Book 6

Programming With Messages

U:\Book\Book_06.doc

Programming With Messages

What to Read in This Part

5

Fehler! Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

1 Message Based Software Development

One of the basic concepts of client server technologies is the communication between client and server through messages and message queues. Instead of calling a distinct program, the requesting client posts a query to a central message queue where it is picked up and processed by a message handler or an interested application. The results of the processing are then posted back to the queue according to an agreement between client and server.

1.1 About Messages and Communication

A client posts its inquiry to a common bulletin board to be picked up by a server

The thought behind messages is like in real life. Someone (the client) needs help of a competent service person (the server). However, the client does not know the server that may be capable to fulfil the requirement. So it posts its message to a public bulletin board and any interested server that reads the message, picks up the message, executes the required activity and returns with a result.

Tasks can be executed secretly or the inquiry may be used to establish a private communication channel

Of course, this can be done through a variety of processing protocols. The server can perform the task silently and eventually come back with a result. The server could also read the message, inform the client that it is interested and ask for detailed instructions what to do and how to do it. After a client and server have contacted each other they may establish a private communication channel to continue the task.

Imagine a student looking for a flat and a landlord looking for a tenant

Simple imagine the scenario, when a landlord has a free apartment to let and is looking for a tenant, while at the same time a student looks for a reasonable apartment. The two would not know each other, so how can they come together. If they are clever they go to a public bulletin board where the landlord posts his offer. When the student finds the quote attractive he will contact the landlord under the posted code or telephone number. Thereafter the two will establish a private conversation to negotiate the deal.

The Microsoft MSMQ Message Queue Sever is a central message server through which all applications may communicate

The Microsoft Message Queue or any other message server within your enterprise network server the purpose of the bulletin board for all running applications. The MSMQ is actually a central database in which a client posts a message until a server picks them up. Physically the MSMQ is simply a database where all messages and returned results are centrally stored in a categorised manner.

Imagine a student looking for a flat and a landlord looking for a tenant

As an intelligent message system the MSMQ does not only store the messages and results, but can also wake up interested applications if it recognises the type of message. To do this job Microsoft supplies an MSMQ application called *Microsoft MSMQ Trigger*. This application can scan message strings for keywords, codes, sender or receiver addresses and trigger an appropriate application. Microsoft Outlook's rule based mail organiser works in a similar way.

1.2 Message Queue Application

Message queue allow asynchronous processing of application service requests where an application can submit a request and instead of waiting for a result spend the time doing something else.

The ASP page submits its requests to the message queue and receives a ticket

When an application has a resource intensive request it submits this request to the message queue. When the queue accepts the request it returns a unique ticket number which will be needed later to pick up results returned by the application that served the message queue.

The message queue serializes and stores the requests

When a message has been received by the queue it holds the request until a service demon asks to process them. Depending on the action requested the message queue submits the task and either holds the request package open for a result message to be returned by the service or it stores a compliant message immediately to the queue to be picked up as success message by the calling

process.

An interest handler demon picks up the next in line and processes it

Parallel there are a number of demons running which can handle the message queue requests. They would usually scan the queue for compliant requests and pick up the next in line.

Results of the queue demon are stored back to the queue under the ticket reference

The handler routine that served the request then stores a result message back to the queue. How this result message looks like depends on the agreement made between queue requester and the service demon. Usually it would be an XML wrapped data table.

Message queuing lets you place requests in a queue and retrieve the result at a later time

Imagine the familiar situation where you connect to your database through a busy web page. With many simultaneous sessions trying to access the database you may experience long waiting times. With a message queue you can deposit your request to a server, which eventually processes it and places the propagated result back in the queue for you to pick it up, when you find it appropriate. While your server takes care of the queue you can spend time doing something more useful.

Message queues are a central momentum in client server developments

Actually a powerful message queuing system plays a central role in any client server environment especially if you find your application back in a huge server farm. The message queue together with the transaction server will relieve you of the burden of handling load balancing and pooling threads. A message queue server could start an instance of every important task on every application server and send it immediately to sleep. Only when it detects a heavy request for a certain application will it wake that application up.

70

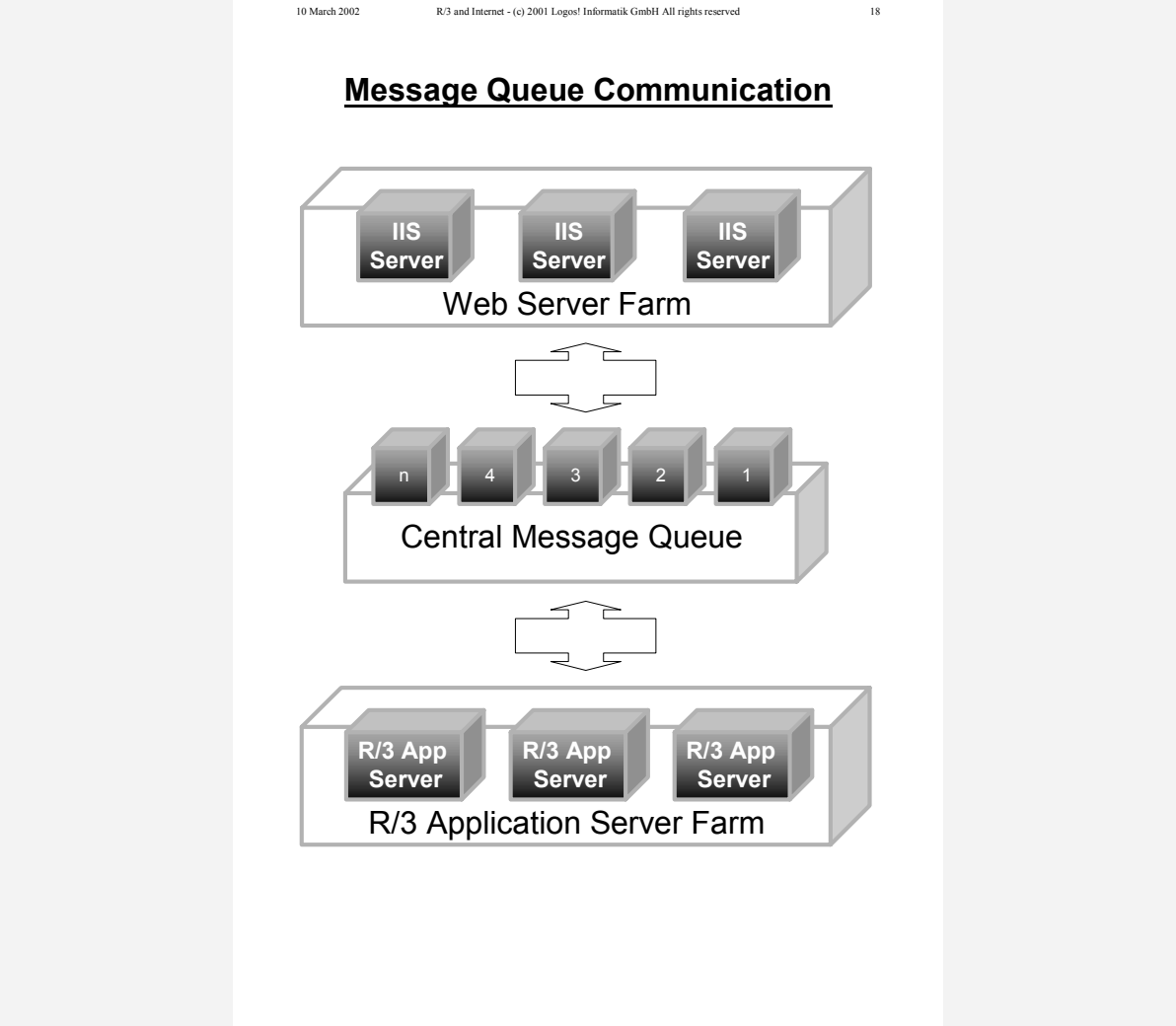
Message queues are also used to collect tasks for later or batch execution

Another common application of a message queue is to collect requests for delayed processing, e.g. as a batch job during off-peak times. Typically this is done with print documents like invoices which are due to be sent and thus printed only once per week. The SAP R/3 NAST table is such a message queue and the ABAP RSNAST00 is the matching handler for it.

Message queuing is part of the windows NT or Windows 2000 system

The Microsoft Message Queue is part of the Windows NT and 2000 server installation.

Figure 1: Use of a message queue as dispatcher in app server farms



2 R/3 Internal Messaging

The R/3 programming model is built heavily around a communication model. There are principally three different message pipes installed in R/3, each of them serving different purposes and with individual advantages and disadvantages. They are message controls, workflow events and IDocs.

2.1 NAST Message Control

Message control serves as a mail drop to trigger subsequent actions

The message control is also known as the NAST messaging due to the name of the database table NAST (germ: NAchrichtenSTeuerung). The mechanism is, that a requesting application deposits a request message in table NAST and an interested consumer application eventually picks them up and decides upon an appropriate action. The consuming action is fully independent from the requester. If the requester expects a response, it has to poll the expected result tables.

- The requesting application stores a message in table NAST
- A program (usually RSNAST00) polls the table NAST for unprocessed messages and activates the program that has been dedicated to process that message
- The processing function finally finalizes the message by marking the message as processed and storing status information and eventual log data to it

2.2 Workflow Events

Workflow events signal that a program has reach a certain state and there may be a workflow handler to react on it or not

Workflow events are more direct way to trigger a subsequent function. The idea is that the calling program signals to the public that it reached a certain state, it *fires an event*. The workflow handling mechanisms evaluates the event and determines if there is an interested application that should react on the event. Contrary to the classical way of calling a subroutine or chaining a subsequent program a workflow event does not require a subsequent action. The calling program simply tells the public that it reach some state, but it does not care if there a program to react on. That is like someone, who comes home and calls back his spouse to report the arrival. The informed partner may just take notice of it, but is not obliged to do something. Indeed, it is the rule that a workflow event is simply burnt and triggers no reaction at all.

The decision between NAST or workflow communication depends on the circumstances. As a rule of thumb you may consider the following questions:

- Which mechanism is supported by the calling application? Not all apps do actually support workflow or NAST message creation in SAP standard.
- If the message requires obligatorily an action and especially if you require a protocol of the action after the message, then you should prefer NAST messaging.
- If the number of required action sparse and the calling app is not interested in any of the following events, then you may consider workflow.

2.3 Intermediate Documents (IDocs)

Intermediate Documents (IDocs) are used if the message holds additional data which is not stored anywhere else

SAP's IDocs (Intermediate Documents, read also [Axel Angeli; The R/3 Guide to EDI and Interfaces]) have been originally designed as a quarantine mailbox for incoming and outgoing EDI data. Their big advantage is that the IDoc system comes with a sophisticated monitoring and error tracking system, that allows controlled reprocessing of IDocs, manipulation of partly erroneous data and stores sorted and categorized protocol information. R/3 can send IDocs to itself and thus you can benefit from the mechanism, by building an IDoc message and submit it to the IDoc processor.

2.4 error tracking

3 Messaging With R/3 IDocs

IDocs are typically used to exchange data asynchronously between R/3 and an external system. IDoc are simple ASCII data files to package all relevant data to process a transaction. Other than calling an RFC function, an IDoc can be sent to R/3 for processing, but the sender would not have to wait for the process to finish. That is the same philosophy which is also behind an HTTP Post request.

4 Programming SAP R/3 Workflow Technology

Workflows are used for to purposes: first they allow the triggering of subsequent actions based on a termination event of a transaction and second they allow trigger whole execution chains of transactions. Workflow allows you to set up chained and conditional program execution without having to modify the R/3 standard and to defined a protocolled and controlled way to create workitems (emails that request a decision or to complete a data form by a human interactor).

4.1 Understanding and Developing R/3 Workflow

Workflow is the automatic triggering of subsequent programs by triggering an event from a calling transaction and executing a handler to process an action. The following chapters will assume that you already have heard of R/3 workflows. We will primarily try to sort out that workflow and object programming are completely independent.

Steps to Do

When you visit SAP's workflow courses you will be told a lot about organisation hierarchies, creating object methods and assigning them to workitems and waiting for events.

