Workflow Utility Classes using ABAP OO (Factory Calendar Deadlines Example)

Jocelyn Dart, 11.10.2004, Technology white paper

Table of Contents

1 Historical Perspective (ABAP OO versus BOR)

Having a history of effective Enterprise Resource Planning solutions, the need for Business Process Management (a way of controlling and monitoring business processes across different teams and different functional areas) had been recognised within SAP for a considerable number of years. Although some exploratory attempts were made in R/3 release 2.0, the first concerted effort to provide Business Process Management services came with the introduction of SAP Business Workflow in R/3 release 3.0C.

At the time SAP Business Workflow was introduced, object oriented programming was still more of an ideal than a reality within ABAP; however it was clear that object oriented techniques were the way of the future and critical to underpin workflow if workflow were to provide efficient and effective services. As a consequence, SAP Business Workflow was delivered with an approximation of object oriented programming called the Business Object Repository (BOR).

The aim of the BOR was clearly to provide object oriented –style techniques and services in systems that were not yet capable of object oriented programming. Major strengths of the BOR was in how well it provided object oriented capabilities - such as inheritance, delegation, association, and polymorphism – to such an extent that it wasn't until R/3 release 4.6C that similar depth of object oriented capabilities was available in ABAP OO, and not until SAP WAS 6.20 that ABAP OO was able to be integrated with SAP Business Workflow to the same degree as the BOR.

However it was clear from the beginning that the BOR was not intended as a long term solution. Even the way in which the BOR uses macros to provide ABAP code fragments that could be replaced later, when ABAP OO was available, was a clear indication that the BOR was intended to have a limited life. However the need for workflow services outstripped the introduction of ABAP OO, and by the time ABAP OO was fully available and capable of replacing the BOR, a large body of business content had already been provided by the BOR. It was no longer a simple matter of replacing code fragments in macros - a major effort would be needed to convert existing workflow content from the BOR to ABAP OO.

Currently the effort required to convert the existing body of BOR content to ABAP OO, and the disruption to customers that would be caused by such a major change in direction, exceeds the ROI of such an activity. However the option of using ABAP OO in workflow has been added. Already new features in Business Process Management (BPM), such as ccBPM (cross-component BPM) in SAP XI (Exchange Infrastructure) are taking advantage of this new option.

Hopefully over time a gradual move from the BOR to ABAP OO can be encouraged in SAP Business Workflow. This whitepaper is intended to assist with this move.

This whitepaper deals in particular with understanding how to create workflow utility classes/methods using ABAP OO.

Throughout this whitepaper the example used is the calculation of deadlines based on factory calendars (rather than absolute days). This example has been used as it is a common and relatively well known utility function needed for workflows. However the principles described within this whitepaper are the same for any utility function.

2 The Factory Calendar Deadline Example

By default, all deadlines in workflow use an absolute calendar. That is: if the deadline is based on start time = Friday at 8 a.m., and the deadline period is 2 days, then the deadline will be raised on the following Sunday at 8 a.m.

As many businesses do not work on weekends a more desirable calculation does not consider the weekend dates as possible working days. That is: if the deadline is based on start time = Friday at 8 a.m., and the deadline period is 2 days, then the deadline will be raised on the following Tuesday (ignoring Saturday and Sunday) at 8 a.m.

This example can be further extended to Public Holidays, for example when Monday is a public holiday. That is: if the deadline is based on start time = Friday at 8 a.m., and the deadline period is 2 days, then the deadline will be raised on the following Wednesday (ignoring Saturday, Sunday and Monday) at 8 a.m.

In SAP systems, working versus non-working days are identified by maintaining factory calendars (and holiday calendars) in transaction SCAL.

Once maintained, the calculation of the end date/time in the above scenarios becomes a simple matter of calling function module END_TIME_DETERMINE, passing the factory calendar id, the start date/time, and the deadline period.

To use function module END_TIME_DETERMINE in a workflow it must be encapsulated with a method of ABAP class or a BOR (Business Object Repository) object.

