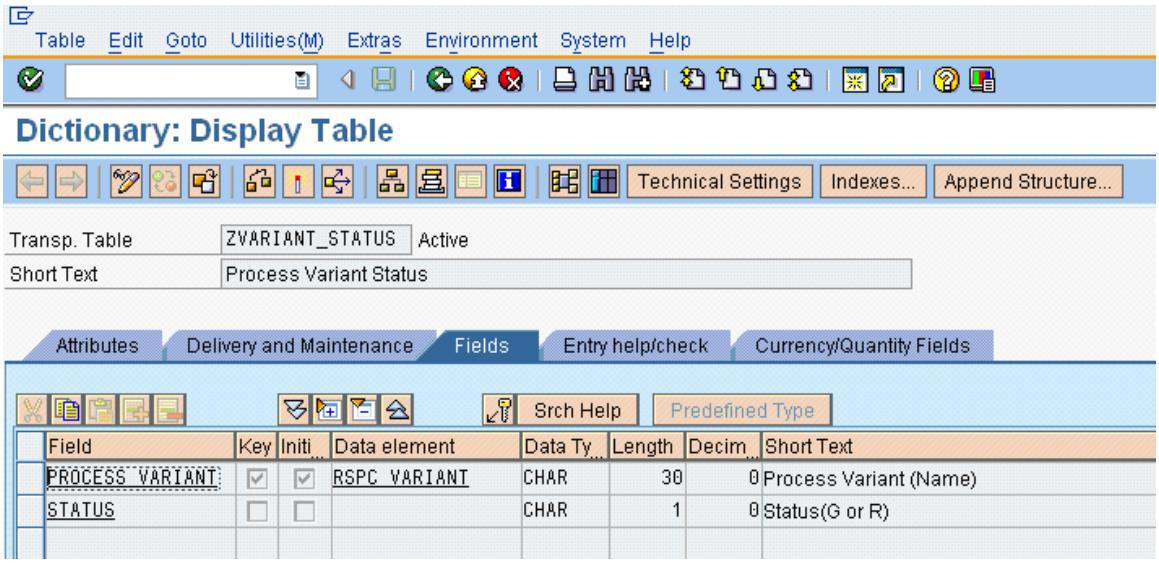
Author Bio

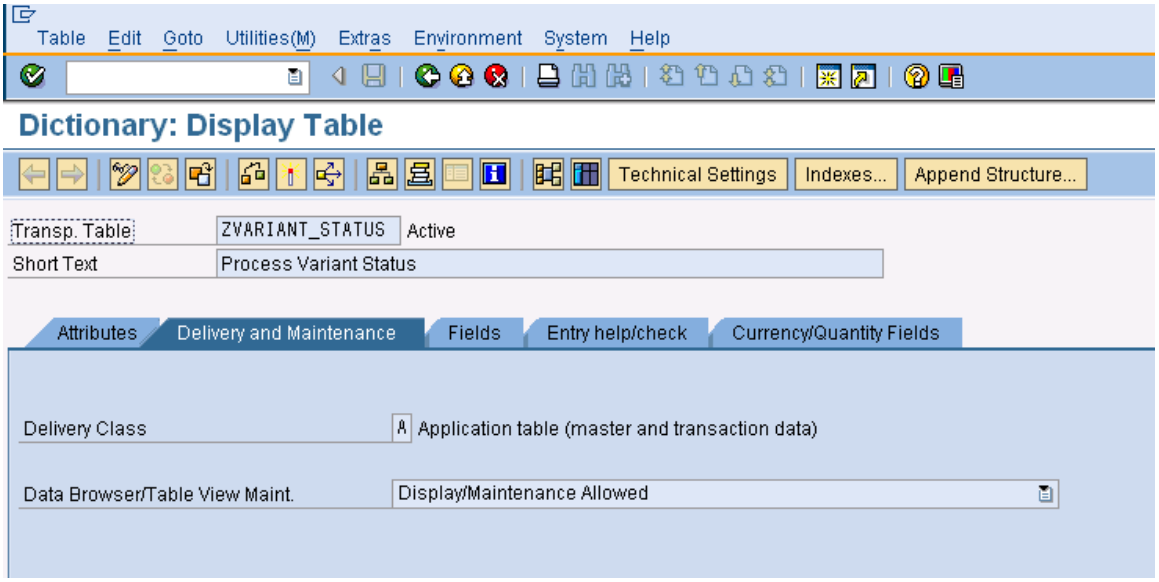
Yokesh Kumar Velusamy is a SAP BW Consultant working in Wipro Technologies since 3.5 years. He has good experience in Support as well as Implementation projects. In addition to modeling, reporting, production support and ETL process, he is good in ABAP and macro.