You are probably not told, how you can create an event on your own, when you have to create object methods for an handler or if it is absolutely necessary to change the HR organization hierarchy.

Learn about workflow what is interesting for a developer

- How to fire a workflow event from a user exit or a user program
- How to trigger a workflow directly without using event coupling
- Using workflows without using the HR organisation hierarchy
- How to write workflow handler without defining object methods

Understand the terminology first. Please.

Terminology is the basis of the whole understanding. Not only in workflow but everywhere in life. In R/3 we find it difficult enough that there are numerous expressions which are used in different context in different manners. Pretty often they are even used ambiguously (e.g. the term *object*). In addition consultants and users are often mistaken when expressions in R/3 are used differently than you are accustomed to from your daily work.

Terms derived from OOP programming

The following terms originate from OOP development and are frequently used in workflow

- Event
- event handler
- object
- object method
- container

Terms originating from workflow

The following terms originate directly from workflow development

- Event coupling
- Workitem
- Actor
- Role
- Organisation hierarchy

OOP Terminology Used in Workflow

There is couple of expressions which are used in workflow development which are taken from the philosophy of Object Oriented Programming (OOP). We list them in different sections and hope this will help to get rid of some of the confusion raised by standard R/3 documentation, which swivels permanently between the two areas, although OOP is pure development which is applied in

workflow every time it appears to be necessary.

Method	A method is an executable part of an object. Technically they are more or less the same as function modules.
Event	An event is a message to other program to take care for subsequent appropriate action
Event handler	An event handler is a program which executes automatically when a matching event is raised
Container	A container is used in OOP to transport data between objects. A container as a universal and public interface structure. Containers are used if the sending and receiving object methods cannot or do not want to use a dedicated data structure or the information passed may change from object method call to the next, e.g. if the container contains text lines or an IDoc structure.
Object	An object in the context of OOP is a collection of event, methods, properties etc. of a development that logically belong together. Mind the different contexts in which the expression object is used. An object can be a development calves, a function group, a data dictionary definition like a table or a data element, a single program etc. In OOP we use the expression in the dedicated meaning as an instance of an object type which adheres to a well defined formal specification.

Special Workflow Terminology

The terms listed here originate primarily from the workflow concept, however they may be used in another context as well. We first list the terms frequently used in workflow developments which refer to elements of classical object oriented programming.

Event coupling	To define in a customizing table which program shall react on a specific event
Workitem	A workitem is a special email sent to an actor. The workitem usually asks for a decision or a confirmation or asks an actor to signal when a manual task is finished
Agent	The workitem agent or actor is the recipient of the workitem email. It is the user or employee who has to make a manual decision to tell the workflow which branch to execute in a forking step.
Role	Actors can be assigned roles in the workflow environment. That is the step done in the organisation hierarchy. Roles are used to group several individuals who are assigned the same responsibilities, e.g. all the purchasers of a purchase group. The workflow engine uses this information to send the workitem mail to all of the actor with the same role and allows of them picks up the workitem and make the respective decision.

In order to understand workflow you have to know the following terms and their meanings.

Event

An event is a message to other program to take care for subsequent appropriate action

About Events, Firing, Burning and Consuming Them

Event	An event is a informational message which is broadcast by an application to other running applications to signal that the program reach a certain state or a special incident, e.g. an error has occurred.
Events are raised or fired	The action to broadcast an event is called to raise an event or to fire an event.
Event handlers are also called consumers and undeliverable events are burnt	A program which is executed after the raise of an event is called an event handler or a consumer of the event. The application that raises an event does normally not know who are the consumers of the event nor whether there are any at all. It raises the event and the rest is treated by the operating system whose responsibility it is

to deliver the event to all candidates. If there are no consumers found, we say that the event is burnt.

R/3 handles event by looking up entries in table SWETYPECOU

R/3 implements an event raise by calling a standard function module which does nothing more than to look in a special table SWETYPECOU whether there is one or more programs listed which have been told to react on the event.

There is also the concept of instance coupling

In addition to link a program firmly to an event it can be dynamically assigned to a special instance. This is not described here yet, because we have not seen cases which cold not be solved with plain type coupling.

About Organisation Hierarchies

The organisation hierarchy is defined in HR and is used to determine the receiver of the email when a workitem is created

The organization hierarchy is only used to determine a user who shall be responsible to process a workitem. A workitem is basically an email which asks a user – called the *agent* or *actor* – to make a decision or to perform a special task which cannot be done automatically.

A typical decision may be to decide whether a sales order can be confirmed to the customer and to complete the expected delivery date before the sales order confirmation document is printed

A typical task to perform may be to complete the data of a document, e.g. filling in the price or assign batch numbers for a material.

Myths About Workflow Developments

There are some rumours about the nature of workflow development we a cannot agree to. Although they describe the usual way of working and implementing a workflow there are alternatives which can and should be considered if your special situation do require it.

Workflow can be triggered by an event or by calling the handler directly

The first myth about workflow is, that it must be triggered by an event. While this is the recommended and good practice, the event handler can always be called directly by the triggering application. However, by doing so, you may loose one of the benefits of workflow development, namely the possibility to dynamically assign or detach handlers via customizing.

A handler needs not necessarily be programmed as an object method, although this is the recommended way

Although handlers are usually declared as object methods you can define workflow handlers as function modules. The restriction you have, is that the function module has to comply to a formal interface, i.e. you have to pass data via import and export parameters with predefined names.

4.2 Tasks in Workflows

A single action in a workflow chain is called a task. SAP distinguishes between a single step task and multiple-step task, which are a sequence of single-step tasks to be executed one after the other.

Task refer to a business object and execute a method of the referred business object

A task is merely a wrapper around an existing business object. In other words, the task is not a program but rather a description that tells the workflow handler which business object is manipulated and which method of the business object has to executed to perform the required action and which event has to be fired when the task has finished (termination event if ended successfully or an error event if ended unsuccessfully).

Single step tasks

Single step tasks are elementary actions built from

- A business object which establishes the object context
- A method to be executed when the task is due
- A message template which can be sent to a user to provide information about the circumstances the task is executed and allow the user to make a decision whether and how to execute the task action
- A set of events which are triggered according the state of execution of the

method

- A container with variables and values which may be needed during the task execution or with information for a human operator

Single-step tasks describe elementary business activities from an organizational viewpoint. These activities refer to one object method.

Multi step tasks are a sequence of single step tasks

Multi step tasks are several task which are executed sequentially. The steps of a multi step task can be executed conditionally, i.e. you may define if-conditions to decide whether the steps should be executed or skipped. A workflow is effectively a multi step tasks and thus both expressions are used equivalently.

Standard Task or Customer Task

SAP provides two task types, the standard tasks and the customer tasks. There is no principle difference between the both. The main difference is, that customer tasks can be client specific and that they may be timely limited.

Figure 2: The table below shows the differences between standard tasks and customer tasks.

Standard task (Tasks T …)	Customer task (Tasks TS…)
Cross-client	Client-specific
Any plan version	Plan version-specific
No validity period For technical reasons, standard tasks are defined with a maximum validity period of 1.1.1900 - 31.12.9999.	Validity period
SAP supplies standard tasks which are available in the customer systems.	

⇨

4.3 SWO1: Defining A New Object Method

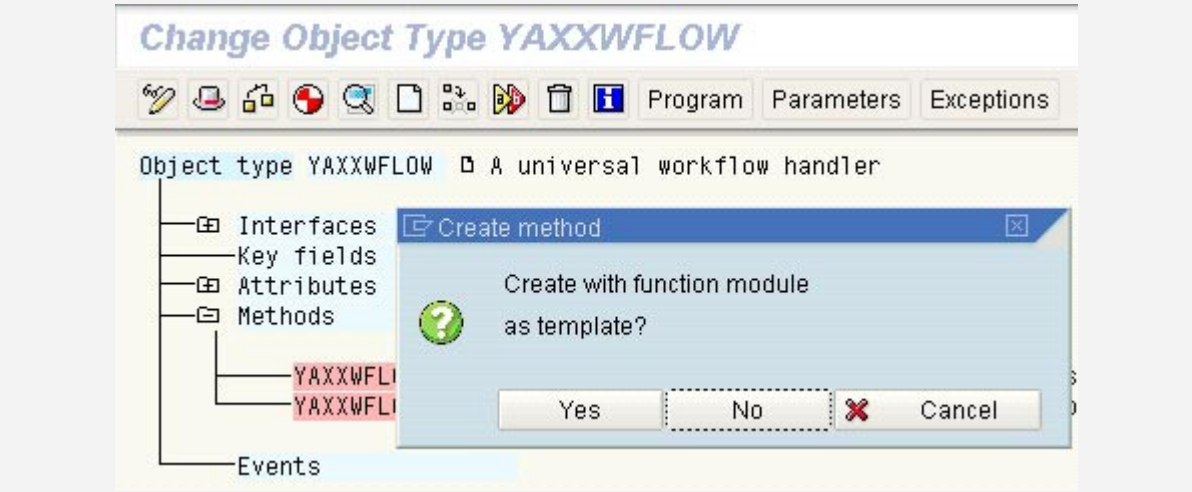
The chapter shows step by step how to define a new method for an object by means of a sequence of commented screen shots.

We will assume that we already own a user written function module Z_BTCI_VA02_TEST that we have created from a BCI recording. Because an object method is simply a standard interface to call programs, we will create the method with appropriate interface parameters and then call the function module from within the method. In terms of OOP we will inherit the function modules functionality to the object method.

R/3 offers you to create a method automatically by using an existing function module as a template

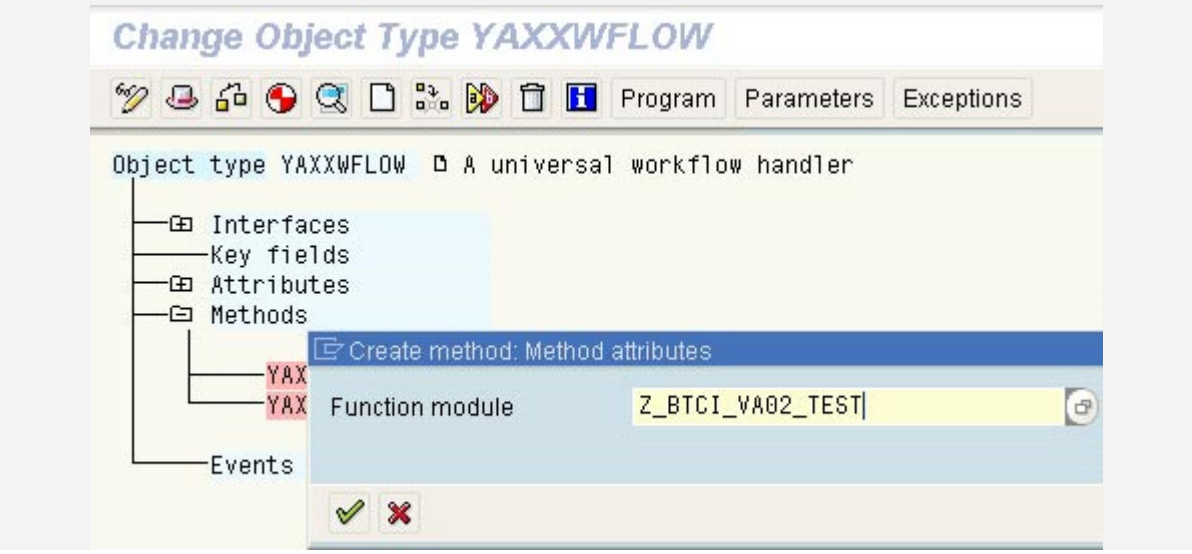
The example will create a new method, which calls an existing SAP R/3 function module. Many object methods are actually nothing than a wrapper for existing programs and functions. Therefore R/3 offers you to create a template code from the definition of an existing function module. When you entered a functions name, it is the easiest way to create a methods coding. Once you have a frame, you can easily adapt and modify it according to your needs.

