SAP Fiori Notifications – Part 2

05. March 2021 von *Igor Muntoreanu*

In this blog we will explain how to connect SAP Fiori Notifications with the classic SAP Workflow. We will explain it along with a Real Business Scenario Case in a step-by-step example with the entire code snippets.

Prerequisites

To such functionality work, you need before customizing the Fiori Notification. To do so, follow the steps discussed in the Part 1 of this blog:

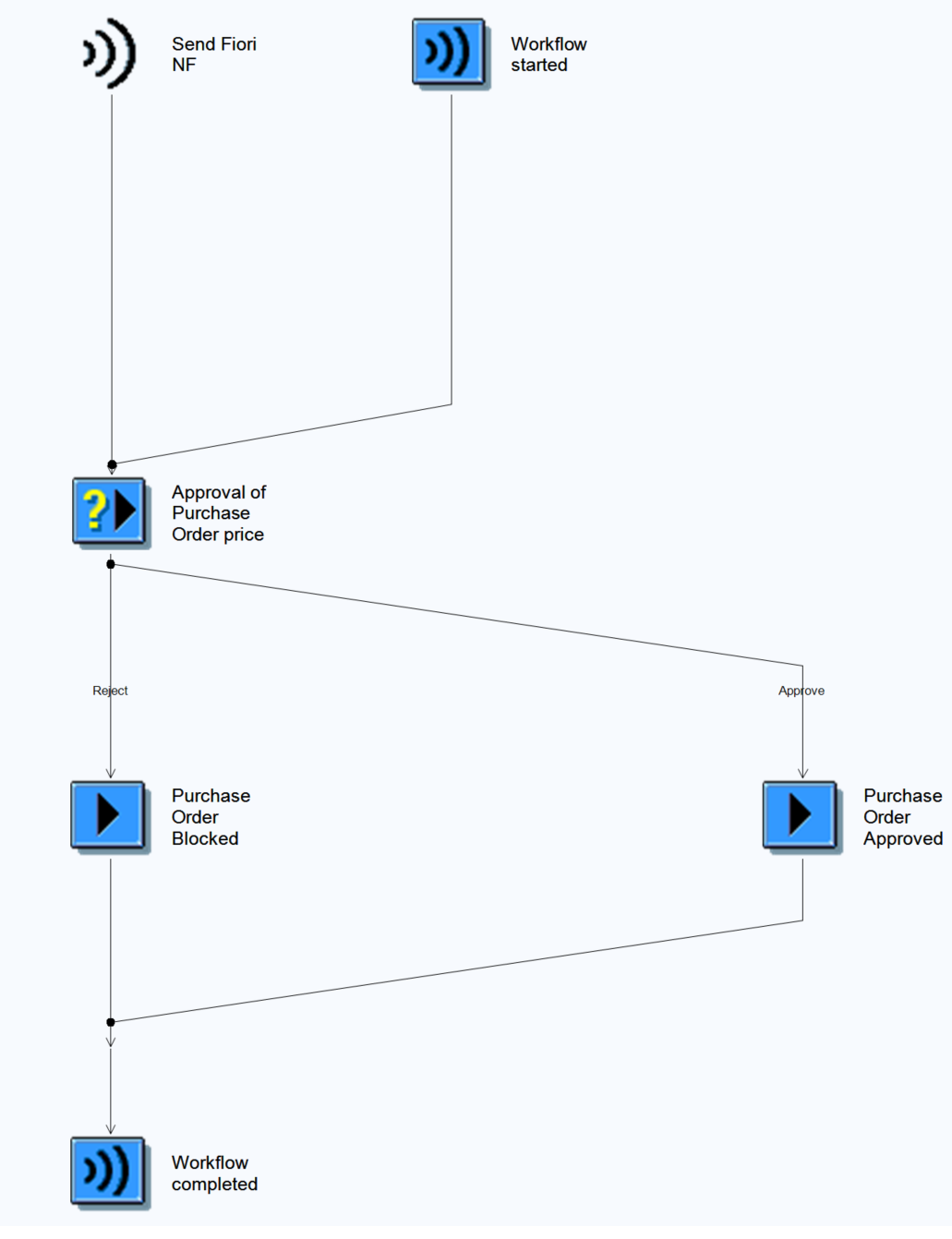
<https://www.exxcellent.de/tech-blog/2021-01-25-sapfiori_notifications1/SAP-Fiori-Notifications-Part-1/>

Real Business Case Scenario 3

In this Scenario we are going to send the Fiori Notification based on the SAP Workflow result that we are going to create from scratch.

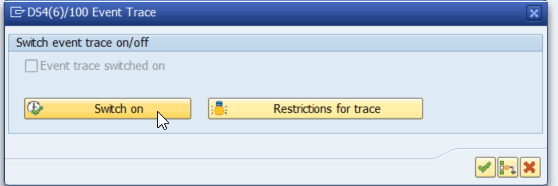
The scope: everytime a Purchase Order is created or changed in transaction ME21N/ME22N with price bigger or equal to 10.000 EUR a Fiori Notification will be sent to the Head of Department. Then, the Head of Department is going to approve or reject the Purchase Order. When it is rejected, the Purchase Order items will be blocked.

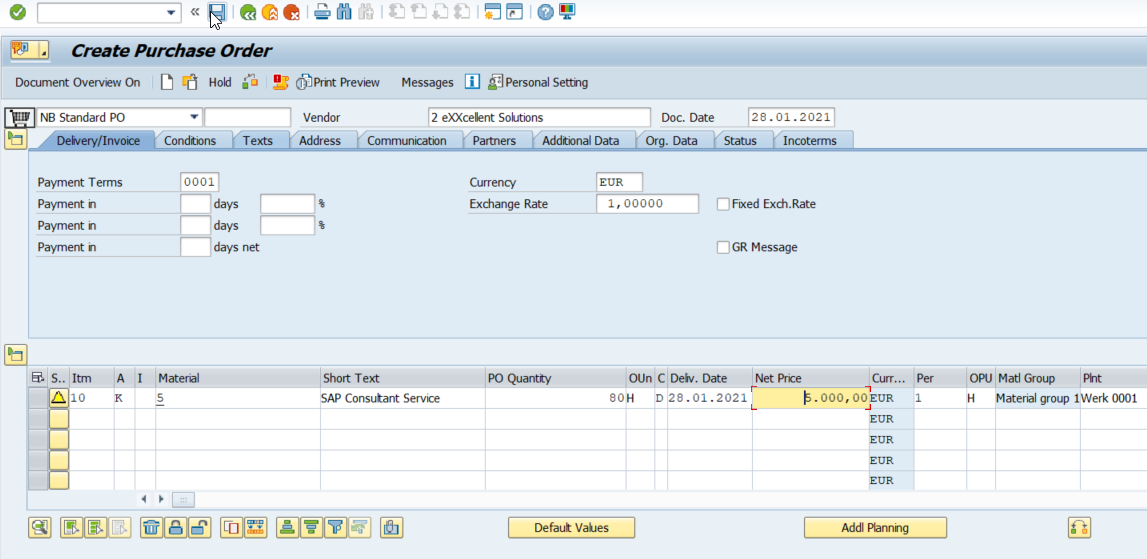
Below we have a big picture of the flow of the process, which is going to start once we create or change a Purchase Order with price over 10.000 EUR:



To do this, first we need to create a custom workflow for this criteria.

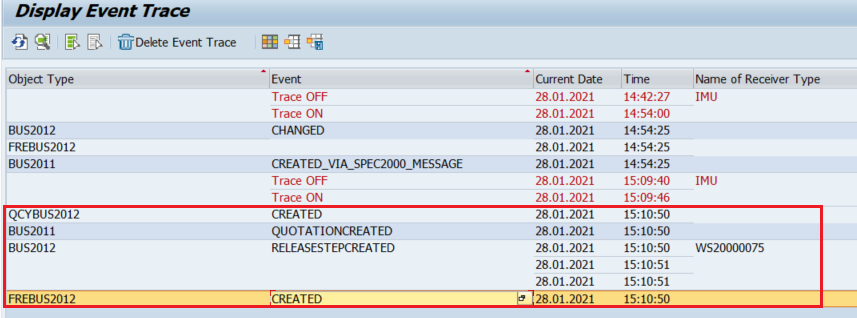
First, lets create a Purchase Order via transaction ME21N and trace the workflow Object Types in background. Before saving the Purchase Order lets activate the workflow trace trough SWELS transaction and check the results:



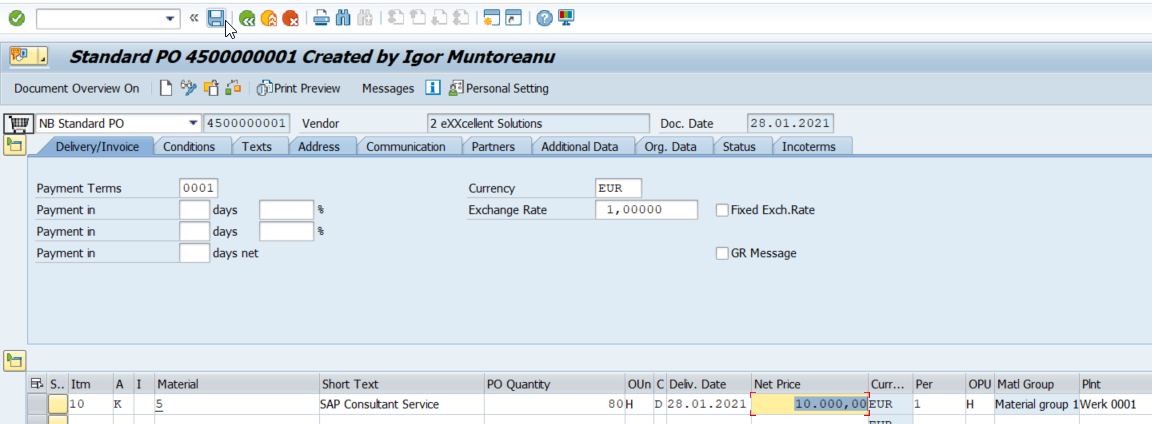




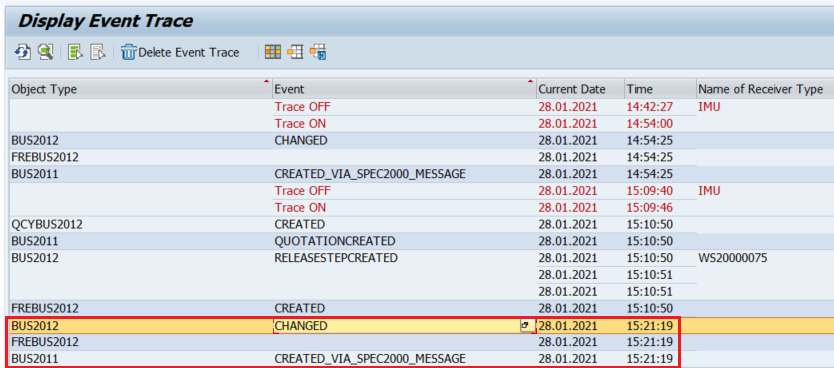
Checking the trace in transaction SWEL, we can see that the following Object Types were called:



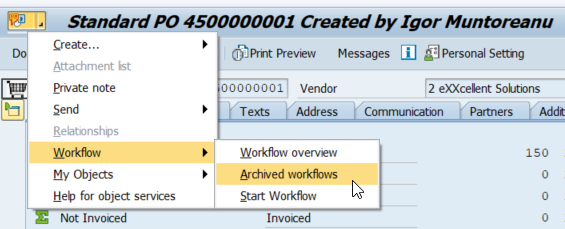
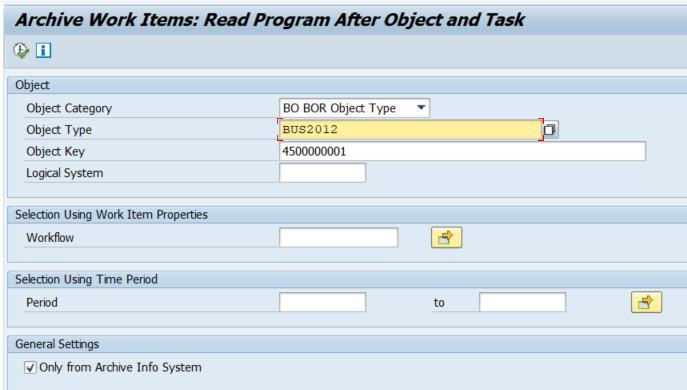
Now let’s change the Purchase Order to 10.000 EUR via ME22n and check the trace again:



Trace:

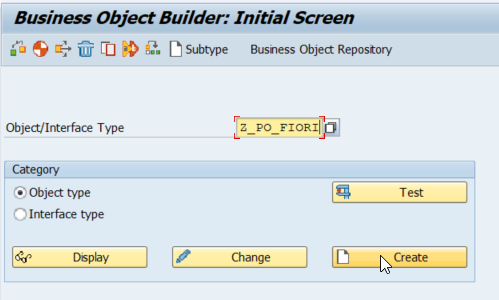


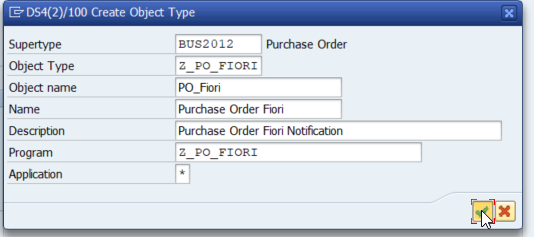
So we have the Business Object BUS2012 as the main Object Type for the Purchase Orders.

Another way to find the correct Object Type, would be by the transaction itself:  
  
  
  


Now, copy this Object Type and create a custom Object Type via transaction SWO1. Use the BUS2012 as the super type:

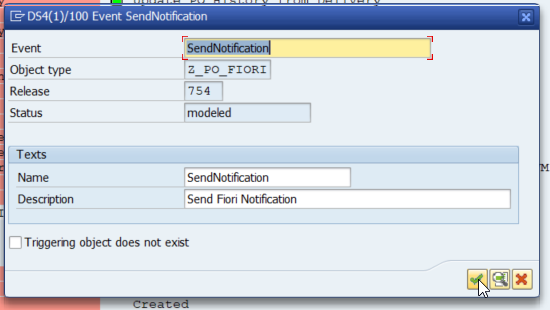
SWO1:



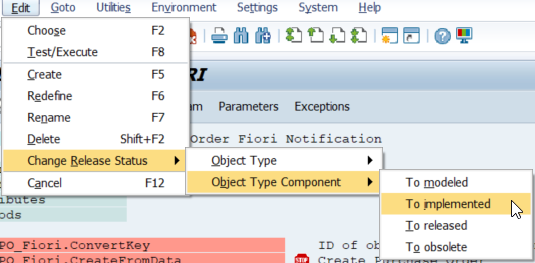


In the next screen make sure to always change the release status of the Object Type and Object Type Component to „implemented“ after you change or create something. Set the release status to „released“ when you want to transport it to the test system or production system.

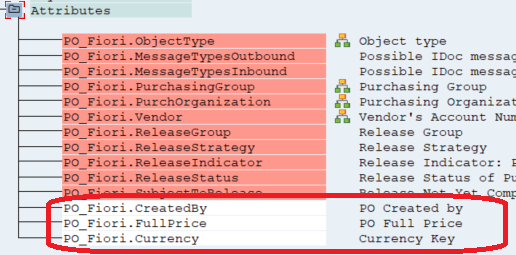
Create a new event in the object type Z\_PO\_FIORI:

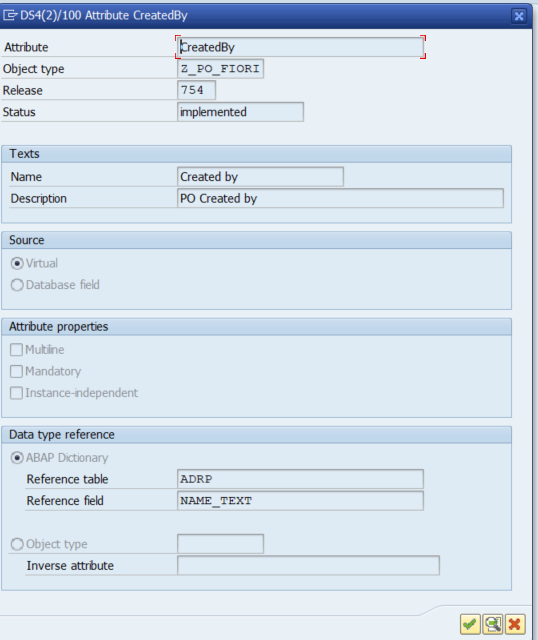


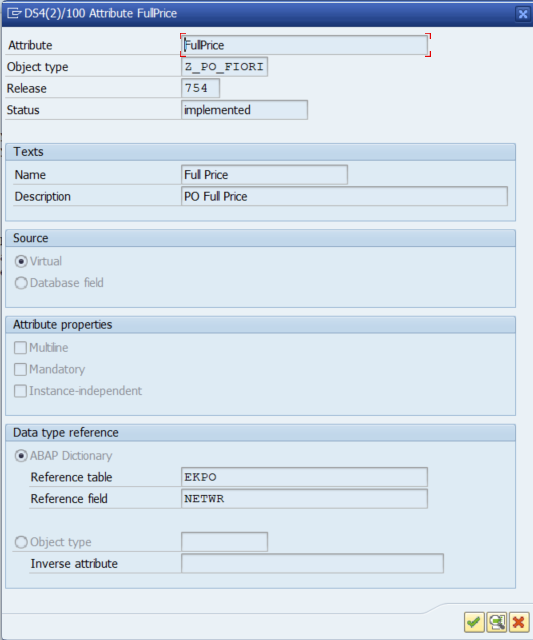
Then update the status to implemented:

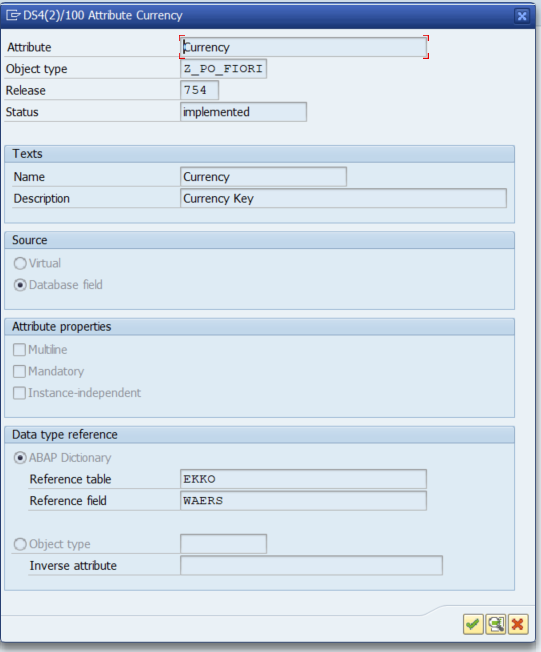


Now lets create 3 new attributes like below:









Now lets code the „GET“ of these attributes, by clicking in „Program“:

CreatedBy:

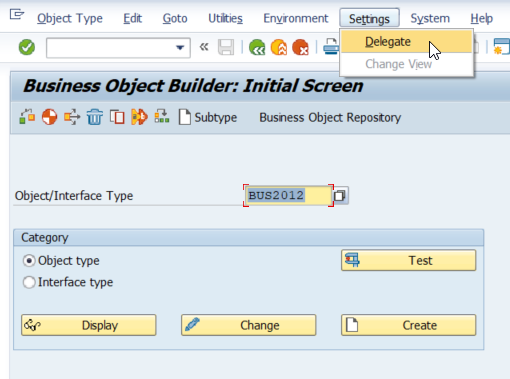
|  |
| --- |
| get\_property createdby changing container.  SELECT SINGLE ernam        FROM ekko        INTO @DATA(lv\_ernam)        WHERE ebeln = @object-key-purchaseorder. IF sy-subrc = 0.   SELECT SINGLE adrp~name\_text INTO @DATA(lv\_name\_text)     FROM usr21 JOIN adrp ON usr21~persnumber = adrp~persnumber     WHERE usr21~bname = @lv\_ernam. ENDIF.  object-createdby = lv\_name\_text.  swc\_set\_element container 'CreatedBy' object-createdby.  end\_property. |

