ABAP OO for SAP Business Workflow (including 6.40)

Jocelyn Dart, 28.10.2004, 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.

There are a number of benefits of moving from the BOR to ABAP OO:

- ABAP OO code can be maintained by any ABAP programmer and does not require a specialist workflow programmer.
- ABAP OO code can be used in workflow and non-workflow programs by non-workflow programmers
- Improved coding support features such as forward navigation, proper separation and identification of different attribute types
- Improved clarity (as a result of enforced coding standards – e.g. TABLES statements and internal tables without header lines are not allowed)
- Utility functions are quicker to develop in ABAP OO

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.

Notes:

1. Although ABAP Classes can be used with workflow from 6.20, there are "restrictions" (i.e. not everything works) until 6.40. Having said that - the differences so far appear to be relatively minor with simple scenarios – for example, searches on object type/method don't work, test tools

are not able to show tables of ABAP instances etc. The major lack is with virtual attributes – there is no good solution for these in 6.20, some approximations are suggested in this document. In 6.40 virtual attributes are implemented as functional methods.

2. Depending on the release (prior to 6.40), before ABAP OO can be used for workflow it may be necessary to run program **RSWF_CATID** to allow ABAP Classes to be used for workflow.

2 ABAP Classes versus BOR Object Types

The main differences between ABAP Classes and BOR Objects for workflow are:

Point of Difference	**BOR Object Types**	**ABAP Classes**
Delegation	Delegation via customizing table	No delegation
Maintenance Transaction	Transaction SWO1	Transaction SE24
Key length	Maximum key length = 70 chars	Maximum key length = 32 chars (after that use GUIDs – Globally Unique Identifiers)
Attribute types	Key, Database, Virtual, Status, Protected, Private	Key, Non-key, Protected, Private
Attribute access	All attributes are read only	Attributes can be changeable – however this cannot currently be used with workflow (workflow treats all attributes as read-only)
Method types	Dialog/Non-dialog, Synchronous/Asynchronous	Not relevant
Method parameters	Importing, Exporting, Result	Importing, Exporting, Returning **Note :** As of 6.40 Functional Methods (methods with a Returning parameter) can be used similar to BOR virtual attributes.
Method exceptions	Temporary, Application or System	Numerous exception classes starting with CX_BO_
Handling tables	“Multiline” parameters	Table types (when used in container elements, table types automatically set the multiline parameter)

3 Defining ABAP classes for Workflow

3.1 Class Definition

ABAP Classes are defined in transaction SE24. When creating an ABAP Class for workflow, create a “usual” (or “normal”) class. The “Final” flag (which determines if subclasses can be created) is optional.

Note: The class is NOT persistent – workflow handles the persistence needed to complete the workflow scenario.

The example is based on a class ZCL_PLANT representing the plant.

Every ABAP Class to be used by workflow needs to implement interface IF_WORKFLOW. This interface is used to recognize the class as workflow-relevant in all workflow functions.

Adding interface IF_WORKFLOW to a class, inherits the interfaces BI_OBJECT and BI_PERSISTENT. These interfaces have methods used to control the conversion from the ABAP Class to the local persistent object reference used by workflow and vice versa.

To ensure workflow is able to do this without runtime errors, BI_OBJECT and BI_PERSISTENT contain methods that must be implemented and activated. If the class contains only static attributes/methods, such as a utility class, it is sufficient to activate the inherited methods without adding any code. However if the class contains instance-dependent attributes or methods then the methods must be filled. The remainder of this section describes and gives examples of what code is needed in each case.

Key attributes must be defined as these identify the business keys used in the business object references passed by workflow. A non-key attribute to show the Local Persistent Object Reference is useful for debugging purposes.

In the example below, the attribute PLANT (type WERKS) has been defined as a key, instance, public, read-only attribute. The attribute M_POR (type SIBFLPOR) has been created as a non-key, instance, public, read-only attribute to show the Local Persistent Object Reference is for debugging purposes.