A workflow “standard task” must then be created to control the class/method call and to pass the import parameters from the workflow container to the method container, and the export parameters from the method container to the workflow container.

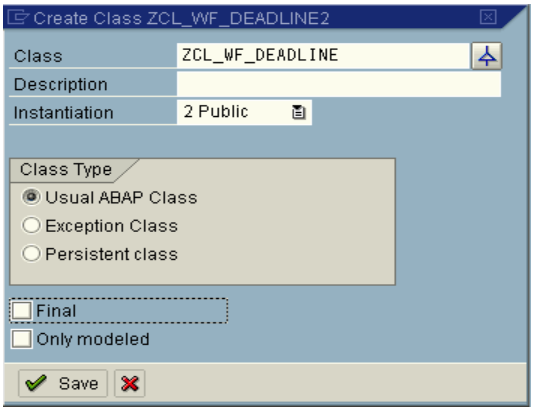
Once the deadline dates/times are passed to the workflow container, calculated deadline dates/times can then be included in the deadline tab of activity workflow steps using deadline “Expressions”.

As using factory calendars and factory calendar deadlines within a workflow is well documented, this whitepaper focuses only with the ABAP OO-related steps in this process that would replace equivalent BOR steps. That is:

1. Creating a workflow-relevant ABAP OO class
2. Creating an ABAP OO instance-independent method
3. Including the ABAP OO method in a workflow standard task

3 Defining ABAP OO classes for Workflow

3.1 Class Definition – Workflow Basics

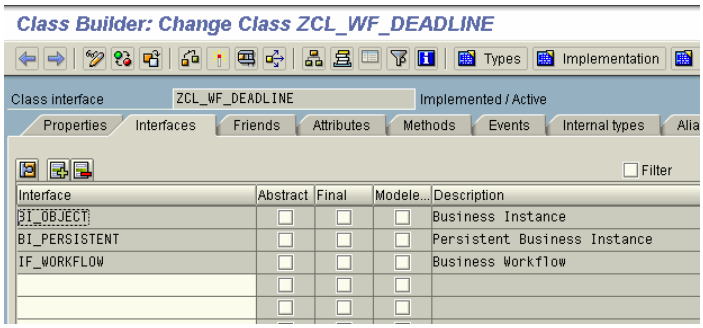


ABAP OO classes are defined in transaction SE24. As a general rule, it is recommended to implement separate classes for dealing with workflow, rather than include them in classes used for other purposes. This is mainly to avoid confusion and conflicts with non-workflow related coding. To use the class within workflow it must be a “Usual ABAP Class” and not a modelled class. Setting of the “Final” option (to control whether subclasses) are allowed is optional.

When defining an ABAP OO class to be used with workflow there is only one major requirement:

The ABAP OO class MUST implement the interface IF_WORKFLOW.

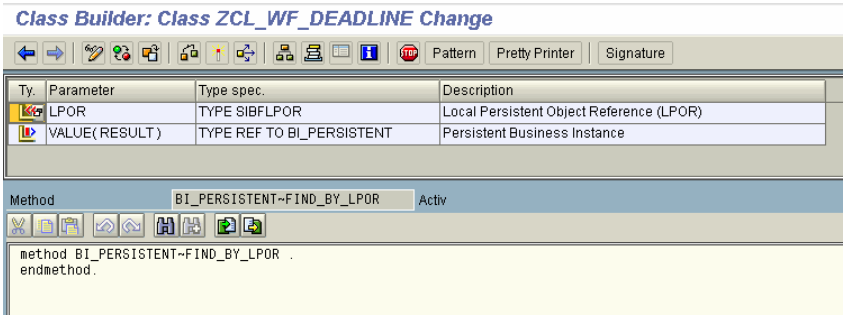
This is done by adding the interface IF_WORKFLOW on the interfaces tab of transaction SE24.



