SAP Business Workflow - Tutorials

HEP.BCBMTWFMTUT

Release 4.6C

Copyright

Icons

Icon	Meaning
⚠	Caution
💬	Example
➡	Note
🧭	Recommendation
SYN	Syntax
💡	Tip

SAP Business Workflow - Tutorials

These tutorials introduce you to *SAP Business Workflow*, its fundamental principles, its operation and some of its functions.

Introduction

If you have little or no experience with *SAP Business Workflow*, work through the following two tutorials first:

- *Tutorial: Workflow Modeling (page 7)*
- *Tutorial: Maintaining the Organizational Plan (page 48)*

We also recommend that you work through these tutorials as preparation for taking part in an SAP training course.

Events

If you are particularly interested in options for event creation, refer to the following tutorials:

- *Tutorial: Event Creation Upon Status Changes and Creation of "Mails" (page 54)*

Object types

Object type definitions are covered in a separate tutorial:

- *Tutorial: Workflow Programming (page 69)*

Programming knowledge is a prerequisite for this tutorial. It is therefore only suitable for advanced users.

Input conventions

At several points in the tutorials, you are requested to enter names, descriptions, or texts. These inputs are shown in **`bold letters`**. You can replace entries in pointed parentheses (**`<example>`**) with your own texts.

You can choose your own IDs and descriptions. All of the abbreviations and IDs you use in this tutorial should start with the same character string, for example, your initials. This will help you to find your own definitions. The points at which an input is required are indicated by the character string *ini*.

Options and outputs form the R/3 Systems are indicated in a `different typeface`.

Prerequisites

To work through this tutorial successfully,

- You should already be familiar with the basic terms in *SAP Business Workflow*.
- You do not have to have any practical experience with *SAP Business Workflow*.
- You must be able to carry out all of the individual steps in the system directly.

Client-independent objects

You will be shown how to create *client-independent* objects. You will only be able to work through this tutorial on the system if you are authorized to create these objects.

Customizing

All of the Customizing settings must be defined for *SAP Business Workflow*. To check these settings, choose *Tools* → *Business Workflow* → *Development* → *Tools* → *Customizing*. If the system detects that some of the Customizing settings have not been defined, carry out the automatic Customizing procedure.

For further information, please refer to *Maintaining the Standard Settings for SAP Business Workflow*.

➡

> You do not need to check the settings if you are already using *SAP Business Workflow*. You can use the automatic Customizing function more than once, since existing settings are not overwritten.

Tutorial: Workflow Modeling

Purpose

This tutorial uses an example in a series of easy-to-follow units to explain the most important tools in *SAP Business Workflow*.

The example used here is based on a scenario for *approving a notification of absence*. At the end of this tutorial, you will have defined and executed a workflow that automatically submits a notification of absence (leave request) to your superior for approval, and informs the requester of the result of the approval process.

You will become familiar with the following areas of *SAP Business Workflow* throughout the course of this tutorial:

- Definition tools
- Business Workplace
- Reporting and analysis tools
- Using the Workflow Builder

The tutorial is not intended to provide a full description of all functions and concepts. This information is available in the documentation on *SAP Business Workflow*.

This tutorial does not deal with the definition of object types. If you want further information on this subject, please work through the tutorial on *Workflow Programming (page 69)*.

Process Flow

Work through the individual units in this tutorial in the specified order.

Important units are followed by tests that you can use to test what you have learned to date. Please make sure to complete these tests.

Result

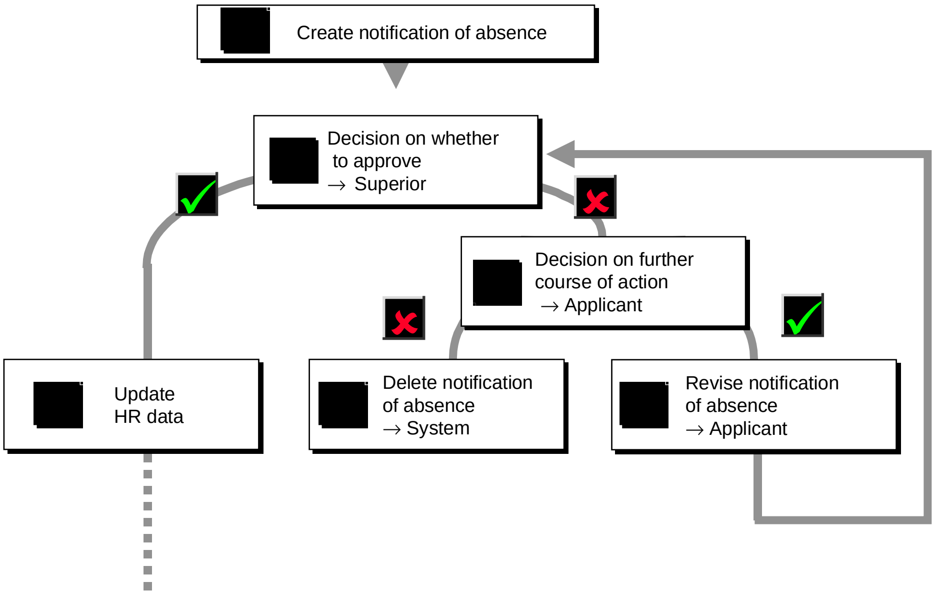
Example - the notification of absence

The scenario in this example begins with the completion of a leave request by an employee (requester or creator of the notification of absence).

The completed form is then forwarded automatically to the head of department (employee's superior).

- If the head of department approves the request, the employee receives a notification and the workflow is terminated.
- If the head of department rejects the request, the employee can decide to revise the request (possibly in accordance with the head of department's wishes) or withdraw it. If the employee decides to revise it, the request form is resubmitted to the head of department after the revision is made.

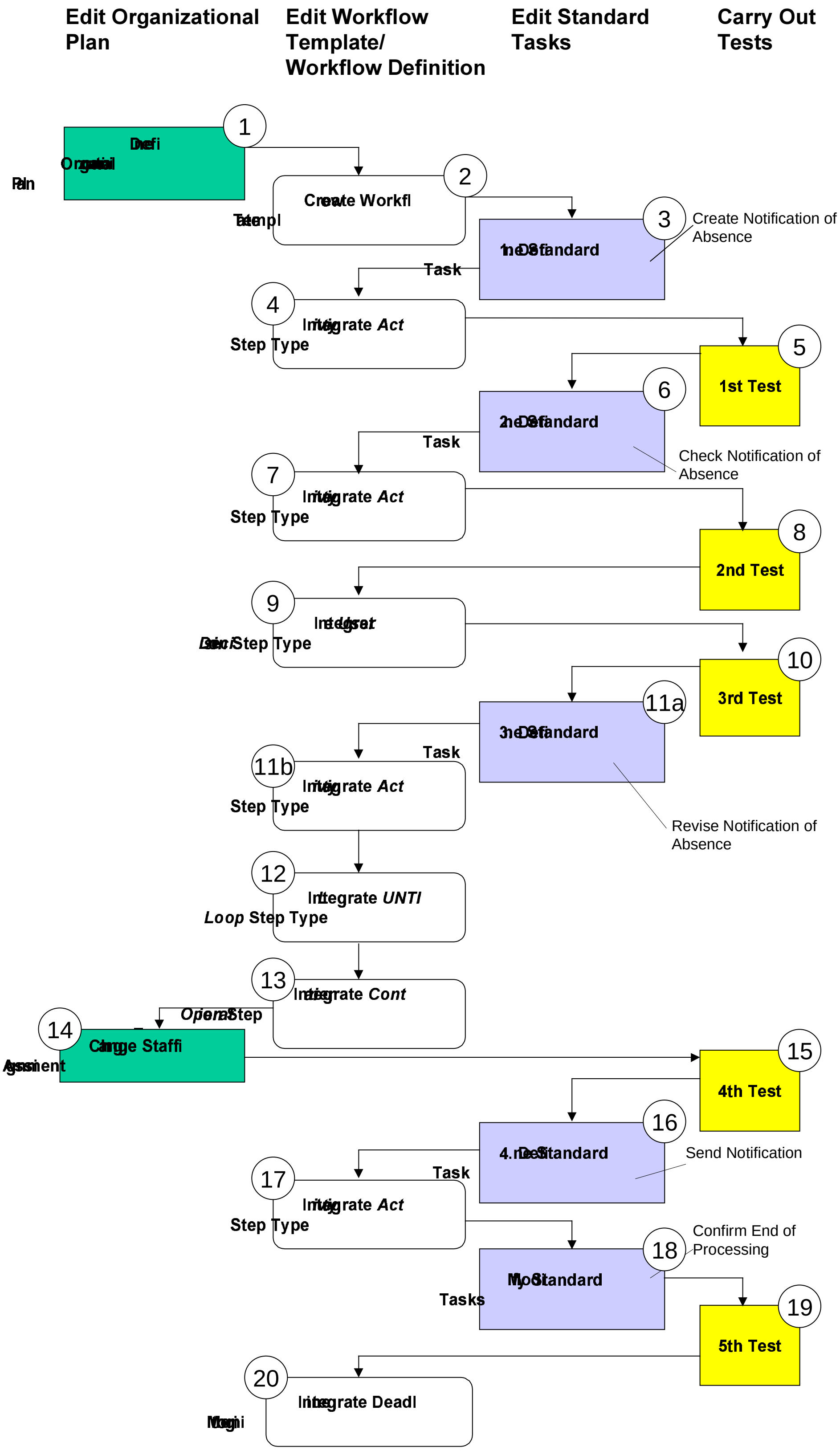
Business Process and Workflow: Example


Create notification of absence
Decision on whether to approve → Superior
✓
✗
Decision on further course of action → Applicant
✗
✓
Update HR data
Delete notification of absence → System
Revise notification of absence → Applicant

The diagram above shows that additional steps could follow the approval, such as updating the leave account, or notifying the secretary. These steps, however, do not arise in this example.

All of the units at a glance

The diagram below shows all of the units in this tutorial. Similar units are listed in the same column.


Edit Organizational Plan
Edit Workflow Template/ Workflow Definition
Edit Standard Tasks
Carry Out Tests
1
Dnefi Orgaatil Pln
2
Create Workfl Tatempl
3
1neDSfandard Task
Create Notification of Absence
4
Intgrate Act Step Type
5
1st Test
6
2neDSfandard Task
Check Notification of Absence
7
Intgrate Act Step Type
8
2nd Test
9
Inteduser Deci Step Type
10
3rd Test
11a
3neDSfandard Task
Revise Notification of Absence
11b
Intgrate Act Step Type
12
Integrate UNTI Loop Step Type
13
Intaegrate Cont Operat Step
14
Change Staffi Assignent
15
4th Test
16
4.neDStandard Task
Send Notification
17
Intgrate Act Step Type
18
MyStandard Tasks
Confirm End of Processing
19
5th Test
20
Intaegrate Deadl Mogni

Unit 1: Organizational Structure

Use

In order for the workflow system to establish the relationship between the requester and their superior, you must create an organizational plan in the system.

For this tutorial, of course, this organizational plan does not have to be complete and valid across the enterprise. To keep the test as simple as possible initially, define an organizational plan that only contains one administrator and one head of department.