So that an instance can be created, a CONSTRUCTOR method (or SUPERCONSTRUCTOR if using static classes) must be created. This will be used by BI_PERSISTENT~FIND_BY_LPOR to convert from the workflow business object reference to the ABAP Class instance.

Similar to Business Objects, it is worthwhile to implement an EXISTENCECHECK method to check the validity of the key values passed by workflow in the business object reference.

Examples of ABAP Classes for workflow are:

- Class CL_SWF_FORMABSENC (6.40 and above)
- Any class implementing IF_WORKFLOW (Where-used list on interface IF_WORKFLOW)

3.1.1 Method CONSTRUCTOR

Method CONSTRUCTOR creates an instance of the ABAP Class.

It makes sense that the key attributes should be importing parameters of the constructor method so that the key attributes can be filled during instantiation. This method can then called by the BI_PERSISTENT~FIND_BY_LPOR method as a result of creating the ABAP Class instance from the workflow reference.

In the example a single importing parameter PLANT (type WERKS) was used.

To prevent invalid data, the validity of the business reference key should be checked, e.g. using an existencecheck method.

This method also needs to fill the Local Persistent Object Reference so that it can be used by method BI_PERSISTENT~LPOR.

To ensure consistency during monitoring and in work item links, the values of the default attributes should be filled at this time also (directly or by calling another method – in the example a supply method is called).

```
METHOD constructor .


  me->plant = plant.
  me->existencecheck( ).

  me->m_por-catid  = 'CL'.
  me->m_por-typeid = 'ZCL_PLANT'.
  me->m_por-instid = me->plant.

* Fill T001W fields as these are used for the default attribute
  me->supply_t001w( ).
ENDMETHOD.
```

3.1.2 Method EXISTENCECHECK

This is optional but useful for checking the validity of the business object reference key.

```
method EXISTENCECHECK .
  data: lv_plant type werks.

 SELECT SINGLE werks FROM t001W into lv_plant
        WHERE werks = me->plant.
  IF sy-subrc NE 0.
    raise exception type cx_bo_error.
  ENDIF.


endmethod.
```

3.1.3 Method DISPLAY

This is optional but useful. This method is called as the default method of the class. Compare this to the coding of the DISPLAY method in business object BUS0008 (Plant).

```
method DISPLAY .
  data: ls_vt001w type v_t001w.

  if me->category is initial.
    CLEAR ls_vT001W.
    ls_VT001W-MANDT = SY-MANDT.
    ls_VT001W-WERKS = me->PLANT.
    CALL FUNCTION 'VIEW_MAINTENANCE_SINGLE_ENTRY'
      EXPORTING
        ACTION    = 'SHOW'
        VIEW_NAME = 'V_T001W'
      CHANGING
        ENTRY     = ls_vT001W.
  else.
    SET PARAMETER ID 'WRK' FIELD me->PLANT.
    CALL TRANSACTION 'WB03' AND SKIP FIRST SCREEN.
  endif.

endmethod.
```

3.1.4 Method BI_PERSISTENT~FIND_BY_LPOR

Method BI_PERSISTENT~FIND_BY_LPOR converts from the Local Persistent Object Reference to the Persistent Business Instance, i.e. using the object reference (business-related) key to instantiate the class. The LPOR-INSTID field contains all attributes marked as “key” attributes.

```
method BI_PERSISTENT~FIND_BY_LPOR .

data: lv_plant type werks.

     move lpor-instid(4) to lv_plant.

     create object result TYPE ZCL_PLANT
        exporting plant = lv_plant.

endmethod.
```

3.1.5 Method BI_PERSISTENT~LPOR

Method BI_PERSISTENT~LPOR converts from Persistent Business Instance to Local Persistent Object Reference, i.e. fill the object reference (business-related) key from the current instance. Field LPOR-INSTID must be filled with the appropriate values for all attributes marked as “key”attributes.

```
method BI_PERSISTENT~LPOR .

  DATA: lpor TYPE sibflpor.

  lpor-catid   = 'CL'.
  lpor-typeid  = 'ZCL_PLANT'.
  lpor-instid  = me->plant.

  result = lpor.


endmethod.
```