Adding interface IF_WORKFLOW, automatically inherits interfaces BI_OBJECT and BI_PERSISTENT, and with them the following methods:

- BI_PERSISTENT~FIND_BY_LPOR
- BI_PERSISTENT~LPOR
- BI_PERSISTENT~REFRESH
- BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE
- BI_OBJECT~EXECUTE_DEFAULT_METHOD
- BI_OBJECT~RELEASE

These methods are only relevant for instance-dependent classes. With a utility class they are not needed. However workflow expects these methods to be available in every class that implements workflow, so to avoid runtime errors, each of these methods should be activated. For a workflow utility class, it's sufficient to activate the methods without adding any other coding to them.



3.2 Class Definition – Workflow Methods

For workflow utility classes, methods can be created like any other method in an ABAP OO class. Of course, the signature of the method must contain the import parameters to be passed from workflow and the export parameters to be passed to workflow. The method coding itself is encapsulated in the class.

There are no special requirements on the method coding, unless you are referencing BOR objects (this will be dealt with in a separate whitepaper.

There are no "dialog", "synchronous", or "result" flags, or result parameters as are found on BOR objects.

The method must of course be "Static" (so that it can be called without first instantiating the class) and "Public" (so that it can be used outside of the class).

Class Builder: Change Class ZCL_WF_DEADLINE

Class interface ZCL_WF_DEADLINE Implemented / Inactive

Properties | Interfaces | Friends | Attributes | Methods | Events | Internal types | Aliases

Methods	Level	Visi	Mo	M	Description
BI_PERSISTENT~FIND_BY_LP	Static	Publ			Find Using Local Persistent Object Reference
BI_PERSISTENT~LPOR	Instan	Publ			Local Persistent Object Reference
BI_PERSISTENT~REFRESH	Instan	Publ			Flag to Reload from Database
BI_OBJECT~DEFAULT_ATTRIB	Instan	Publ			Value of Default "Attribute" (as Data Reference)
BI_OBJECT~EXECUTE_DEFAUL	Instan	Publ			Execute Default Methods
BI_OBJECT~RELEASE	Instan	Publ			Release for Garbage Collector to Delete
CALCULATE_DEADLINE	Static	Publ			Calculated factory calendar deadlines

In the factory deadline calculation example, a CALCULATE_DEADLINE method is created to import the deadline start time, deadline period, and the factory calendar id. The method will call the function module END_TIME_DETERMINE, and then export the calculated date.

Class Builder: Change Class ZCL_WF_DEADLINE

Class interface ZCL_WF_DEADLINE Implemented / Active

Properties | Interfaces | Friends | Attributes | Methods | Events | Internal types | Aliases

Method parameters CALCULATE_DEADLINE

Parameter	Type	P...	O...	Typing ...	Associated Type	Default value	Description
IV_FACTORY_CALENDAR	Importi...	✓	✓	Type	WFCID	SPACE	Factory Calendar Id
IV_TIME_UNIT	Importi...	✓	✓	Type	MSEHI	'DAY'	Time Unit
IV_START_DATE	Importi...	✓	✓	Type	DATUM	SY-DATUM	Start Date
IV_START_TIME	Importi...	✓	✓	Type	UZEIT	SY-UZEIT	Start Time
IV_DURATION	Importi...	✓	✓	Type	SYINDEX	0	Duration
EV_END_DATE	Exporti...	✓		Type	DATUM		End Date
EV_END_TIME	Exporti...	✓		Type	UZEIT		End Time
				Type			

