Integrating Workflow Programming with ABAP Objects

Applies to:

SAP Workflow Programming with ABAP OO Objects – Interfacing Workflow with ABAP Classes and Methods
For more information, visit the ABAP homepage.

Summary

Programming inside the workflow is often needed for complex workflow development. This document provides a description of how to write ABAP classes and methods to use in a workflow. It is a comprehensive "step-by-step" for a template that can be used to write own classes and methods to use in workflow.

There are several ways to call ABAP OO methods from a workflow step. The two basic techniques are either the implementation as a BOR object or the implementation as an ABAP Objects class and method. When you create a Class that will implement the workflow interface and use it inside of a workflow task, you need to implement an Interface IF_WORKFLOW inside your custom class. Without this implementation you will not be able to select your ABAP OO class in a Workflow step, this is the basic communication interface between your methods and the workflow.

We will cover two topics in this article:

Workflow with ABAP OO in Static method scenario

This article will start with an easy scenario where we have a static method. The method will be the workflow equivalent of "Hello World" and will just say hello to us.

Workflow with ABAP OO in Instance method scenario

This article is about implementing an Instance Method in a Workflow Class. Why use an instance method? An instance method is only valid and unique inside the workflow that implements the method. It cannot be shared and therefore specific to the workflow

Author: Holger Stumm, Mario Bajic

Company: log(2) oHG Darmstadt, Germany

Created on: 1. März 2010

Author Bio

Holger Stumm started as a Logistics Consultant and ABAP programmer way back in 1988 on good old mainframe R/2. Later, with R/3 , he spent six years in the US, working as a Managament Consultant for KPMG and Deloitte in Silicon Valley, Palo Alto, CA, USA . He worked on worldwide projects on all five continents as project lead, programmer, and consultant. Worked for SAP in Walldorf on ecommerce and SRM. Since 2001, Holger Stumm has his own SAP consulting company together with hisy wife in Darmstadt, Germany.

Table of Contents

General

Workflow is a broad topic and programming inside the workflow is often needed for complex workflow development.

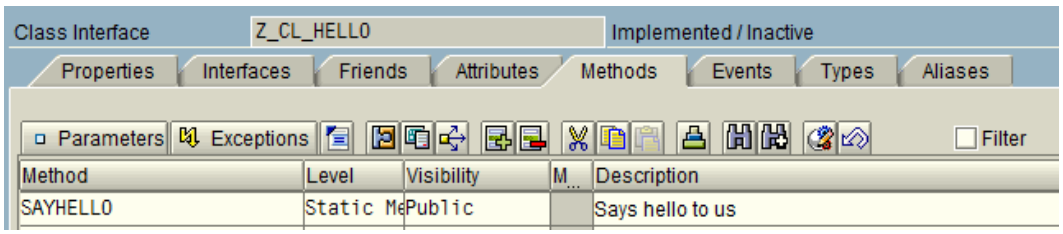
There are several ways to call ABAP OO methods from a Workflow Step. The two basic techniques are either the implementation as a BOR object or the implementation as an ABAP Objects class and method. When you create a Class that will implement the workflow interface and use it inside of a workflow task, you need to implement an Interface IF_WORKFLOW inside your custom classs. Without this implementation you will not be able to select your ABAP OO class in a Workflow step, this is the basic communication interface between your methods and the workflow.

Static method scenario

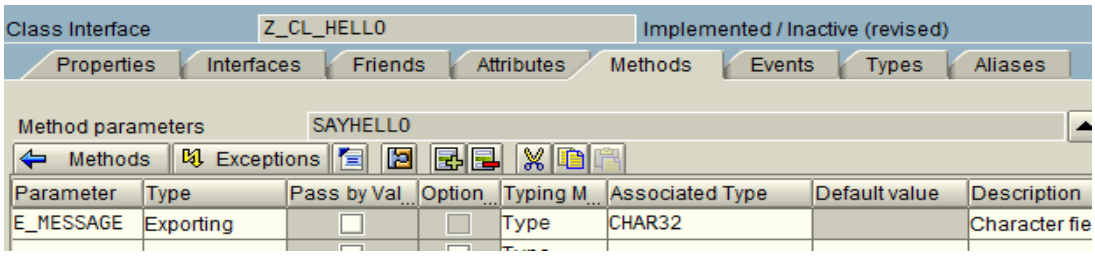
Let´s start with an easy scenario where we have a static method. The method will be the workflow equivalent of “Hello World” and will just say hello to us.

Create ABAP Class

1. Call **SE24** and create an ABAP OO class Z_CL_HELLO which contain our methods SAYHELLO. The method should have an export parameter E_MESSAGE which we use in the Workflow to display the result of the message.



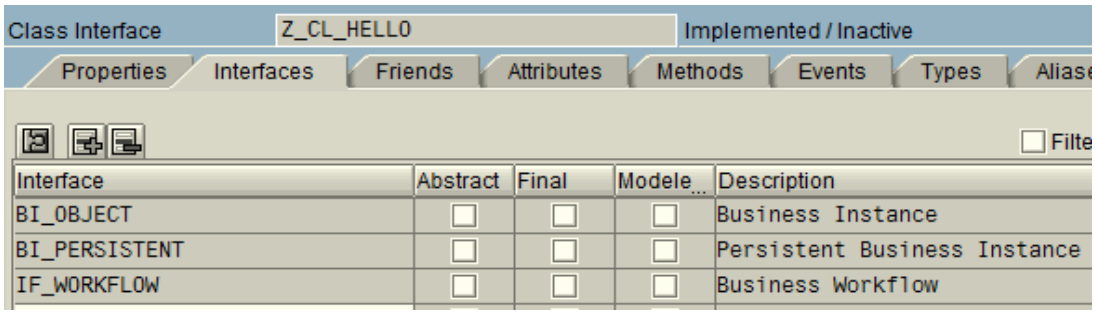
We keep it simple and use only a character export parameter. If you want, you can also use structures or other dictionary objects.