Table of Contents

Creating Class for ABAP Process Type

Prerequisite: Before creating the class, create a Z table (as shown in the below screenshots) to store the status for the process variant.





After creating the Z table, follow the below procedure to create Class for ABAP process type:

Step 1: In SE24, define the new class – *ZCL_ABAP_PROCESS_TYPE*.

(A useful tip here: click on utilities->display object list, gives you an SE80 type tree layout, but without the additional SE80 options cluttering the screen. This also works in SE37, SE38 etc.)

Step 2: Create the class, without adding any methods or attributes etc., and activate it.

Defining Public Section for Class

In change mode, use menu option Goto->public section. Paste this code after the line "public section" as shown below.

```
public section.

 interfaces if_rspc_call_monitor .
 interfaces if_rspc_check .
 interfaces if_rspc_execute .
 interfaces if_rspc_get_status .
 interfaces if_rspc_get_variant .
 interfaces if_rspc_maintain .
 interfaces if_rspc_transport .
 interfaces if_rspv_transport .
```

Implementing the Methods

After done with the public section, we need to implement the methods that are present in the class. They all must be implemented though most are empty. Given below are the few methods which are not empty.

```
method IF_RSPV_TRANSPORT~GET_ADDITIONAL_OBJECTS.
 cl_rspc_abap=>if_rspv_transport~get_additional_objects(
 EXPORTING i_variant = i_variant
 i_cto_mode = i_cto_mode
 i_is_content_system = i_is_content_system
 IMPORTING e_t_cto_object = e_t_cto_object
 e_t_cto_key = e_t_cto_key ).
endmethod.

method IF_RSPC_TRANSPORT~GET_TLOGO.
 cl_rspc_abap=>if_rspc_transport~get_tlogo( EXPORTING i_variant = i_variant
 i_objvers = i_objvers
 IMPORTING e_tlogo = e_tlogo
 e_objnm = e_objnm ).
endmethod.

method IF_RSPC_MAINTAIN~MAINTAIN.
 cl_rspc_abap=>if_rspc_maintain~maintain( EXPORTING i_variant = i_variant
 i_t_chain = i_t_chain
 IMPORTING e_variant = e_variant
 e_variant_text = e_variant_text ).
endmethod.

method IF_RSPC_MAINTAIN~GET_HEADER.
 cl_rspc_abap=>if_rspc_maintain~get_header( EXPORTING i_variant = i_variant
 i_objvers = i_objvers
 IMPORTING e_variant_text = e_variant_text
 e_s_changed = e_s_changed
 e_contrel = e_contrel
 e_conttimestmp = e_conttimestmp ).
endmethod.

method IF_RSPC_GET_VARIANT~GET_VARIANT.
 cl_rspc_abap=>if_rspc_get_variant~get_variant( EXPORTING i_variant = i_variant
 i_t_chain = i_t_chain
```

```
 i_t_select = i_t_select
 i_objvers = i_objvers
 IMPORTING e_variant = e_variant
 e_variant_text = e_variant_text
 EXCEPTIONS nothing_selected = 1 ).
 IF sy-subrc EQ 1.
  MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
  WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4
  RAISING nothing_selected.
 ENDIF.
endmethod.

method IF_RSPC_GET_VARIANT~WILDCARD_ENABLED.
 result = cl_rspc_abap=>if_rspc_get_variant~wildcard_enabled( ).
endmethod.

method IF_RSPC_GET_VARIANT~EXISTS.
 r_exists = cl_rspc_abap=>if_rspc_get_variant~exists( i_variant = i_variant
 i_objvers = i_objvers ).
endmethod.

method IF_RSPC_GET_STATUS~GET_STATUS.
 cl_rspc_abap=>if_rspc_get_status~get_status( EXPORTING i_variant = i_variant
 i_instance = i_instance
 i_dont_update = i_dont_update
 IMPORTING e_status = e_status ).
endmethod.

method IF_RSPC_EXECUTE~GIVE_CHAIN.
 return = cl_rspc_abap=>if_rspc_execute~give_chain( i_variant ).
endmethod.

method IF_RSPC_CHECK~CHECK.
 cl_rspc_abap=>if_rspc_check~check(
 EXPORTING
  i_s_process = i_s_process
  i_t_chain = i_t_chain