Figure 3: Create a new method by using a function module as template



⇨

Figure 4: Creating a new univesal object with SWO1



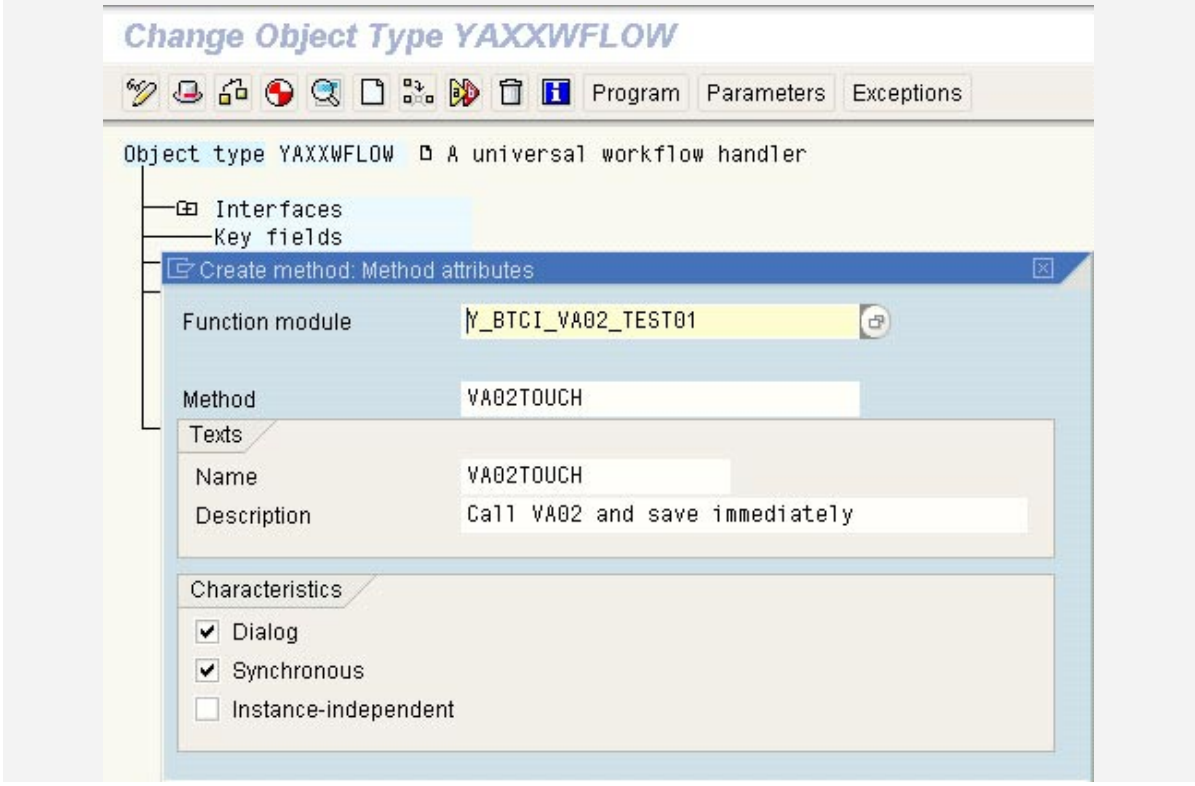
⇨

Choose same names for technical and long name of the object

There is a lot of confusion caused by the fact, that SAP uses two names, a technical name and a long name for a method. E.g. the SalesOrder object has the technical name BUS2032. We advise you to choose identical names for the object technical name, the object long name and also for the name of the ABAP to store the object coding. The reason for this advise is, that the tools to search for objects, will look for the long name only, while you always need the object's technical name in your programming and workflow definitions.

280

Figure 5: Setting the method name and characteristics

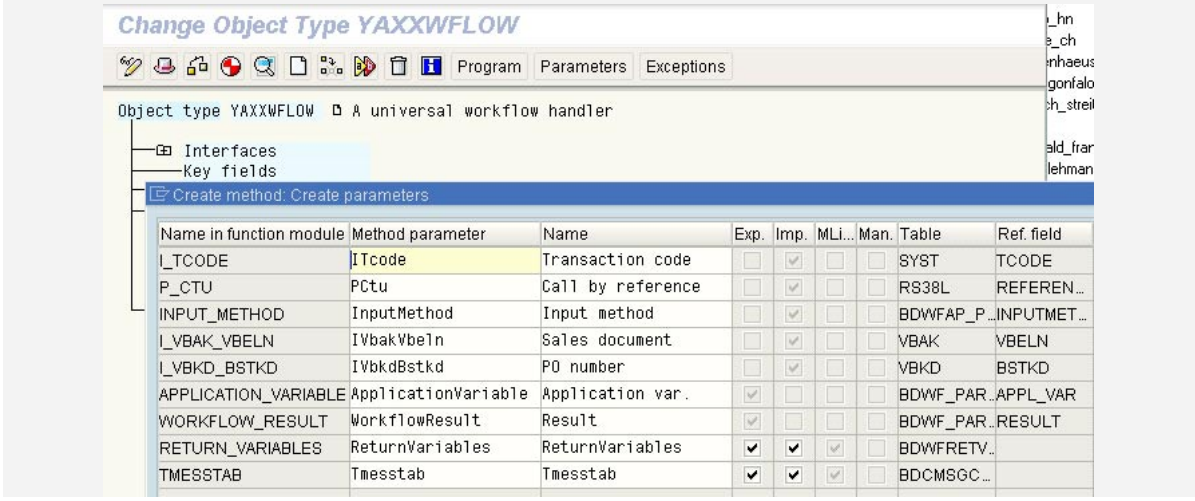


⇨

Take over the parameters and revise or delete them later

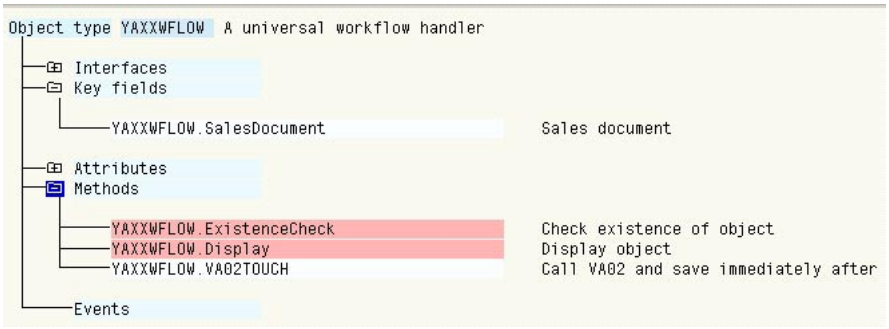
You can take over the proposed parameters and revise them later properly. Usually you will be proposed the maximum number of parameters, while you need probably only a few of them or none at all. Every object has also an object key, which is not part of the method parameter, but belongs to the entire object. E.g. the sales order number is the key of the BUS2032 (sales order) object. For many methods, this number will be all you need to execute it properly.

Figure 6: Setting the method name and characteristics



⇨

Figure 7: After generating the object the newly defined method is ready to test


Object type YAXXWFLOW A universal workflow handler
Interfaces
Key fields
YAXXWFLOW.SalesDocument
Sales document
Attributes
Methods
YAXXWFLOW.ExistenceCheck
Check existence of object
YAXXWFLOW.Display
Display object
YAXXWFLOW.VA02TOUCH
Call VA02 and save immediately after
Events

⇨

Listing 1: Implementation of method VA02TOUCH.txt

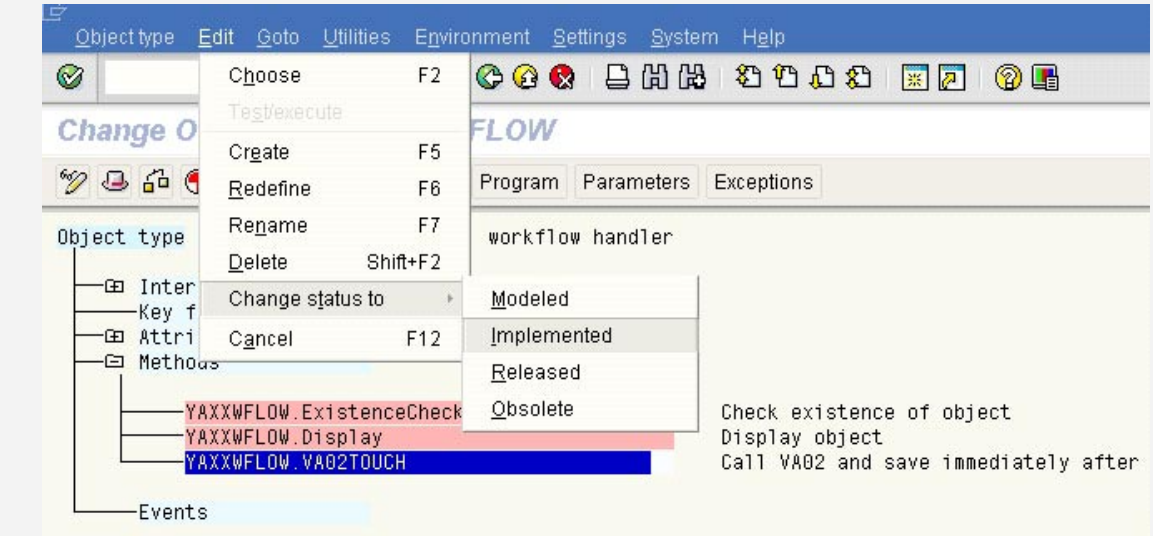
```
*****           Implementation of object type YAXXWFLOW            *****
INCLUDE <OBJECT>.
BEGIN_DATA OBJECT. " Do not change.. DATA is generated
* only private members may be inserted into structure private
DATA:
" begin of private,
"   to declare private attributes remove comments and
"   insert private attributes here ...
" end of private,
  BEGIN OF KEY,
      SALESDOCUMENT LIKE VBAK-VBELN,
  END OF KEY.
END_DATA OBJECT. " Do not change.. DATA is generated

BEGIN_METHOD VA02TOUCH CHANGING CONTAINER.
DATA:
      INPUTMETHOD LIKE BDWFAP_PAR-INPUTMETHD,
      IVBAKVBELN LIKE VBAK-VBELN,
      IVBKDBSTKD LIKE VBKD-BSTKD,
      APPLICATIONVARIABLE LIKE BDWF_PARAM-APPL_VAR,
      WORKFLOWRESULT LIKE BDWF_PARAM-RESULT,
      RETURNVARIABLES LIKE BDWFRETVAR OCCURS 0 WITH HEADER LINE,
      TMESSTAB LIKE BDCMSGCOLL OCCURS 0 WITH HEADER LINE.

* let it to 'A' for testing, but must be set to 'N' for true Workflows
  INPUTMETHOD = 'A'.
  MOVE OBJECT-KEY TO IVBAKVBELN.

* This is a dummy text to write to the salesorder to see a change
  CONCATENATE 'Touched' SY-DATUM SY-UZEIT INTO IVBKDBSTKD.

  CALL FUNCTION 'Y_BTCI_VA02_TEST01'
    EXPORTING
      I_VBKD_BSTKD = IVBKDBSTKD
      I_VBAK_VBELN = IVBAKVBELN
      INPUT_METHOD = INPUTMETHOD
    IMPORTING
      WORKFLOW_RESULT = WORKFLOWRESULT
      APPLICATION_VARIABLE = APPLICATIONVARIABLE
    TABLES
      RETURN_VARIABLES = RETURNVARIABLES
      TMESSTAB = TMESSTAB
    EXCEPTIONS
      OTHERS = 01.
  CASE SY-SUBRC.
    WHEN 0.            " OK
    WHEN OTHERS.       " to be implemented
  ENDCASE.

* Putting the result values into the container
  SWC_SET_ELEMENT CONTAINER 'ApplicationVariable' APPLICATIONVARIABLE.
  SWC_SET_ELEMENT CONTAINER 'WorkflowResult' WORKFLOWRESULT.
  SWC_SET_TABLE CONTAINER 'ReturnVariables' RETURNVARIABLES.
  SWC_SET_TABLE CONTAINER 'Tmesstab' TMESSTAB.
* Raise an exception event if workflow result is not zero.
  EXIT_RETURN WORKFLOWRESULT
   TMESSTAB-MSGV1 TMESSTAB-MSGV2 TMESSTAB-MSGV3 TMESSTAB-MSGV4.
END_METHOD.
```