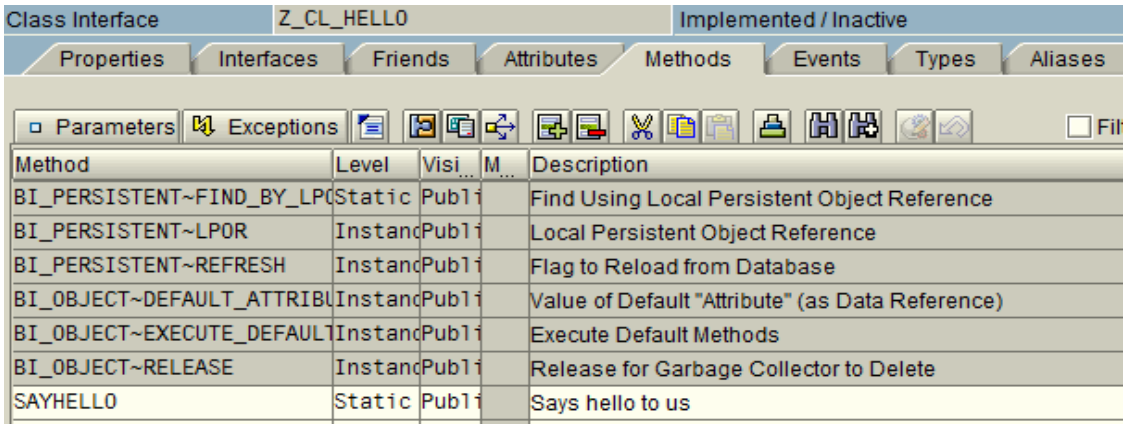
The implementation could look like this:

```
method SAYHELLO.
  DATA: l_message TYPE char32.
  CONCATENATE 'Hello, ' sy-uname INTO l_message.
  e_message = l_message.
endmethod.
```

Add the IF_WORKFLOW interface to our class.



Move to Method tab and we see some methods which have been automatically inherited.



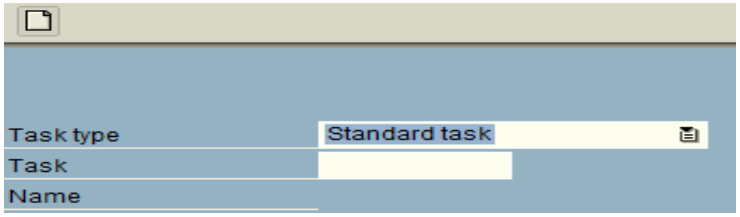
That's all we have to do in our class.

The class is static, this means, they can be accessed at runtime by all workflows who hare and implement this class. For right now, for a "Hello World", this is really next, we will implement an instantiated method.

Create Workflow Task

Calling of the implemented method is done in the Workflow with a task. In order to use the task you must create it and have your class and method activated and ready to go.

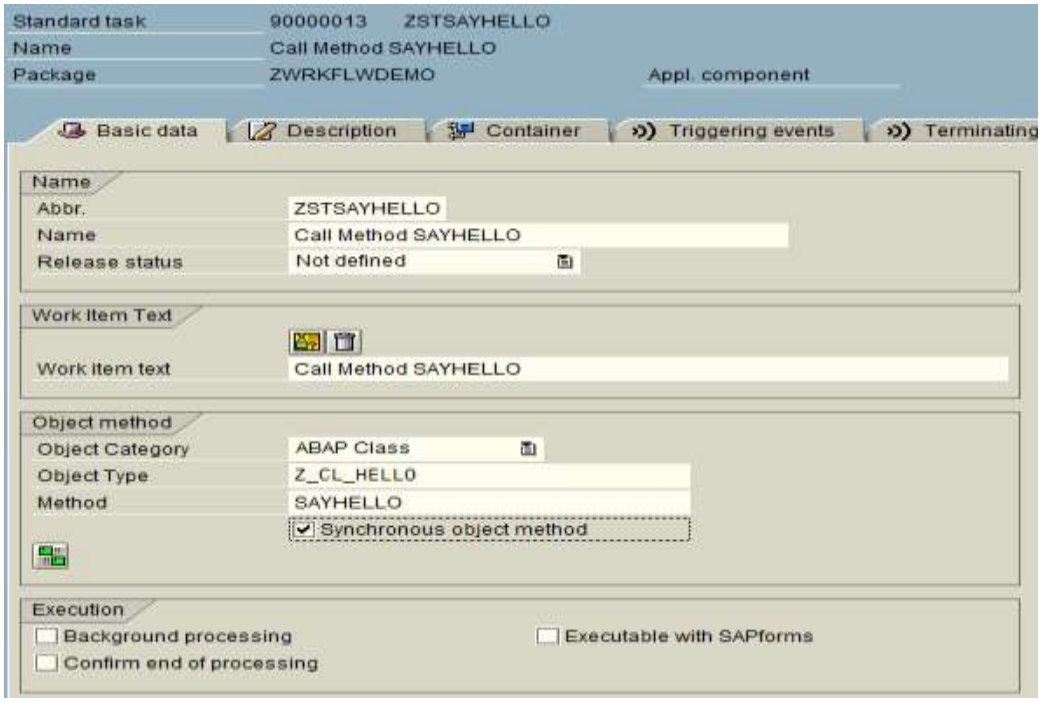
1. Call transaction PFTC_INS and choose Task type Standard Task



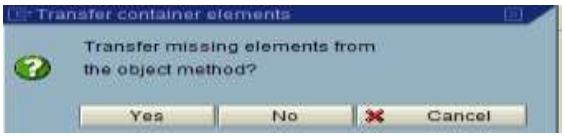
Press the icon on the left side or F5 to create a new task