  i_t_chains = i_t_chains
 IMPORTING
  e_t_conflicts = e_t_conflicts
 ).
endmethod.

method IF_RSPC_CHECK~GIVE_ALL.
 return = cl_rspc_abap=>if_rspc_check~give_all( i_variant ).
endmethod.

method IF_RSPC_CALL_MONITOR~CALL_MONITOR.
 cl_rspc_abap=>if_rspc_call_monitor~call_monitor(
 i_variant = i_variant
 i_instance = i_instance ).
endmethod.

method IF_RSPC_EXECUTE~EXECUTE.
 cl_rspc_abap=>if_rspc_execute~execute(
 EXPORTING
```

```
  i_variant = i_variant
  i_event_start = i_event_start
  i_eventp_start = i_eventp_start
  i_t_processlist = i_t_processlist
  i_logid = i_logid
  i_t_variables = i_t_variables
  i_synchronous = i_synchronous
  i_simulate = i_simulate
  i_repair = i_repair
 IMPORTING
  e_instance = e_instance
  e_state = e_state
  e_eventno = e_eventno
  e_hold = e_hold
 ).

***Send the Success/Failure message to Process chain by passing value to e_state
select single status from zvariant_status into e_state
  where process_variant = i_variant.
```

Things to be noted

- In Method - IF_RSPC_EXECUTE~EXECUTE, *i_variant* contains the name of the ABAP process variant which triggered the ABAP Program.
- Based on the variant name, get the status value from the table *–ZVARIANT_STATUS* and pass the value to *'e_state'* which will report the success or failure message to the process chain.

Business Scenario

Example

Let us say that there are three different loads starting at different point of time and we need to carry out some loads after the successful completion of all the above 3 loads. In such cases, we can write an ABAP program to meet the above requirement and implement it based on the new ABAP process type which we have created.

Sample ABAP Program

Program which is being called by the ABAP process type needs to get the input for Process Type Variants from the ABAP process type and based on the input it should send the success/failure message to the process chain using Z table.

Follow the below procedure to write the program and make sure that the requirement is satisfied

Under the declaration part, declare *Parameter* and *Data* statement as shown below. Those will be helpful in getting the name of the variant which triggered the ABAP program and for updating the status to the Z table.

```
*****<Declaration Part>*****
*TYPES:....
*TABLES:...
*RANGES:...
PARAMETERS: pc_var TYPE rspc_variant.
"Above statement is to get the process variant name which triggered the program
Data:  BEGIN OF wa_variant_status occurs 0,
         process_variant type rspc_variant,
```

```
          status type c length 1,
        END OF wa_variant_status.
"Above statement is update the process variant name and its status to Z table
*****</Declaration Part>*****

Write your ABAP code to find out whether the requirement is met.
*****<Program Logic to find out whether the requirement is met/not>*****
*
*Enter your code here
*
*****</Program Logic to find out whether the requirement is met/not>*****

Once you find that execution of the program is a success or failure, set the status
flag ('R' for failure and 'G' for success) and update the entry in Z table.
***<Update the Z table with status of the Process variant>***
if sy-subrc = 0. " The ABAP program failed
 wa_variant_status-PROCESS_VARIANT = pc_var.
 wa_variant_status-status = 'R'.
else. " The ABAP program was successful
 wa_variant_status-PROCESS_VARIANT = pc_var.
 wa_variant_status-status = 'G'.
endif.
MODIFY zvariant_status from wa_variant_status.
***</Update the Z table with status of the Process variant>***
```