Assign both items to yourself. As a result, all work items will appear in your own Business Workplace. Later you will modify the organizational plan such that you will have to work through the scenario with two users.

Procedure

The organizational plan required for this tutorial consists, when complete, of **one** *organizational unit* (= “department”), which contains **two** *positions*: a head of department and an administrator.

Each position is described by one *job* and each position is assigned one user as holder. The head of department position is also designated as *chief position* of the organizational unit.

> ➡
>
> Of course, a “real” organizational plan is created by arranging several organizational units with their positions in a hierarchy. Usually *several* positions are described by *one* job.
>
> For further information, refer to the documentation *Organizational Plan* (see SAP Library).

The procedure in this unit is divided into four parts:

1. You create an organizational unit.
2. Enter necessary jobs as required.
3. You create a position for the head of department in the new organizational unit.

   You define a position in three steps:

   i. You create a position that is assigned your organizational unit.

   ii. You assign a holder to the position.

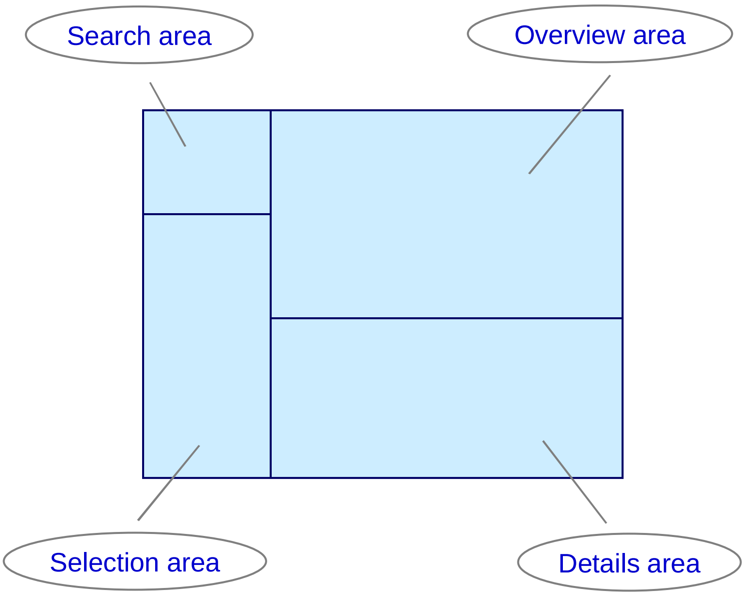
   iii. You assign a job to the position.

3. You create a position for the administrator in the organizational unit.

Creating an organizational unit

1. Choose *Tools* → SAP *Business Workflow* → *Development* → *Definition tools* → *Organizational Management* → *Organizational plan* → *Create*.
2. Confirm the validity period proposed in the dialog box *Creating a Root Organizational Unit*.

   This takes you to the *Create Organization and Staffing (Workflow)* screen. This user interface is divided into four screen areas:



3. On the Basic Data tab in the details area, enter an abbreviation and a name in the *Organizational unit* input fields.

   Abbreviation: <*ini*_**sales**>

   Name: <**OrgUnit: Sales (***ini***)**>

4. Choose 💾.

   You can now create the position for a head of department and one administrator.

Create jobs

When enhancing an organizational unit, the necessary jobs are usually already available. For this tutorial however, you create the necessary jobs for head of department and administrator yourself.

1. Choose *Edit* → *Create jobs.*

   You go to the dialog box *Create jobs*. The lower area contains a list of existing jobs and the upper area contains an input table in which you can create new jobs by entering abbreviations and names.

2. In the input table, enter an abbreviation and a name for each of the new jobs.

   Job - head of department:

   Object abbreviation: <*ini*_**dhead_C**>

   Name: <**job: head of department (***ini***)**>

   Job administrator:

   Object abbreviation: <*ini*_**admi_C**>

   Name: <**job: administrator (***ini***)**>

3. Choose ✓.

Creating Position for Head of Department

1. Change to the overview area in the *staff assignments* of the organizational unit, in order to assign positions, jobs and holders. Choose the arrow ▤ on the right next to the ⧉ and then the *staff assignments (list)*.
2. Choose ⊞.

   A new position is then created in the staff assignments and is displayed in a new line in the table. The position is vacant and no job is assigned to it.
3. Open the details view for the new position in the details area by double-clicking on the entry in the table.
4. On the *Basic Data* tab, enter a code and a description in the *Position* input fields. Overwrite the previous contents.

   Abbreviation: <*ini*_dhead_S>

   Description: <position: head of department (*ini*)>

Assigning a holder to the position

You now assign R/3 users to the positions. The staff assignments for your organizational unit are displayed and you see the vacant position in the table.

5. Select *User* in the search area and enter the search criteria in order to find your user names.

   All of the user names that match your search criterion are listed in the selection screen.
6. Select your user name in the selection area and drag it to the *Person/User* column of the position in the overview area.

   Confirm the message that the relationship period of the validity has been changed.
7. Set the *Head of own organizational unit* indicator in the details area.

Assigning a job to the position

Assign the job of the head of department you created earlier to the position.

8. Select *Job* in the search area and enter the search criteria in order to find the job of the head of department.

   All jobs that match your search criterion are listed in the selection screen.
9. Select job: head of department (*ini*) in the selection area and drag it to the *Job* column of the position in the overview area.
10. Choose 💾.

    The job is assigned the position. Check this by switching to the staff assignments of the organizational unit. Select the organizational unit in the overview area, choose the arrow ▤ on the right next to the ⧉ and then the *staff assignments (list)*. The newly created job is displayed in the *job* column.

Create position for administrator and assign holder and job

You are now in the staff assignments of the organizational unit.

1. Choose ⊞.

   A new position is then created in the staff assignments and is displayed in a new line in the table. The position is vacant and no job is assigned to it.
2. On the Basic Data tab in the details area, enter an abbreviation and a name in the *Position* input fields. Overwrite the previous contents.

   ID: <*ini*_admi_S>

Description: <position: administrator (*ini*)>

3. Choose 💾.
4. Select *User* in the search area and enter the search criteria in order to find your user names.
5. Select your user name in the selection area and drag it to the *Person/User* column of the position in the overview area.

   Confirm the message that the relationship period of the validity has been changed.
6. Select *Job* in the search area and enter the search criteria in order to find the job of the administrator.
7. Select job: administrator (*ini*) in the selection area and drag it to the *Job* column of the position in the overview area.
8. Choose 💾.

Result

Display your entire staffing schedule again and make sure that all the information listed is correct.

You can display a detailed view of jobs, users, and positions. Choose the relevant cell in the table by double-clicking it.

In the details view of a position or job, all of the assigned tasks are displayed on the *Tasks* tab.

You have now completed the first unit (defining the organizational plan). You can now start on the next unit. To exit processing of the organizational structure, choose *Back*.

Unit 2: Creating a Workflow

Use

To define the flow of the approval process in the system, you first create a multistep task workflow template.

For further information, refer to *Definition of Multistep Tasks* in the *SAP Business Workflow* documentation.

Procedure: Creating a workflow template

1. Choose *Tools* → *Business Workflow* → *Development* → *Definition Tools* → *Tasks/Task Groups* → *Create*.

   This takes you to the *Task: Maintain* screen.
2. In the *Task type* field, choose the Workflow template entry.
3. Choose 🗋.

   The *Workflow Template: Add* screen is then displayed.

Entering the basic data for the workflow template

1. Enter an *abbreviation* (of your choice) and a *name* (of your choice) for the workflow template to be created.

   Abbreviation: <*ini*_ws>

   Name: <Workflow: notification of absence (*ini*)>
2. Choose 💾.

3. In the *Create Object Directory Entry* dialog box, choose *Local object*.

   The system has now allocated an 8-digit number for your workflow template. This number is made up of the 3-digit prefix number (Customizing setting) and a 5-digit number taken from a number range. This number together with the letters **WS** forms the workflow ID. Make a note of the ID so that you can later call the workflow more quickly.

Entering a workflow description

4. Go to the *Description* tab.
5. Choose the text type `Task description` and then ✎.

   The *SAPscript* text editor is then launched.
6. Enter a text that describes the task. You can use the following proposal:

   ```
   <You can use this workflow to create a notification of absence.
   This will then be sent to your superior for review. You will be
   notified of the result of this review.

   If your request is rejected, you can revise it and resubmit it
   for review or withdraw it completely.>
   ```
7. Choose ⬅ to return to the workflow template and then save the workflow template.

Determining agents

To enable the workflow template to be started in dialog, the workflow template must be assigned to its possible agents.

8. Select *Additional data → Agent assignment → Maintain*

   The *workflow template: agent assignment* screen is displayed.
9. Position the cursor on the name of your workflow template.
10. Choose *Properties…*

    The dialog box *Task: <Workflow: Notification of absence (ini)>* is displayed.
11. Set the *General task* flag.
12. Choose ✓ *Copy* and then ⬅.

    ⚠

    Designating your workflow template as a *general task* means that every user in the system is authorized to start this workflow template in dialog.

Result

You have now created a workflow template as a "framework" for a workflow definition.

To create a workflow definition, you need tasks that are used as steps in the workflow definition.

Additional information: Tasks

A *task* can either be a *single-step task* or a *multistep task*. In our example of the notification of absence, making the application and checking the application are single-step tasks. The entire procedure involved in processing a leave request consists of these (and other) single steps, and is therefore a multistep task.

➡

In this tutorial, a single step task is generally described as a **task**. The difference between the task types is only explicitly explained if there is a risk of confusion over multistep tasks. The term **workflow** is used as the umbrella term for the workflow template (and thus the multistep task) and the workflow definition. More precise differentiation only takes place if there is a risk of confusion.

On a technical level, the main difference between single-step tasks and multistep tasks is that:

- Every single-step task refers to an object method
- Every multistep task refers to a workflow definition

  A workflow definition is made up of a sequence of different steps (for example, all steps required to process a leave request).

Standard tasks and *workflow templates* are client-independent. *Customer* and *workflow tasks* are client-dependent. Customer and workflow tasks are no longer supported and are therefore no longer used in this tutorial.

An *Activity* is a possible step type in a workflow definition. An activity refers to a task. It references a task, which can be a single-step or multistep task (subworkflow).

For further information, refer to *Tasks and Task Groups* in the *SAP Business Workflow* documentation.

Unit 3: Starting the Workflow Builder

Use

A *workflow definition* is created with the help of the *Workflow Builder*.

Prerequisites

You have now created a workflow template (*Unit 2: Creating a Workflow (page 13)*).