2. Give some appropriate name to our task.

   In the Object Method category we can assign our ABAP Class method to the Task

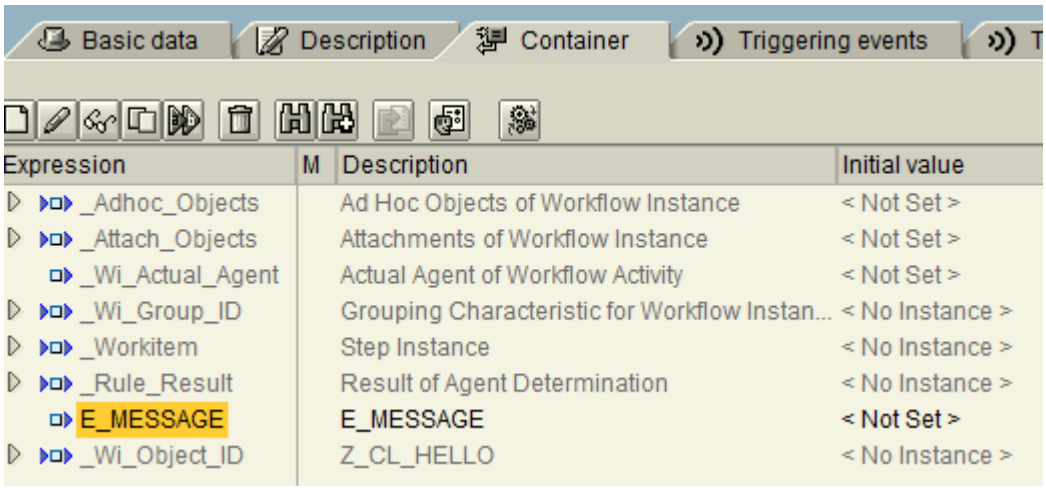


3. Save your task and confirm the popup which is asking to transfer missing element.

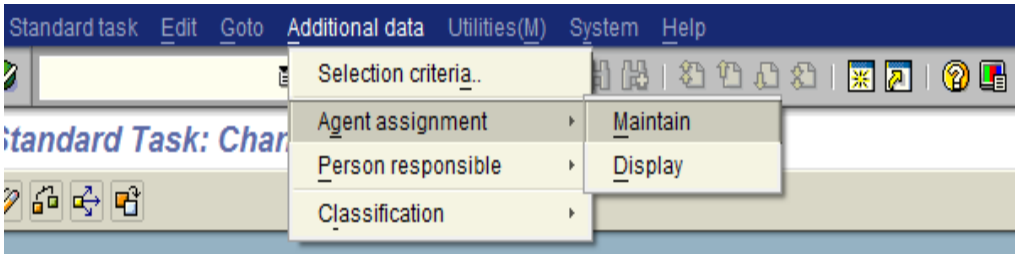


4. After you have saved your task we should check **Synchronous object method**. This will also create an exit to the workflow step.

Let´s look at the Container tab here we can see our export parameter E_MESSAGE is available in the container.



5. Now we need to specify who can actually execute our task.
   Go to Additional Data - Agent Assignment – Maintain



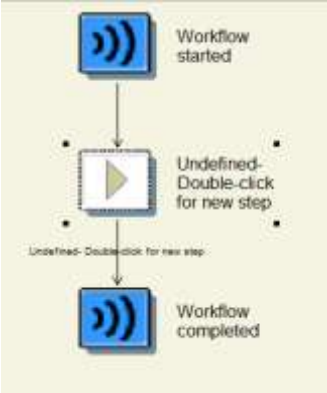
Choose Attributes an choose General Task

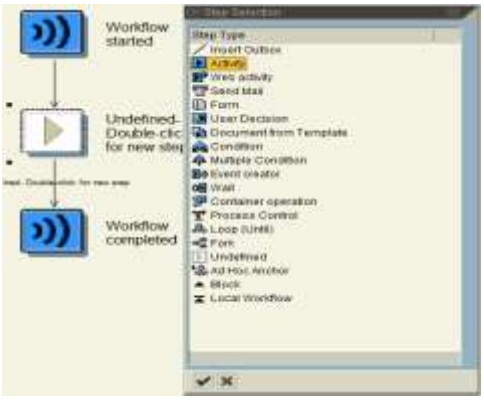
Workflow Design

In this chapter, we will create a workflow embed the ABAP Objects classes in a task

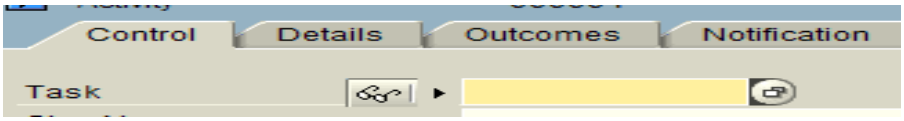
Workflow design

1. Call transaction SWDD and create new Workflow.
2. Double-click on the undefined step in the middle of the window and insert an Activity step.

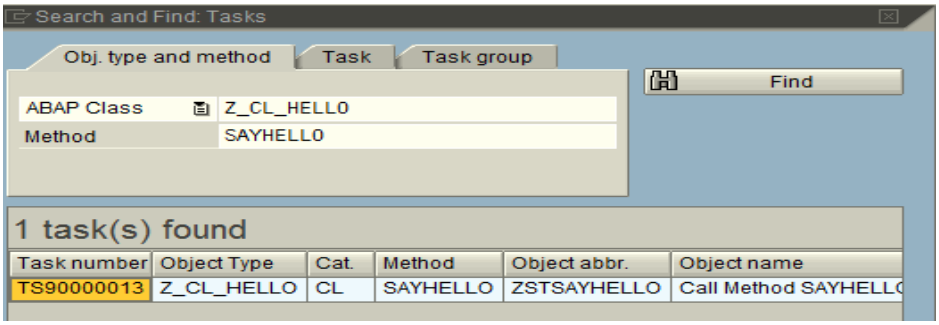




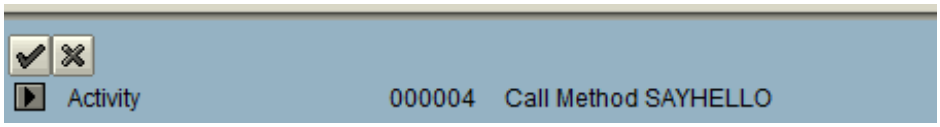
3. Choose the value help of the Task step of press F4