Things to be noted

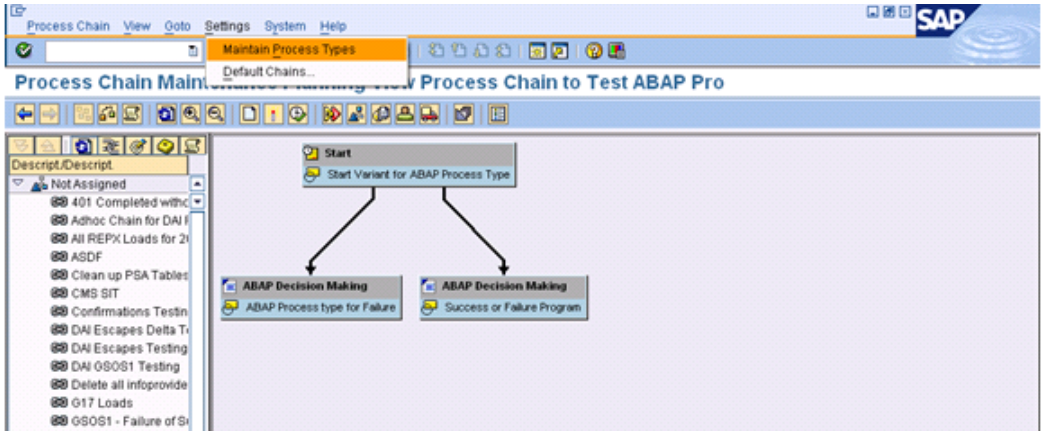
- The variant - *pc_var* is nothing but the input which will be entered while executing the program and it should hold the name of the ABAP Process type variant which triggered the ABAP program.
- In order to achieve that we need to pass the name of the process variant from the ABAP process type using program variant.This will be taken care in the backend if you have defined the process chain variant for ABAP process type.(Refer the screenshots given below)
- Using the ABAP program, find out whether the execution of program is success or a failure and based on that update the status field in Z table for the corresponding process variant.

Implementing in Process Chain

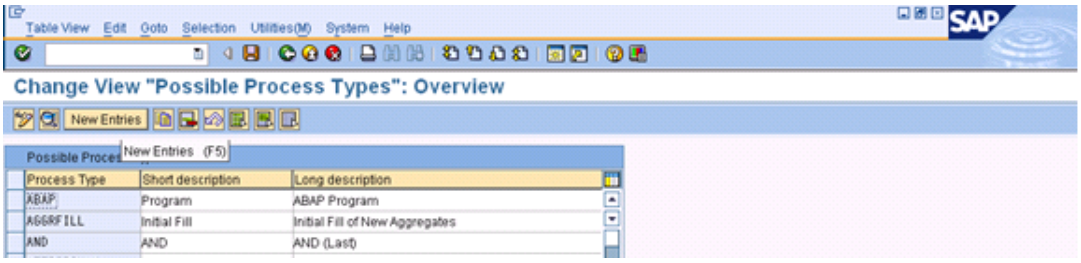
Follow the below steps to implement the ABAP process type in the system and to use it in the Business Scenario which we have discussed above.