3.1.6 Method BI_PERSISTENT~REFRESH

Method BI_PERSISTENT~REFRESH refresh attributes. This is the equivalent of swc_object_refresh macro in Business Object Types. This method is optional, e.g. any methods that extract attribute values from the database could be called here to ensure the values are up to date.

```
method BI_PERSISTENT~REFRESH .
endmethod.
```

3.1.7 Method BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE

Method BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE sets up default attribute value for work item links and monitoring. This method is optional. If no code is added to this attribute the concatenated value of all attributes marked as “key” attributes will be used.

```
method BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE .
 GET REFERENCE OF me->description INTO result.
endmethod.
```

3.1.8 Method BI_OBJECT~EXECUTE_DEFAULT_METHOD

Method BI_OBJECT~EXECUTE_DEFAULT_METHOD sets up default method value for work item links and monitoring. This method is optional. If no default method is defined, no method will be executed from object links in work items and monitoring.

```
method BI_OBJECT~EXECUTE_DEFAULT_METHOD .
  try.
      call method me->display.
    catch cx_root.
  endtry.
endmethod.
```

3.1.9 Method BI_OBJECT~RELEASE

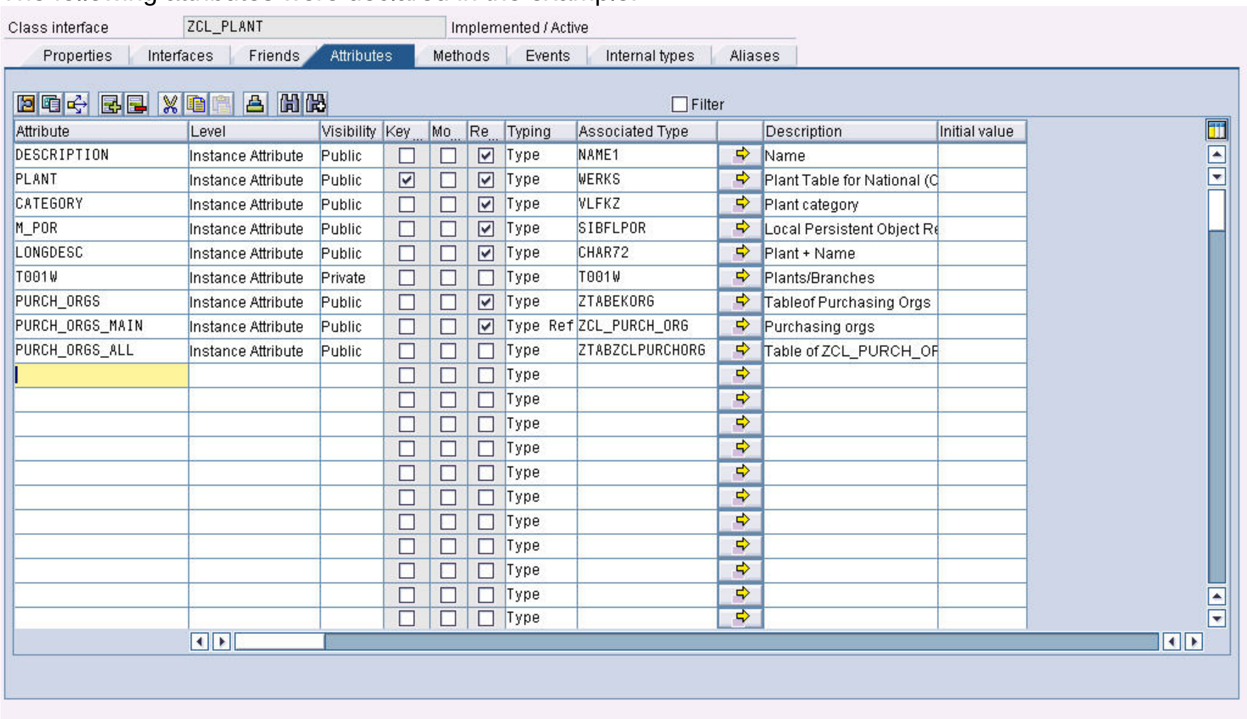
Method BI_OBJECT~RELEASE release the object for garbage collection when it is no longer required. This method is optional.

```
method BI_OBJECT~RELEASE .
endmethod.
```

3.2 Attributes

Attributes to be used in workflows must be “Public” attributes so that they can be used outside the class itself.