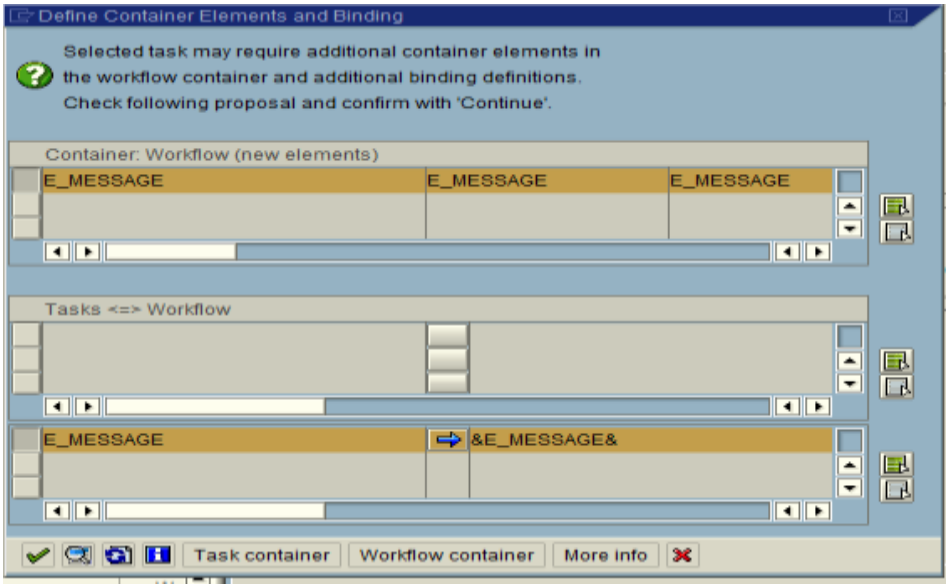


Search for our generated task

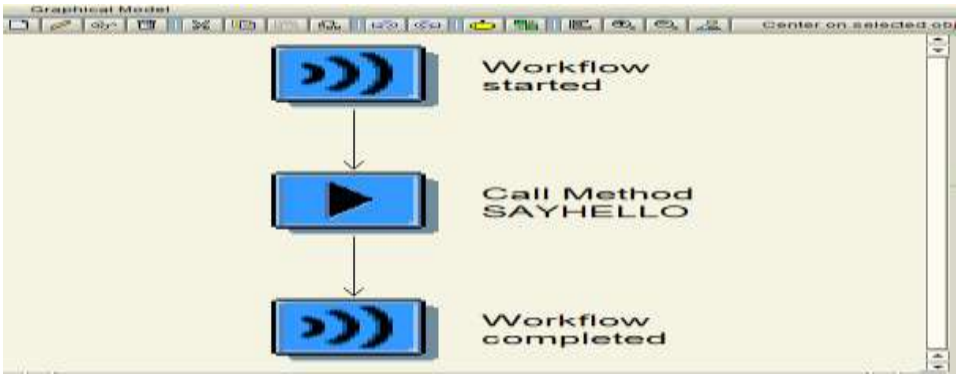


4. Choose the small transfer icon on the left side of the Activity step





Our simple Workflow is now finish and we can test it.



5. Save your workflow

Test Workflow

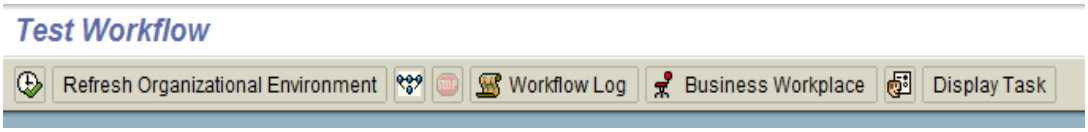
In this chapter, we will actually test the wlow

Start testing

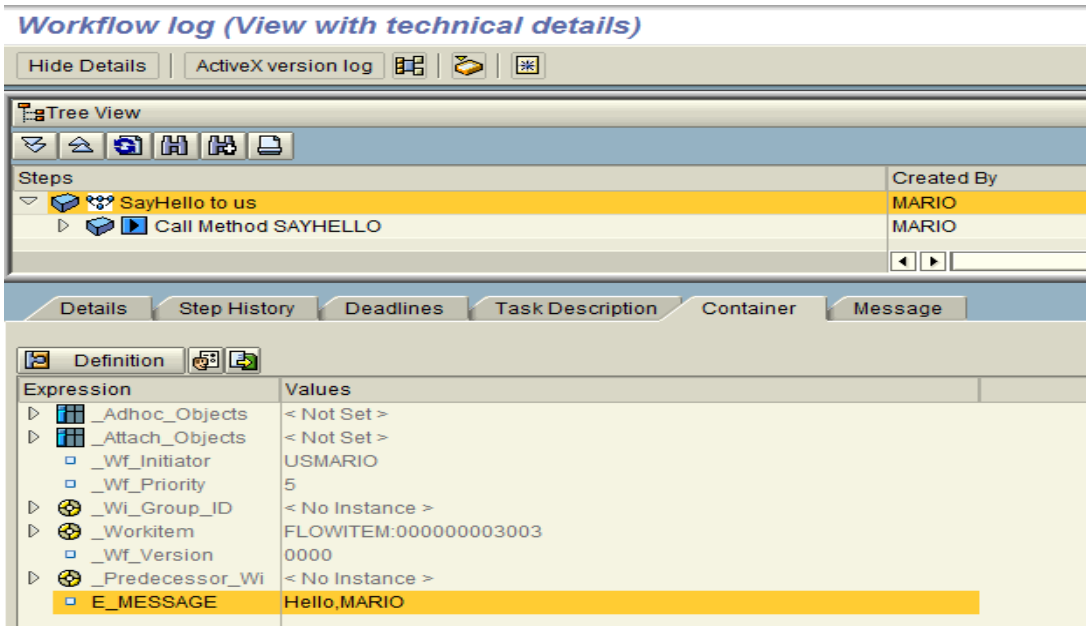
1. Press F8 for testing your workflow.