⇨

Do not forget to set the object and method status to implemented or higher

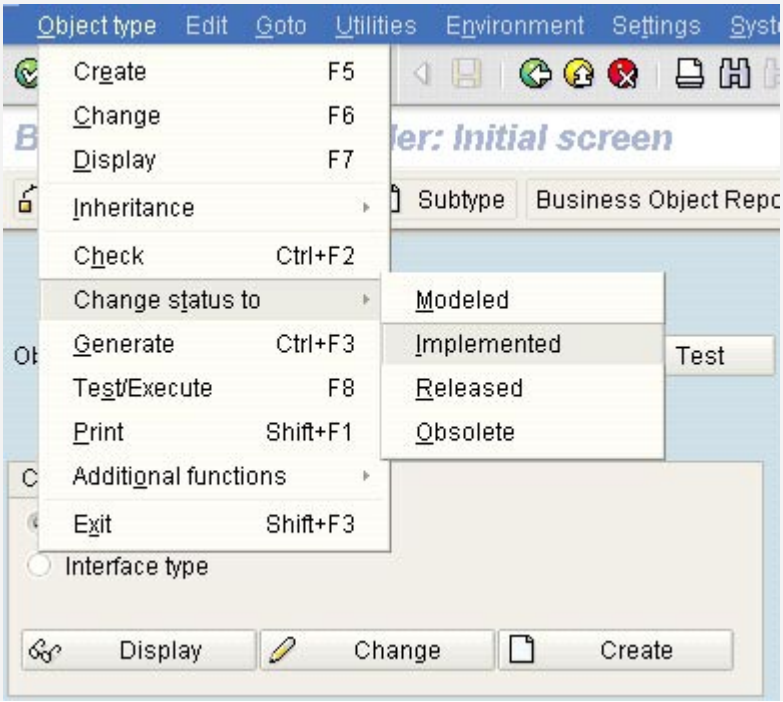
Both objects and methods have a status attribute. They serve mainly the same purpose like the activation and deactivation of function groups. Before you can use an object or a method, you have to set the status of both of them to at least *implemented*.

Figure 8: The status of the method must be set to implemented or higher



⇨

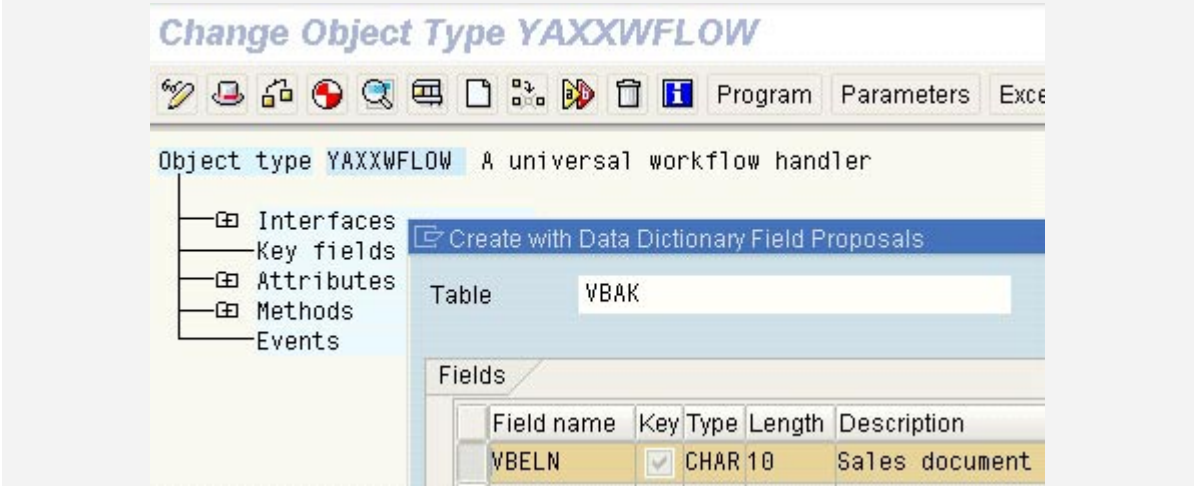
Figure 9: The status of the object itself must also be set to implemented or higher



⇨

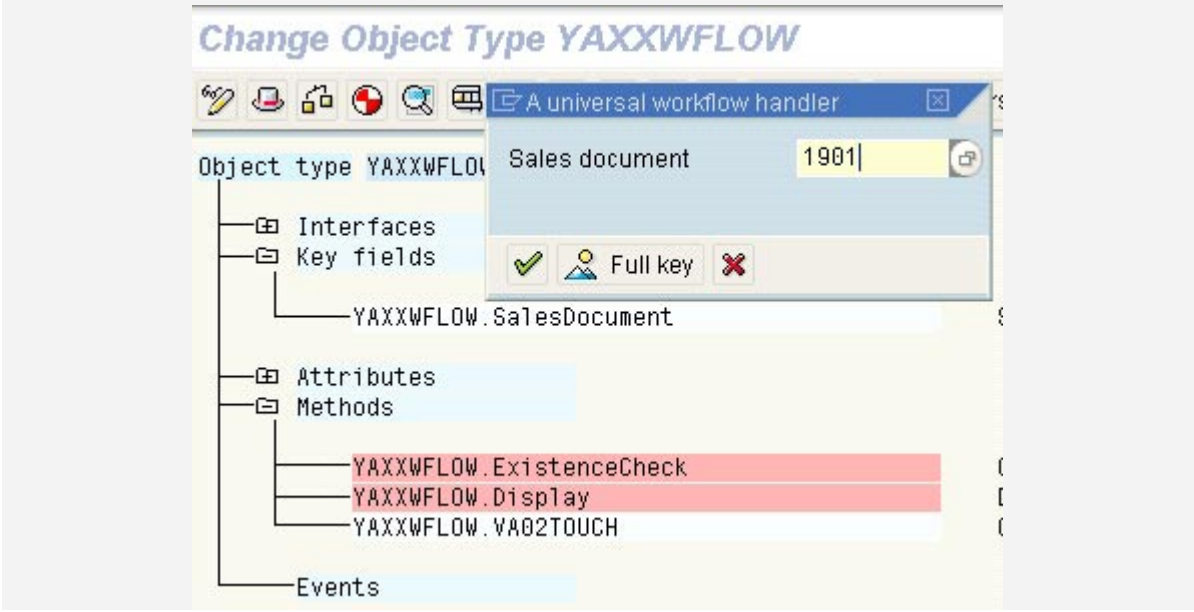
Title	Explanation

Figure 10: An object needs to have at least one key field which can reference a DDic object



Title	Explanation

Figure 11: Now the method can be tested with the test tool



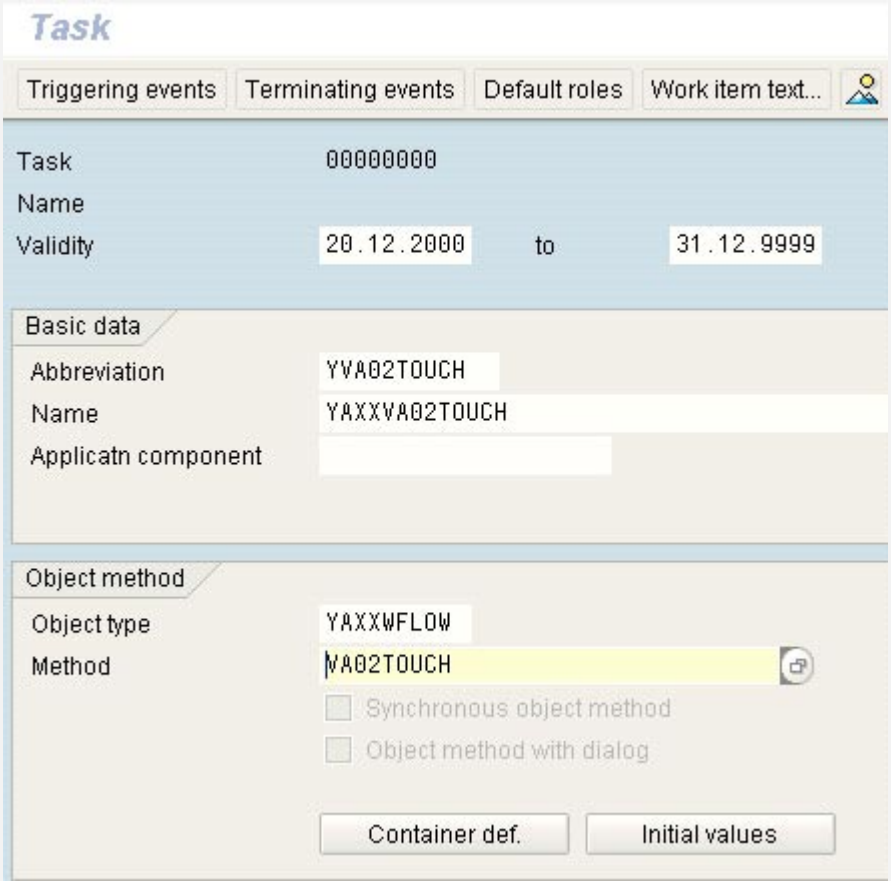
⇨

Object methods need to be wrapped into a workitem task to be used in a workflow

300

To make use of an object method in a workflow, it needs to be wrapped into a *workitem task*. This is done by transaction PFTC. Workitem tasks are actually workflow steps, which combine an email text and an executable object method. The email text comes only in action, if there is a dialog task. In that case, every time a task is addressed, it will send an email to an appropriate agent and wait, until the user reads and activates it. Then the associated object method will be executed. Non-dialog tasks simply bypass the email part and execute the method immediately. Non-dialog tasks are thus only a standardized wrapper for object methods.

Table 1: Creating a task with PFTC which makes use of the VA02TOUCH method



⇨

4.4 SWO1: Deriving A Delegation Object Subtype

According to the principles of OOP you can add and overwrite methods and events of a predefined object type. SAP calls these derived type: Subtypes. You can declare such a subtype as a delegation type which will substitute the original object throughout the system.

In order to enhance the functionality of an object or add additional methods you can define a subtype

Subtypes are derived object types which inherent the complete functionality and interface structure of its base type. SAP needs subtypes because it cannot allow the modification of the SAP owned base types in a client system. Object types are development object in SAP's name space. When the modify them you need a repair key for this object. SAP introduced the subtypes to bypass the problem. Subtypes are defined in the customer name space, i.e. they have to start with Z… or Y.. and are declared as child type to its standard parent object.

310

In order to invoke the new object sub type it has to be called by the application which usually requires to change the existing call for the new object subtype

Now you encounter a serious problem, when you want to invoke the new subtype. Most application fire there events by specifying the object name directly within the application. In order to allow the new subtype to be called instead of the original base type object, you would have to change all the programs that invoke the object. E.g. a sales order identifies itself as object BUS2032. If you want to use the methods of a newly defined replacement object ZZBUS2032 you have to change the module SAPMV45A to fire an event for ZZBUS2032 instead for BUS2032.