The coding is a simple matter of calling the function module. No exceptions are raised in this example. (How to deal with exceptions will be covered in a future whitepaper).

```
METHOD calculate_deadline .

  IF iv_factory_calendar IS NOT INITIAL.
    CALL FUNCTION 'END_TIME_DETERMINE'
      EXPORTING
        duration                    = iv_duration
        unit                        = iv_time_unit
        factory_calendar            = iv_factory_calendar
      IMPORTING
        end_date                    = ev_end_date
        end_time                    = ev_end_time
      CHANGING
        start_date                  = iv_start_date
        start_time                  = iv_start_time
      EXCEPTIONS
        factory_calendar_not_found = 1
        date_out_of_calendar_range = 2
        date_not_valid              = 3
        unit_conversion_error       = 4
        si_unit_missing             = 5
        parameters_no_valid         = 6
```

```
        OTHERS                      = 7.
  ELSE.
    CALL FUNCTION 'END_TIME_DETERMINE'
    EXPORTING
      duration                          = iv_duration
      unit                              = iv_time_unit
*      FACTORY_CALENDAR                  = iv_factory_calendar
    IMPORTING
      end_date                          = ev_end_date
      end_time                          = ev_end_time
    CHANGING
      start_date                        = iv_start_date
      start_time                        = iv_start_time
    EXCEPTIONS
      factory_calendar_not_found        = 1
      date_out_of_calendar_range        = 2
      date_not_valid                    = 3
      unit_conversion_error             = 4
      si_unit_missing                   = 5
      parameters_no_valid               = 6
      OTHERS                            = 7
            .
  ENDIF.
  IF sy-subrc <> 0.
    ev_end_date = iv_start_date.
    ev_end_time = iv_start_time.
  ENDIF.


ENDMETHOD.
```

Of course both the method and the class must be activated before they can be used.

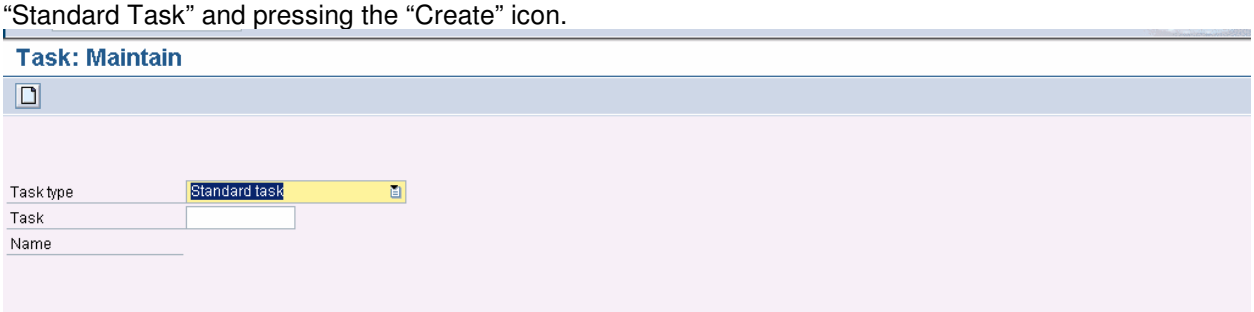
As always, it's a good idea to test the class/method in transaction SE24 before including it in the workflow standard task.

4 Using ABAP OO methods in workflow tasks

ABAP OO methods are included in workflow standard tasks in a very similar way to BOR methods. The main differences are:

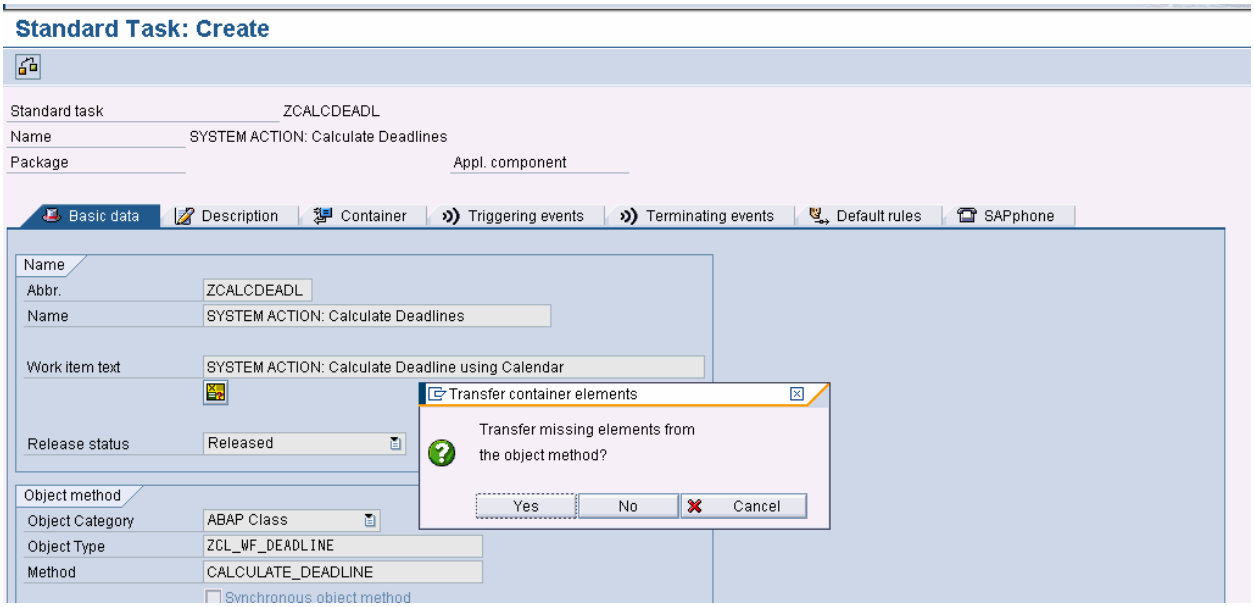
- The object category is "ABAP Class"
- The "dialog" and "synchronous" flags must be specified in the task (as they are not available at the class/method).