After you have run the workflow we can use the Workflow log to check the execution.

2. Choose **Workflow Log** on the next window press **Shift+F9**



Now we are in the technical details View, here we can drill down all step of our workflow and check the values of the Container elements.



Instance method scenario 1

This chapter is about implementing an Instance Method in a Workflow Class. Why use an instance method? An instance method is only valid and unique inside the workflow that implements the method. It cannot be shared and therefore specific to the workflow. If you work with specific IDs and environments with unique key requirements (pending transactions etc) you should implement your class as instance method.

As with other methods that will interface through Tasks with the workflow you have to implement the **IF_WORKFLOW** methods. The configuration of the Workflow Task to call an instance Method is the same like in the previous example "Static method scenario".

Implement IF_WORKFLOW

1. Open each method inherited form IF_WORKFLOW and activate the empty source code. This means you have to click in every method (even if it is empty) and activate it.

The most important methods are **FIND_BY_LPOR** and **LPOR**. If Workflow needs to instantiate an ABAP Class, Workflow will call the FIND_BY_LPOR Method. The simplest implementation has to return only a new instance of our ABAP Object.

2. Implement FIND_BY_LPOR

```
method BI_PERSISTENT~FIND_BY_LPOR.
  CREATE OBJECT result TYPE Z_CL_HELLO.
endmethod.
```

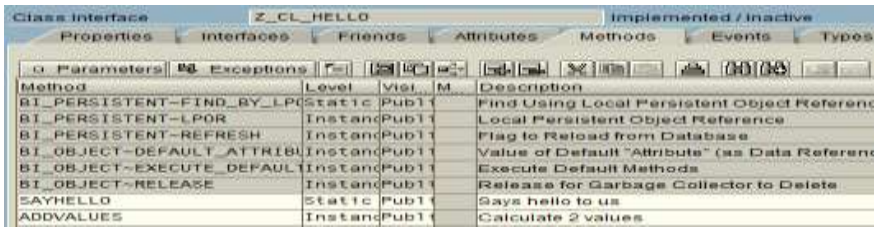
The FIND_BY_LPOR method contains an import parameter LPOR which is passed from Workflow to our ABAP class. This Local Persistence Object Reference (LPOR) has three parameters.

CATID - For ABAP Classes this is always "CL".
TYPEID - Contains the technical name of the ABAP Class.
INSTID - This is the unique identifier of a ABAP Class Instance.

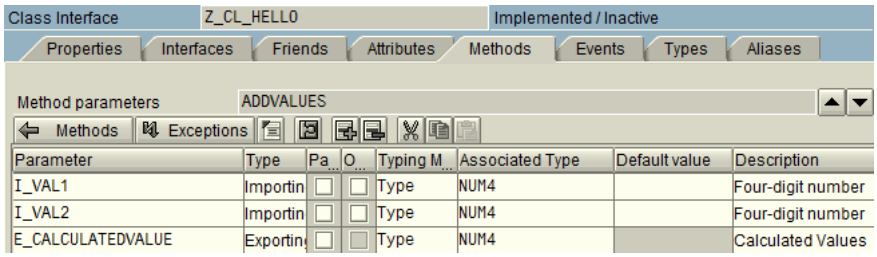
Implement an instance Method

1. Create a new method ADDVALUES

   This method should calculate 2 Values and exporting the result.



   Parameters



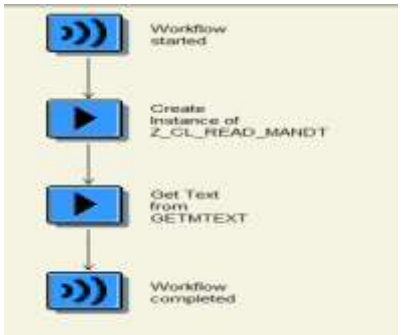

```
method ADDVALUES.
  DATA: l_calcval type num4.
  l_calcval = i_val1 + i_val2.
  e_calculatedvalue = l_calcval.
endmethod.
```

Now we are able to call our ABAP instance method from a Workflow task. The configuration of the Workflow Task to call an instance Method is the same like in “Static method scenario”.

Instance method scenario 2

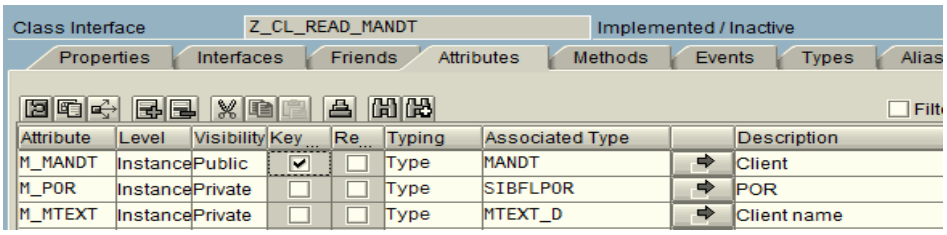
Why use an instance method? An instance method is only valid and unique inside the workflow that implements the method. It cannot be shared and therefore specific to the workflow. If you work with specific IDs and environements with unique key requirements (pending transactions etc) you should implement your class as instance method. In this example we will show the use of an instantiated method in a workflow by keeping up calling the instances.