Every invoked object is automatically replaced by its delegation type, if such one is defined

To circumvent this difficulty SAP invented the concept of delegation types. A *delegation type* is assigned as a deputy object to the base type. E.g. you tell the system that ZZBUS2032 shall take over the role of BUS2032. Every time when the object BUS2032 is invoked, the object handler checks the customizing for a

325

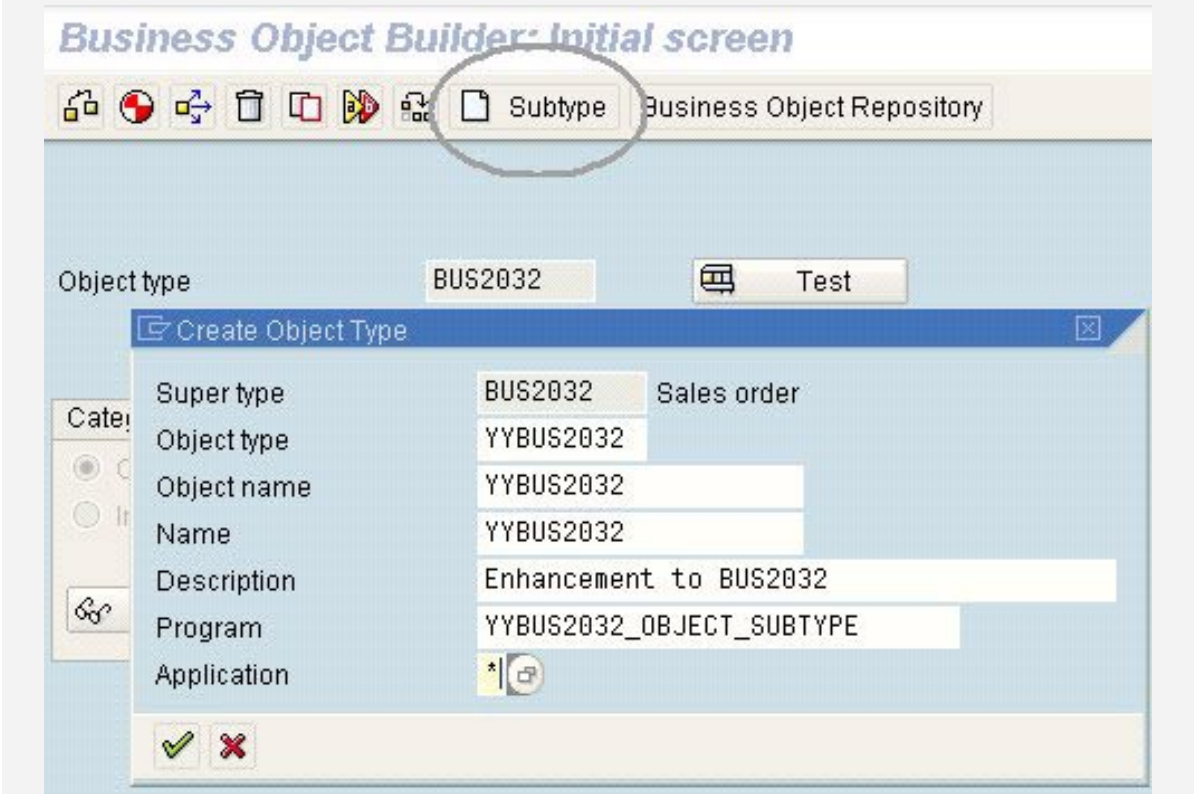
delegation type and if found, redirects all base type calls to the delegation type. E.g. if an application asks to fire the event BUS2032->EventCreated the handler will transparently and automatically invoke the event ZZBUS2032->EventCreated.

Subtypes are created like any custom object type with the exception that they have the field supertype filled in with Example to make ZZBUS2032 a delegation type for BUS2032

There is now special deal with the creation of sub types. They are defined using SWO1 like any object type. The only difference is, that you have to specify the supertype's name when you create the sub type. As an effect your new subtype inherits all the properties and methods of its parent.

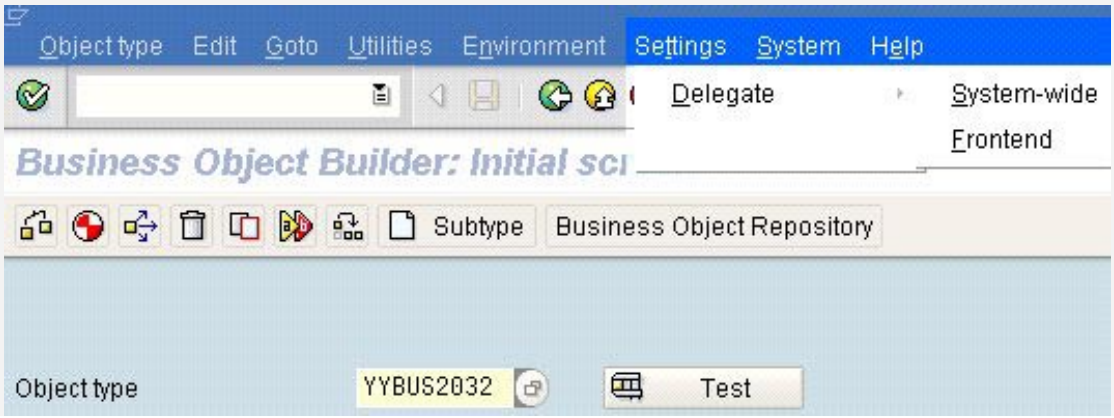
The following slide show demonstrates how to declare a subtype as delegation type for Explanation

Figure 12: Create a subtype to enhance an existing read-only SAP standard object type



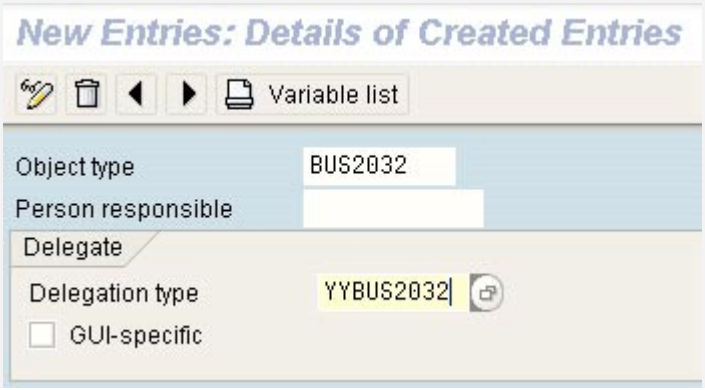
⇨

Figure 13: Make the subtype a delegation type for its supertype



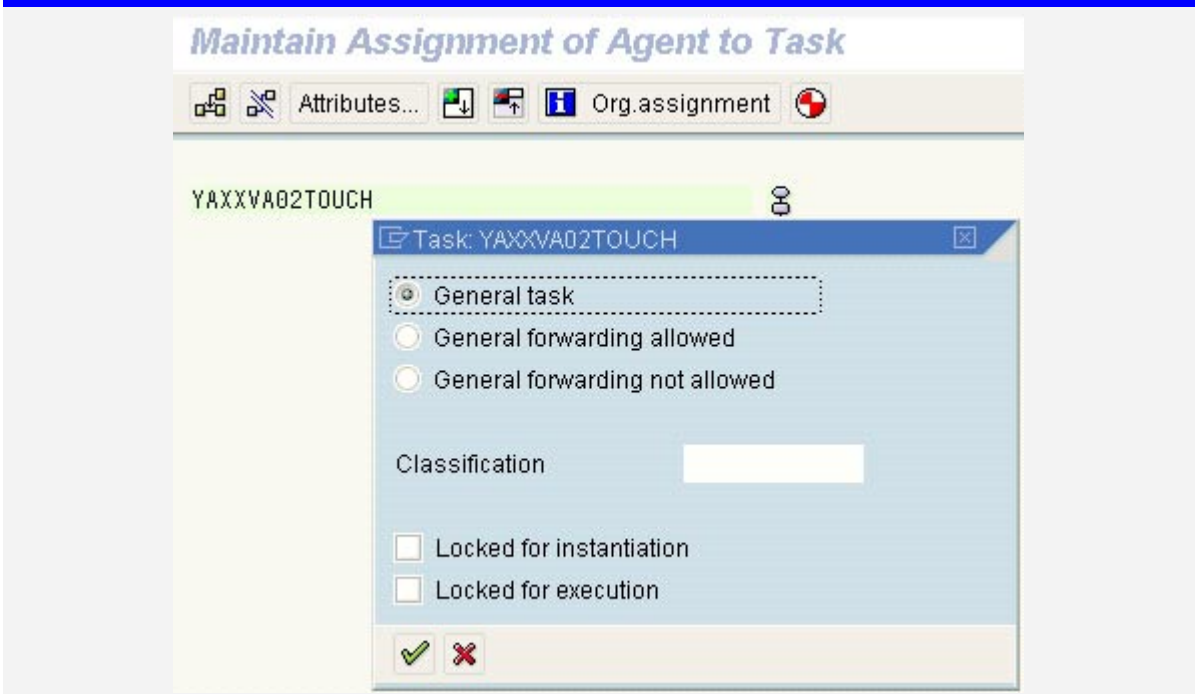
⇨

Figure 14: Set the delegation type which will execute in place of its supertype



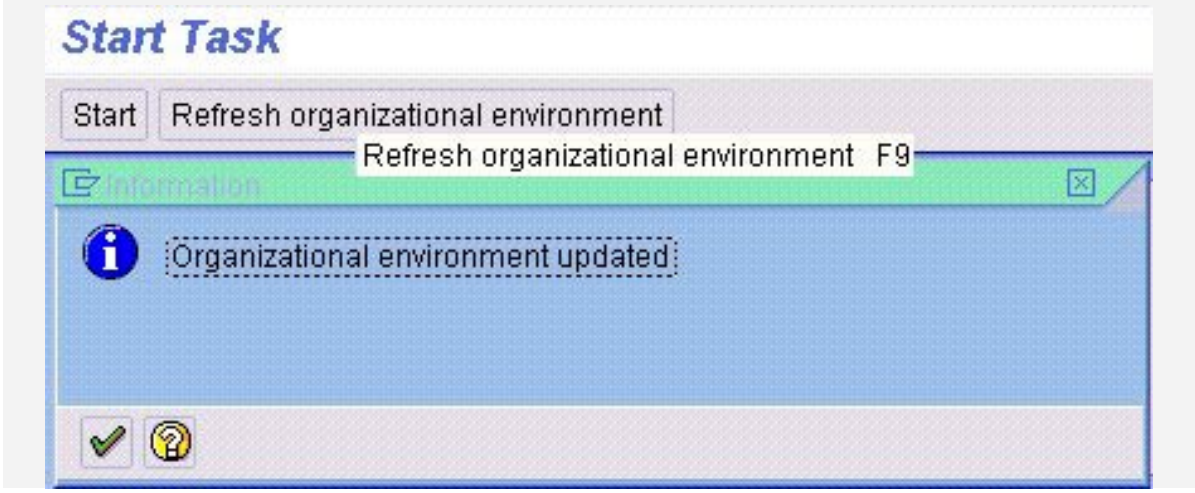
4.5 Defining The Class Of People To Process A Workitem

Figure 15: To allow everybody to execute a task you have to set the General Task flag



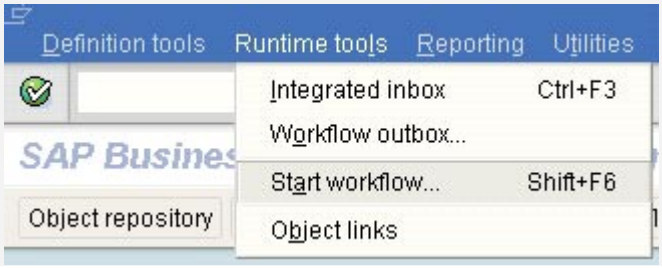
⇨

Figure 16: Caution, bug! After changing the general task attribute you should refresh the organisational environment



⇨

Figure 17: You can test your task via the Start Workflow tool in SWO1



⇨

4.6 Creating Workflow Events Via Message Control

You can easily trigger any event of a business object by means of message control and media type 9. It is not possible to raise events for an external object.

Messages lets you define conditions when events are fired

R/3 message controls a standard processing routine found in ABAP RSWEMC01 that raises a predefined event when the message is processed. This allows you to customise the workflow event creation the same way as you set up message processing for SapScript or IDocs. When the message is defined with processing time set to *immediate processing* this has pretty much the same effect than the direct firing of an event by the application. The advantage of message are, that you can define condition rules that allow the firing of an event under certain circumstances only, e.g. when the document is complete or goods are issued.