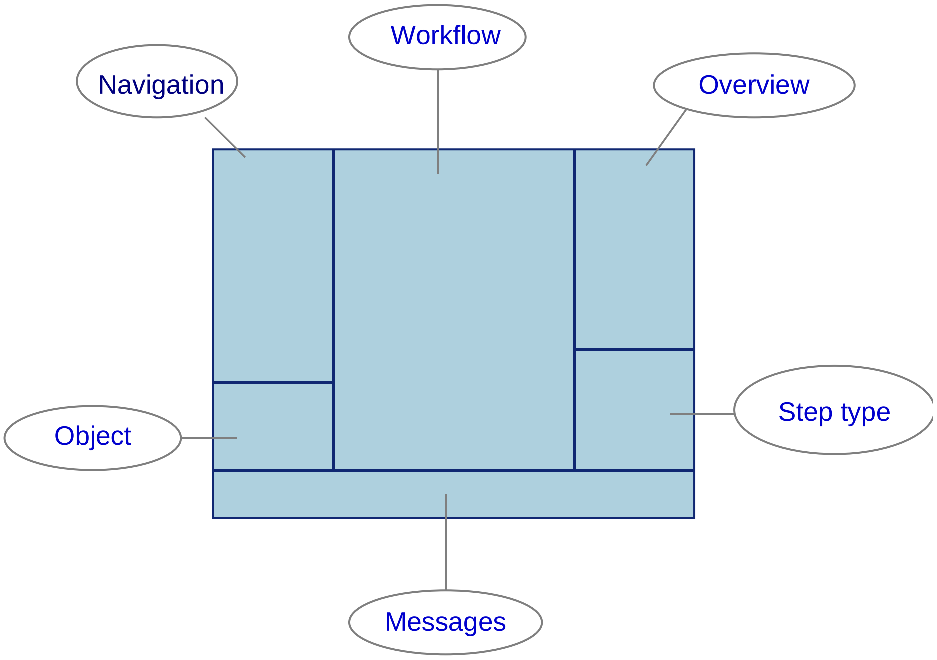
Procedure

Calling the Workflow Builder

1. Choose *Tools* → *Business Workflow* → *Development* → *Definition Tools* → *Workflow Builder.*

   The Workflow Builder is then launched with the workflow definition that was processed last. If the Workflow Builder has not yet been called, then an initial workflow definition is displayed.

   The Workflow Builder screen is divided into the following areas.



The workflow definition is displayed in the Workflow area on the screen. If the workflow is shown in display mode, the step type area is not displayed. For further information, refer to *Workflow Builder* in the *SAP Business Workflow* documentation.

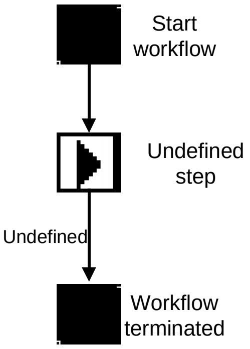
2. Choose 🗗 and enter the abbreviation of your workflow template (<**WS***eight digit number*>) .

    You can use the *F4 input help*. Enter the start of the abbreviation for your workflow (for example: <***ini****>) and choose ✓ All of the workflows that match are displayed. Select your workflow and choose ✓.

Result

The workflow definition is displayed and if necessary can be processed. If the Workflow Builder is called for the first processing of a newly created initial workflow definition, the following graphic is displayed:

Initial Presentation of a Workflow Definition



> If the graphic displayed differs from that above, check the graphic settings in the Workflow Builder. Choose *Extras* → *Options*, and click the *Graphic* tab. Choose the *Without event-driven process chains* view.

- The start of the workflow definition is indicated by *Start Workflow*. If the new workflow definition has been called from the definition of a multistep task for which *triggering events* are defined, these are represented by the symbol and their description.
- The end of the workflow definition is indicated by *Workflow terminated*.
- The area in which the new workflow definition can be added is indicated by an undefined step with an outcome. Steps are displayed in the form of symbols. The description of an outcome is illustrated in the standard view with an arrow.

This unit is now complete. In the following unit you create the first step in your workflow and define a task for this purpose at the same time.

Unit 4: Defining and Inserting Task "Create Notification of Absence"

Use

The notification of absence is created in the first step of the workflow. You will also need a task that executes this function. You can create the task separately from the workflow definition or directly from the Workflow Builder. This unit describes how you create the task from the Workflow Builder.

For further information on creating an individual task, refer to *Definition of Single-Step Tasks* in the *SAP Business Workflow* documentation.

Prerequisites

This task is a single-step task and it references *one* object method. The object type used (**FORMABSENC** (notification of absence)) and the required method (**Create**) are already defined and implemented in the *Business Object Repository*.

You are in the Workflow Builder and the initial workflow definition is displayed.

Procedure

Creating an Activity as a Step in the Workflow Definition

Add a first step of the type *Activity* to the workflow definition.

1. Choose ▶ under step type. The cursor changes shape.
2. Position the cursor on the *Undefined* step and click.

   By doing so, you add an *activity* to your workflow definition. The step definition is shown in the right-hand part of the screen. You are on the tab page *Control*.

Defining a Task "Create Notification of Absence"

You now create a new task, with which a notification of absence can be created.

1. Choose the arrow on the right next to the and then *Create task*.

   The *Basic data* tab in the *Standard task: Create* screen is then displayed.

Entering the basic data

2. Enter an *abbreviation* (of your choice) and a *name* (of your choice) for the standard task to be created.

Abbreviation: *<ini_*`creat`*>*

Name:`<Create notification of absence (ini)>`

Work item text

The work item text appears as an information text in the Business Workplace as soon as there is a work item for the step with this task.

3. Enter `Create notification of absence` in the *Work item text* field.

Object type and method

4. Enter an object type and a method:

   Object type: `FORMABSENC`

   Method: `CREATE`

   You can also use the F4 input help to select the method. Additional indicators associated with this method are set automatically.

   ➡

   You can view the definition of the object type entered by double-clicking on it. Information about its components is provided.

5. Save your task as a local object.

   The system has now allocated an 8-digit number for your task. This number is made up of the 3-digit prefix number (Customizing setting) and a 5-digit number taken from a number range. The identification of this task consists of **TS** and the eight digit number.

Determining agents

Selecting the *possible agents* for this task determines who is to be authorized to make leave requests. For this task, this should be all employees in your enterprise. Define the task as a *general task*.

1. Select *Additional data* → *Agent assignment* → *Maintain*

   This takes you to the *Standard task: Maintain Agent Assignment* screen.

2. Position the cursor on the name of your task.
3. Choose *Properties…*

   The dialog box *Task: <Notification of absence (ini)>* is displayed.

4. Set the *General task* indicator.
5. Choose ✓ *Copy* and then ⬅.
6. Exit the task definition with ⬅.

Interim result

You have now defined your first task in full. This task is used in the activity that you created in the Workflow Builder.

Completing Activity

The *Define Container Elements and Binding* dialog box is then displayed in which the system proposes a change to the workflow container and a binding definition.

- Upper part of the dialog box: Proposal for new container elements in the workflow container.

The system proposes that you create a local container element in the workflow container, which can store the reference to the *notification of absence* created in this step. The element has the technical name AbsenceForm.

- Lower part of dialog box: Proposed binding definition

  The system proposes that you define a binding from the task container to the workflow container. This binding transports the reference to the generated object (i.e. the created notification of absence) **from** the task (from the _WI_Object_ID element in the task container) **to the** workflow (to the AbsenceForm element in the workflow container).

Further information on automatic binding definition proposals is available under *Binding Definitions from the Workflow Container* and *Binding Definitions from the Task Container* in the documentation on *SAP Business Workflow*.

1. Confirm the proposal with ✓.

   The new AbsenceForm container element is displayed in the workflow container. Local container elements are marked ▙. The remaining elements in the workflow container are workflow system elements.

2. Enter a text to describe the step.

   You can, of course, leave the description of the task that is proposed by the system.

   (The current step in your workflow definition appears here.)

3. Go to the *Outcomes* tab.

   All of the outcomes for this step are displayed here.

4. Enter a text for the description.

   The description (proposal: <Application filled out>) in the *Task executed synchronously* line.

   Each step has at least one subsequent event (in this case: *Task executed synchronously*). The description is the text with which the outcome is labeled in the workflow definition.

   The *Form does not exist* and *Form could not be generated* outcomes are the possible exceptions that could occur in the method execution. You “run the risk” of not intercepting these exceptions and not modeling any exception handling in your workflow definition. If one of the exceptions actually occurs at runtime, the workflow will assume the *error* status.

Checking the binding definition (optional)

1. Go to the *Control* tab.

2. Choose ⇄ *Binding (present)*.

   The *WF Builder: Binding for Step ‘<Create notification of absence>’* dialog box is then displayed.

   – In the **upper part of the screen**, the binding defined from the workflow container to the task container is displayed. The system executes this binding before providing the task for execution.

     This binding direction is not relevant for this step since no information is to be transported from the workflow to the task.

   – In the **lower part of the screen**, the binding defined from the task container to the workflow container is displayed. This system executes this binding after the user has processed the task.

     Here, the binding is _WI_ObjectId. ⇨ &ABSENCEFORM& has been automatically defined here by the system. This binding ensures that the reference to the notification of absence created is not only known in the single-step task, but also in the workflow.

You can view the contents of both containers by choosing *Workflow container* or *Task container*.

(The element of the task container `_WI_Object_ID` is called `Notif. of absence`.)

3. Choose ✓.

You then return to the step definition.

Entering the agents responsible

On the tab page *Control,* choose the entry *Workflow initiator (expression)* in the area *agent* as the *agent responsible*.

The system enters the expression `&_WF_INITIATOR&` in the *Expression* field and in the input field beside the checkbox.

➡

You start your executable workflow later manually. At this point in time, the system fills the workflow container element **_WF_Initiator** automatically with your user name.

The above assignment informs the workflow system that the first work item for creating the notification of absence should be addressed as the "starter" as the workflow.

You return to the screen *Workflow definition: Create step: Activity*

Concluding step definition

1. Choose to check the step definition.
2. Choose ✓ to exit the step definition.
3. Choose to view the entire workflow definition in the workflow area on the screen.

   You will notice that your workflow definition now contains a new step (the activity just created). The undefined step is also still there.
4. Choose .

   The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.
5. Exit the Workflow Builder.

Result

You have now completed this unit and can start the next unit.

Unit 5: First Test

Use

You are now going to start a workflow in dialog for the first time, which will run according to your workflow definition. This workflow definition only contains a step for creating a notification of absence so far.

For further information on starting workflows, refer to *Starting Workflows (Test Environment)* in the *SAP Business Workflow* documentation.

Prerequisites

The creation of a notification of absence, which you carry out within your workflow, is reported on a system-wide basis by an event. Other workflows entered as event receivers for this event may therefore be started. The notification of absence created by you (and published by the event) is then also processed with these workflows.

This event is **not** used in connection with this tutorial.

To ensure that only you work exclusively with the notification of absence, you can deactivate any existing linkages between the event and its receivers. Proceed as follows:

> ⚠
>
> If you carry out the following steps, you prevent other workflows being started unintentionally by your notification of absence.
>
> However, you may interrupt other workflow demonstrations. Therefore, be careful and speak to the colleagues involved if necessary.

1. Choose *Tools* → *Business Workflow* → *Development* → *Utilities* → *Events* → *Simulate Event.*
2. Enter `FORMABSENC` in the *Object type* field.
3. Enter `created` in the *Event* field.
4. Choose 🕓.

   The system simulates the event **FORMABSENC.Created** and establishes which workflows would be started.
5. If you find one or more entries in the list under the branch *Tasks without syntax errors to be started*, position the cursor on an entry and choose ✳ *Event linkage*.