The following attributes were declared in the example:



Attribute	Level	Visibility	Key...	Mo...	Re...	Typing	Associated Type		Description	Initial value
DESCRIPTION	Instance Attribute	Public	[ ]	[ ]	[x]	Type	NAME1	⇨	Name	
PLANT	Instance Attribute	Public	[x]	[ ]	[x]	Type	WERKS	⇨	Plant Table for National (C	
CATEGORY	Instance Attribute	Public	[ ]	[ ]	[x]	Type	VLFKZ	⇨	Plant category	
M_POR	Instance Attribute	Public	[ ]	[ ]	[x]	Type	SIBFLPOR	⇨	Local Persistent Object Re	
LONGDESC	Instance Attribute	Public	[ ]	[ ]	[x]	Type	CHAR72	⇨	Plant + Name	
T001W	Instance Attribute	Private	[ ]	[ ]	[ ]	Type	T001W	⇨	Plants/Branches	
PURCH_ORGS	Instance Attribute	Public	[ ]	[ ]	[x]	Type	ZTABEKORG	⇨	Tableof Purchasing Orgs	
PURCH_ORGS_MAIN	Instance Attribute	Public	[ ]	[ ]	[x]	Type Ref	ZCL_PURCH_ORG	⇨	Purchasing orgs	
PURCH_ORGS_ALL	Instance Attribute	Public	[ ]	[ ]	[ ]	Type	ZTABZCLPURCHORG	⇨	Table of ZCL_PURCH_OF	

3.2.1 Key Attributes

Both Business Object Types and ABAP OO Classes have “Key” attributes that are used for the same purpose, i.e. to uniquely identify a single object instance based on a business-related key. When an object is instantiated, the business-related key must be passed, and the existence of this key in the relevant database tables is checked.

The key attributes determine the contents of the INSTID field in the Local Persistent Object Reference. The maximum length of the key is 32 characters. If the business key is longer, a GUID (Globally Unique Identifier) should be used as the key, and a relationship established between the GUID and the business key (e.g. held on a database table).

3.2.2 Non-Key Attributes

The main differences between BORand ABAP OO attributes are for public, non-key attributes.

In Business Objects, there are three (3) types of attributes for non-key attributes:

- Database – Public, read-only attributes based on database table SELECTs called at the time the object is instantiated. All database attributes referencing the same database table share common code to execute the SELECT statement.
- Virtual – Public, read-only attributes calculated at the time of usage. Each virtual attribute has its own code. There is no guarantee of the order in which virtual attributes will be evaluated if more than one attribute is used (e.g. in a work item description).
- Status – Public, read-only attributes calculated via status management. Shows “X” if the status is active and blank if not. Requires additional virtual attributes to be implemented to determine the status object and status object type.

In ABAP OO Classes, these distinctions do not exist, and attributes are not directly linked with any code.

From an ABAP OO perspective, BOR database attributes most closely resemble attributes filled in the CONSTRUCTOR method when an instance is created. (Of course static attributes are filled in the SUPERCONSTRUCTOR method instead of the CONSTRUCTOR method.)

From an ABAP OO perspective, BOR virtual and status attributes most closely resemble public ABAP OO methods with a single returning parameter, i.e. Functional Methods.