New events have to be assigned to a delegation type

Every R/3 business object has a couple of standard events defined. E.g. the sales order business object BUS2032 has standard events like *changed* or *created*. To raise additional events you have to add it to the business object. For R/3 standard objects you can add events only by creating a subtype and making the subtype a delegation type for the parent object.

Example how to add new events to a delegation subtype of a business object

E.g. you want to signal the event *OrderBlockedNoCredit* to raise appropriate action when the credit approval for the sales order failed. To define an event for the standard business object BUS2032 you have to create a subtype for BUS2032, let us give it the name ZZBUS2032. The transaction to define subtypes can be accessed via SWO1. A subtype is a derived type and inherits all the object characteristics from its parent. This subtype then is to be declared as a delegation type for its parent object.

Finally you can add the event to the message control

Now, that the event exists you can specify it on the message detail screen of the message to be triggered. This is done the same way as a printer is specified for print messages or the fax number is specified for fax messages. Check out transaction VV12 for sales orders, VV22 for deliveries for examples.

Standard NAST processing routine CREATE_EVENT(RSWEMC01)

SAP provides a simple processing routine `create_event` stored in ABAP `RSWEMC01`. This one does a simple `CALL FUNCTION 'SWE_EVENT_CREATE'` for the specified event and the object type of the application.

Listing 2: SAP R/3 standard handler to create an event from NAST entry

```
REPORT RSWEMC01.
INCLUDE <CNTAIN>.
TABLES: NAST, T681Z.

*---------------------------------------------------------------------*
*       FORM CREATE_EVENT                                             *
*---------------------------------------------------------------------*
FORM CREATE_EVENT USING RETURNCODE
                        US_SCREEN.
  INCLUDE RSWUINCL.
  DATA: L_EVENT_CREATOR LIKE SWHACTOR .
  DATA: L_OBJKEY LIKE SWEINSTCOU-OBJKEY.
  DATA: L_EVENTID LIKE SWEDUMEVID-EVTID.
*
  IF NAST-EVENT EQ SPACE OR NAST-OBJTYPE EQ SPACE.
      CLEAR: SYST-MSGID, SYST-MSGNO, SYST-MSGTY, SYST-MSGV1,
             SYST-MSGV2, SYST-MSGV3, SYST-MSGV4.
      SYST-MSGID = 'VN'.
      SYST-MSGTY = 'E'.
      SYST-MSGNO = 075.
      SYST-MSGV1 = NAST-EVENT.
      SYST-MSGV2 = NAST-OBJTYPE.
      CALL FUNCTION 'NAST_PROTOCOL_UPDATE'
           EXPORTING
                MSG_ARBGB              = SYST-MSGID
                MSG_NR                 = SYST-MSGNO
                MSG_TY                 = SYST-MSGTY
                MSG_V1                 = SYST-MSGV1
                MSG_V2                 = SYST-MSGV2
                MSG_V3                 = SYST-MSGV3
                MSG_V4                 = SYST-MSGV4
           EXCEPTIONS
                MESSAGE_TYPE_NOT_VALID = 01
                NO_SY_MESSAGE          = 02.
      RETURNCODE = 4.
  ELSE.
    SWC_CONTAINER L_CONT.
    L_EVENT_CREATOR-OTYPE = ORG_OBJTYPE_USER .      "global in RSWUINCL
    L_EVENT_CREATOR-OBJID = SY-UNAME.               "hopefully sy-uname
                                                    "does exist!
* Ereignis absetzen
    L_OBJKEY = NAST-OBJKY.

    IF NAST-KAPPL EQ 'MR' AND NAST-OBJKY(4) EQ '$$$$'.
      L_OBJKEY = NAST-OBJKY+4.
    ENDIF.

    CALL FUNCTION 'SWE_EVENT_CREATE'
         EXPORTING
              OBJTYPE           = NAST-OBJTYPE
              OBJKEY            = L_OBJKEY
              EVENT             = NAST-EVENT
              CREATOR           = L_EVENT_CREATOR
         IMPORTING
              EVENT_ID          = L_EVENTID
         TABLES
              EVENT_CONTAINER   = L_CONT
         EXCEPTIONS
              OBJTYPE_NOT_FOUND = 01.
* Ereignis konnte nicht erzeugt werden, da Objkttyp nicht gefunden
    IF SY-SUBRC <> 0.
      CLEAR: SYST-MSGID, SYST-MSGNO, SYST-MSGTY, SYST-MSGV1,
             SYST-MSGV2, SYST-MSGV3, SYST-MSGV4.
      SYST-MSGID = 'VN'.
      SYST-MSGTY = 'E'.
      SYST-MSGNO = 074.
      SYST-MSGV1 = T681Z-OJ_NAME.
      CALL FUNCTION 'NAST_PROTOCOL_UPDATE'
           EXPORTING
                MSG_ARBGB              = SYST-MSGID
                MSG_NR                 = SYST-MSGNO
                MSG_TY                 = SYST-MSGTY
```

```
                MSG_V1                  = SYST-MSGV1
                MSG_V2                  = SYST-MSGV2
                MSG_V3                  = SYST-MSGV3
                MSG_V4                  = SYST-MSGV4
           EXCEPTIONS
                MESSAGE_TYPE_NOT_VALID = 01
                NO_SY_MESSAGE          = 02.
     RETURNCODE = 4.
   ELSEIF NOT L_EVENTID IS INITIAL.
     RETURNCODE = 0.
   ELSE.
     CLEAR: SYST-MSGID, SYST-MSGNO, SYST-MSGTY, SYST-MSGV1,
            SYST-MSGV2, SYST-MSGV3, SYST-MSGV4.
     SYST-MSGID = 'VN'.
     SYST-MSGTY = 'E'.
     SYST-MSGNO = 355.
     SYST-MSGV1 = NAST-EVENT.
     SYST-MSGV2 = NAST-OBJTYPE.
     CALL FUNCTION 'NAST_PROTOCOL_UPDATE'
          EXPORTING
                MSG_ARBGB               = SYST-MSGID
                MSG_NR                  = SYST-MSGNO
                MSG_TY                  = SYST-MSGTY
                MSG_V1                  = SYST-MSGV1
                MSG_V2                  = SYST-MSGV2
                MSG_V3                  = SYST-MSGV3
                MSG_V4                  = SYST-MSGV4
          EXCEPTIONS
                MESSAGE_TYPE_NOT_VALID = 01
                NO_SY_MESSAGE          = 02.
     RETURNCODE = 4.
   ENDIF.
  ENDIF.
ENDFORM.
```

⇨

4.7 Defining Subsequent Actions Via Message Control

You can trigger a workflow by calling the handler from the handler routine which processes a message, which had been created by the SAP R/3 message. The handler is a program with a well-defined interface that is called when the message is due to be processed. Using messages allows you usually to circumvent a modification of standard transactions.

Most SAP R/3 applications have an interface to the standard message control. Message control determines messages according conditions defined in the table. When the application quits, it stores all appropriately found messages in table NAST along with matching processing hints. Message control is regularly used to
370 trigger print output via SapScript or to send out IDocs. But messages allow also to perform any arbitrary action by defining special processing routines.

Figure 18: List of most importing processing media types

Sample routines are listed in the appendix	What it does	Example processing routine
1	Print via SapScript	ENTRY(ZSNASTPR)
2	Fax via SapScript	ENTRY(ZSNASTPR)
6	EDI	EDI_PROCESSING(RSNASTED)
7	SAPoffice	
8	Special function	ENTRY_TEST(ZSNASTWF)
9	Events (SAP Business Workflow)	CREATE_EVENT(RSWEMC01) CREATE_EVENT(ZSNASTWF)
A	Distribution (ALE)	ALE_PROCESSING(RSNASTED)
T	Tasks (SAP Business Workflow)	

Please note, that the programs starting with Z… are not part of the standard R/3 systems. They can be downloaded from http://logosworld.com . In the appendix

we list a couple of sample handling routines for the different media types. Some
375 of them are described in detail in the following chapters.