   The dialog box *Event linkage: Triggering events* is displayed.
6. Position the cursor on the entry with the red background for the event and choose ✳.

   The linkage between workflow and event is deactivated.
7. Choose ✓.
8. Repeat steps 5 to 7 or repeat the whole simulation as applicable.

Procedure

Starting the workflow and filling out the form

1. If you are still in the Workflow Builder, choose 🖥.

   The *Start Workflow (Test Environment)* screen is then displayed. The number of your workflow template is entered.

   If you are no longer in the Workflow Builder, choose *Tools* → *Business Workflow* → *Development* → *Runtime Tools* → *Start Workflow (Test Environment)*. Enter the identification of your workflow or use the F4 input help.
2. Choose 🕓.

To handle work processes more quickly, SAP Business Workflow supports *Advancing with Immediate Dialog*. When a workflow is started in dialog, this means that the first work item of the workflow is made available for processing immediately provided that the user who starts the workflow is also one of the *recipients* of this first work item. This is the case here because you entered the container element `_WF_Initiator` as the agent in the step definition.

The *Create notification of absence* screen, therefore, is displayed immediately where you can see the notification of absence form. It is the method `Create` that is executed with the first work item.

3. Fill out the form with entries of your choice.

4. Choose 💾.

   The *Start Workflow (Test Environment)* screen is then displayed again.

5. Choose ⬅ and exit the Workflow Builder if necessary.

Starting a workflow / Business Workplace

Before you extend your workflow definition, carry out the following test. It will familiarize you with the *Business Workplace*.

1. Start your workflow in dialog again. Choose *Tools* → *Business Workflow* → *Development* → *Runtime Tools* → *Start Workflow*.

   You go to the screen *Start task*.

2. Choose your workflow in the table on the left.

3. Choose *Start* on the right-hand side of the screen.

   The notification of absence form is displayed.

4. Do not make any entries and do **not** save. Instead, choose ✖.

   You have now started the workflow but canceled the processing of its first step. But the processing of the work item is not yet completed. You have only broken the processing chain of advancing with immediate dialog.

5. Choose ⬅.

6. Choose *Tools* → *Business Workflow* → *Development* → *Runtime Tools* → *Business Workplace*.

7. Open the *Inbox* node and choose the *Workflow* folder.

   Your workflow inbox now contains (at least) one work item for processing. This is the work item for creating a notification of absence, the processing of which you canceled previously.

8. Select the work item, if necessary, and choose 🕒.

   You return to the *Create notification of absence* screen where the notification of absence form is displayed.

9. Fill out the form and choose 💾.

   The work item disappears from your workflow inbox.

Result

You have now completed this unit and can start the next unit.

Reporting and analysis

In this first test, you will take a look at the work item analysis function.

1. Choose *Tools* → *Business Workflow* → *Development* → *Reporting* → *Work Item Analysis* → *Work Items Per Task*

   The *Work Items Per Task* screen is then displayed.

2. Select the monitoring period *Today*.

3. Select the work item type *(Sub-)Workflow* only.

4. Choose 🕒.

The system then determines all of the workflows that were started today. These are then listed on the *Work Items Per Task from <Date> to <Date>* screen. The number of work items in each workflow is also shown here.

⚠

Make sure that you only ever analyze the work items for your workflow and your absence notification.

5. Display the list of associated work items by double-clicking the workflow ID.

   Both the workflows you just started for this workflow identification are displayed with status *Completed*.

6. Display the workflow log by double-clicking a work item ID.

   The *Workflow Log* screen is then displayed.

   For information on the workflow log, see *Workflow Log* in the *SAP Business Workflow* documentation.

7. Exit the work item analysis function.

Unit 6: Creating a Task "Check Notification of Absence"

Use

You define the task *Check notification of absence*. This task is incorporated into your workflow definition as the second step.

As this is not the first task you have defined, many steps will doubtless be familiar to you. (Refer to *Unit 4: Defining and Inserting Task "Create Notification of Absence (page 15)*.)

Prerequisites

The object type used (here: **FORMABSENC** (notification of absence)) and the required method (here: **Approve**) are already defined and implemented in the *Business Object Repository*.

Procedure

1. Choose *Tools* → *Business Workflow* → *Development* → *Definition Tools* → *Tasks/Task Groups* → *Create*.

   The screen *Task: Maintain* is displayed.

2. In the *Task type* field, choose the `Standard task` entry.

3. Choose 🗋.

   The *Basic data* tab in the *Standard task: Create* screen is then displayed.

Entering the basic data

1. Enter an *abbreviation* (of your choice) and a *name* (of your choice) for the task to be created.

Abbreviation:	<*ini*_`check`>
Name:	<`Check notification of absence` (ini)>

Object type and method

2. Enter an *object type* and a *method*:

Object type:	`FORMABSENC`

Method: `APPROVE`

You can also use the F4 input help to select the method.

Work item text

3. In the *Work item text* field, enter the text `Check notification of absence from`.
4. Choose 💾. Save your standard task as a local object.
5. You can include variables in your work item text to be filled with values from the task container at runtime..

   The name of the creator of the notification of absence is to be included into the work item text. As soon as the work item appears in the superior's Business Workplace, the name of the creator appears in the work item information text.

   Position the cursor in the work item text after the word `of`.
6. Choose ▦.

   The *Please choose an expression* dialog box is then displayed.
7. Choose the expression *Notif. of absence* by double-clicking the *Container* node. →*Issuer object ref.* → *Name.*

   ➡

   If **_WI_Object_ID** is displayed instead of *Notif. of absence*, choose ▣. The system then displays the description of the expressions instead of their technical names.

   The variable `&_WI_Object_ID.Creator.Name&` has been added to the work item text automatically. The variable name displayed here is the technical name of the expression.

You can integrate all of the elements in the task container in your work item text. This enables you to include information into the work item text, which is not available until runtime.

Entering the description text

1. Go to the *Description* tab.
2. Choose `Task description` in the *Test type* field.

   The current task description is then displayed. This text is intended to inform future recipients of a work item in which this task is referenced and help them in their work.

   The text will often be similar to the work item text or may supplement it.
3. Choose 📝.
4. Enter the following text:

   `Please check notification of absence no. &_WI_OBJECT_ID.NUMBER& from &_WI_OBJECT_ID.CREATEDATE& of employee &_WI_OBJECT_ID.CREATOR.NAME&.`

   `Decide whether to approve the request.`

   If you want to insert an expression from the task container as a text variable, choose *Include → Expression.* Then choose the relevant expression in the *Please choose an expression* dialog box by double-clicking the *Container* node.
5. Choose ⬅.
6. Choose 💾.

Determining possible agents

1. Select *Additional data* → *Agent assignment* → *Maintain*

    This takes you to the *Standard task: Maintain Agent Assignment* screen.

2. Position the cursor on the name of your task.
3. Choose 👥.

    You go to the dialog box *Choose agent type.*

4. Double-click *Job*.

    The dialog box *Choose Job* is displayed.

5. Enter either part of or the full abbreviation of the job that you created for the head of department in *Unit 1: Organizational Plan (page 10)*.

    You return to the screen *Standard task: Maintain agent assignment.*

6. Choose ⤓.

    You have linked the task to the job of a head of department. In the organizational plan, you use this job to describe a position. The holder of the position is also displayed.

7. Choose ←.
8. Choose 💾.

Result

The task for checking the notification of absence is defined in full. You can now exit the screen for defining a task.

In the next unit, you will incorporate this task into the workflow definition.

➡

Display your organizational plan again.

a. Choose *Tools* → *Business Workflow* → *Development* → *Definition Tools* → *Organizational Management* → *Organizational Plan* → *Display.*

b. Choose your organizational unit in the search area. Display it by double-clicking the entry in the table.

c. In the overview screen, choose the arrow ▾ on the right next to the ⧉ and then *Staff assignments (list)*.

d. Display a detailed view of the position of the head of department by double-clicking the entry in the list.

Here, you can see that this position is assigned to the job and, on the *Tasks* tab, that it is also assigned to the task *Check notification of absence*.

Unit 7: Including "Check Notification of Absence" in the Workflow Definition

Use

The task with which an employee can check a notification of absence is ready to be included into your workflow definition as the next step.

Prerequisites

You have defined a workflow with a step (*Unit 4: Task Defining and Inserting "Create Notification of Absence" (page 20)*) and created the task *Check Notification of Absence* (*Unit 6: Create Task "Notification of Absence" (page 23)*).

Procedure

Calling the Workflow Builder

1. Choose *Tools* → *SAP Business Workflow* → *Development* → *Definition Tools* → *Workflow Builder.*

   The Workflow Builder is then launched in display mode with the workflow that was processed last.

2. Work through the following steps to display your workflow for this tutorial in change mode.

   i. If the workflow you created for this tutorial is already displayed, choose ✎. Go to step 3.

   ii. If a different workflow is displayed, check whether the workflow you created is listed in the *My available tasks* area. Double-click it and choose ✎. Go to step 3.

   iii. Choose 🗀. Enter the complete abbreviation in the *Task* field on the *Open other workflow definition* dialog box and press RETURN. You can now also enter the workflow abbreviation. Enter either the full abbreviation or just the first characters (for example: `<i*>`). Press ***F4***. The search result is then displayed. Choose your workflow by double-clicking it. When you enter the abbreviation in the *Task* field on the *Open other workflow definition* dialog box, all of the available versions are displayed. Since you have not created any versions, select version 0000 and choose ✓. The workflow is then loaded in display mode. To switch to change mode, choose ✎.

3. Choose ⛰ to display the entire workflow in the workflow area on the screen.

Creating a container element in the workflow container

You need an additional container element in the workflow container to store the name of the user who will execute this step. This user name is to be used later in the notification text that is sent to the requester.

1. Choose the entry <*Double-click to create*> by double-clicking in the *Workflow Container*.

   The dialog box for entering a container element is displayed.

2. Make the following entries:

   Element: `Approver`

   Name: `Approver`

   Description: `Approver of the absence request`

   Reference table: `WFSYST`

   Reference field: `AGENT`

3. Choose ✓.

   You have now added a local container element to the workflow container, which is ready to take an agent name, based on its data type reference. However, you have not yet determined a value for this container element.

Creating an Activity as a Step in the Workflow Definition

1. Choose ▶ under step type. The cursor changes shape.

2. Position the cursor on the *Undefined* step that follows the outcome *Request completed* and click.

   By doing so, you add an *activity* to your workflow definition. The step definition is shown in the right-hand part of the screen. You are on the tab page *Control*.

Entering basic data for control

3. In the *Task* field, enter the abbreviation (**TS<***8-digit number***>**) of the task *Check Notification of Absence* that you defined in the previous unit. Choose ✓.

   If you no longer know the full abbreviation, you can use different methods to search for it:

   – Use the *F4 input help* to search via the object type and method of the task.

     This brings you to the *Search and Find: Tasks* dialog box.

     On the tab page *Obj. type and method*, enter the *object type* **FORMABSENC** and the *method* **CREATE**.

     Then double-click your task in the list. You should recognize your task from your initials in the abbreviation. Choose ✓.

   – Enter a character string in the abbreviation or description and choose ✓.

     A list of tasks is then displayed in which you can double-click the desired task.