The difference to the previous example is the use of the method FIND_BY_LPOR, which enables you to search the instantiated methods.
How to instantiate a Class and pass INSTID to FIND_BY_LPOR? We will create a static method which returns an instance of our class then we can define a mapping to **_WI_OBJECT_ID** in workflow.
Let´s create a small scenario where we read the client table and show the text of the client. This is really not the best way to read the client, but our intension is to illustrate how you can implement FIND_BY_LPOR and LPOR.
At the end our workflow looks like below picture

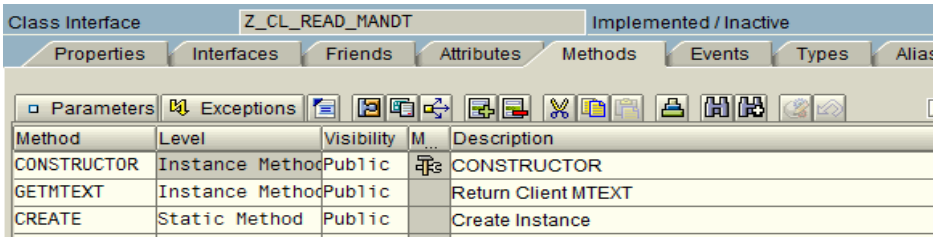


Create instance method

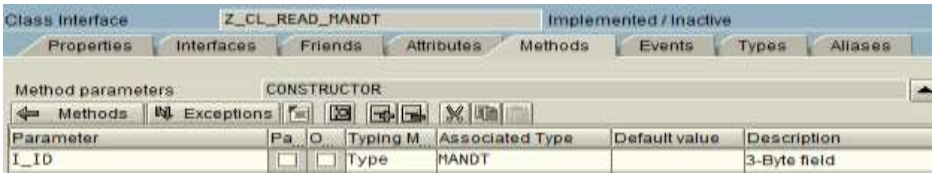
1. Create a new ABAP Class Z_CL_READ_MANDT which has three Attributes
   M_MANDT is the Client key
   M_POR used in LPOR
   M_TEXT which holds the Name of the client



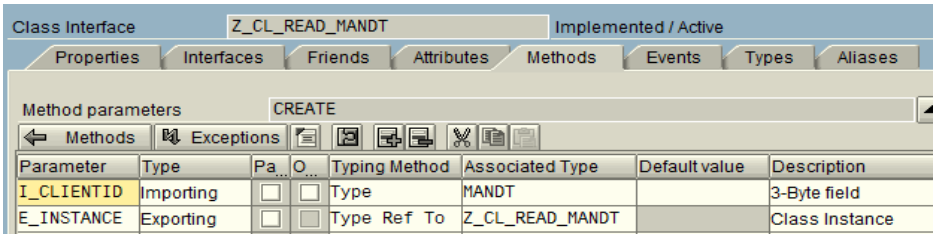
2. Implement Constructor, Create Method and a really simple business Method.



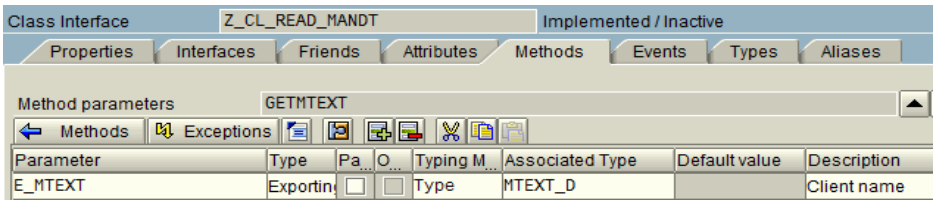
CONSTRUCTOR Parameter



CREATE Method Parameter



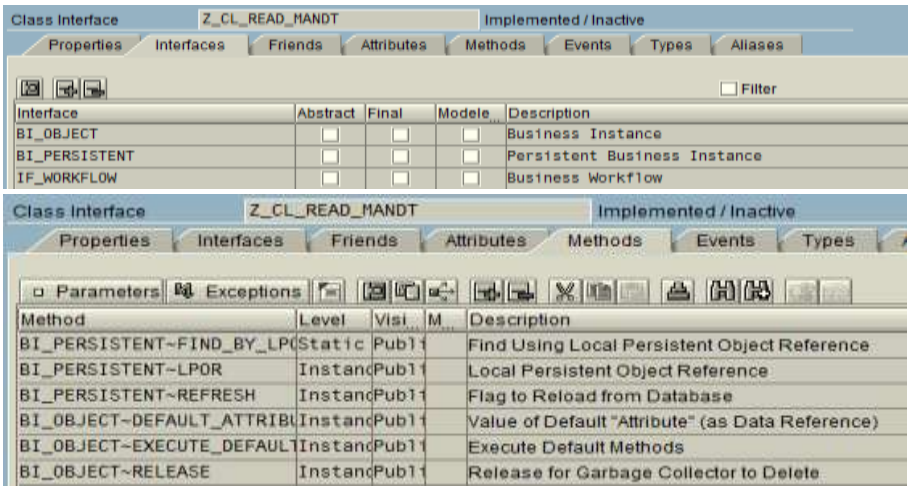
GETMTEXT Parameter



Instance Method Implementation

```
method CONSTRUCTOR.
  DATA: i_tab TYPE SORTED TABLE OF T000 WITH UNIQUE KEY MANDT,
        i_tabstr LIKE LINE OF i_tab,
        i_client type mandt .


  UNPACK i_id to i_client.

  m_mandt = i_client.
  m_por-INSTID = i_client.
  m_por-CATID = 'CL'.
  m_por-TYPEID = 'Z_CL_READ_MANDT'.

 SELECT * FROM T000 INTO TABLE i_tab WHERE MANDT = m_mandt.

 IF sy-subrc = 0.
  LOOP AT i_tab INTO i_tabstr.
    me->m_mtext = i_tabstr-mtext.
  ENDLOOP.
 ENDIF.


endmethod.
method GETMTEXT.
 e_mtext = me->m_mtext.
endmethod.
method CREATE.
    CREATE OBJECT e_instance type z_cl_read_mandt exporting i_id = i_clientid.
endmethod.
```