Figure 19: Sample NAST processing routines to be used for own enhancements

```
************************************************************************
* Collection of NAST processing routines                               *
*  for media = 8 : special processing                                  *
*  for media = 9 : Workflow Event                                      *
*  for media = T : Workflow Task                                       *
*                                                                      *
************************************************************************

* <object> contains call declarations for object handling
INCLUDE <OBJECT>.
* Following are the declarations for standard NAST
INCLUDE RVADTABL .
DATA: RETCODE LIKE SY-SUBRC.
DATA: XSCREEN.
DATA: XNAST LIKE NAST OCCURS 0 WITH HEADER LINE.
*---------------------------------------------------------------------*
PARAMETERS: P_KAPPL  LIKE NAST-KAPPL MEMORY ID Z01.
PARAMETERS: P_KSCHL  LIKE NAST-KSCHL MEMORY ID Z03.
PARAMETERS: P_OBJKY  LIKE NAST-OBJKY MEMORY ID Z02.
PARAMETERS: P_SPRAS  LIKE NAST-SPRAS.
PARAMETERS: P_PARNR  LIKE NAST-PARNR.
PARAMETERS: P_PARVW  LIKE NAST-PARVW.
PARAMETERS: P_ERDAT  LIKE NAST-ERDAT.
PARAMETERS: P_ERUHR  LIKE NAST-ERUHR.
*---------------------------------------------------------------------*

*---------------------------------------------------------------------*
AT SELECTION-SCREEN.
*---------------------------------------------------------------------*

*---------------------------------------------------------------------*
INITIALIZATION.
*---------------------------------------------------------------------*
*---------------------------------------------------------------------*
START-OF-SELECTION.
*---------------------------------------------------------------------*
  WRITE: / 'Start processing NAST record', SY-DATUM, SY-UZEIT, SY-UNAME.
  NEW-LINE.
  WRITE: P_OBJKY.
  WRITE: P_KAPPL.
  WRITE: P_KSCHL.
  WRITE: P_SPRAS.
  WRITE: P_PARNR.
  WRITE: P_PARVW.
  WRITE: P_ERDAT.
  WRITE: P_ERUHR.
  IF SY-BATCH NE SPACE.
    WRITE: / 'Batch processing detected'.
    SELECT SINGLE * FROM NAST WHERE OBJKY EQ P_OBJKY
                                AND KAPPL EQ P_KAPPL
                                AND KSCHL EQ P_KSCHL
                                AND SPRAS EQ P_SPRAS
                                AND PARNR EQ P_PARNR
                                AND PARVW EQ P_PARVW
                                AND ERDAT EQ P_ERDAT
                                AND ERUHR EQ P_ERUHR.
  ELSE.
    SELECT SINGLE * FROM NAST WHERE OBJKY EQ P_OBJKY
                                AND KAPPL EQ P_KAPPL
                                AND KSCHL EQ P_KSCHL.
   BREAK ANGELIAX.
  ENDIF.
  IF SY-SUBRC EQ 0.
    WRITE: / 'PERFORM einzelnachricht(rsnast00)'.
    PERFORM EINZELNACHRICHT(RSNAST00) USING RETCODE.
    WRITE: / 'End processing NAST record', SY-DATUM, SY-UZEIT, SY-UNAME.
  ELSE.
    WRITE: / 'NAST Entry not found. Stop processing.'.
  ENDIF.

*---------------------------------------------------------------------*
*       FORM CREATE_EVENT                                             *
```

```
*---------------------------------------------------------------------*
*  Routine is used to process NAST media type "9" (Workflow Event)    *
*  It will actually raise the specified workflow event
*  as defined in NAST-OBJECT and NAST-EVENT
*  Note: NAST-EVENT must exist for object NAST-OBJECT
*        or its delegation type (=derived subtype)
*---------------------------------------------------------------------*
*  -->  RETURN_CODE                                                   *
*  -->  US_SCREEN                                                     *
*---------------------------------------------------------------------*
FORM ENTRY_CREATE_EVENT USING RETURN_CODE US_SCREEN.
  PERFORM CREATE_EVENT(RSWEMC01) USING RETURN_CODE US_SCREEN.
ENDFORM.                               " create_event.
*---------------------------------------------------------------------*
*       FORM ENTRY_VA02TOUCH                                          *
*---------------------------------------------------------------------*
*  Routine is a sample handler which will call a transaction.
*  This special routine call VA02.
*  The speciality here is, taht it can handle immediate processing
*  (NAST-VSZTP = 4) by submitting the processing in a background task
*  The submit in background task is done by
*       CALL FUNCTION 'Y_NAST_PROCESSING' IN BACKGROUND TASK
*  Of course the function has to be defined.
*---------------------------------------------------------------------*
*  -->  RETURN_CODE                                                   *
*  -->  US_SCREEN                                                     *
*---------------------------------------------------------------------*
FORM ENTRY_VA02TOUCH USING RETURN_CODE US_SCREEN.
  CLEAR RETCODE.
  XSCREEN = US_SCREEN.
  BREAK ANGELIAX.
  IF NAST-VSZTP EQ '4'.
*    PERFORM submit_in_backgroundtask USING nast.
    PERFORM SUBMIT_TO_SPOOL USING NAST.
    RETURN_CODE = 3.
  ELSE.
    PERFORM DO_VA02TOUCH USING US_SCREEN NAST-OBJKY.
    CASE RETCODE.
      WHEN 0.
        RETURN_CODE = 0.               "all well done, sets VSTAT = 1
      WHEN 3.
        RETURN_CODE = 3.  "means not processed, leaves VSTAT = 0
      WHEN OTHERS.
        RETURN_CODE = 1.               "Errors, sets VSTAT = 2
    ENDCASE.
  ENDIF.
ENDFORM.

*---------------------------------------------------------------------*
*       FORM PROCESSING                                               *
*---------------------------------------------------------------------*
*       ........                                                      *
*---------------------------------------------------------------------*
FORM DO_VA02TOUCH USING USCREEN PVBELN.
  PERFORM PROTOCOL_UPDATE USING '38' '000' 'I'
      'do_va02touch called' SY-REPID SY-TCODE SY-CALLD.
  RETCODE = 0.
ENDFORM.                               "processing.

*---------------------------------------------------------------------*
*       FORM SUBMIT_IN_BACKGROUNDTASK                                 *
*---------------------------------------------------------------------*
*       ........                                                      *
*---------------------------------------------------------------------*
*  -->  PNAST                                                         *
*---------------------------------------------------------------------*
FORM SUBMIT_IN_BACKGROUNDTASK USING PNAST LIKE NAST.
  NAST = PNAST.
  NAST-VSZTP = '1'.
  CALL FUNCTION 'Y_NAST_PROCESSING' IN BACKGROUND TASK
       EXPORTING
            MSG_NAST = NAST.
ENDFORM.                               "submit_in_backgroundtask
```

```
*---------------------------------------------------------------------*
*       FORM SUBMIT_TO_SPOOL                                          *
*---------------------------------------------------------------------*
*       ........                                                      *
*---------------------------------------------------------------------*
*  -->  PNAST                                                         *
*---------------------------------------------------------------------*
FORM SUBMIT_TO_SPOOL USING PNAST LIKE NAST.
  DATA: TBTCJOB LIKE TBTCJOB.
  NAST = PNAST.
  NAST-VSZTP = '1'.

  TBTCJOB-JOBNAME = 'YSNASTWF'.
  CALL FUNCTION 'JOB_OPEN'
       EXPORTING
            JOBNAME          = TBTCJOB-JOBNAME
       IMPORTING
            JOBCOUNT         = TBTCJOB-JOBCOUNT
       EXCEPTIONS
            CANT_CREATE_JOB  = 1
            INVALID_JOB_DATA = 2
            JOBNAME_MISSING  = 3
            OTHERS           = 4.

  SUBMIT YSNASTWF VIA JOB TBTCJOB-JOBNAME NUMBER TBTCJOB-JOBCOUNT
   AND RETURN TO SAP-SPOOL KEEP IN SPOOL 'X' WITHOUT SPOOL DYNPRO
   WITH  P_OBJKY = NAST-OBJKY
   WITH  P_KAPPL = NAST-KAPPL
   WITH  P_KSCHL = NAST-KSCHL
   WITH  P_SPRAS = NAST-SPRAS
   WITH  P_PARNR = NAST-PARNR
   WITH  P_PARVW = NAST-PARVW
   WITH  P_ERDAT = NAST-ERDAT
   WITH  P_ERUHR = NAST-ERUHR.

   CALL FUNCTION 'JOB_CLOSE'
        EXPORTING
             JOBCOUNT                    = TBTCJOB-JOBCOUNT
             JOBNAME                     = TBTCJOB-JOBNAME
*       IMPORTING
*            JOB_WAS_RELEASED            =
        EXCEPTIONS
             CANT_START_IMMEDIATE        = 1
             INVALID_STARTDATE           = 2
             JOBNAME_MISSING             = 3
             JOB_CLOSE_FAILED            = 4
             JOB_NOSTEPS                 = 5
             JOB_NOTEX                   = 6
             LOCK_FAILED                 = 7
             OTHERS                      = 8.

ENDFORM.                               "submit_in_backgroundtask
*---------------------------------------------------------------------*
*       FORM PROTOCOL_UPDATE                                          *
*---------------------------------------------------------------------*
*       ........                                                      *
*---------------------------------------------------------------------*
FORM PROTOCOL_UPDATE USING
            MSG_ARBGB
            MSG_NR
            MSG_TY
            MSG_V1
            MSG_V2
            MSG_V3
            MSG_V4.
  CHECK XSCREEN = SPACE.
  SYST-MSGID  = MSG_ARBGB.
  SYST-MSGNO  = MSG_NR.
  SYST-MSGTY  = MSG_TY.
  SYST-MSGV1  = MSG_V1.
  SYST-MSGV2  = MSG_V2.
  SYST-MSGV3  = MSG_V3.
  SYST-MSGV4  = MSG_V4.
```

```
  CALL FUNCTION 'NAST_PROTOCOL_UPDATE'
       EXPORTING
            MSG_ARBGB = SYST-MSGID
            MSG_NR    = SYST-MSGNO
            MSG_TY    = SYST-MSGTY
            MSG_V1    = SYST-MSGV1
            MSG_V2    = SYST-MSGV2
            MSG_V3    = SYST-MSGV3
            MSG_V4    = SYST-MSGV4
       EXCEPTIONS
            OTHERS    = 1.
ENDFORM.

END-OF-SELECTION.
* nix
```

⇨

4.8 An ABAP to Workflow Trigger from NAST

A workflow in R/3 can be triggered from any application which can handle NAST messages. When the message refers to a special handler routine (type 8) than a workflow event can be raised even if the calling object does not provide a convenient event.

Messages control calls FORM einzelnachricht IN PROGRAM rsnast00

When R/3 creates a message it stores this message in the table NAST. When the message is set to immediate processing (event time = 4), R/3 message control makes a call to the routine FORM einzelnachricht IN PROGRAM rsnast00. This standard handler looks up the table TNAPR where you can define in customizing the name of an arbitrary routine to be called when the message is due to execute. When the message is not set to immediate processing the program RSNAST00 is usually called in batch to collect all waiting message, then it calls FORM einzelnachricht IN PROGRAM rsnast00 accordingly.

380

Called routine can be any ABAP sub routine which is allowed in an update procedure

The routine can execute anything which is allowed in an update routine. If you want to call a transaction from the NAST workflow handler you must submit the call transaction in a background task or an RFC call to destination NONE. Or destination WORKFLOW_LOCAL.

Messages allow direct call of workflow actions without using the workflow mechanism

When we use workflow only to execute a program when another one ended, we can bypass the R/3 workflow mechanism completely and instead be calling the handler directly. Every R/3 transaction which allows message processing via NAST messages can call such an individual routine when the message is set to output media 8.

NAST processing allows conditional message finding

The big advantage of NAST messages is, that they usually can be determined via the conditional message finding. If a convenient message scheme is set up in customizing, the message is created only and only if certain data constellation is true. So you can execute a workflow action depending on certain data, e.g. if goods issue has posted or if a document has been stored incompletely.