     ➡

     Every activity, as a step in a workflow definition, refers to a task.

     At this point, the unique reference to this task is entered. The identification is made up of an abbreviation (T, TS, WF, or WS) followed by an 8-digit number.

     If you select a workflow (WS or WS) as your task, your activity is a *subworkflow*. If you select a task (TS or T), your activity is a single step.

   You go to a dialog box in which the system proposes binding from the workflow container to the task container.

   Here, the object reference to the notification of absence is transported **from** the workflow container (container element `AbsenceForm`) to the task container (to the container element `_WI_Object_ID`).

4. Choose ✓ to confirm the proposal.

Extending the binding definition

Now define another binding via which the *current agent* of the work item (the superior) is stored in a container element of the workflow container.

5. Choose ▦ *Binding (present).*

   The *Binding for Step 'Check notification of absence'* dialog box is then displayed. If the first column is entitled *Element*, choose ▦ to display the *element description*.

   The system has created the following binding definition from the workflow container to the task container:

   `Notif. of absence` ⇐ `&AbsenceForm&`

   This binding ensures that the correct notification of absence is checked in the task.

6. Choose ▦.

   All of the container elements in the task container are now displayed on the left-hand side. You can define a binding from these container elements to the workflow container.

7. Position the cursor in the lower half of the screen, in the empty input field beside the container element *Actual agent* and call the F4 input help.
8. Double-click *Approver*.

   In addition to the bindings already existing, the following binding is now defined:

   `Actual Agent` ⇨ `&Approver&`
9. Choose ✓.

Determining outcomes

The method **APPROVE** of the object type **FORMABSENC**, to which you refer in the standard task definition, is defined with a *result*. The three possible values of this result are offered on the tab page *Outcomes* as outcomes of this step:

- *Approved*
- *rejected*
- *New*

The ● indicates that these outcomes have already been transferred to the workflow definition. The outcome *New* is not really appropriate at this point. At this stage of processing, the notification of absence cannot be “new” any more. It should therefore not be incorporated in the definition.

1. Click on the symbol ● in front of the outcome *New.*

   The symbol changes to ◆.

   ➡

   The outcome *Processing obsolete* is processed if the relevant work item is set to obsolete via a *process control* step. This functionality is not used in this tutorial.

   The outcome *Form not available* results from the *exception* defined for the method **APPROVE**. You could include this outcome into the workflow definition and would then have to model appropriate subsequent steps. However, you do not model anything for the purposes of this tutorial and accept that your workflow will have an error at runtime if this exception is raised.
2. Choose ✓.

   Two branches are now inserted in your workflow definition for the two marked outcomes *Approved* and *Rejected*.

Inputs for the agent

If you are working through this tutorial alone, do not specify anything for responsibility at this point. If it is processed when the workflow is executed, therefore, this step will be “offered” for processing to all employees declared as *possible agents* of the single-step task *Check notification of absence*. This means all user whose positions are described with the job *head of department*.

In your organizational plan, there is only one users who is a possible agent -> you.

- You have maintained the organizational plan with jobs, positions, and user assignments in *Unit 1: Organizational Plan (page 10)*.
- You have defined the single-step task “Check notification of absence” and have assigned the possible agents in *Unit 6: Definition of the Standard Task “Check Notification of Absence” (page 23)*.

Concluding step definition

Choose 🪄.

The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

Result

You can subject your workflow definition to a second test.

Unit 8: Second Test

Use

You start your workflow in dialog and two steps are executed. The notification of absence is created in the first step and then checked in the second.

Procedure

Starting the workflow and filling out the form

6. If you are still in the Workflow Builder, choose .

   The *Start Workflow (Test Environment)* screen is then displayed. The number of your workflow is entered.

   If you are no longer in the Workflow Builder, choose *Tools* → *Business Workflow* → *Development* → *Runtime Tools* → *Start Workflow (Test Environment)*. Enter the number of your workflow or use the F4 input help.

7. Choose .

   You go to the screen *Create Notification of Absence*.

From the requester's perspective

When the workflow is started, the work item for creating the notification of absence is provided to you directly for processing. This is because of *advancing with immediate dialog*. You were introduced to this in the first test.

3. Fill out the form and choose .

From the head of department's perspective

You created the step for checking the notification of absence in your workflow definition without specifying the agents responsible. This means that all of the possible agents of the single-step task are recipients of the work item. Since you occupy the position of both requester and head of department, you are **automatically and immediately** presented with the work item for approving the application. This is again because of *advancing with immediate dialog*.

As the superior, you have several options:

- Approve the request
- Reject the request
- Cancel processing of the request.

4. Choose to cancel processing.

The *Start Workflow (Test Environment)* screen is then displayed again.

Exit this dialog and, if necessary, the Workflow Builder.

Reporting and analysis: Workflow outbox

In the first test of your workflow, you were introduced at this point to *work item analysis*. Now, you will be introduced to the *workflow outbox*.

Amongst other things, the work items which you started in dialog along with date, time and current status are displayed in the workflow outbox.

1. Choose *Tools* → *Business Workflow* → *Development* → *Runtime Tools* → *Business Workplace*.

2. In the tree displayed, choose *Outbox* → *Started workflows*.

3. Choose the arrow on the right next to the and then *Today only*.

   The work items for all the workflows you started today are then displayed. Note the statuses of the work items displayed in the Status column of the Business Workplace. Read the F1 help for the Status column.

4. Position the cursor on the work item for the workflow just started. This work item has the status *in process*.

   From the current workflow data, you can see that the step *Create notification of absence* has been completed successfully, when that was, and who processed it. The work item has the status *completed*.

   You can also see that the work item representing the task for checking the request still has status *ready* and can therefore still be seen in the superior's Business Workplace.

Business Workplace - check notification of absence

You now once again occupy the position of the superior who canceled processing earlier when the application was to be checked.

5. In the tree, choose *Inbox* → *Workflow*.

   The work items for you to process are displayed with their work item texts and certain other attributes.

   A preview of the selected work item is displayed below the list and contains the description of the work item.

6. Select the work item for checking the notification of absence and choose .

   You go to the work item display. There you find amongst other things the description text, also with replaced text variables.

7. Choose to return to the Business Workplace.

8. Choose or start execution by double-clicking the entry.

9. Either reject or approve the request. You should start the workflow twice and test each variant.

To check whether the workflow was completed correctly, go to your workflow outbox. The procedure is described above.

Result

Your workflow definition has now been tested and you can continue with the next unit.

Unit 9: Adding a User Decision

Use

In the previous units, you created a workflow whose definition contains two activities. These activities are based on two standard tasks that you created.

In this unit, you will add a *user decision* to your workflow definition.

With the user decision, the requester can decide to revise and resubmit the notification of absence if the head of department rejects it.

For further information, refer to *Maintaining a User Decision* in the *SAP Business Workflow* documentation.

Procedure

You start the Workflow Builder and open your workflow definition in Change mode. Proceed as described in *Unit 7: Including "Check Notification of Absence" in the Workflow Definition (page 25)*.

Creating a user decision as a step in the workflow definition

1. Choose ▣ under step type. The cursor changes shape.
2. Position the cursor on the *Undefined* step that follows the outcome *Rejected* and click.

   By doing so, you add a *user decision* to your workflow definition. The step definition is shown in the right-hand part of the screen. The *Decision* tab is displayed.

Entering basic data for user decision

3. Enter `Your request was rejected by &. Revise?` in the *Title* field.

   This text is used as the title for the alternative decisions when the workflow is executed. The variable & is a placeholder for a parameter that is filled from the task container at runtime.

4. Choose the F4 input help in the *Parameter 1* field.

   The *Expression for first parameter of user decision* dialog box is then displayed.

5. Choose the *Approver* container element by double-clicking it.

Entering alternative decisions

6. Define the alternative decisions. Enter the following texts:

Decision texts	*Name*
`Decision: Revise request?`	`Revise`
`Decision: Withdraw request?`	`Withdraw`

   The *decision texts* you enter here appear as decision options on the screen that can be processed by the requester after their leave request has been rejected.

   The *descriptions* are the terms used to describe the outcomes in the workflow definition.

Entering the agents responsible

7. Choose *Workflow initiator (expression)* under *Agents*.

   The system enters the expression &`_WF_INITIATOR`& in the *Expression* field and in the input field beside the checkbox.

Concluding step definition

1. Choose to check the step definition.
2. Choose ✓ to exit the step definition.
3. Choose to view the entire workflow definition in the workflow area on the screen.

   You can see that your workflow definition contains a user decision in the *rejected* branch. This has two outcomes whose branches converge again in the *rejected* branch. Each new branch contains an undefined step.

4. Choose .

The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

Result

You can now subject your workflow definition to another test.

Unit 10: 3rd Test

Use

You start your workflow in dialog and three steps are executed: Two activities, for creating and checking a notification of absence, and one user decision.

Procedure

Starting the workflow and filling out the form, checking the notification of absence, user decision

8. If you are still in the Workflow Builder, choose .

   The *Start Workflow (Test Environment)* screen is then displayed. The number of your workflow is entered.

   If you are no longer in the Workflow Builder, choose *Tools* → *SAP Business Workflow* → *Development* → *Runtime tools* → *Start workflow (test environment)*. Enter the number of your workflow or use the F4 input help.

9. Choose .

   You go to the screen *Create Notification of Absence.*

From the requester's perspective

10. Fill out the form and choose .

    When you save the form, you (as superior) can immediately approve or reject the application (reason: *advancing with immediate dialog*).

From the superior's perspective

11. Do not approve the application.

12. Choose .

    If the request is not approved, the next step in the workflow definition is the user decision. The requester was entered as the agent for the step. The decision is therefore offered to you again as the requester (reason: *advancing with immediate dialog*).

From the requester's perspective

You are given three options:

- *Revise request?*
- *Withdraw request?*
- *Cancel*

It is irrelevant which option you choose since you have not defined any follow-up steps. You should start the workflow several times and try out the different options. The complete process flow can be checked each time in the work item analysis or workflow outbox.

Unit 11: Define and Include "Revise Notification of Absence" in the Workflow Definition

Use

You will now add the activity *Revise notification of absence* to your workflow definition. You will create the necessary task within the workflow definition as in unit 4.

Procedure

You start the Workflow Builder and open your workflow definition in Change mode. Proceed as described in *Unit 7: Including "Check Notification of Absence" in the Workflow Definition (page 25)*.

Creating an activity as a step in the workflow definition

1. Choose ▶ under step type. The cursor changes shape.
2. Position the cursor on the *Undefined* step that follows the outcome *Revise* after the user decision, and click.

   By doing so, you add an *activity* to your workflow definition. The step definition is shown in the right-hand part of the screen. You are on the tab page *Control*.

Creating a standard task for use in this step

1. Choose the arrow ▤ on the right next to the ⚙ and then *Create task*.

   The *Basic data* tab in the *Standard task: Create* screen is then displayed.

2. Enter the following data.

   Abbreviation: **<*ini*_revis>**

   Name: **<Revise notif. of absence (*ini*)>**

   Work item text: **Revise notification of absence**

   Object type: **FORMABSENC**

   Method: **UPDATE**

   You can also use the F4 input help to select the method.