3. Add the IF_WORKFLOW Interface



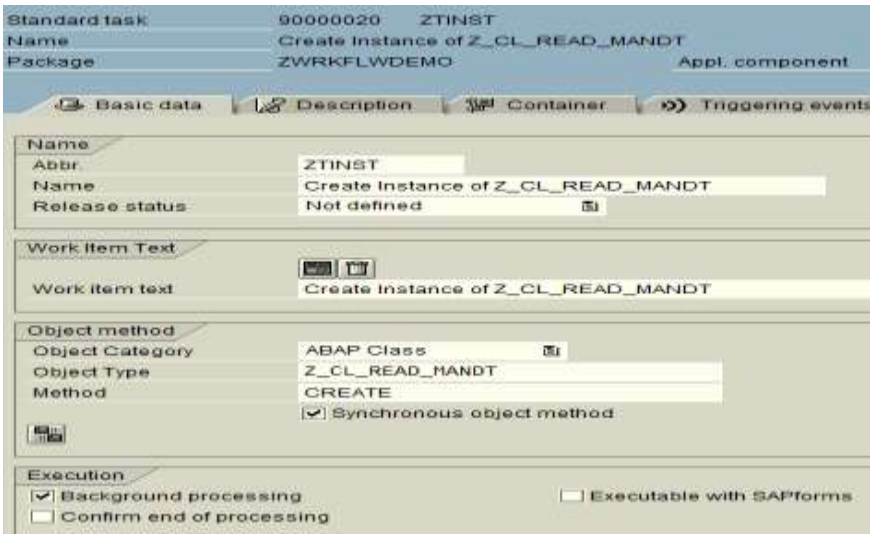
4. Implement BI_PERSISTENT~FIND_BY_LPOR

```
method BI_PERSISTENT~FIND_BY_LPOR.
  CREATE OBJECT result type z_cl_read_mandt exporting i_id = lpor-instid(3).
endmethod.
```

5. Implement LPOR

```
method BI_PERSISTENT~LPOR.
  result = me->m_por.
endmethod.
```

6. CREATE method workflow task (PFTC_INS)
   Create a Workflow Standard task´s like described in “Static method scenario”

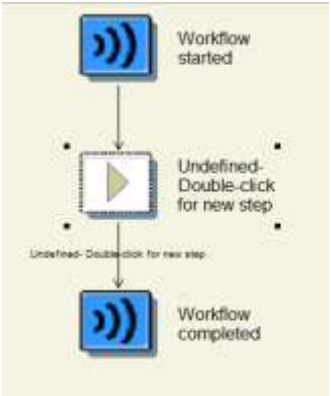


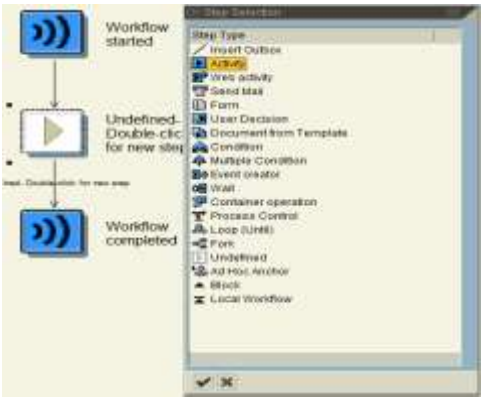
7. GETMTEXT method workflow task
   Create a second Workflow Standard task´s like described in “Static method scenario”


Standard task
90000022
ZTSGETTEXT
Name
Get Text from GETMTEXT
Package
ZWRKFLWDEMO
Appl. component
Basic data
Description
Container
Triggering events
Name
Abbr.
ZTSGETTEXT
Name
Get Text from GETMTEXT
Release status
Not defined
Work Item Text
Work item text
Get Text from GETMTEXT
Object method
Object Category
ABAP Class
Object Type
Z_CL_READ_MANDT
Method
GETMTEXT
Synchronous object method

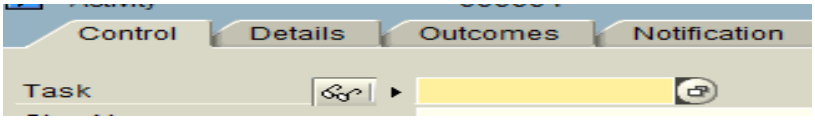
8. Create new Workflow (SWDD)

Click on the undefined step and insert an Activity which point to the CREATE Method Task

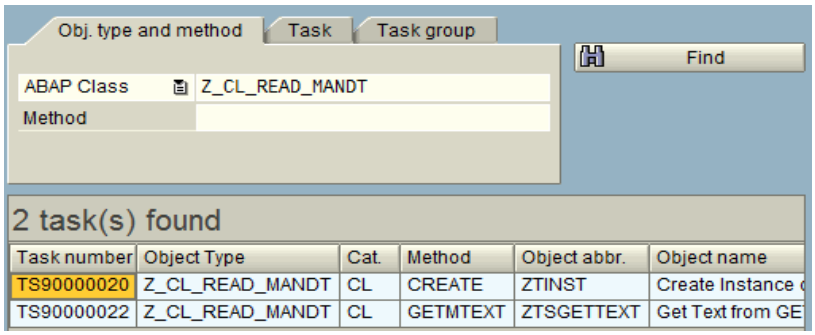




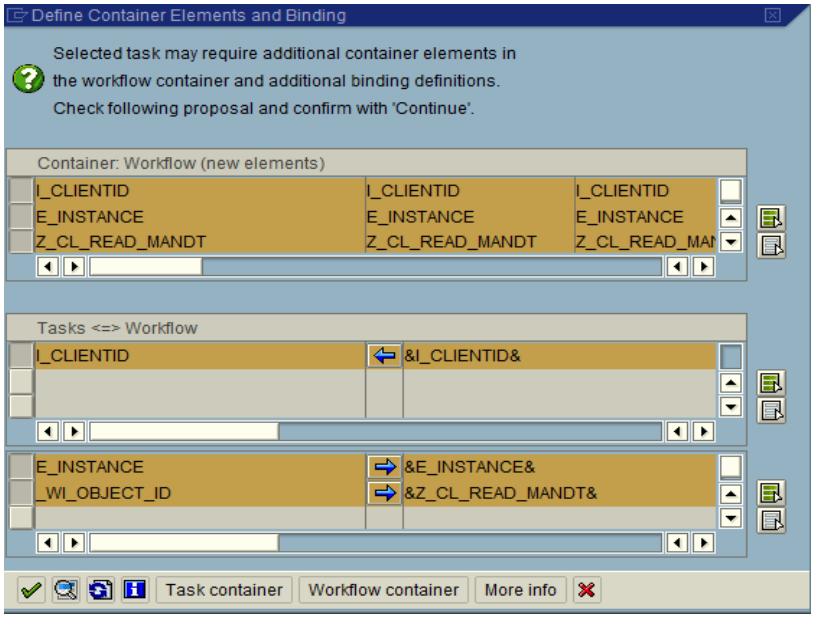
Choose the value help of the Task step of press F4