Workflow is able to use ABAP OO attributes and Business Object attributes directly in descriptions and bindings. However ABAP OO methods cannot be used directly in descriptions and bindings before 6.40 – they must be called via a standard task. As one of the main advantages of virtual attributes was to avoid creating standard tasks to minimize overhead, the lack of a virtual attribute equivalent is a potential disadvantage of using ABAP OO prior to 6.40. If each attribute has its own supply method, then potentially many tasks and work items will need to be created to fill all the attributes needed for a single workflow.

One option to get around this in systems prior to 6.40 may be to use a single public SUPPLY_INSTANCE method to initiate supply of all non-key attributes (or perhaps SUPPLY_STATIC for static attributes). The workflow would then only need to call a single method/task/work item to fill all attributes needed for the workflow.

This SUPPLY_INSTANCE method would have a true/false flag to request the attribute be filled for each attribute, but no export parameters. The method then calls the appropriate SUPPLY_<attribute> (where

<attribute> is the attribute name) methods to fill the attributes requested. This approach gives added flexibility as to when and in what order supply methods are called, which is particularly useful when attributes are dependent on each other.

As of 6.40, functional methods can be called directly in work item texts, descriptions, and bindings similar to BOR virtual attributes. Parameters can be passed to functional methods, e.g.:

&_WI_OBJECT_ID.DESC_AND_PARAM(PARAM="X")&

Regardless of the release, each "virtual" attribute should have its own public functional method, with a returning parameter that supplies the virtual attribute value. This method can buffer the calculated value in an attribute of the object – to avoid calculating the value more often than necessary.

If some attributes are closely related, such as database attributes from the same table, a single supply method can be used for the group e.g. SUPPLY_T001W (for table T001W). This method is then responsible for filling all of the attributes in the group. This method can be called directly or as part of other methods such as the CONSTRUCTOR, or SUPPLY_<attribute> methods.

3.2.3 Read-only versus Changeable Attributes

Workflow treats all attributes as read-only.

ABAP OO can also have public attributes which are changeable. These were not available in BO. However it is not possible to change the value of these attributes via workflow e.g. you cannot use an attribute of an ABAP Class as the result of a container operation, or use a binding to set the value of an attribute of an ABAP Class.

Where attributes are truly read-only it may be useful (although not mandatory) to enforce this by marking them as read-only in the attribute definition. This avoids confusion with non-workflow programs using the workflow class.

3.2.4 Protected and Private Attributes

Both BOR and ABAP OO may have protected (visible to the object and its children) and private (visible to the object only) attributes and use them for much the same purposes, e.g. to hold already found intermediary values used in public attributes and methods to save re-reading the same data multiple times.

For example where multiple values are retrieved from the same table, it may be useful to have a private attribute with all fields of that table. This can make adding new public attributes based on the same table easier.

3.3 Methods

Methods to be used in workflows must be "Public" methods so that they can be used outside of the class. Protected and private methods of the class can be used for intermediate processing (similar to the use of subroutines in Business Object Types).

3.3.1 Method Parameters

Similar to Business Object methods, ABAP Class methods must formally declare all parameters. Parameters can be Importing, Exporting or Returning. Only one parameter can be a returning parameter (a special export parameter).

The concept of returning parameter is very similar to the Business Object result parameter however workflow treats the returning parameter the same as any exporting parameter, i.e. the special task container element _WI_Result is not automatically generated from a returning parameter.

As of 6.40 the task definition "Container" tab includes a special button "Add Returned Element" to create _WI_RESULT from the returned parameter. This does not affect the definition of the method (i.e. the method does NOT need to pass a returning parameter called "RESULT" as was the case with BOR methods).

3.3.2 Dialog and Synchronous Settings

The "Dialog" flag is not available. This is set in the standard task as follows:

- Dialog – Background Processing flag is off (default)
- Non-dialog – Background Processing flag is on

The "Synchronous" flag is not available. This is set in the standard task as follows:

- Synchronous = Synchronous Object Method flag is on
- Asynchronous = Synchronous Object Method flag is off (default)

Note: The Synchronous Object Method flag can only be changed once the task has been created (i.e. saved to the database).

3.3.3 Exception Handling

Similar to Business Object methods, ABAP Class methods must formally declare any exceptions to be used.