3. Save your standard task as a local object.
4. Select *Additional data* → *Agent assignment* → *Maintain*
5. Position the cursor on the name of your standard task.
6. Choose *Properties*…
7. Set the *General task* indicator.
8. Choose ✓ *Copy* and then ⬅.

   The screen for editing the standard task is then displayed.

9. Choose 💾 and then ⬅.

   You then return to the step definition in the Workflow Builder. For this task, a binding must be defined between the workflow and task containers. The system proposes a binding and displays it for you to check in the *Define container elements and binding* dialog box. This proposal defines that the object reference to the notification of absence is transported **from** the workflow container (container element `AbsenceForm`) **to** the task container (container element `_WI_Object_ID`).

10. Confirm the proposal with ✓.

The system has entered the abbreviation for the new standard task in the *Task* field. The *Step description* field contains a description of the standard task. You can change this entry if necessary.

11. Switch to the *Outcomes* tab and enter a description (for example `request revised`) for the *Task executed synchronously* outcome.

Entering the agents responsible

12. Choose *Workflow initiator (expression)* under *Agents*.

The system enters the expression &_WF_INITIATOR& in the *Expression* field and in the input field beside the checkbox.

By doing so, you choose the *agent responsible*.

Concluding step definition

1. Choose to check the step definition.
2. Choose ✓ to exit the step definition.
3. Choose to view the entire workflow definition in the workflow area on the screen.

   You will notice that your workflow definition now contains a new step (the activity just created). The undefined step is also still there.
4. Choose .

   The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

Result

You can now subject your workflow definition to another test. Carry out the test as described in *Unit 10: Third Test (page 32)*.

Test proposal

To test the new step, proceed in the following order:

1. Create notification of absence.
2. Reject application
3. Revise user decision.
4. **Revise request** (the new step)

Since you are both the requester and approver and because *advancing with immediate dialog* is activated, all of the steps are presented to you directly.

Unit 12: Integrating the UNTIL Loop for Reapproval

Use

If the requester decides to revise and resubmit the notification of absence to their superior, the step *Check notification of absence* must be executed again.

There are different ways of solving this problem. The solution outlined here is only one example.

- You create a container element in the workflow container that you use as a flag. This flag contains different values, depending on the status of the notification of absence:

   – Approved

   – Not approved and revised

   – Not approved and not revised

   To assign values to the container element, container operations must be executed on the workflow container.

   For information on the workflow log, see *Definition of the Workflow Container* in the *SAP Business Workflow* documentation.

- You use this container element to define the condition in the *UNTIL loop*.

   For further information, refer to *Maintaining an UNTIL loop* in the *SAP Business Workflow* documentation.

Procedure

You start the Workflow Builder and open your workflow definition in Change mode. Proceed as described in *Unit 7: Including "Check Notification of Absence" in the Workflow Definition (page 25)*.

Create container element in the workflow container

1. Choose the entry *Double-click to create* by double-clicking in the *Workflow Container*.

   The dialog box for entering a container element is displayed.

2. Make the following entries:

   Element: `Flag`

   Name: `Flag`

   Description: `Flag for approval status`

   Reference table: `SYST`

   Reference field: `INPUT`

3. Choose ✓.

   You have now added a local container element to the workflow container, which is ready to store a flag, based on its data type reference. However, you have not yet determined a value for this container element.

Integrating the UNTIL loop as a step

10. Choose under step type. The cursor changes shape.

11. Move the cursor to the *Undefined* step before the event workflow end and click.

   By doing so, you add an *UNTIL loop* to your workflow definition. The step definition is shown in the right-hand part of the screen.

Entering the basic data

12. Enter `Resubmission required?` in the *Step description* field.

13. Click on the condition area to open the condition editor.

   The bottom half of the screen contains the empty condition, whereas the top half shows the system fields as well as the content of the workflow container. The field in the *Expression 1* column is ready for input and marked ➨.

14. Choose the *Flag* container element as the first expression by double-clicking it.

   The first expression is now entered in the condition in the lower part of the screen. The *Expression 2* field is now ready for input and marked ➨.

15. Choose = as the *operator*.
16. Enter the constant **X** in the field for *Expression 2* and confirm by pressing **Enter**.
17. Choose ✓.

You have now defined the condition `Flag = X`.

The comparison between the container element `Flag` and the constant **X** returns either the result *true* or the result *false*. Two outcomes are therefore possible.

18. Enter the following texts for the outcomes:

    True: **Flag equal to X - no resubmission**

    False: **Flag not equal to X - resubmission**

    If the result of the evaluated condition is *false*, the loop is processed again. If the notification of absence is to be resubmitted to the superior, the container element *flag* must not have the value X. For this purpose, you will insert *container operations* in the workflow definition in the next unit.

19. Choose ✓.

You have now inserted an isolated UNTIL loop into the workflow definition, which does not yet contain any steps. You must change the workflow definition so that all of the steps that are required to check and resubmit the notification of absence are within the UNTIL loop.

In order to understand the steps required, you must familiarize yourself with block operations.

Block operations

1. Choose 👤 to display the entire workflow definition in the workflow area on the screen.
2. Display the block structure of your workflow definition. Choose *Graphic* → *Blocks* → *Show*.

   ➡

   The workflow definition is block-oriented. Every block represents a basic structural element that is a self-contained and consistent arrangement of steps and outcomes.

   If you, for example, create a new step that has **one** outcome, this pair (step, outcome) represents a block. Operations that are performed on one step (delete, cut,..) always affect the entire block associated with that step.

   In the above example, in which the block consists of the pair (step, outcome), using the *delete* function will also delete the corresponding outcome as well.

   You should test the operations listed under *Edit* → *Block* on blocks:

   - First save your workflow definition.
   - Test the various block operations.

   You can undo changes at any time by choosing ↶. Another possibility is to exit the Workflow Builder without saving your work and to call it again.

Copying steps to the UNTIL loop

The step *Check notification of absence*, which follows the outcome *Request completed*, is the first step in a new block. To implement the resubmission, the workflow must send the notification of absence to this step after it has been revised by the requester. You have used an UNTIL loop for this purpose. The step *Check notification of absence* must be located within this loop. In order to do so, first cut the block that begins with this step. Then paste the cut block in the UNTIL loop.

1. Select the step *Check notification of absence*.

2. Choose ✂ to cut the block.
3. Select the *Undefined* outcome in the UNTIL loop and choose 📋.
4. Choose ▤ to align the graphic.
5. Choose 🏞 to view the entire workflow definition in the workflow area on the screen.

   You will notice that your workflow definition now contains steps within the UNTIL loop.
6. Choose ✨.

   The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

Result

The integration of the UNTIL loop is now complete. The workflow up to this point, however, delivers an incorrect result, since the *Flag* container element is not assigned a value during execution. The result of the evaluated condition in the UNTIL loop is always *false* and the workflow can never be completed. To ensure that the workflow is executed correctly, you will add further container operations to the workflow definition in the next unit.

Unit 13: Inserting Container Operations

Use

You have added an UNTIL loop to your workflow definition. To ensure that this loop condition is executed correctly, you have to add *container operations* to your workflow definition that change the value of the *flag* container element in accordance with how the workflow is executed.

Prerequisites

You have added an UNTIL loop to your workflow definition.

Procedure

You start the Workflow Builder and open your workflow definition in Change mode. Proceed as described in *Unit 7: Including "Check Notification of Absence" in the Workflow Definition (page 25)*.

Integrating container operations as a step

You can manipulate the individual elements of the workflow container using a *container operation* type step.

You need to insert a container operation at three different points in your workflow definition.

- After the outcome *Request revised*

  At this point, you assign the workflow container element *Flag* the value *Z*. The comparison in the UNTIL loop then returns the result *false*. In this case, the workflow continues via the outcome *Flag does not have value X* and the request is *resubmitted* to the superior's *Business Workplace*.
- After the outcome *Withdraw*

  At this point, you assign the workflow container element *Flag* the value *X*. The comparison in the UNTIL loop returns the result *true*. The loop is exited.
- After the outcome *Approved:*

  You assign the container element *Flag* the value *X*. The comparison in the UNTIL loop returns the result *true*. The loop is exited.

Container operation after the outcome *revised* event

1. Choose under step type. The cursor changes shape.
2. Position the cursor on the *Undefined* step that follows the outcome *Request revised* and click.

   By doing so, you add a *container operation* to your workflow definition. The step definition is shown in the right-hand part of the screen.
3. Enter the following basic data:

   Step name: **`Set flag to Z`**

   Outcome name: **`Flag = Z`**
4. Specify the following for the operation:

   Result element: `Flag` (select using F4 input help)

   Expression: `Z`

   The input fields for the operator and second expression must be empty.

   When this step is executed, the *Flag* container element in the workflow container receives the value ***Z***. This value remains as long is it is not changed explicitly.
5. Choose ✓.

Container operation after the *Withdraw* outcome

1. Choose under step type. The cursor changes shape.
2. Position the cursor on the *Undefined* step that follows the outcome *Withdraw* and click.

   By doing so, you add a *container operation* to your workflow definition. The step definition is shown in the right-hand part of the screen.
3. Enter the following basic data:

   Step name: **`Set flag to X`**

   Outcome name: **`Flag = X`**
4. Specify the following for the operation:

   Result element: `Flag` (select using F4 input help)

   Expression: **`X`**

   The input fields for the operator and second expression must be empty.
5. Choose ✓.

Container operation after the *Approved* outcome

Since this container operation is identical to the one created above, you can simply copy it.

1. Select the container operation *Set flag to X* after the *Withdraw* outcome and choose .
2. Select the *Undefined* step that follows the *Approved* event and choose .

Concluding step definition

Choose .

The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

Result

You have now extended the workflow definition in such a way that the results of the condition evaluated in the UNTIL loop are meaningful. You can now subject your workflow definition to another test. Carry out the test as described in *Unit 10: Third Test (page 32)*.

Unit 14: Changing Staff Assignments

Use

In the tests you have carried out so far you acted both as the requester and as the superior.

In this unit, you will change the organizational plan in such a way that the requester and superior are two different R/3 users.

➡

You can change the organizational plan without modifying the workflow definition.

Procedure

You enter a different user as the superior in the staff assignments of your organizational plan. Ask a colleague whether they would like to take part in this test. Or have yourself a second user created.

1. Choose *Tools* → *SAP Business Workflow* → *Development* → *Definition tools* → *Organizational Management* → *Organizational plan* → *Change.*

   This takes you to the *Change Organization and Staffing (Workflow)* screen.

2. Search for your head of department position in the search area. The positions that were found are displayed in the selection area. Open yours by double-clicking on the entry in the change mode.

Deleting position holders

3. Select the row in the overview area in which the assignment to the user is entered.

4. Choose the arrow ▤ on the right next to the 🗑 and then *Assignment*.

   The position of the department head is no longer occupied.

Reassigning position holders

5. Choose *User* in the search area and search for the user whom you want to assign to the position.

   All of the users that match the search criteria are displayed in the selection area.

6. Select the user whom you want to assign to the position and drag him or her to the line *position: head of department (ini)* in the overview area. Confirm the adaptations of the relationship period.