Step-By-Step Procedure:

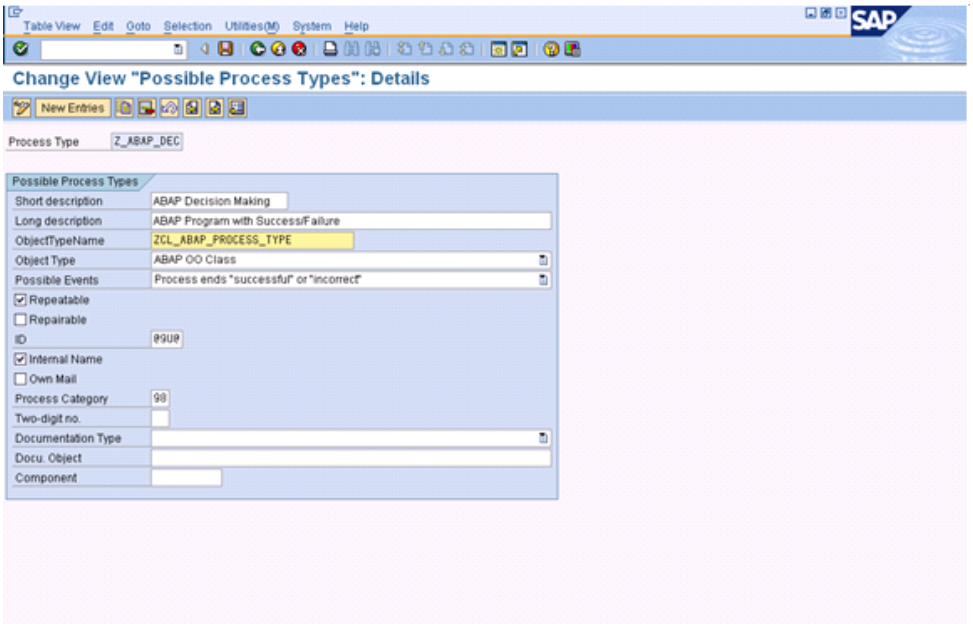
1) In RSPC, go to the plan view of any process chain and choose menu option *'Settings' -> 'Maintain Process Types'* as shown below.



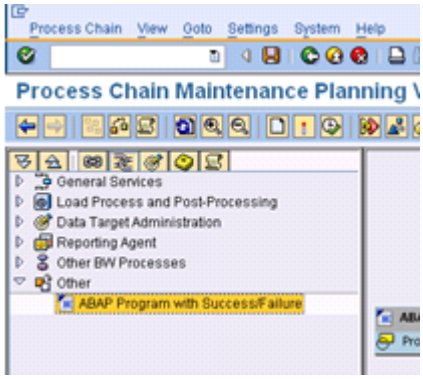
2) Create an entry for the ABAP Process type by clicking on the button *"New Entries".*



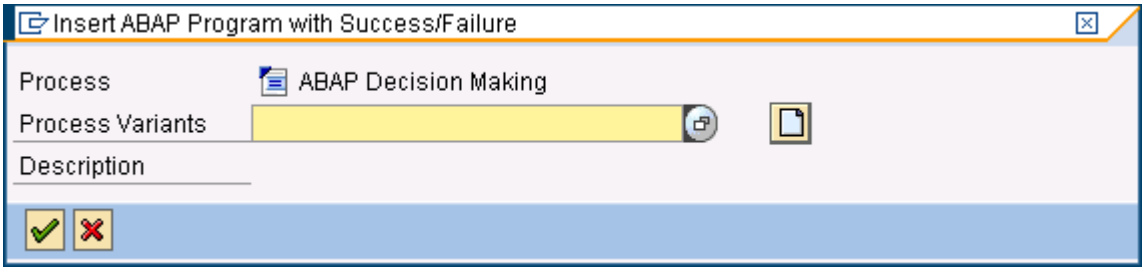
3) Fill the details for the process type as shown below and give the name of the class which you have created for the ABAP process type. Then save the process type.