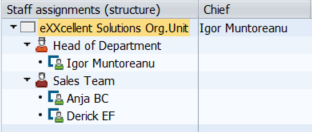
FullPrice:

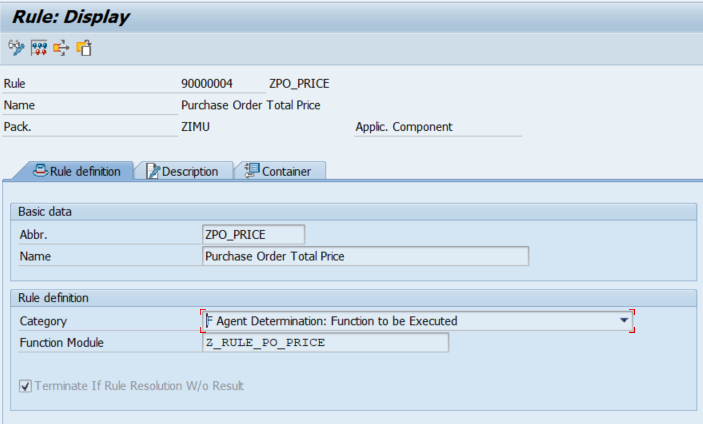
|  |
| --- |
| get\_property fullprice changing container.  DATA: lv\_fullprice TYPE ekpo-netwr.  SELECT a~ebeln, a~ebelp, a~netwr, a~menge, b~waers, b~ernam   FROM ekpo AS a   INNER JOIN ekko AS b ON ( a~ebeln = b~ebeln )   INTO TABLE @DATA(lt\_ekpo)   WHERE a~ebeln = @object-key-purchaseorder. IF sy-subrc = 0.    LOOP AT lt\_ekpo ASSIGNING FIELD-SYMBOL(<fs\_ekpo>).     lv\_fullprice = <fs\_ekpo>-netwr + lv\_fullprice.   ENDLOOP.    swc\_set\_element container 'FullPrice' lv\_fullprice.  ENDIF.   end\_property. |

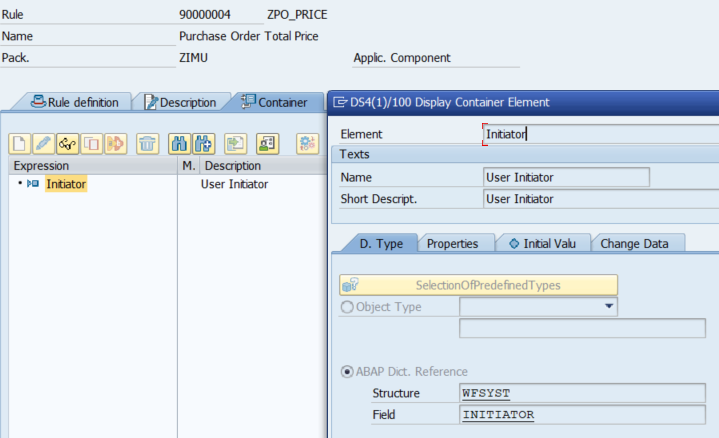
The Currency is automatically inherited from object type BUS2012, so it is not necessry to do a customizing code for it.

Now, delegate the BUS2012 to our Object Type Z\_PO\_FIORI trough transaction SWO1:

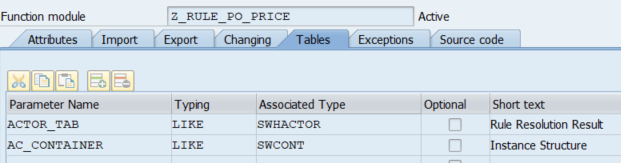


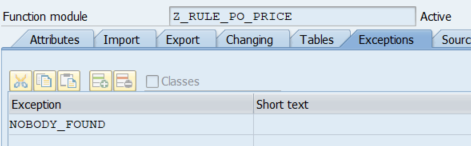
To send the notification to the Head of Department, we need to create a Rule. This rule will get the workflow initiator, in other words, the user who created or changed the Purchase Order and then get the Head of Department, who is responsible for this user. For example:  
  
  
  
If the user Anja or Derick, create or change a Purchase Order over 10.000, then the Head of Department Igor will receive a notification to approve or reject the Purchase Order.

We create the rule through transaction PFAC:  
  


Create a new import element in the Container tab:  


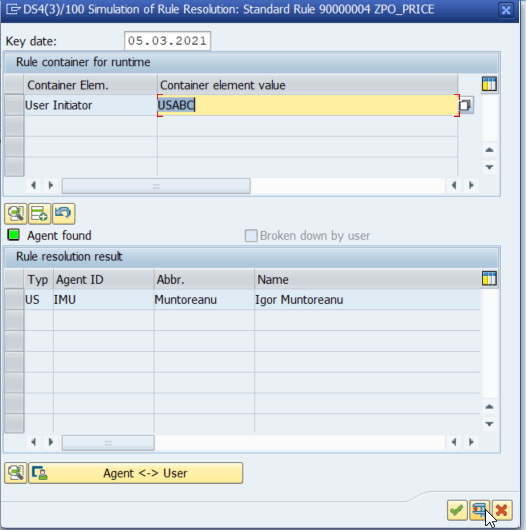
Now lets code the Function Module „Z\_RULE\_PO\_PRICE“:





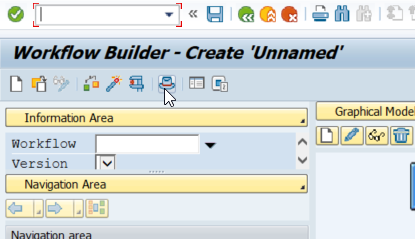
|  |
| --- |
| FUNCTION z\_rule\_po\_price. *\*"----------------------------------------------------------------------* *\*"\*"Local Interface:* *\*"  TABLES* *\*"      ACTOR\_TAB STRUCTURE  SWHACTOR* *\*"      AC\_CONTAINER STRUCTURE  SWCONT* *\*"  EXCEPTIONS* *\*"      NOBODY\_FOUND* *\*"----------------------------------------------------------------------*    INCLUDE <cntn01>.    DATA: lt\_objec     TYPE STANDARD TABLE OF objec,         lv\_initiator TYPE wfsyst-initiator,         lv\_user      TYPE sy-uname.    swc\_get\_element ac\_container 'Initiator' lv\_initiator.    lv\_user = lv\_initiator+2.    *"The Notification will be sent only to the Head of the Department*   *"Get the Organization Unit*   CALL FUNCTION 'RH\_STRUC\_GET'     EXPORTING       act\_otype      = 'O'       act\_objid      = '50000077' *" eXXcellent Solutions Organization Unit*       act\_wegid      = 'AI\_ORGUS' *" All users of the Organization Unit*     TABLES       result\_objec   = lt\_objec     EXCEPTIONS       no\_plvar\_found = 1       no\_entry\_found = 2       OTHERS         = 3.   IF sy-subrc = 0.     *"Send notification only to the Head of Department*     READ TABLE lt\_objec TRANSPORTING NO FIELDS WITH KEY objid = '50000079'.     IF sy-subrc = 0.       DATA(lv\_tabix) = sy-tabix + 1.        TRY.           DATA(ls\_object) = lt\_objec[ lv\_tabix ].            CHECK lv\_user <> ls\_object-realo. *"Avoid the Head of sending the notification to himself*            READ TABLE lt\_objec TRANSPORTING NO FIELDS WITH KEY realo = lv\_user. *"Make sure that the user is in the Organization Unit*           IF sy-subrc = 0.             actor\_tab-otype = 'US'.             actor\_tab-objid = ls\_object-realo.             APPEND actor\_tab.           ENDIF.          CATCH cx\_sy\_itab\_line\_not\_found.       ENDTRY.     ENDIF.   ENDIF.    IF actor\_tab IS INITIAL.     RAISE nobody\_found.   ENDIF.  ENDFUNCTION. |

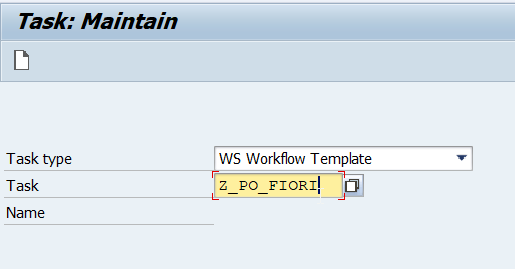
Now lets simulate the rule:



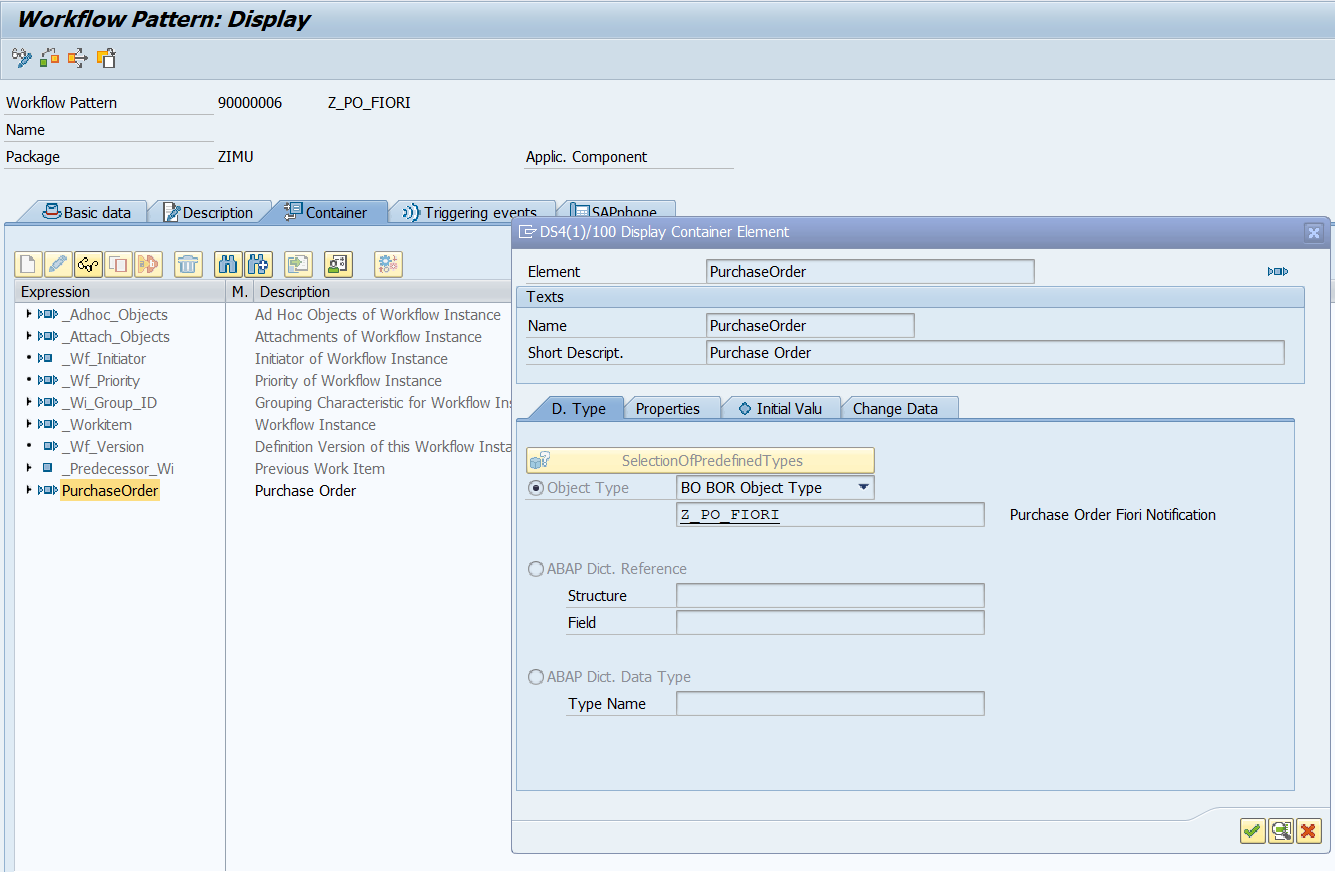
As we can see, it is working.

Now enter in the Workflow Builder through PFTC or SWDD and lets create the flow. The flow is going to have a task where the user will decide to approve or not the Purchase Order:

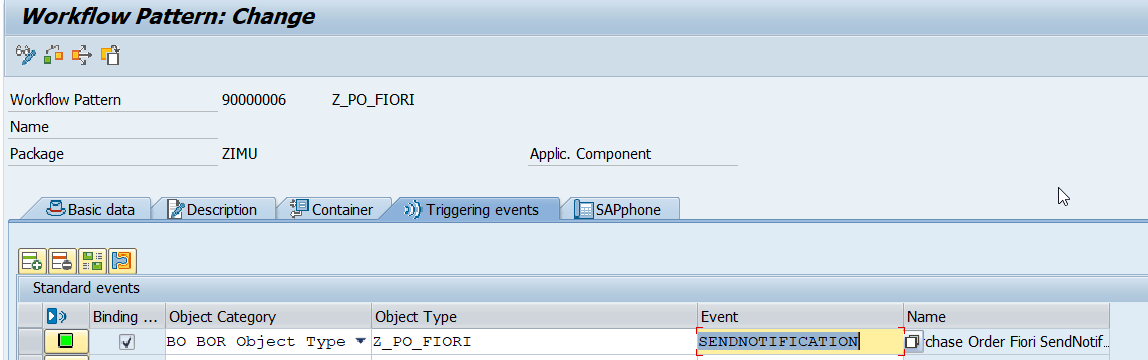




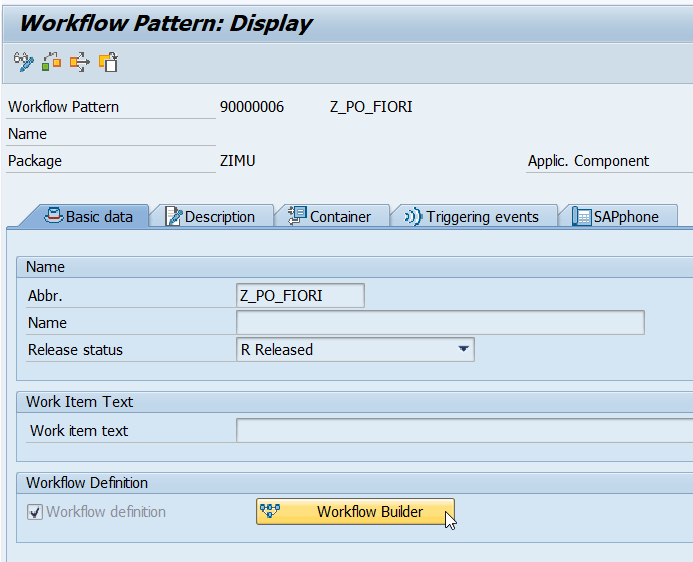
Create the element „PurchaseOrder“ in the container and link it to the Object Type „Z\_PO\_FIORI“:



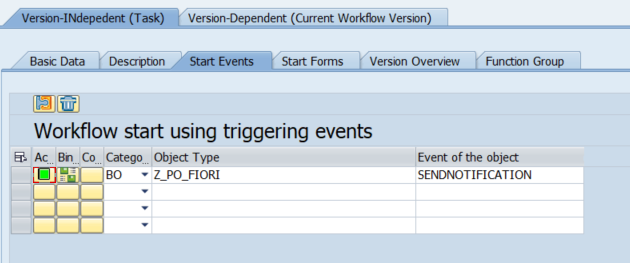
Do the connection between the Task and the Object type at the „Triggering Events“ Tab:



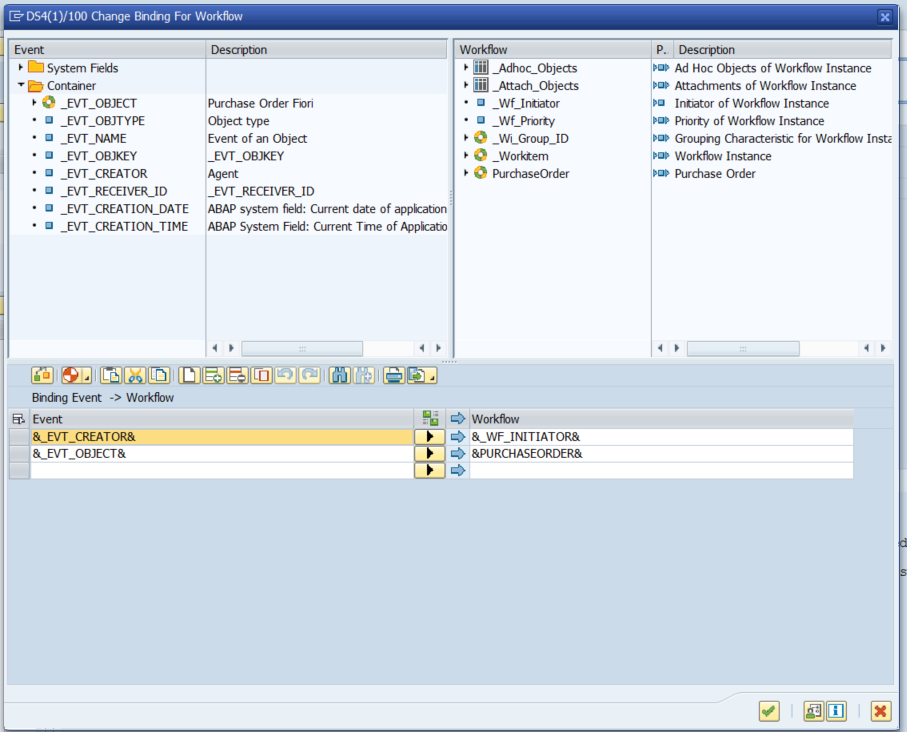
Navigate to the workflow builder through the button at the „Basic data“ tab or through the transaction SWDD:

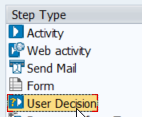


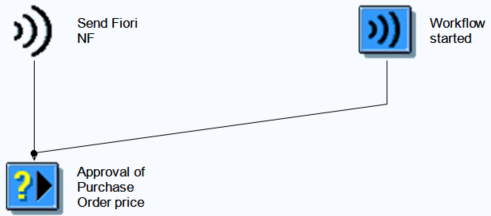
Click on the Header, through the Hat button and configure the Start Events like below:

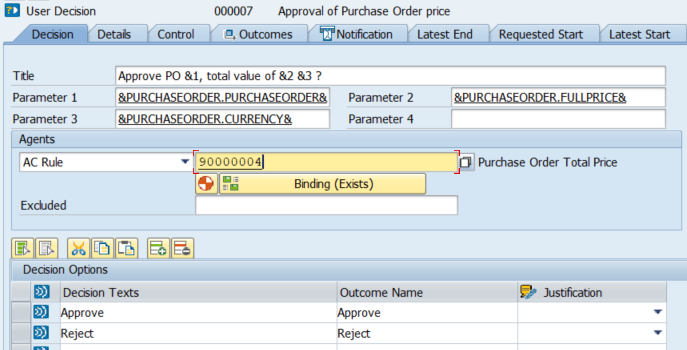


Do the Binding:

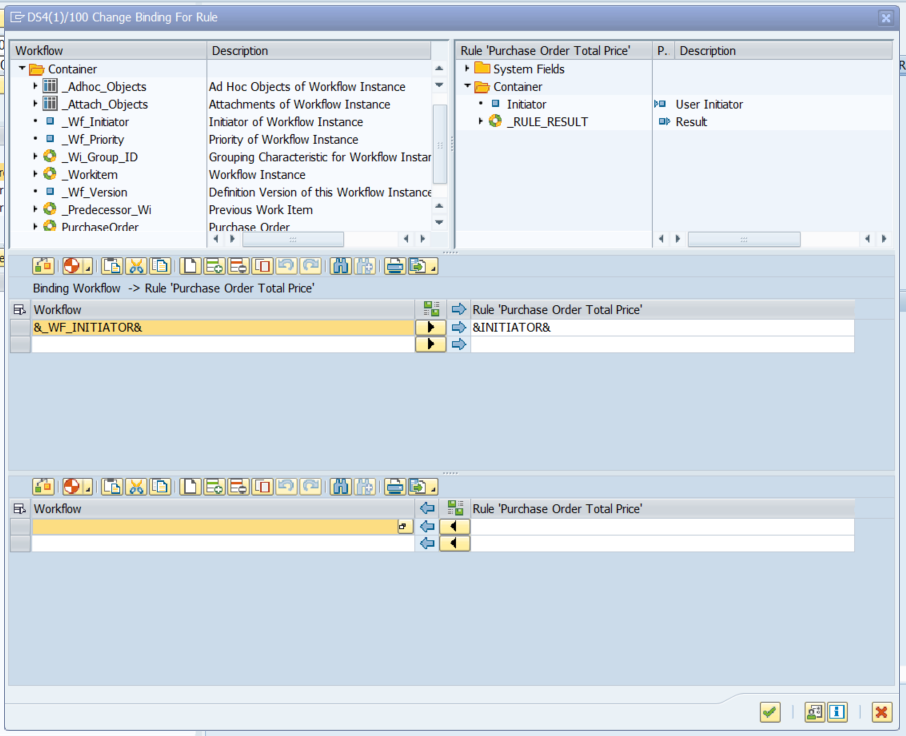


Now lets create a User Decision task:  
  


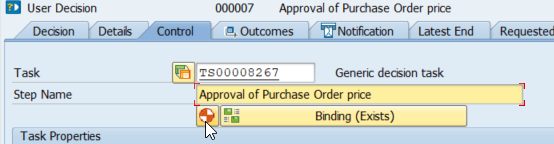


Give it a Title, set the parameters, give the rule created and write the decision options:

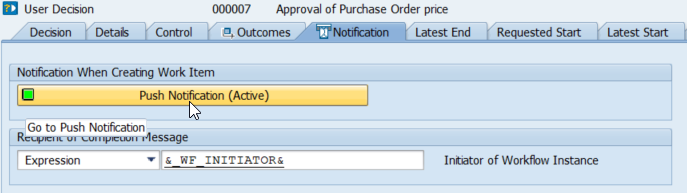
Now do the Binding between the Workflow and the Rule:

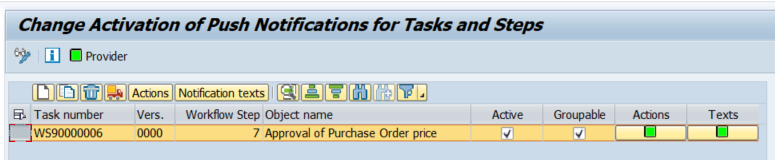


Go to the Control tab and do the automatic Binding between the Workflow and the Task:

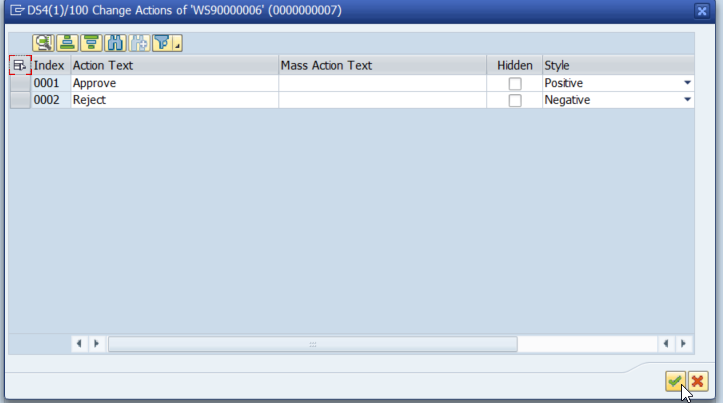


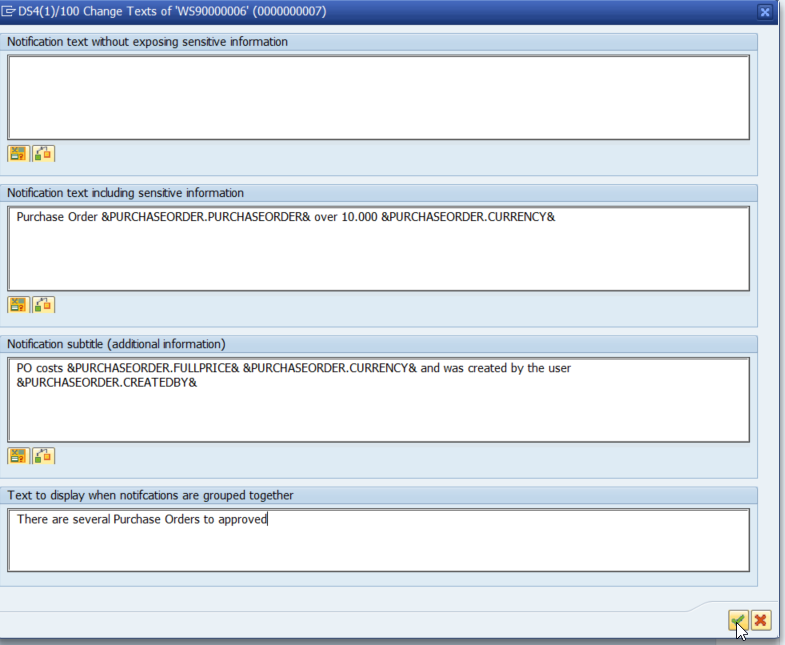
Now, go to the Notification tab, give the Workflow initiator as the Recipient of Completion Message and click on the Push Notification:



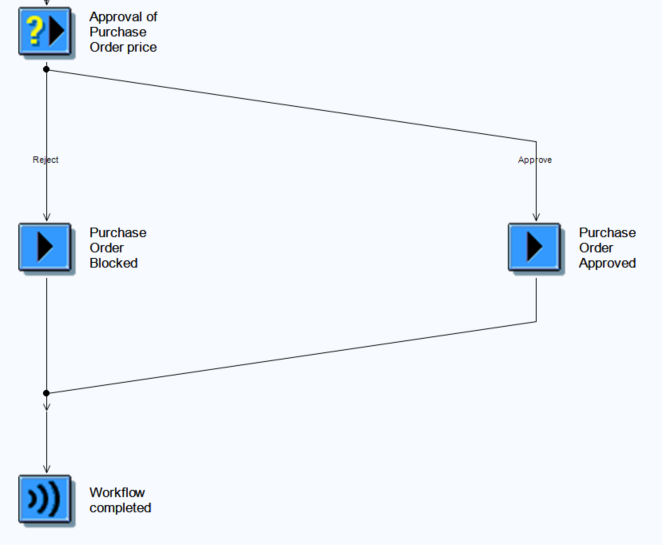


Now, fill the Actions and Texts. Save it:

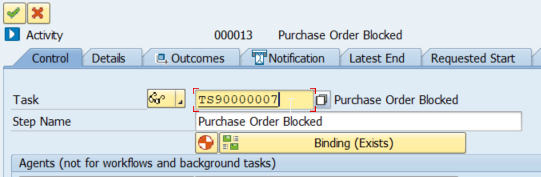




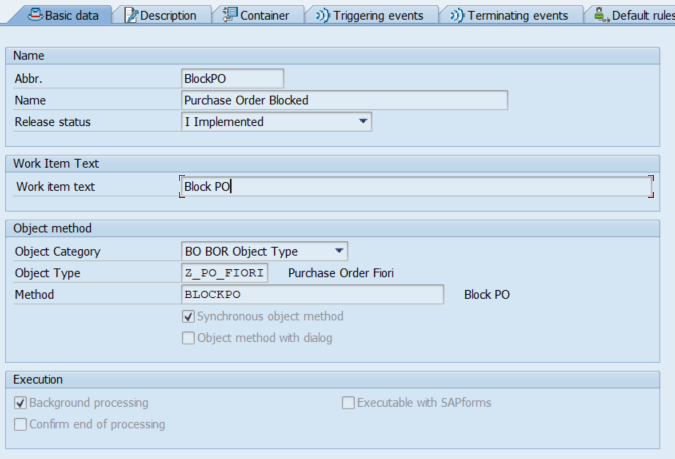
Now we need to create 2 Tasks, one to approve and another one to reject the Purchase Order:

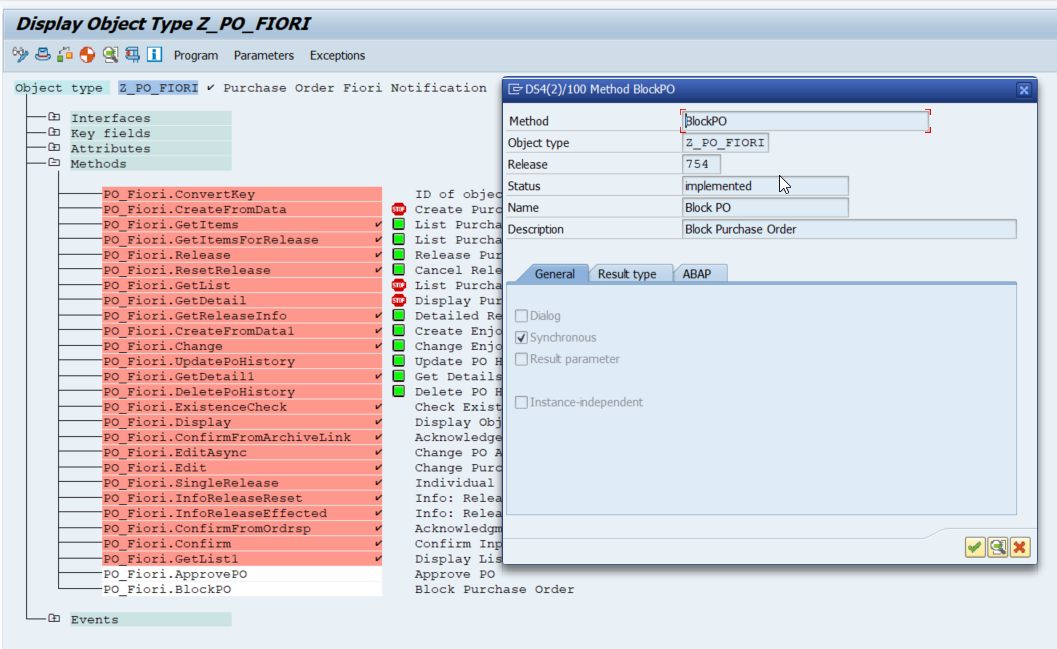


Reject Task:



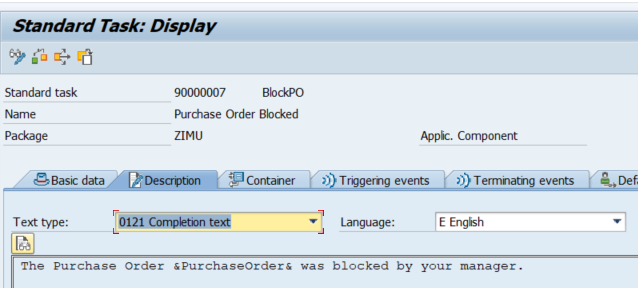
Enter in the task and fill the Basic data:



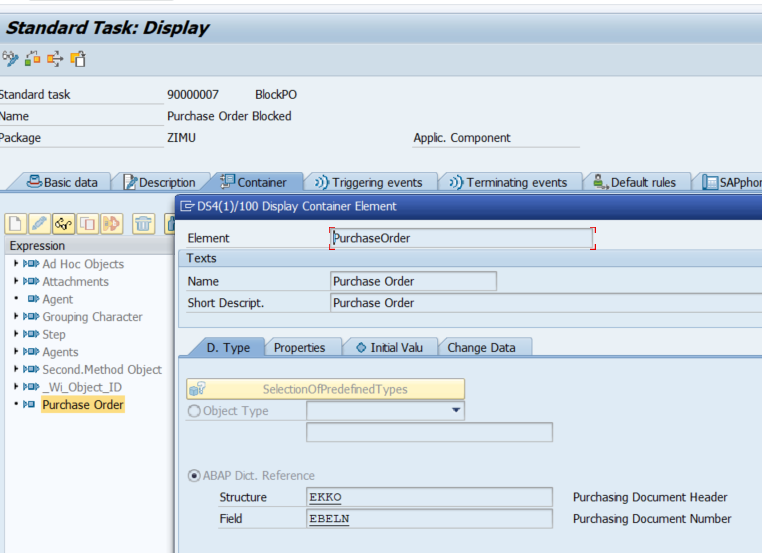
The method BLOCKPO should be created in the Object Type Z\_PO\_FIORI in transaction SWO1:  


In this Method we will block the Purchase Order:

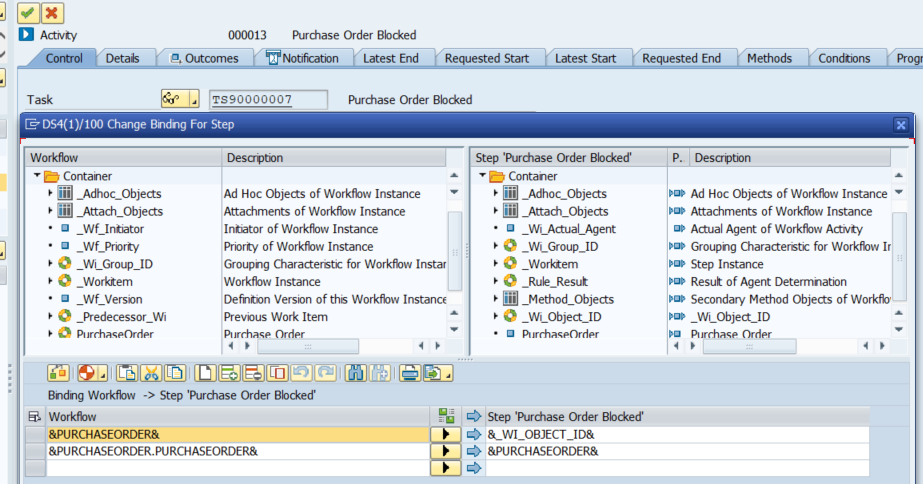
|  |
| --- |
| begin\_method blockpo changing container.  DATA: lt\_poitem  TYPE STANDARD TABLE OF bapimepoitem,       lt\_poitemx TYPE STANDARD TABLE OF bapimepoitemx,       ls\_poitemx TYPE bapimepoitemx,       lt\_return  TYPE STANDARD TABLE OF bapiret2.  CALL FUNCTION 'BAPI\_PO\_GETDETAIL1'   EXPORTING     purchaseorder = object-key-purchaseorder   TABLES     return        = lt\_return     poitem        = lt\_poitem.  LOOP AT lt\_return TRANSPORTING NO FIELDS WHERE type CA 'EAX'.   DATA(lv\_error) = abap\_true. ENDLOOP.  IF lv\_error IS INITIAL.   LOOP AT lt\_poitem ASSIGNING FIELD-SYMBOL(<fs\_poitem>).     <fs\_poitem>-delete\_ind = 'S'. *"Block*      ls\_poitemx-po\_item  = <fs\_poitem>-po\_item.     ls\_poitemx-po\_itemx = abap\_true.     ls\_poitemx-delete\_ind = abap\_true.      APPEND ls\_poitemx TO lt\_poitemx.     CLEAR ls\_poitemx.   ENDLOOP.    REFRESH lt\_return.    CALL FUNCTION 'BAPI\_PO\_CHANGE'     EXPORTING       purchaseorder  = object-key-purchaseorder     TABLES       return         = lt\_return       poitem         = lt\_poitem       poitemx        = lt\_poitemx.  LOOP AT lt\_return TRANSPORTING NO FIELDS WHERE type CA 'EAX'.   lv\_error = abap\_true. ENDLOOP.  IF lv\_error IS INITIAL. CALL FUNCTION 'BAPI\_TRANSACTION\_COMMIT'. ENDIF.  ENDIF.  end\_method. |

Fill the Completion Text as below:  


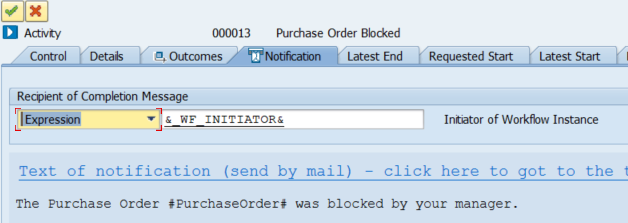
Create an import element in the Container:



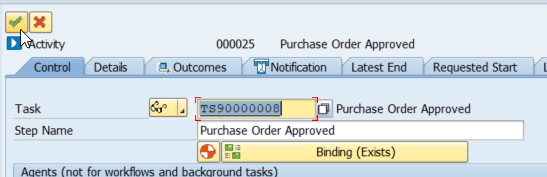
Save it, return to the Task in SWDD and do the binding as below:

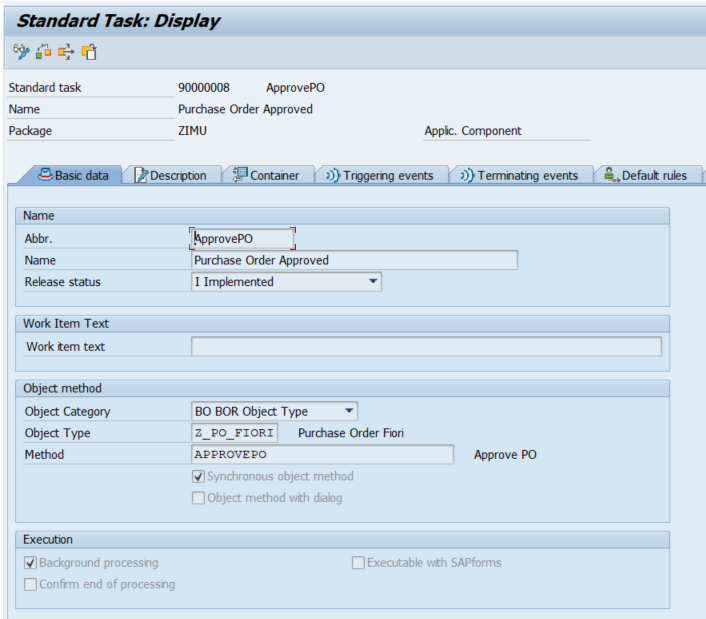


Fill the Recipient of Completion Message:



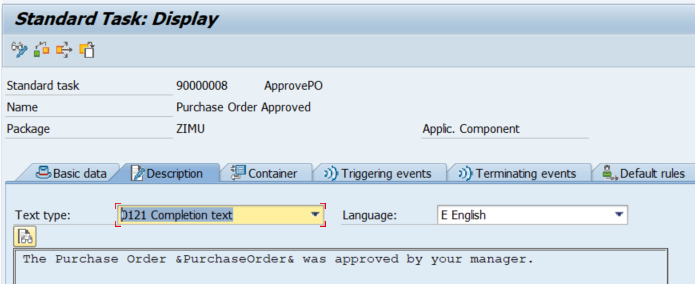
Now, we must do the same steps as above for the Approve task:



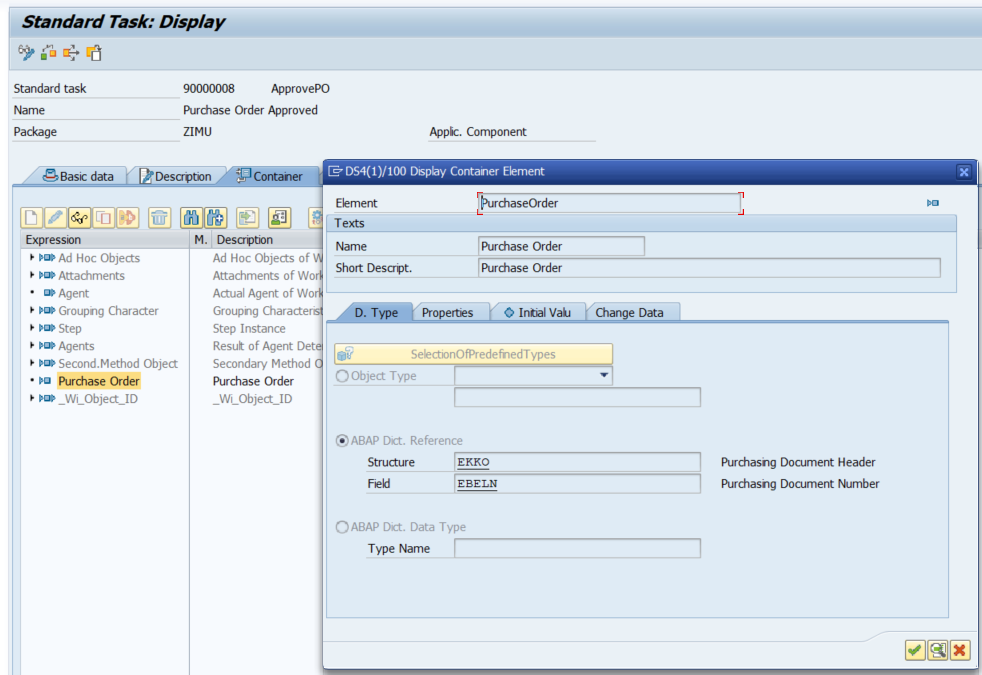
  
  
Code for the „APPROVEPO“ Method:

|  |
| --- |
| begin\_method approvepo changing container.  DATA: lt\_poitem  TYPE STANDARD TABLE OF bapimepoitem,       lt\_poitemx TYPE STANDARD TABLE OF bapimepoitemx,       ls\_poitemx TYPE bapimepoitemx,       lt\_return  TYPE STANDARD TABLE OF bapiret2.  CALL FUNCTION 'BAPI\_PO\_GETDETAIL1'   EXPORTING     purchaseorder = object-key-purchaseorder   TABLES     return        = lt\_return     poitem        = lt\_poitem.  LOOP AT lt\_return TRANSPORTING NO FIELDS WHERE type CA 'EAX'.   DATA(lv\_error) = abap\_true. ENDLOOP.  IF lv\_error IS INITIAL.   LOOP AT lt\_poitem ASSIGNING FIELD-SYMBOL(<fs\_poitem>).     <fs\_poitem>-delete\_ind = space. *" Free*      ls\_poitemx-po\_item  = <fs\_poitem>-po\_item.     ls\_poitemx-po\_itemx = abap\_true.     ls\_poitemx-delete\_ind = abap\_true.      APPEND ls\_poitemx TO lt\_poitemx.     CLEAR ls\_poitemx.   ENDLOOP.    REFRESH lt\_return.    CALL FUNCTION 'BAPI\_PO\_CHANGE'     EXPORTING       purchaseorder  = object-key-purchaseorder     TABLES       return         = lt\_return       poitem         = lt\_poitem       poitemx        = lt\_poitemx.  LOOP AT lt\_return TRANSPORTING NO FIELDS WHERE type CA 'EAX'.   lv\_error = abap\_true. ENDLOOP.  IF lv\_error IS INITIAL. CALL FUNCTION 'BAPI\_TRANSACTION\_COMMIT'. ENDIF.  ENDIF. end\_method. |

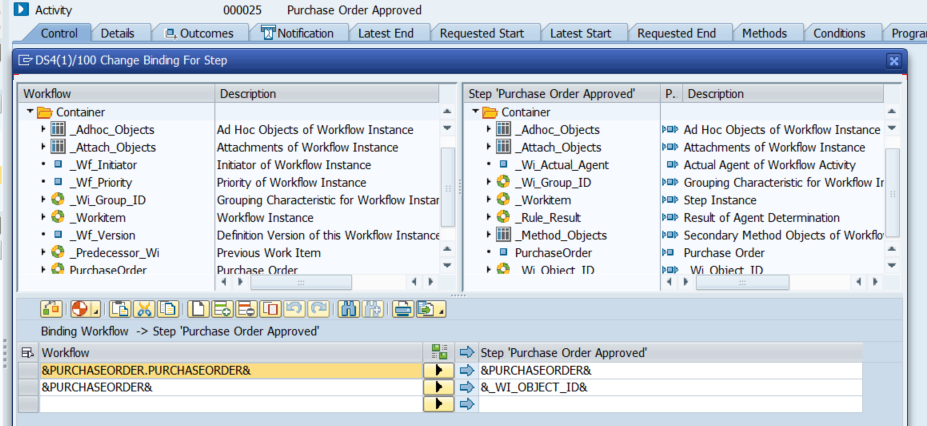
Fill the Completion Text as below:



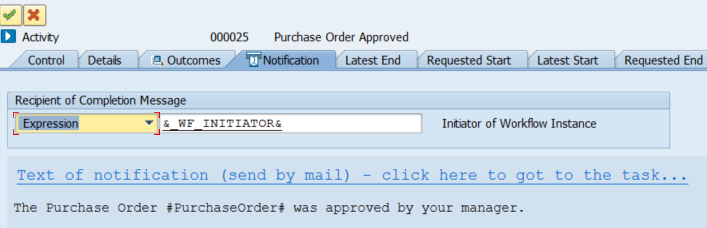
Create the import element in the task Container:



Do the Binding:

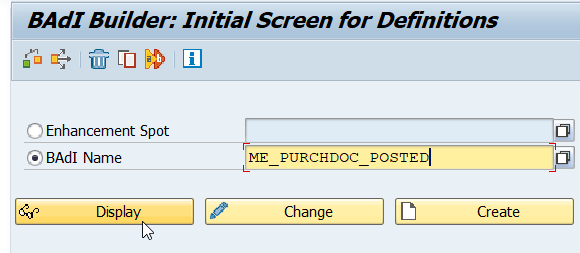


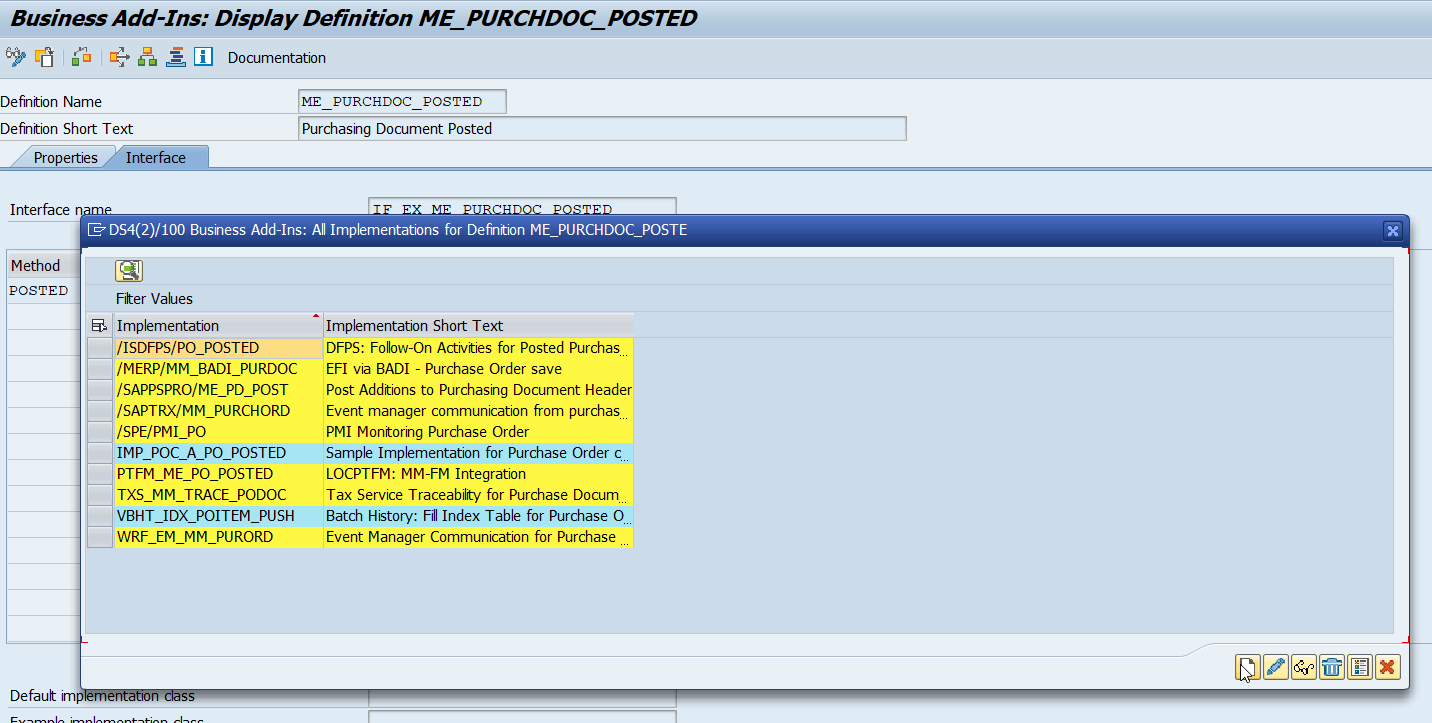
Fill the Recipient of Completion Message:



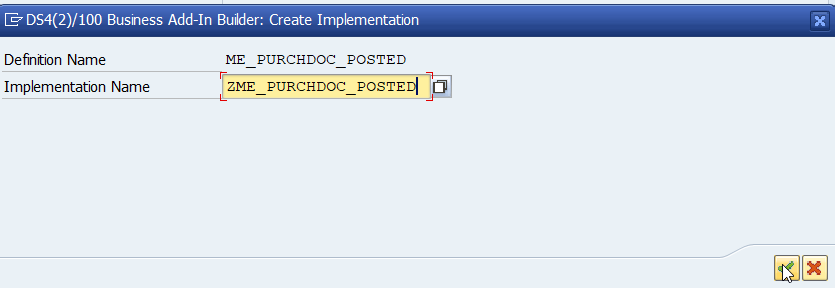
Finally as a final step, we need to call the workflow event through abap code. But where to do it exactly? Well, when we save the Purchase Order in the transaction ME21N or ME22N. That’s why we have chosen the BADI „ME\_PURCHDOC\_POSTED“ , if you want to learn more about this step and how to find the correct BADI for your process, please follow the link: <https://wiki.scn.sap.com/wiki/pages/viewpage.action?pageId=133758980>