7. Choose 💾.

   This completes the changes to the staff assignments. The two positions are held by different persons.

Result

You have changed the staff assignments by changing the assignment of users to a position. You can check this by displaying the staff assignments. Proceed as described in *Unit 1: Organizational Plan (page 10)*.

You can also start with *Unit 16: Fourth Test (page 41)* directly after this unit. The recipients of the work item for the step *Check notification of absence* are then determined from the possible agents for the single-step task. You assigned the job of the head of department as the possible agents of this task in *Unit 6: Create task “Check Notification of Absence” (page 23)*.

Unit 15: Using a Role

Use

In the organizational plan you have created up to now, there are two *positions*, each of which is described by a *job*. Each position must be linked to a different user.

The job of department manager is linked to the department manager position. The job of department manager, in turn, is linked to the task *Check notification of absence*. In a more extensive organizational plan containing several positions for department managers, each position can be linked to the job of department manager.

These links ensure that each user who is assigned to a position as department manager is one of the possible agents of the task *Check notification of absence*. If you want to assign all of the department managers as possible agents of a different task, all you have to do is link this task to the job of department manager. You do not need to make any detailed changes.

> You can add a further organizational unit to your organizational plan by creating a new position as department manager and linking it to a third user. When you link this new position to the existing job of department manager, the third user is automatically added to the list of possible agents for the task *Check notification absence*.

If you then create a notification of absence, it is offered to all of the possible agents for processing. This is clearly undesirable, since the notification of absence should only be presented to your own superior. This superior is the head of department who has the chief position in the organizational unit to which you too belong.

To ensure that the notification of absence is not submitted to all of the *possible agents*, you must define *agents responsible* in your step definition. All of the agents responsible who are also possible agents for this task then become recipients of the work item for the task. In this example, the agents responsible must be defined dynamically, since the relevant agent depends on the requester (person initiating the workflow). To do so, you use a *role*.

The role *Superior of* is shipped as standard with the R/3 System. This role first establishes a user’s position and the relevant organizational unit, and then finds the chief position in this organizational unit.

> This role only works correctly if you are **not** your own superior. You should only work through this unit if you have completed *Unit 14: Changing Staff Assignments (page 39)* and adapted your organizational plan accordingly.

By using roles to assign responsibilities, you do not need to have any specific knowledge of the organizational plan when you define a workflow definition. When the role is resolved at runtime, all of the necessary information is provided via the binding definition from the workflow container to the role container.

Procedure

You start the Workflow Builder and open your workflow definition in Change mode. Proceed as described in *Unit 7: Including "Check Notification of Absence" in the Workflow Definition (page 25)*.

Selecting a role

1. Open the step definition for the activity *Check notification of absence* by double-clicking the activity icon in the workflow area on the screen.
2. Choose *Superior of workflow initiator* under Agents.

   The system then enters the role ID (`00000168`) in the *Role* selection field and in the input field.

   ➡

   The binding between the workflow container and the role container is created automatically here. If you choose *Role* in the checkbox and enter the role in the input field using the F4 input help, you must define the binding manually.

Concluding step definition

5. Choose to check the step definition.
6. Choose ✓ to exit the step definition.
7. Choose .

   The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

Result

You can now subject your workflow definition to another test.

Unit 16: Fourth Test

Use

In the organizational plan that you changed in *Unit 14: Adjusting the Staff Assignments (page 39)*, your colleague has now assumed the tasks of the superior. As the requester, you must start the workflow. The work item for checking the request will then be sent to the Business Workplace of the superior.

Procedure

Procedure as requester

1. Choose *Tools* → *Business Workflow* → *Development* → *Runtime Tools* → *Start Workflow.*

   All of the workflows that you are allowed to start are listed on the left-hand side of the screen. A description of the selected workflow is shown on the right.

2. Select your workflow and choose *Start.*

   The work item for creating the notification of absence is proposed to you immediately for processing.

3. Fill out the form and choose 💾.

You have now submitted your request and your superior must decide whether to approve it. Since you are not now the recipient of the next work item, *advancing with immediate dialog* does not have any effect.

Procedure as superior

1. Choose *Office* → *Workplace* to start the Business Workplace.

2. Open the folder *Inbox* → *Workflow*.

   The request awaiting approval is displayed as a work item in this folder.

3. Select the work item.

   The task description of the single-step task is displayed in the work item preview. As you can see, the expressions used in the task description have been replaced with the current content.

4. Choose ✓ but do not approve the request.

5. Choose ↩.

   You are now in your Business Workplace. The work item you have just executed is no longer displayed in the list.

Procedure as requester

6. Choose *Office* → *Workplace* to start the Business Workplace.

1. Open the folder *Inbox* → *Workflow*.

   This folder contains a work item awaiting a user decision.

7. Select the work item and choose ✓.

8. Select *Revise application* in the user decision dialog box.

   Since the flag *advancing with dialog* is set in the step definition for the activity *Revise notification of absence*, the request is immediately presented to you for revision. If this flag is not set, you must execute the work item from your Business Workplace.

9. Change the request and choose 💾 and ↩.

   You are now in your Business Workplace. The work item you have just executed is no longer displayed in the list.

Procedure as superior

If you have not closed your Business Workplace, choose 🔄 to refresh the work item display.

You have received the request from the requester again. They can proceed as above to approve the request or reject it again. This workflow does not have a modeled response to a situation in which the requester and superior do not reach agreement. The workflow definition can be easily extended for this situation (for example, automatic notification of a third person if the request is exchanged between the employee and head of department several times).

Reporting and analysis with the workflow log

You should start the workflow several times together with your colleague and try out the alternatives. These alternatives can then be analyzed via work item analysis or the workflow outbox.

Use the workflow log in this unit. If the workflow you want to analyze is displayed in the work item analysis or in the workflow outbox, select the entry and choose 📜. The workflow log provides information about all the stages of processing. This includes the following:

- Which steps were executed?
- Who was the agent?
- When was processing carried out?
- What was the result of the processing?

If the workflow log is displayed with ActiveX, you can view the various steps in the workflow in the form of a graphic. To do so, choose ▦. For further information on the workflow log, please refer to *Workflow Log: Standard View*, and on the ActiveX view under *Personal Settings*.

Unit 17: Including "Send Notification" into the Workflow Definition

Use

You have created a complete workflow with which a notification of absence can be created, revised and approved.

In this unit, you will insert a step in the workflow definition that sends a notification to the requester after the request has been approved.

Procedure

You start the Workflow Builder and open your workflow definition in Change mode. Proceed as described in *Unit 7: Including "Check Notification of Absence" in the Workflow Definition (page 25)*.

A separate step type is available for the sending of mails.

1. Select the outcome *Approved* of the step *Check notification of absence* and choose *Create* in the context menu.

   The dialog box *Select step* is displayed.

2. Choose *Send mail*.

   The step definition is then displayed. The input fields required for sending a mail are found on the tab page *Mail*.

3. Due to the fact that the mail is to be sent to the requester (the workflow initiator), you do not need to change the standard settings. Inspite of this, choose the entry *Workflow initiator* in the field *Recipient type*.

   When entering a recipient, you must first select a recipient type. The entries have the following meanings:

   - *Workflow initiator*: Choose this entry if only the workflow initiator is to receive a mail. The recipient type is automatically set to *Organizational object* and the expression `&_WF_INITIATOR&` is entered in the field *Recipient*. These entries are the standard settings.
   - *Organizational object*: You must enter an *Organizational object* in the field *Recipient* or choose an expression that contains one or several organizational objects. Use the F4 input help when making the entries. When you enter an expression, the system automatically creates a container element in the task container and adds the binding definition.
   - *E-Mail address*: In the field *Recipient,* enter either an e-mail address directly or by using the F4 input help, select a container element from the workflow container that contains one or several Internet addresses at runtime. The corresponding expression is entered. The system automatically creates a container element in the task container and adds the binding definition.

     When you change the recipient type for the first time, the system automatically creates the necessary task for this step. You must also enter the abbreviation and the name of this task.

4. Enter the basic data for the task:

   Abbreviation: <*ini*_**nottxt**>

   Name: <**Notification request accepted (***ini***)**>

   In the *Create Object Directory Entry* dialog box, choose *Local object*.

5. Enter the following text for the mail subject:

```
<Your leave request was reserved>
```

6. Enter the following text that the system is to send to the requester if the request is approved:

```
<Dear colleague,

Your leave request number &AbsenceForm.Number& from
&AbsenceForm.CreateDate& was approved on
&AbsenceForm.ApprovDate& by &AbsenceForm.Approver.Name&.>
```

   If you want to include an expression that is to be replaced at runtime, choose ▦. Choose the expression from the container elements of the workflow container. The system automatically creates a container element in the task container and adds the binding definition.

7. Choose ✓.

Activating workflow definition

8. Choose ✨.

   The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

Test

Test your workflow definition in the usual way.

You receive a mail when your “superior” has approved your "notification of absence". You find the mail in your Business Workplace in the folder *Inbox* → *Unread documents*.

Unit 18: Work Item Attachments

Use

The superior is to have the opportunity to add an attachment to their work item for checking the notification of absence. This attachment is a document (any SAP*script* or PC document), which is then made available to all subsequent recipients in the workflow. The superior can use this to give a full explanation for a rejection.

An attachment can be added to a workflow **before** the workflow is actually executed in the Business Workplace.

To create an attachment **after** the workflow has been executed, you have to change the definition of the associated standard task.

To enable the superior to create an attachment after the notification of absence has been checked, you must set the *Confirm end of processing* flag for the task *Check notification of absence*. This property means that the agent has to confirm the end of processing of the work item explicitly. The agent can also add an attachment before this confirmation.

For further information about using documents in workflows, refer to *Document Processing* in the *SAP Business Workflow* documentation.

Procedure

You start the Workflow Builder and open your workflow definition in Change mode. Proceed as described in *Unit 7: Including "Check Notification of Absence" in the Workflow Definition (page 25)*.

Changing the task "check notification of absence"

1. Open the step definition for the step *Check notification of absence* by double-clicking it.
2. Double-click the task ID to open the task definition.

   You go to the screen *Standard task: Display*.

3. Choose ✎ to switch to change mode.
4. Set the *Confirm end of processing* flag under *Execution*.
5. Choose 💾 and then ⮌.

Concluding step definition

1. Choose ⧉ to check the step definition.
2. Choose ✓ to exit the step definition.
3. Choose ✳.

   The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

Result

Execution of the work item for the step *Check notification of absence* must be confirmed expressly. This enables you to create an attachment after the request has been approved or rejected.

> ➡
>
> You can also create attachments **while** a work item is being executed. An example of this is provided in the demo workflows. Choose *Tools* → *Business Workflow* → *Development* → *Environment* → *Start demo workflows.* Start the demo workflow *Demo for WI execution with WF toolbox*. Further information is provided on the right-hand side of the screen.

Unit 19: Fifth Test

Use

After you have executed the step *Check notification of absence*, you must confirm the end of processing.

Procedure

Procedure as requester

4. Choose *Office* → *Start Workflow*.

   All of the workflows that you are allowed to start are listed on the left-hand side of the screen. A description of the selected workflow is shown on the right.