Listing 3: ZSNASTWF – Sample NAST processing routines to trigger a subsequent workflow

```
************************************************************************
* Collection of NAST processing routines                               *
*  for media = 8 : special processing                                  *
*  for media = 9 : Workflow Event                                      *
*  for media = T : Workflow Task                                       *
*                                                                      *
************************************************************************

* <object> contains call declarations for object handling
INCLUDE <OBJECT>.
* Following are the declarations for standard NAST
INCLUDE RVADTABL .
DATA: RETCODE LIKE SY-SUBRC.
DATA: XSCREEN.
DATA: XNAST LIKE NAST OCCURS 0 WITH HEADER LINE.
*---------------------------------------------------------------------*
PARAMETERS: P_KAPPL  LIKE NAST-KAPPL MEMORY ID Z01.
PARAMETERS: P_KSCHL  LIKE NAST-KSCHL MEMORY ID Z03.
PARAMETERS: P_OBJKY  LIKE NAST-OBJKY MEMORY ID Z02.
PARAMETERS: P_SPRAS  LIKE NAST-SPRAS.
PARAMETERS: P_PARNR  LIKE NAST-PARNR.
PARAMETERS: P_PARVW  LIKE NAST-PARVW.
PARAMETERS: P_ERDAT  LIKE NAST-ERDAT.
PARAMETERS: P_ERUHR  LIKE NAST-ERUHR.
*---------------------------------------------------------------------*

*---------------------------------------------------------------------*
AT SELECTION-SCREEN.
*---------------------------------------------------------------------*

*---------------------------------------------------------------------*
INITIALIZATION.
*---------------------------------------------------------------------*
*---------------------------------------------------------------------*
START-OF-SELECTION.
*---------------------------------------------------------------------*
  WRITE: / 'Start processing NAST record', SY-DATUM, SY-UZEIT, SY-UNAME.
  NEW-LINE.
  WRITE: P_OBJKY.
  WRITE: P_KAPPL.
  WRITE: P_KSCHL.
  WRITE: P_SPRAS.
  WRITE: P_PARNR.
  WRITE: P_PARVW.
  WRITE: P_ERDAT.
  WRITE: P_ERUHR.
  IF SY-BATCH NE SPACE.
    WRITE: / 'Batch processing detected'.
    SELECT SINGLE * FROM NAST WHERE OBJKY EQ P_OBJKY
                                AND KAPPL EQ P_KAPPL
                                AND KSCHL EQ P_KSCHL
                                AND SPRAS EQ P_SPRAS
                                AND PARNR EQ P_PARNR
                                AND PARVW EQ P_PARVW
                                AND ERDAT EQ P_ERDAT
                                AND ERUHR EQ P_ERUHR.
  ELSE.
    SELECT SINGLE * FROM NAST WHERE OBJKY EQ P_OBJKY
                                AND KAPPL EQ P_KAPPL
                                AND KSCHL EQ P_KSCHL.
   BREAK ANGELIAX.
  ENDIF.
  IF SY-SUBRC EQ 0.
    WRITE: / 'PERFORM einzelnachricht(rsnast00)'.
    PERFORM EINZELNACHRICHT(RSNAST00) USING RETCODE.
    WRITE: / 'End processing NAST record', SY-DATUM, SY-UZEIT, SY-UNAME.
  ELSE.
    WRITE: / 'NAST Entry not found. Stop processing.'.
  ENDIF.

*---------------------------------------------------------------------*
*       FORM CREATE_EVENT                                             *
```

```
*---------------------------------------------------------------------*
*  Routine is used to process NAST media type "9" (Workflow Event)    *
*  It will actually raise the specified workflow event
*  as defined in NAST-OBJECT and NAST-EVENT
*  Note: NAST-EVENT must exist for object NAST-OBJECT
*        or its delegation type (=derived subtype)
*---------------------------------------------------------------------*
*  -->  RETURN_CODE                                                   *
*  -->  US_SCREEN                                                     *
*---------------------------------------------------------------------*
FORM ENTRY_CREATE_EVENT USING RETURN_CODE US_SCREEN.
  PERFORM CREATE_EVENT(RSWEMC01) USING RETURN_CODE US_SCREEN.
ENDFORM.                               " create_event.
*---------------------------------------------------------------------*
*       FORM ENTRY_VA02TOUCH                                          *
*---------------------------------------------------------------------*
*  Routine is a sample handler which will call a transaction.
*  This special routine call VA02.
*  The speciality here is, taht it can handle immediate processing
*  (NAST-VSZTP = 4) by submitting the processing in a background task
*  The submit in background task is done by
*       CALL FUNCTION 'Y_NAST_PROCESSING' IN BACKGROUND TASK
*  Of course the function has to be defined.
*---------------------------------------------------------------------*
*  -->  RETURN_CODE                                                   *
*  -->  US_SCREEN                                                     *
*---------------------------------------------------------------------*
FORM ENTRY_VA02TOUCH USING RETURN_CODE US_SCREEN.
  CLEAR RETCODE.
  XSCREEN = US_SCREEN.
  BREAK ANGELIAX.
  IF NAST-VSZTP EQ '4'.
*    PERFORM submit_in_backgroundtask USING nast.
    PERFORM SUBMIT_TO_SPOOL USING NAST.
    RETURN_CODE = 3.
  ELSE.
    PERFORM DO_VA02TOUCH USING US_SCREEN NAST-OBJKY.
    CASE RETCODE.
      WHEN 0.
        RETURN_CODE = 0.               "all well done, sets VSTAT = 1
      WHEN 3.
        RETURN_CODE = 3.  "means not processed, leaves VSTAT = 0
      WHEN OTHERS.
        RETURN_CODE = 1.               "Errors, sets VSTAT = 2
    ENDCASE.
  ENDIF.
ENDFORM.

*---------------------------------------------------------------------*
*       FORM PROCESSING                                               *
*---------------------------------------------------------------------*
*       ........                                                      *
*---------------------------------------------------------------------*
FORM DO_VA02TOUCH USING USCREEN PVBELN.
  PERFORM PROTOCOL_UPDATE USING '38' '000' 'I'
      'do_va02touch called' SY-REPID SY-TCODE SY-CALLD.
  RETCODE = 0.
ENDFORM.                               "processing.

*---------------------------------------------------------------------*
*       FORM SUBMIT_IN_BACKGROUNDTASK                                 *
*---------------------------------------------------------------------*
*       ........                                                      *
*---------------------------------------------------------------------*
*  -->  PNAST                                                         *
*---------------------------------------------------------------------*
FORM SUBMIT_IN_BACKGROUNDTASK USING PNAST LIKE NAST.
  NAST = PNAST.
  NAST-VSZTP = '1'.
  CALL FUNCTION 'Y_NAST_PROCESSING' IN BACKGROUND TASK
       EXPORTING
            MSG_NAST = NAST.
ENDFORM.                               "submit_in_backgroundtask
```

```
*---------------------------------------------------------------------*
*       FORM SUBMIT_TO_SPOOL                                          *
*---------------------------------------------------------------------*
*       ........                                                      *
*---------------------------------------------------------------------*
*  -->  PNAST                                                         *
*---------------------------------------------------------------------*
FORM SUBMIT_TO_SPOOL USING PNAST LIKE NAST.
  DATA: TBTCJOB LIKE TBTCJOB.
  NAST = PNAST.
  NAST-VSZTP = '1'.

  TBTCJOB-JOBNAME = 'YSNASTWF'.
  CALL FUNCTION 'JOB_OPEN'
       EXPORTING
            JOBNAME          = TBTCJOB-JOBNAME
       IMPORTING
            JOBCOUNT         = TBTCJOB-JOBCOUNT
       EXCEPTIONS
            CANT_CREATE_JOB  = 1
            INVALID_JOB_DATA = 2
            JOBNAME_MISSING  = 3
            OTHERS           = 4.

  SUBMIT YSNASTWF VIA JOB TBTCJOB-JOBNAME NUMBER TBTCJOB-JOBCOUNT
   AND RETURN TO SAP-SPOOL KEEP IN SPOOL 'X' WITHOUT SPOOL DYNPRO
   WITH  P_OBJKY = NAST-OBJKY
   WITH  P_KAPPL = NAST-KAPPL
   WITH  P_KSCHL = NAST-KSCHL
   WITH  P_SPRAS = NAST-SPRAS
   WITH  P_PARNR = NAST-PARNR
   WITH  P_PARVW = NAST-PARVW
   WITH  P_ERDAT = NAST-ERDAT
   WITH  P_ERUHR = NAST-ERUHR.

   CALL FUNCTION 'JOB_CLOSE'
        EXPORTING
             JOBCOUNT                    = TBTCJOB-JOBCOUNT
             JOBNAME                     = TBTCJOB-JOBNAME
*       IMPORTING
*            JOB_WAS_RELEASED            =
        EXCEPTIONS
             CANT_START_IMMEDIATE        = 1
             INVALID_STARTDATE           = 2
             JOBNAME_MISSING             = 3
             JOB_CLOSE_FAILED            = 4
             JOB_NOSTEPS                 = 5
             JOB_NOTEX                   = 6
             LOCK_FAILED                 = 7
             OTHERS                      = 8.

ENDFORM.                               "submit_in_backgroundtask
*---------------------------------------------------------------------*
*       FORM PROTOCOL_UPDATE                                          *
*---------------------------------------------------------------------*
*       ........                                                      *
*---------------------------------------------------------------------*
FORM PROTOCOL_UPDATE USING
            MSG_ARBGB
            MSG_NR
            MSG_TY
            MSG_V1
            MSG_V2
            MSG_V3
            MSG_V4.
  CHECK XSCREEN = SPACE.
  SYST-MSGID  = MSG_ARBGB.
  SYST-MSGNO  = MSG_NR.
  SYST-MSGTY  = MSG_TY.
  SYST-MSGV1  = MSG_V1.
  SYST-MSGV2  = MSG_V2.
  SYST-MSGV3  = MSG_V3.
  SYST-MSGV4  = MSG_V4.
```

```
  CALL FUNCTION 'NAST_PROTOCOL_UPDATE'
       EXPORTING
            MSG_ARBGB = SYST-MSGID
            MSG_NR    = SYST-MSGNO
            MSG_TY    = SYST-MSGTY
            MSG_V1    = SYST-MSGV1
            MSG_V2    = SYST-MSGV2
            MSG_V3    = SYST-MSGV3
            MSG_V4    = SYST-MSGV4
       EXCEPTIONS
            OTHERS    = 1.
ENDFORM.

FORM ENTRY_WORKITEM USING RETURN_CODE US_SCREEN.
  CLEAR RETCODE.
  XSCREEN = US_SCREEN.
  BREAK ANGELIAX.
  IF NAST-VSZTP EQ '4'.
*    PERFORM submit_in_backgroundtask USING nast.
    PERFORM SUBMIT_TO_SPOOL USING NAST.
    RETURN_CODE = 3.
  ELSE.
    PERFORM DO_WORKITEM USING US_SCREEN NAST-OBJKY.
    CASE RETCODE.
      WHEN 0.
        RETURN_CODE = 0.               "all well done, sets VSTAT = 1
      WHEN 3.
        RETURN_CODE = 3.  "means not processed, leaves VSTAT = 0
      WHEN OTHERS.
        RETURN_CODE = 1.               "Errors, sets VSTAT = 2
    ENDCASE.
  ENDIF.
ENDFORM.


FORM DO_WORKITEM USING RETURN_CODE US_SCREEN.
DATA: WI_CONTAINER LIKE SWCONT   OCCURS 0 WITH HEADER LINE.
DATA: AGENTS       LIKE SWHACTOR OCCURS 0 WITH HEADER LINE.
DATA: SWWWIHEAD    LIKE SWWWIHEAD.
DATA: BEGIN OF OBJECT_ID,
       LOGSYS  LIKE T000-LOGSYS,
       OBJTYPE LIKE SWOTENTRY-OBJTYPE,
       KEY  LIKE SWCONT-VALUE,
      END OF OBJECT_ID.
*
DATA: TASKNAME LIKE HRP1000-MC_SHORT.
DATA: OBJNAME  LIKE SWOTENTRY-OBJTYPE.
*ARAMETERS: taskname LIKE hrp1000-mc_short DEFAULT 'ZAXXVA02'.
*ARAMETERS: objname  LIKE swotentry-objtype DEFAULT 'BUS2032'.
*  *** ************ Object ID ***************************************
* The logical system is retrieved from table T000-logsys
 CALL FUNCTION 'OWN_LOGICAL_SYSTEM_GET'
      IMPORTING OWN_LOGICAL_SYSTEM = OBJECT_ID-LOGSYS.
* BUS2032 is the sales order BAPI
 OBJECT_ID-OBJTYPE = NAST-OBJTYPE.
 TASKNAME          = NAST-EVENT.
 OBJECT_ID-KEY     = NAST-OBJKY.
* *** Load Container
 REFRESH WI_CONTAINER.
* The object_id must be passed to the macro as an unstructured type
* If the type is structured, one single entry for every sub-field
* of the parameter is created. This is not what we want.
 WI_CONTAINER-VALUE = OBJECT_ID.
 SWC_SET_ELEMENT WI_CONTAINER '_WI_OBJECT_ID' WI_CONTAINER-VALUE.
*swc_set_element wi_container 'SALESDOCUMENT' object_id-key.
 SWC_SET_ELEMENT WI_CONTAINER 'VBELN' OBJECT_ID-KEY.
* *** Identify yourself
SWWWIHEAD-WI_CREATOR = SY-UNAME.
* *** Set the action that continues the Workitem
SELECT OBJID INTO SWWWIHEAD-WI_RH_TASK "need internal object id
        FROM HRP1000 UP TO 1 ROWS
        WHERE MC_SHORT EQ TASKNAME.
ENDSELECT.
IF SY-SUBRC NE 0. MESSAGE A000(38) WITH 'Task not found' TASKNAME.ENDIF.
```

```
* *** Define the users, which are meant to check the workitem
REFRESH AGENTS.
AGENTS-OTYPE = 'US'.
AGENTS-OBJID = 'ANGELIAX1'.
APPEND AGENTS.
CALL FUNCTION 'SWW_WI_START_SIMPLE'
     EXPORTING
          CREATOR                      = SWWWIHEAD-WI_CREATOR
          TASK                         = SWWWIHEAD-WI_RH_TASK
     TABLES
          AGENTS                       = AGENTS
          WI_CONTAINER                 = WI_CONTAINER
     EXCEPTIONS
            ID_NOT_CREATED               = 1
            READ_FAILED                  = 2
            IMMEDIATE_START_NOT_POSSIBLE = 3
            EXECUTION_FAILED             = 4
            INVALID_STATUS               = 5
            OTHERS                       = 6.
ENDFORM. " entry_workitemh USING return_code us_screen.

END-OF-SELECTION.
* nix
```

⇨

4.9 Workflow Handlers

When a workflow event is triggered you need a program which has to handle the next step in the workflow. Although there are plenty of convenient handlers in R/3 standard that are applicable for many all day situations, a workflow handler is in general a simple R/3 RFC enabled function module.

SWW_WI_CREATE_VIA_EVENT is a good example to use as a master copy for a new workflow handler

When you start writing a workflow handler routine I suggest to copy the standard function SWW_WI_CREATE_VIA_EVENT to a new function. This function shows you exactly the interface structure which is required for a handler. All IMPORT, EXPORT or TABLES parameter must have exactly the names which are also used in SWW_WI_CREATE_VIA_EVENT, because the handler function will be normally called dynamically from within the workflow processor (function SWE_EVENT_CREATE).

405

Listing 4: Interface structure of a sample workflow handler

```
FUNCTION SWW_WI_CREATE_VIA_EVENT.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(EVENT) LIKE  SWETYPECOU-EVENT
*"             VALUE(RECTYPE) LIKE  SWETYPECOU-RECTYPE
*"             VALUE(OBJTYPE) LIKE  SWETYPECOU-OBJTYPE
*"             VALUE(OBJKEY) LIKE  SWEINSTCOU-OBJKEY
*"       TABLES
*"              EVENT_CONTAINER STRUCTURE  SWCONT
*"----------------------------------------------------------------------
```

⇨