

# Creating an ABAP Process Type for Process Chains in BI that Reports Success or Failure.



## Applies to:

SAP BI 3.5, 7.0

## Summary

Within standard BI, there is an ABAP process type for use in Process Chains. This type runs a 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 I've found them to be rather code heavy. This article explains how to create a new Process Type, the reports success or failure, with minimal, but robust coding. It also demonstrates the useful Object Oriented programming technique of pass-through methods.

**Author:** Matthew Billingham

**Company:** Yireh GmbH

**Created on:** 20 December 2007

## Author Bio

Matthew Billingham, Consultant

Matthew Billingham holds a BSc in Pure Mathematics and Theoretical Computer Science from the University of Essex, England. Matthew has worked in IT since 1988, both as an employee and as an independent consultant for many companies including: Eurotunnel, Vodafone, British Gas and Coats PLC, where as Development Manager, he reported to the director of IT. From 2002 to the end of 2004, Matthew was employed as Head of Applications Development within Novartis Pharma in Basel, Switzerland, managing the development side of a global SAP rollout. He now runs his own technical consultancy company, Yireh GmbH, providing guidance and designing solutions for third parties, covering all aspects of SAP technologies. Matthew's expertise is in program design, troubleshooting and knowing just how SAP works! He particularly enjoys finding solutions to seemingly intractable problems.

## Table of Contents

Creating the new Process Type class .....	3
The ABAP Program.....	6
Setting up the Process Type.....	6
Disclaimer and Liability Notice.....	7

## Creating the new Process Type class

Unfortunately, the standard ABAP Process Type class is final. So the opportunity to use inheritance to define our new class is lost. Instead, I've used a different technique, that allows me to reuse the standard SAP functionality, adding my own enhancements, **without making a copy of standard SAP code**, with all the risk that entails.

In SE24, we define our new class. ZCL\_ABAP\_PT\_DECISION. ( 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. ). Create the class, without adding an methods or attributes etc., activate it.

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

constants failed type rspc_state value 'R'. "#EC NOTEXT
constants status_mem_id type char20 value 'PC_ABAP_STATUS'. "#EC NOTEXT
constants success type rspc_state value 'G'. "#EC NOTEXT

```

The protected and public section contain nothing. Now we need to implement the methods of the interface. They all must be implemented - though most are empty. Those that are not empty are:

```

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_control      = e_control
                                           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_eventtp_start = i_eventtp_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
    ).

    IMPORT e_state TO e_state FROM MEMORY ID zcl_abap_pt_decision=>status_mem_id.

ENDMETHOD.

```

You'll notice, that with the exception of the last method, all I've done is pass and retrieve identical parameters to CL\_RSPC\_ABAP static methods. This is known as passing through. In the last method, I also do this, but I have an additional statement. This method executes the ABAP program in the Process Type variant, and then sets the state of the Process Type according to that stored in the memory id.

## The ABAP Program

The ABAP program that the process type will actually call, needs to set its status - whether it succeeded or failed - and report that to the process type handling class. This I've done simply using memory ids. You could follow the example in Jürgen Noe's blog, where he addresses the same issue, and use a table to store the result of the called program.

```
* Set the flag for the process chain to read
if ... " The ABAP program failed
  l_flag = zcl_abap_pt_decision =>failed.
else. " The ABAP program was successful
  l_flag = zcl_abap_pt_decision =>success.
endif.
export e_state from l_flag to memory id zcl_abap_pt_decision=>status_mem_id.
```

## Setting up the Process Type

The final bit of work is setting up the process type for use within a process chain. From transaction RSPC, go to Settings->Maintain Process types. ( You might have to select a process chain first to activate this menu option ). Create a new process type:

The screenshot shows the 'Maintain Process Types' transaction in SAP. The process type 'Z\_ABAP\_DEC' is selected. The configuration is as follows:

Possible Process Types	
Short description	Decision ABAP
Long description	ABAP program with result
ObjectTypeName	ZCL_ABAP_PT_DECISION
Object Type	ABAP OO Class
Possible Events	Process ends "successful" or "incorrect"
<input checked="" type="checkbox"/> Repeatable	
<input type="checkbox"/> Repairable	
ID	@9UE
<input checked="" type="checkbox"/> Internal Name	
<input type="checkbox"/> Own Mail	
Process Category	98
Two-digit no.	
Documentation Type	
Docu. Object	
Component	

## **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.