New workflow standard tasks are created using transaction PFTC_INS by selecting the task type "Standard Task" and pressing the "Create" icon.

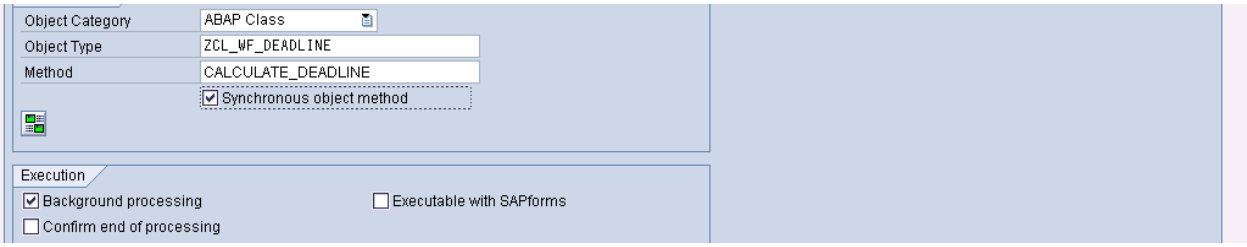


When creating the class, specify the object category "ABAP Class" then select the ABAP Class and method. Note: Only ABAP Classes that have implemented the interface IF_WORKFLOW can be

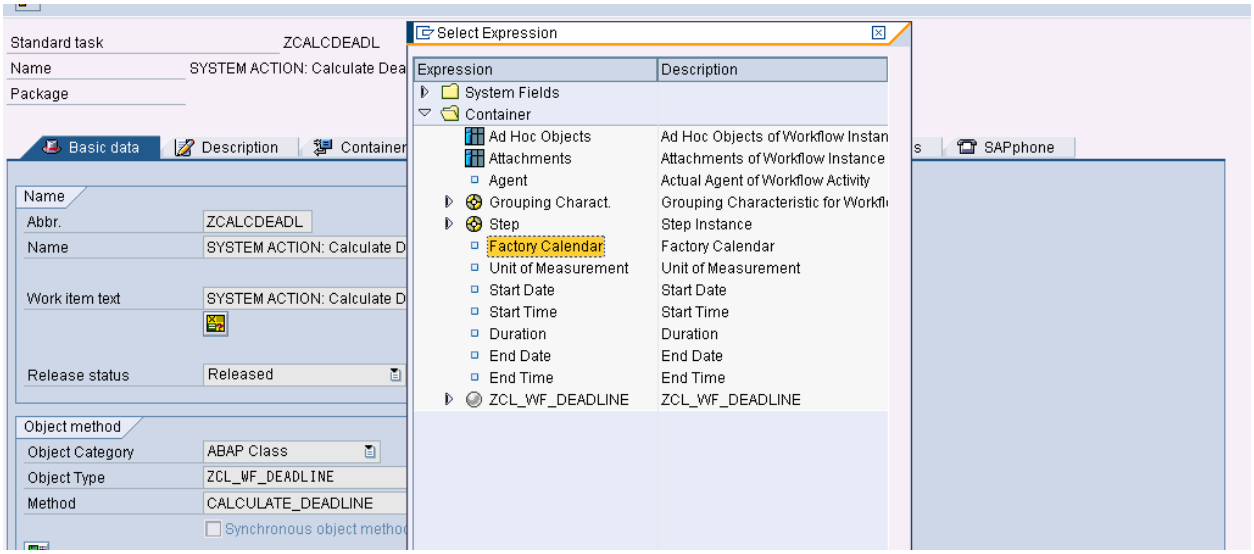
selected. Additional parameters will be copied as usual when “Yes” is answered to the question “Transfer missing elements from the object method?”.