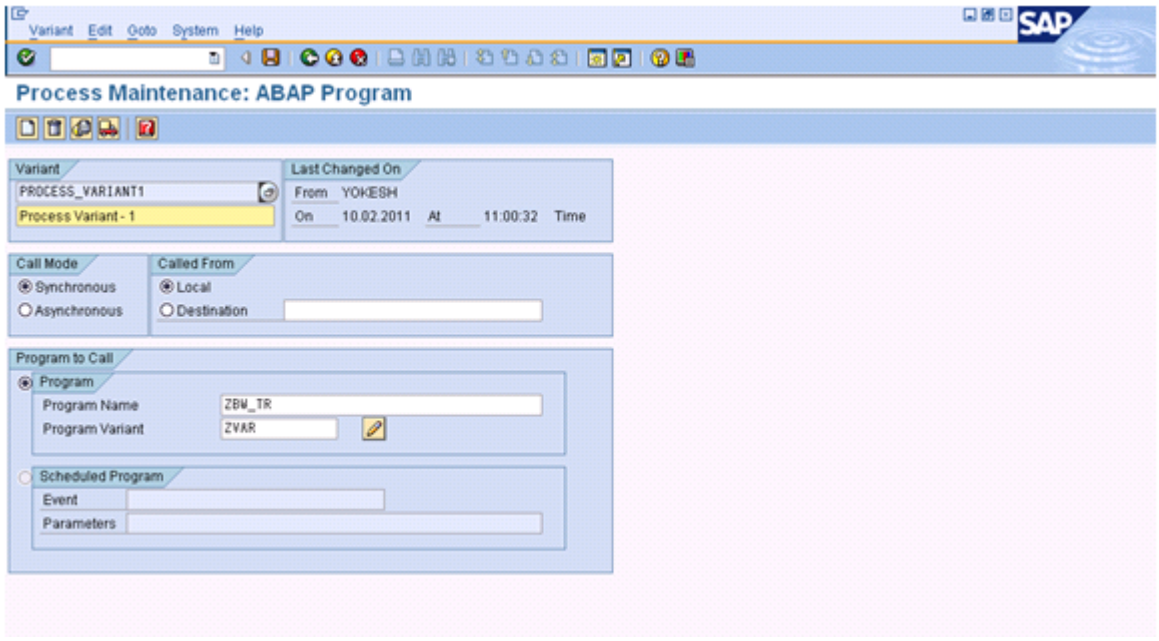
4) Now, the ABAP process type which we have defined will be available under *'Other'* Process type as shown in the below screenshot.



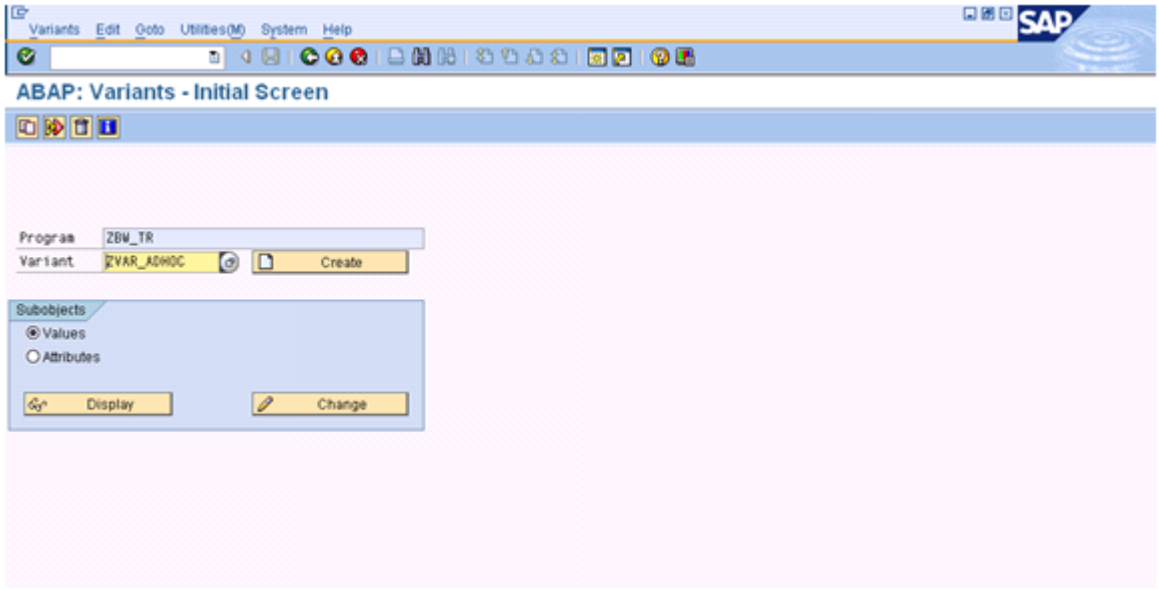
5) Create a variant for the ABAP Process type.