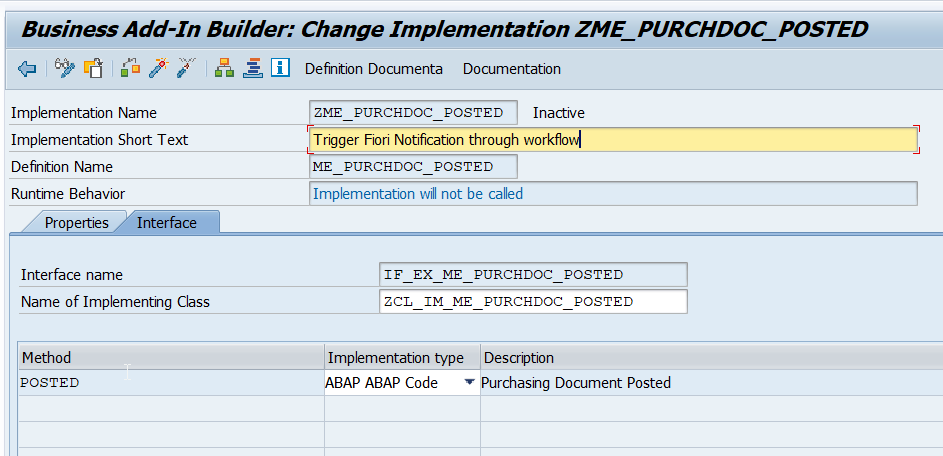
Implement the BADI „ME\_PURCHDOC\_POSTED“ through se18 transaction:





Give it a name:



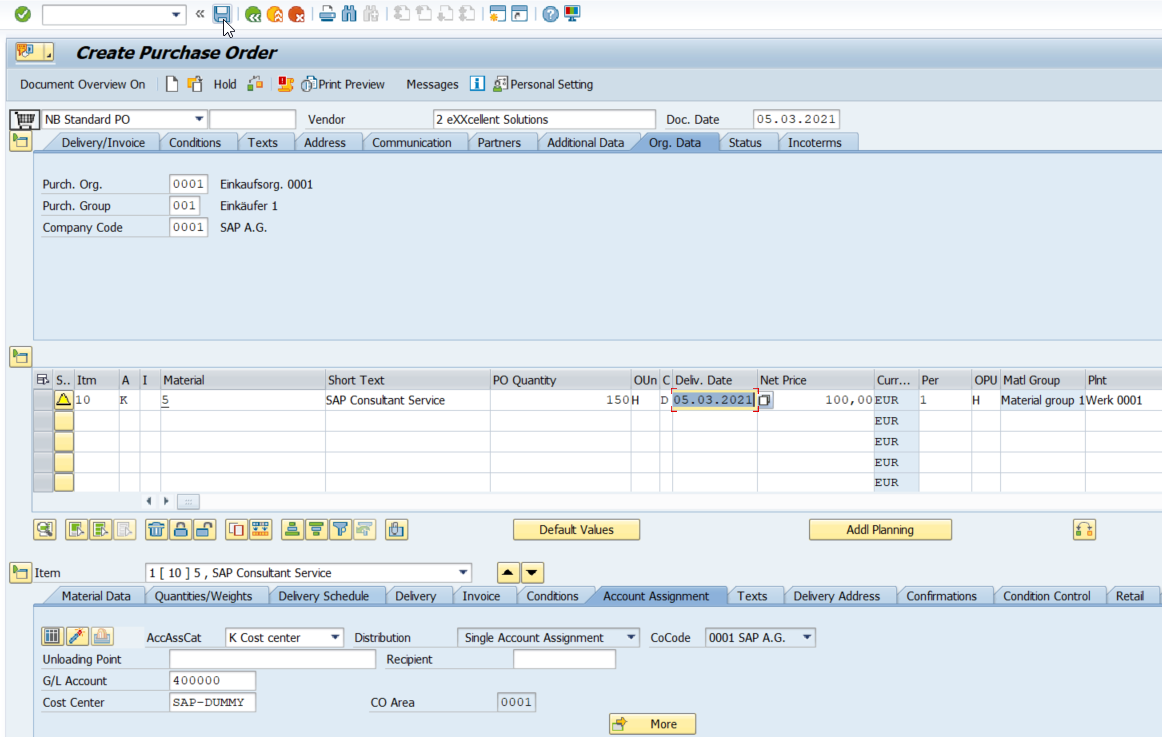


Code the method like below and activate the code. Note that here we will call our workflow event only when the total price of the Purchase Order is equal or more expensive than 10.000.

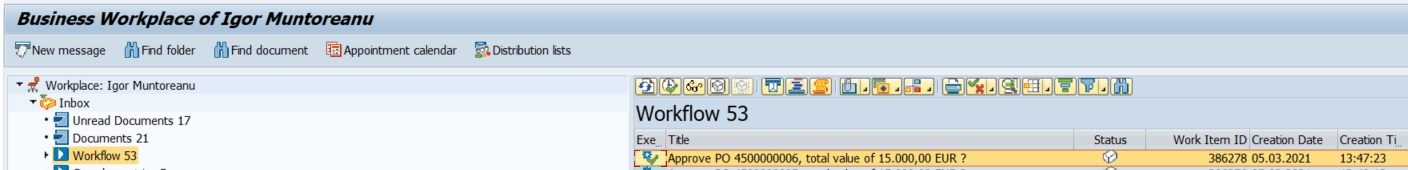
|  |
| --- |
| METHOD if\_ex\_me\_purchdoc\_posted~posted.    DATA: lv\_ebeln       TYPE swr\_struct-object\_key,         lv\_total\_price TYPE ekpo-netwr,         lt\_msg\_lines   TYPE STANDARD TABLE OF swr\_messag.    LOOP AT im\_ekpo ASSIGNING FIELD-SYMBOL(<fs\_ekpo>).     lv\_total\_price = lv\_total\_price + <fs\_ekpo>-netwr.   ENDLOOP.    CHECK lv\_total\_price >= 10000.    lv\_ebeln = im\_ekko-ebeln.    CALL FUNCTION 'SAP\_WAPI\_CREATE\_EVENT'     EXPORTING       object\_type   = 'Z\_PO\_FIORI'       object\_key    = lv\_ebeln       event         = 'SENDNOTIFICATION'     TABLES       message\_lines = lt\_msg\_lines.  ENDMETHOD. |

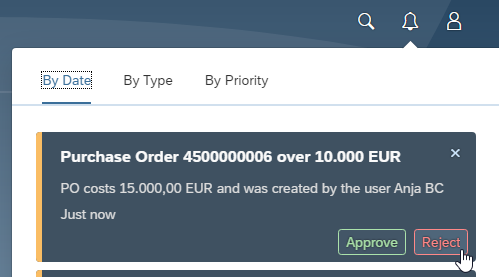
Testing

With the user ABC, we are going to create a Purchase Order through ME21N transaction with a price over 10.000 EUR:



Notification in transaction SBWP from the Head of Department:

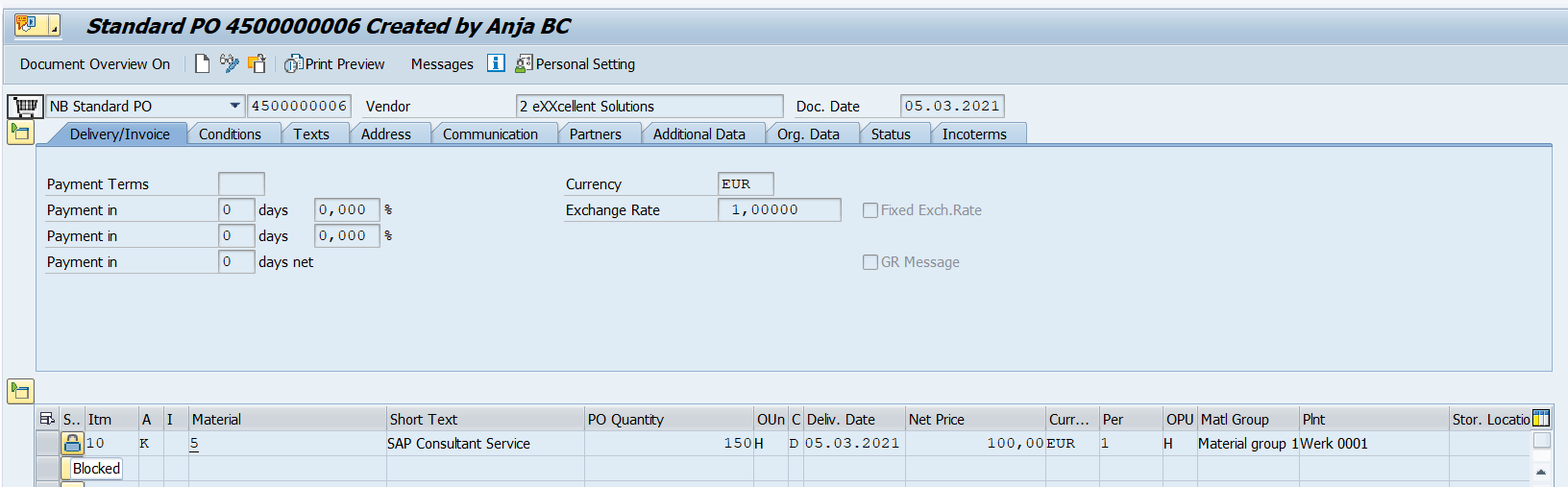


Notification in Fiori Launchpad from the Head of Department:  
  


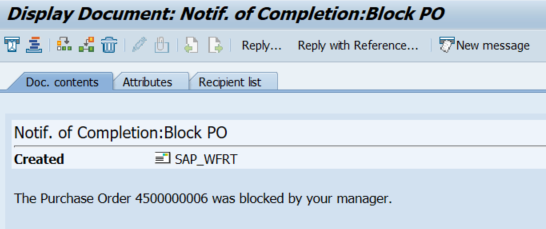
Now lets press Reject and check what happens with the Purchase Order:



As we can see it was blocked:



Then the user who created or changed the Purchase Order, receive also a notification of Completion:



We could also send a Fiori Notification as feedback instead of only an entry in SBWP. To do it, you can use what you learn in the Part 1 of this Blog Post and implement it inside the Task „Approve“ and „Reject“. Since this 2 methods are executed in background, then you need to send the notification manually by calling the method: /iwngw/cl\_notification\_api=>create\_notifications

And by configuring a new Notification Channel Provider in the SPRO transaction.

Conclusion

As we can see the SAP Fiori Notification is very well integrated with the classic SAP Workflow and both technologies complement each other.