Specify that the method is to be run in Background by turning on the “Background Processing” flag. Note: You must save the task before you can turn on the “Synchronous object method” flag.



Container elements of the task, including attributes of the ABAP class, can be included in the work item text and description by using the “Insert Expression” option as usual.



As always, it's a good idea to test the new task using transaction SWUS before including it in a workflow.

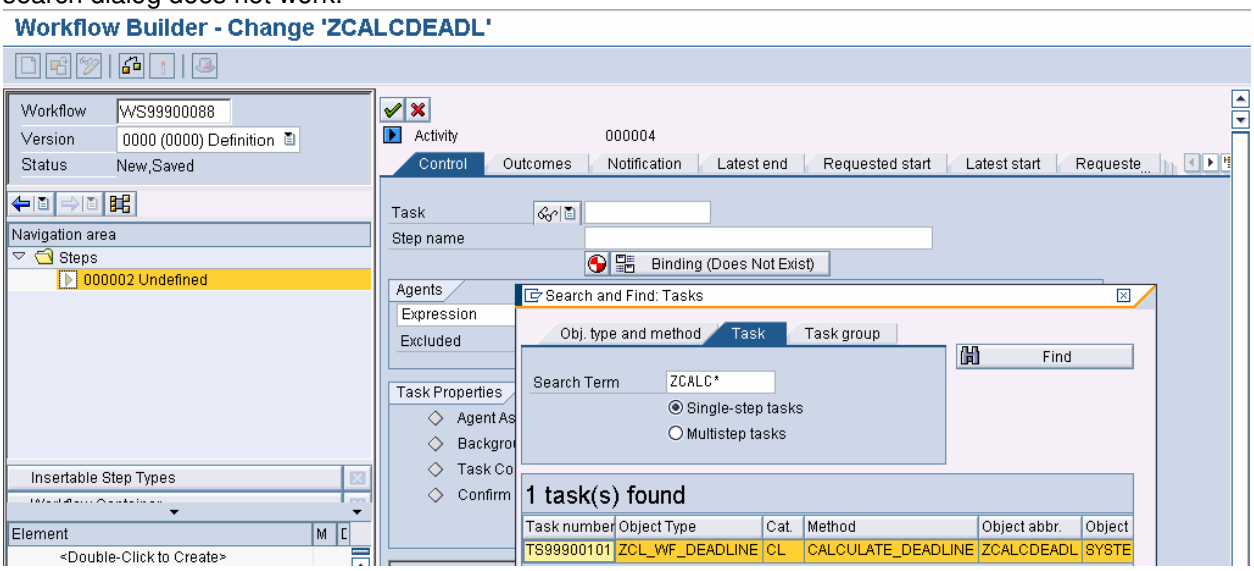
5 Calling the task from the workflow

Calling a standard task based on an ABAP OO class/method is no different to calling a standard task based on BOR.