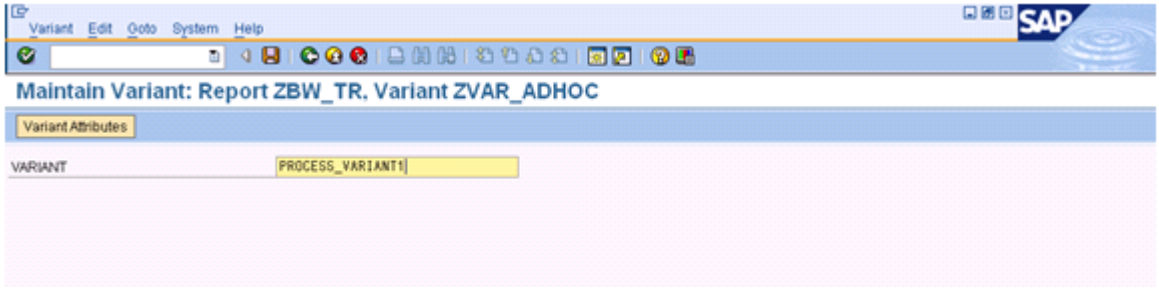
6) Give the name of the ABAP program and create a program variant(PROCESS_VARIANT1) for sending the process type variant name to the ABAP Program.



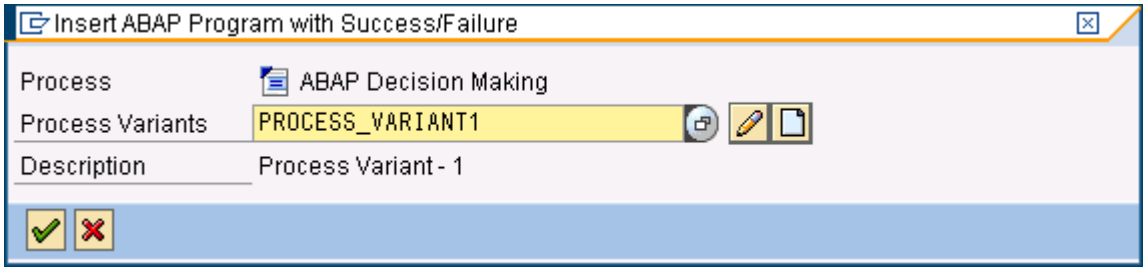
7) Give the input value to the ABAP Program as *'PROCESS_VARIANT1'* by creating a program variant.



(Value of the program variant should be same as the process type variant as shown below)



8) Once you have successfully created the program variant, insert the process variant to the process chain.



9) You can follow the steps from *4 - 8* when you need to create the ABAP Process type variant.

Related Content

http://help.sap.com

http://help.sap.com/saphelp_nw70/helpdata/en/06/efd63b54e56276e10000000a11402f/content.htm

http://help.sap.com/saphelp_nw70/helpdata/en/35/43e4424db15604e10000000a155106/content.htm

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.