With Business Objects, each exception was linked to a message and also marked as "temporary", "application" or "system".

Note: Temporary exceptions do not send the work item into error – they remain in the inbox and can be retried later. Application and system exceptions send the work item into error.

With ABAP Classes, exception handling is via exception classes. A number of different exception classes are available including:

Exception Class	**Purpose**
CX_BO_TEMPORARY	Temporary exception, e.g. locking error
CX_BO_ACTION_CANCELLED	User cancelled dialog processing (subclass of CX_BO_TEMPORARY
CX_BO_SYSTEM	System exception
CX_BO_APPLICATION	Application exception
CX_BO_INSTANCE_NOT_FOUND	Instance not found
CX_BO_INSTANCE_NOT_EXISTING	Instance does not exist
CX_BO_INSTANCE_EXISTING	Instance already exists
CX_BO_INSTANCE_DELETED	Instance already deleted

CX_BO_FIND_ERROR	Cannot find instance
CX_BO_CREATE_ERROR	Cannot create instance
CX_BO_DELETE_ERROR	Cannot delete instance
CX_BO_ERROR	General error
CX_BO_ABORT	General abort

The command RAISE EXCEPTION is used to raise the exception, e.g.

RAISE EXCEPTION TYPE cx_bo_action_cancelled.

3.3.4 Example method SUPPLY_LONGDESC

The instance, private SUPPLY_LONGDESC method fills the LONGDESC attribute.

The LONGDESC attribute is the equivalent of a virtual attribute in a Business Object. When calling this method in work item and long text descriptions the insert expression would generate a reference such as:

```
&_WI_OBJECT_ID.SUPPLY_LONGDESC()&
```

Notice that the SUPPLY_INSTANCE method ensures that the DESCRIPTION attribute has been filled first.

```
METHOD supply_longdesc .

  CONCATENATE me->plant me->description INTO me->longdesc.

ENDMETHOD.
```

3.3.5 Example method SUPPLY_T001W

The instance, private SUPPLY_T001W method fills the T001W and related attributes.

This approach is the equivalent to handling database attributes in Business Objects.

```
method SUPPLY_T001W .

  SELECT SINGLE *
         FROM  T001W
         INTO me->T001W
         WHERE WERKS = plant.

  category = me->T001w-vlfkz.
  description = me->T001w-name1.

endmethod.
```

3.3.6 Example method SUPPLY_PURCH_ORGS

The instance, private SUPPLY_PURCH_ORGS method fills all attributes related to purchasing organizations.

This method shows filling of attributes that are themselves ABAP OO instances, or tables of ABAP OO instances.

```
METHOD supply_purch_orgs .

  DATA: lv_purch_org TYPE ekorg,
        lo_purch_org type ref to zcl_purch_org.

  clear: me->purch_orgs, me->purch_orgs_main, me->purch_orgs_all.

  SELECT ekorg FROM  t024w
         INTO TABLE me->purch_orgs
         WHERE  werks  = me->plant.

  READ TABLE me->purch_orgs INTO lv_purch_org INDEX 1.
  IF sy-subrc EQ 0.
    CREATE OBJECT me->purch_orgs_main TYPE zcl_purch_org
      EXPORTING purch_org = lv_purch_org.
  ENDIF.

  loop at me->purch_orgs into lv_purch_org.
      CREATE OBJECT lo_purch_org TYPE zcl_purch_org
      EXPORTING purch_org = lv_purch_org.
      append lo_purch_org to me->purch_orgs_all.
  endloop.


ENDMETHOD.
```

3.3.7 Example method SUPPLY_INSTANCE

The instance, public SUPPLY_INSTANCE method calls private supply methods to fill attributes.