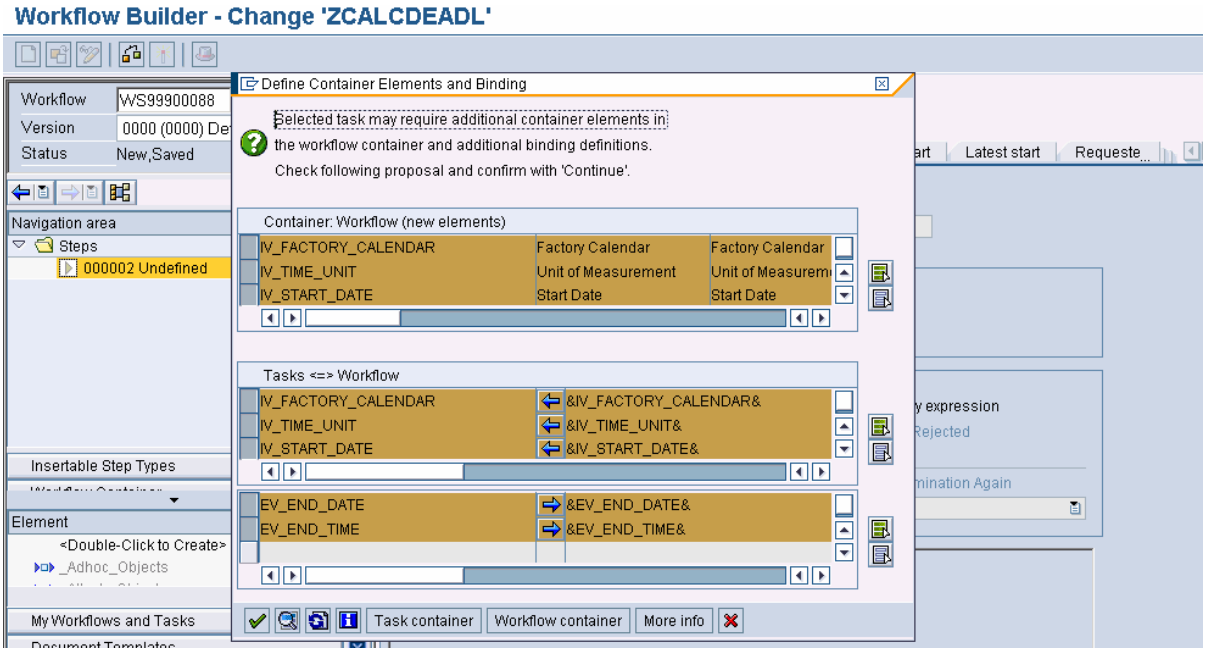
Workflows are created and maintained using transaction SWDD.

Use an activity step to call the new task.

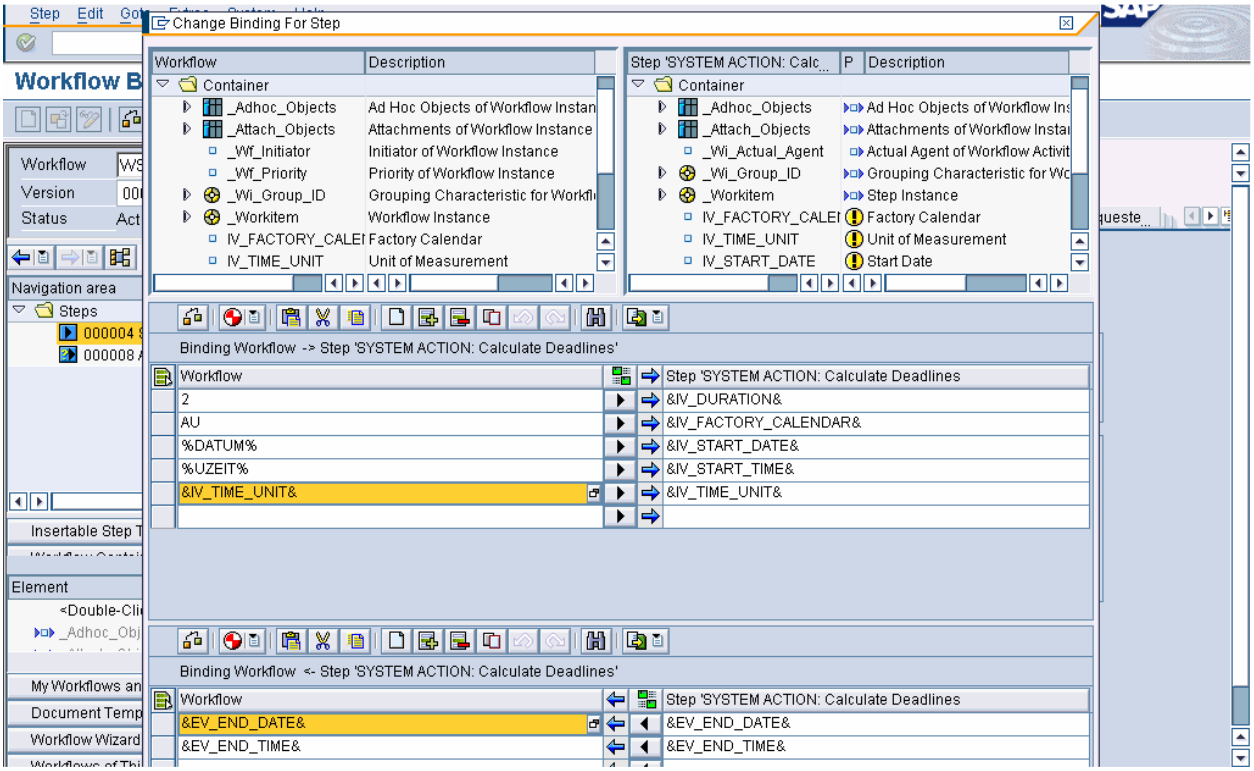
The task can be found using the search dialog. Note: In 6.20 the Object Type and Method tab in the task search dialog does not work.