Search for our new created Standard task

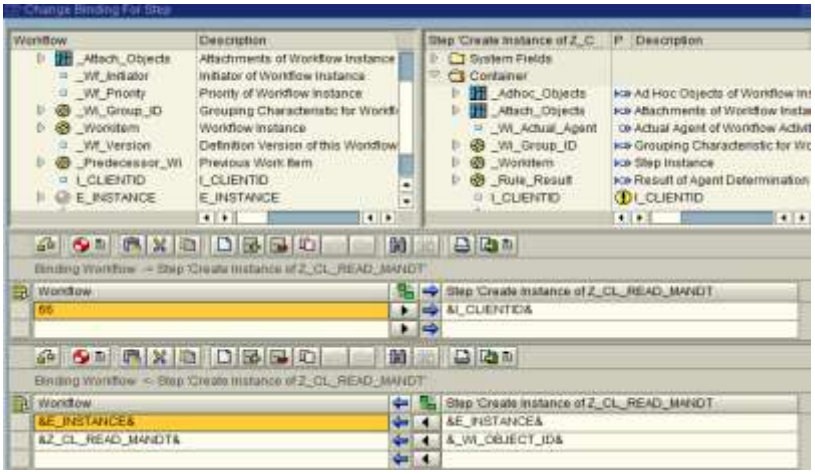


Accept the popup Window which appear when transfer the Task

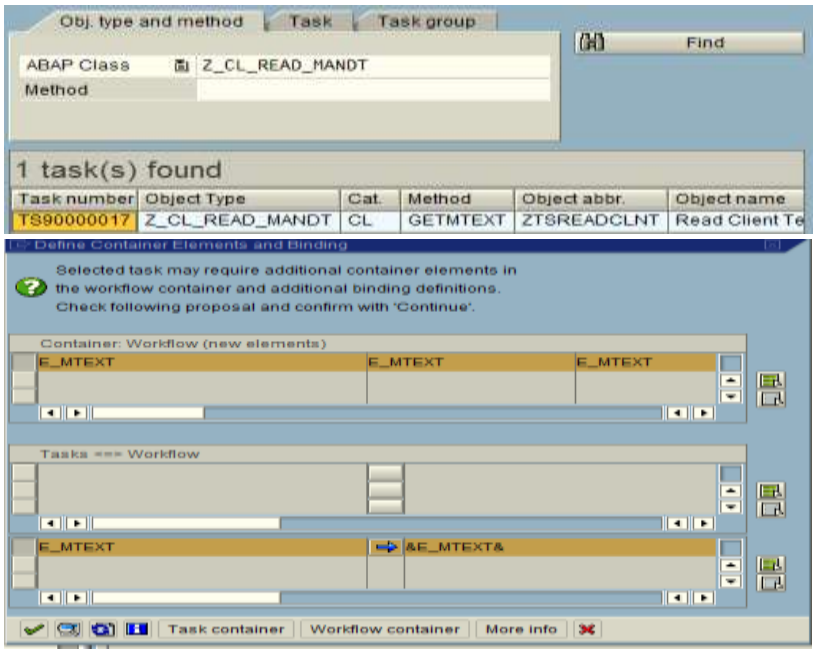


9. Create Binding on CREATE method activity

   In our example we will add a static constant to I_CLIENTID for instantiation

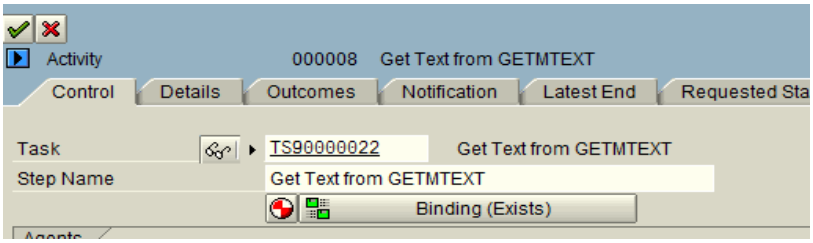


10. Insert a second activity which point to the GETMTEXT Method Task

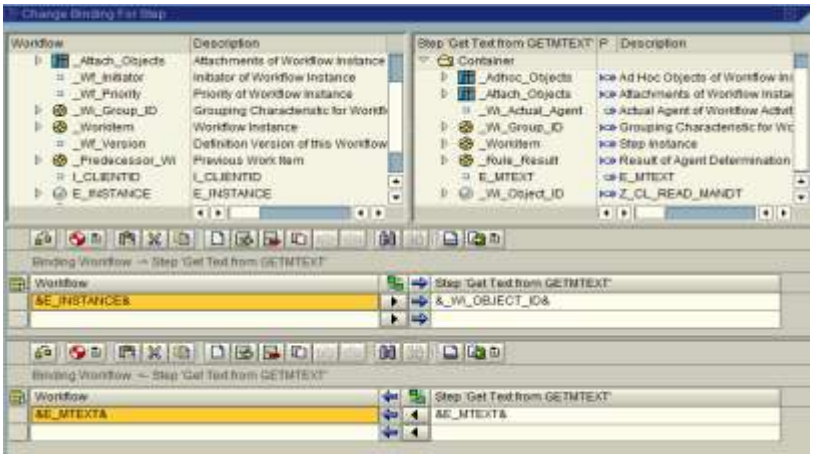


   Accept Popup Window

11. Create Binding on GETMTEXT activity



Here we map our instance to the _WI_OBJECT_ID



Related Content

http://wiki.sdn.sap.com/wiki/pages/viewpage.action?pageId=60654596

http://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/3907

https://weblogs.sdn.sap.com/pub/wlg/3948

For more information, visit the ABAP homepage.

Disclaimer and Liability Notice