Parameters are:

Ty.	Parameter	Type spec.	Description
	VALUE(IV_LONGDESC)	TYPE XFELD OPTIONAL	Checkbox
	VALUE(IV_DESCRIPTION)	TYPE XFELD OPTIONAL	Checkbox
	VALUE(IV_CATEGORY)	TYPE XFELD OPTIONAL	Checkbox
	VALUE(IV_PURCH_ORGS)	TYPE XFELD OPTIONAL	Checkbox

```
METHOD supply_instance .

  IF NOT iv_description IS INITIAL
  OR NOT iv_category IS INITIAL.
    me->supply_t001w( ).
  ENDIF.

  IF NOT iv_longdesc IS INITIAL.
    me->supply_longdesc( ).
  ENDIF.

  IF NOT iv_purch_orgs IS INITIAL.
    me->supply_purch_orgs( ).
  ENDIF.

ENDMETHOD.
```

3.4 Events

Events to be used in workflows must be “Public” events so that they can be used outside of the class.

Events may have exporting parameters similar to Business Object events.

Events can be static instead of instance-dependent. This is the only major difference from Business Object events. Of course a static event does not pass the instance. This may be useful for terminating events.

Events can be raised in methods. The following example is an instance, public method RAISE_EVENT with a single importing parameter IV_EVENT (type SWO_EVENT).

```
method RAISE_EVENT .

        TRY.
            CALL METHOD cl_swf_evt_event=>raise
              EXPORTING
                im_objcateg = me->m_por-catid
                im_objtype  = me->m_por-typeid
                im_event    = iv_event
                im_objkey   = me->m_por-instid.
          CATCH cx_swf_evt_exception.
        ENDTRY.
        COMMIT WORK.

endmethod.
```

4 Using ABAP OO Attributes in Workflow

Attributes of ABAP Classes are used in workflow in exactly the same way as BOR attributes:

- To provide values for process control (container operations, loops, start conditions, etc.)
- To provide values for bindings (e.g. event to workflow, workflow to task, task to workflow, task to rule, etc.)
- In descriptions (e.g. work item text, work item long description).

Using an ABAP OO attribute in work item texts, work item descriptions or bindings, results in a technical reference that looks like this:

```
&_WI_OBJECT_ID.DESCRIPTION&
```

Functional methods can also be used similar to BOR virtual attributes. Using an ABAP OO functional method in work item texts, work item descriptions or bindings, results in a technical reference that looks like this:

```
&_WI_OBJECT_ID.SUPPLY_LONGDESC()&
```

5 Using ABAP OO methods in workflow tasks

Methods of ABAP Classes are used in workflow in a similar way to BOR methods. When including a method of an ABAP Class in a standard task, the following differences are noticeable:

- The “Object Category” must be set to “ABAP Class”
- The ABAP Class name is specified as the “Object Type”
- The “Background Processing” flag must be set for non-dialog methods
- The “Synchronous Object Method” flag must be set for synchronous methods
-

Note: The Synchronous Object Method flag can only be changed once the task has been created (i.e. saved to the database).

Generation of container elements from the method parameters, definition of texts and terminating events, are all essentially the same as for BOR methods, other than the _WI_Result container element.

Prior to 6.40, the_WI_Result container element is never used (returning parameters are treated the same as any other export parameter).

As of 6.40, “Returning” parameters are not automatically marked as export parameters. The developer can choose to:

- Either change the properties of the generated task container element to allow export
- Or use the “Add Returned Element” button on the Container tab of the task definition to create a _WI_RESULT parameter.

If using “Add Returned Element” the developer only has to complete the defined type (object type, Dictionary field, or Dictionary type) of the returned element.

6 Using ABAP OO events in workflow

Events of ABAP Classes are used in workflow in a similar way to BOR events. When including an event of an ABAP Class in an event linkage, the following differences are noticeable:

- The “Object Category” must be set to “ABAP Class”
- The ABAP Class name is specified as the “Object Type”

The event linkage acts as an event handler for the ABAP Class event.

To call events from programs use the class CL_SWF_EVT_EVENT. There’s also an interface for adding event parameters – check the SAP Library help.

7 Further information

Further information can be found in the SAP Library Help in the SAP Business Workflow Reference Documentation.