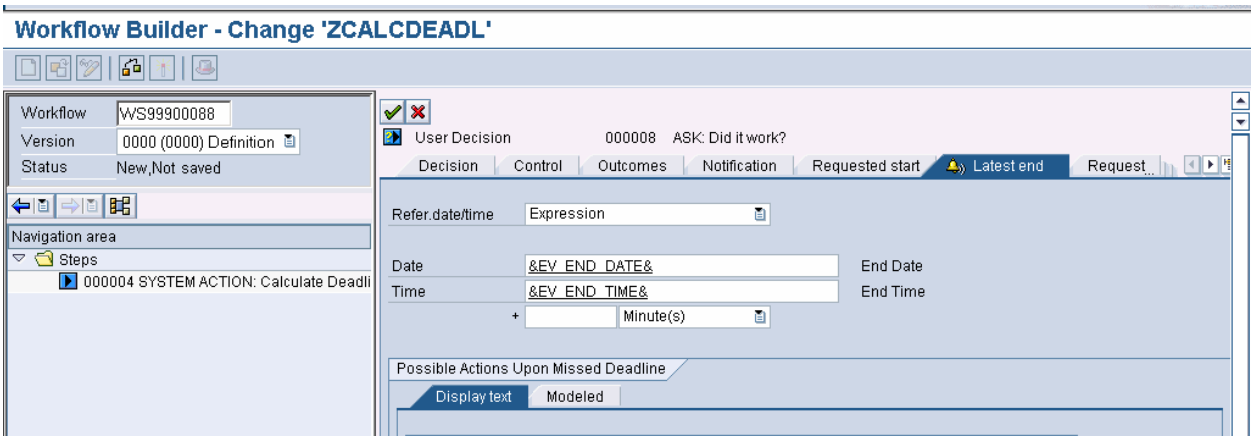
Bindings can be generated automatically or entered manually as usual.



In the binding definition you can either provide workflow container elements or pass constants to fill the import parameters with appropriate values. Note: The conversion of the field and the help depends on the definition of the individual fields, e.g. to set the time unit to DAY the technical id TAG is entered.



The values passed back to the workflow container can then be used in subsequent steps of the workflow as usual.



6 Future Direction

Writing utility classes using the Business Object Repository has always been awkward. As ABAP OO is easier to use for utility classes, by preference, ABAP OO should be used for utility classes instead of the BOR.

7 Further information

Further information can be found in OSS using component id BC-BMT-WFM and in the SAP Library Help under SAP Web Application Server -> Business Process Management.