5. Select your workflow and choose *Start.*

   The work item for creating the notification of absence is proposed to you immediately for processing.

6. Fill out the form and choose 💾.

You have now submitted your request and your colleague must decide whether to approve it. Since you are not now the agent of the next activity, *advancing with immediate dialog* does not have any effect.

Procedure as superior

10. Choose *Office* → *Workplace* to start the Business Workplace.

11. Open the folder *Inbox* → *Workflow*.

    The request awaiting approval is displayed as a work item in this folder.

12. Select the work item.

    The task description of the task is displayed in the work item preview. As you can see, the expressions used in the task description have been replaced with the current content.

13. Choose ⏱ but do not approve the request.

14. Choose ⬅.

    The dialog box for confirming the end of processing appears.

15. Choose the arrow ▾ on the right next to the 📎 and then *Create attachment*.

    The *Create Document: Header* dialog box is then displayed.

16. Choose `RAW` as the *Type* and enter a *title*:

    Title: `Reason for rejecting the request`

17. Choose ✓.

    The SAPscript editor is then displayed.

18. Enter your reason and choose 💾. Choose ⬅.

    You then return to the dialog box for *User confirmation* of end of processing. The attachment is displayed under *Objects and attachments*.

19. Choose ✓ *Complete work item.*

    You are now in your Business Workplace and the work item you have just processed has been removed from the worklist.

Procedure as requester

2. Choose *Office* → *Workplace* to start the Business Workplace.

3. Open the folder *Inbox* → *Workflow*.

    This folder contains a work item awaiting a user decision. The 📎 symbol in the *Attachment* column indicates that the work item has an attachment.

4. Select the work item.

    All of the attachments for this work item are displayed in the work item preview.

5. Choose the attachment in the work item preview.

    The attachment is then opened and displayed in a new screen.

6. Execute the work item.

Further Procedure

Continue processing the workflow and analyze it using the workflow log.

Unit 20: Monitoring Missed Deadlines

Use

When defining steps that require dialog with the user, you can instruct the workflow system to monitor certain deadlines. The workflow system can monitor the following types of deadline:

- Requested start
- Latest start
- Requested end
- Latest end

This unit explains how missed deadlines are usually handled: The standard system response to an exceeded deadline is to notify a *recipient of the message for missed deadlines* by sending a *work item for missed deadlines* to his or her Business Workplace.

For further information, please refer to *Maintaining the Deadline Tabs* in the *SAP Business Workflow* documentation.

In this unit, the workflow is to be extended so that the requester should be notified if the superior has not made the decision about the leave request after 10 minutes.

Procedure

You start the Workflow Builder and open your workflow definition in Change mode. Proceed as described in *Unit 7: Including "Check Notification of Absence" in the Workflow Definition (page 25)*.

Determining the recipient of the message for missed deadlines

1. Open the step definition for the step *Check notification of absence* by double-clicking it.
2. Go to the *Latest end* tab.
3. Choose the entry *Work item creation* as the *reference time*.

   The active deadline is indicated by the 🔔 symbol in the tab index of the relevant tab.
4. Enter the value `10` in the field **+** and choose *Minutes* as the time unit.
5. On the *Display text* tab, choose the entry *Workflow initiator (expression)* as the recipient.

   As a result, the entry *_WF_Initiator* is automatically copied to the adjacent field and the entry in the checkbox changes to *Expression*.

   If a deadline is exceeded, the text visible in the display is automatically copied to the work item for missed deadlines. You will now change this text in the task definition.
6. Choose the highlighted text in the display to open the task definition.
7. Choose ✎ and go to the *Description* tab.
8. Choose the text type `Text for latest end` and then 📝.

   The *SAPscript* text editor is then launched.
9. Enter the following text:

   ```
   Your notification of absence no. &_WI_OBJECT_ID.NUMBER& from
   &_WI_OBJECT_ID.CREATEDATE& has not been approved yet.

   You should remind your superior.
   ```

   If you want to insert an expression from the task container as a text variable, choose *Include → Expression.* Then choose the relevant expression in the *Please choose an expression* dialog box by double-clicking the *Container* node.
10. Choose ⬅.
11. Choose 💾 and then ⬅.

Concluding step definition

9. Choose ✓ to check the step definition.

10. Choose ✓ to exit the step definition.

11. Choose 🪄.

    The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

Test

7. Choose *Office* → *Start Workflow*.

   All of the workflows that you are allowed to start are listed on the left-hand side of the screen. A description of the selected workflow is shown on the right.

8. Select your workflow and choose *Start.*

   The work item for creating the notification of absence is proposed to you immediately for processing.

9. Fill out the form and choose 💾.

You have submitted your request and should now allow the deadline to expire.

Once the latest end passes, a work item for missed deadlines appears in your Business Workplace notifying you of the missed deadline. This work item is a deadline monitoring work item, for which there is no executable method. You can go from the display of this work item to the work item display of the late work item.

Notes on extensions

The *recipient of a message for missed deadlines* is notified with a work item. The notification text is stored in the task definition of the monitored task.

If you want to customize the notification, you can define you own *Text for latest end* on the *Description* tab in the task definition for the single-step task *Check notification of absence*. This text can contain text variables that are replaced at execution time.

Tutorial: Maintaining the Organizational Plan

Purpose

This tutorial shows you how to create the framework for an organizational plan quickly and effectively. An organizational plan describes how an *employee* is assigned within the organization. You can store the employees who are responsible for carrying out individual business activities in your organizational plan.

The organizational plan is maintained for specific clients in the *Organizational Management* component. An organizational plan created for HR purposes can also be used in *SAP Business Workflow* as long as the workflow functionality and the HR application are in the same client.

Creating your organizational plan also creates the prerequisites for the Workflow System to determine the appropriate agents of a work item at runtime.

Process Flow

The individual units of this tutorial contain step-by-step instructions on how to create the basic framework of your organizational plan in order to reflect the structure and HR environment of your enterprise. The following items, each of which represents a separate unit, provide you with an overview of the tutorial procedure.

1. You create a *root organizational unit*. Once you have done so, you create the subordinate *organizational units*.

2. You create *jobs*, provided that they do not already exist in your job index. You also create *positions*, which represent the specific instances of jobs in your enterprise. You assign R/3 users as position holders. You can also indicate that a position is a chief position for an organizational unit.
3. You assign tasks to the objects you have created. These objects can be jobs, positions, or organizational units. You can use tasks to describe jobs and positions.

Examples are included to help you work through the individual units. These are intended to illustrate the step-by-step procedure for creating your organizational plan.

For further information, see *Organizational Management* (see SAP Library).

Unit 1: Creating an Organizational Structure

An organizational plan consists of the organizational units that exist in the company. The organizational units are related one to the other in a hierarchical reporting structure. However, they can also be created independently of each other.

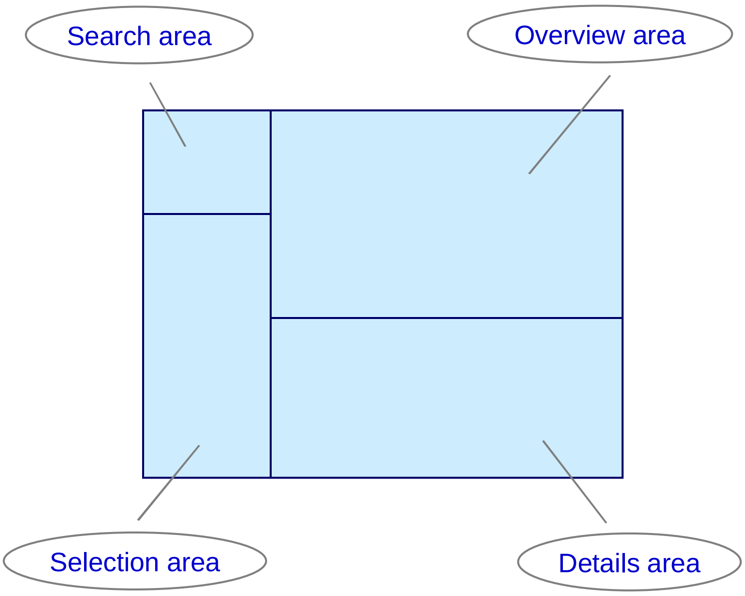
To create a new organizational plan, you create a root organizational unit. This is the highest unit within the organizational structure, such as the Executive Board. You then set up the organizational structure, starting at the root organizational unit and working downwards.

Procedure

Creating a Root Organizational Unit

5. Choose *Tools* → *Business Workflow* → *Development* → *Definition Tools* → *Organizational Management* → *Organizational Plan* → *Create*.
6. Confirm the validity period proposed in the dialog box *Creating a Root Organizational Unit*.

   This takes you to the *Create Organization and Staffing (Workflow)* screen. This user interface is divided into four screen areas:



7. On the Basic Data tab in the details area, enter an abbreviation and a name in the *Organizational unit* input fields for your root organizational unit.

   Abbreviation: **<*ini*_org>**

Name: **<**`Organizational plan`**(***ini***)>**

8. Choose 💾.

The root organizational unit you have just created is then selected in the overview area.

Creating further organizational units

Starting from the root organizational unit, you create subordinate organizational units. In this unit, you will create two further organizational units.

1. Choose ⊞.

   A new organizational unit is now created below the root organizational unit.

2. Open the details view of the new organizational unit by double-clicking the corresponding line in the display area.

3. In the details area, enter an abbreviation and a description for the *Organizational unit* in the input fields.

   Abbreviation: **<***ini*`_org_sa`**>**

   Name: **<**`Sales` **(***ini***)** >

   ➡

   When determining the validity period of objects and relationships, the system chooses the validity period of the superordinate object as standard.

   If you want to change the validity period for the organizational units to be created and their relationships, change the values in the corresponding input fields in the details area.

4. Select your root organizational unit.

5. Choose 🗋.

   A new organizational unit is now created below the root organizational unit.

6. In the details area, enter an abbreviation and a description for the *Organizational unit* in the input fields.

   Abbreviation: **<***ini*`_org_rd`**>**

   Name: **<**`Research and development` **(***ini***)** >

7. Choose 💾.

   ➡

   You can also include existing organizational units in your organizational plan. To do so, drag them from the selection area to your organizational plan.

Reassigning organizational units

If changes are made to the organizational structure of your company, you can reassign the organizational units involved.

➡

If you reassign organizational units, please note the following:

- If you change the assignment of an organizational unit, you also change the associated relationship records.
- If the validity period of the new relationship overlaps with the validity period of the original relationship, the system delimits the validity period of the original relationship accordingly.

- Root objects cannot be reassigned. In your example, the root object has the abbreviation ***ini*_org**.

1. Select the organizational unit that you want to reassign in the selection area.
2. Drag it to the organizational unit under which it is to be assigned.

Delimiting organizational units

You delimit organizational units if you want to change their validity period, that is, bring forward the validity end date.

> When you delimit objects (in this case, organizational units), all of the associated links are automatically delimited at the same time.

1. Double-click the organizational unit to open the details view.
2. Switch to display with periods. Choose .
3. Change the entry in the *Valid to* field.

> You can only change a period if you also change a different entry of the organizational unit at the same time. For example, change the description.

Unit 2: Creating Staff Assignments

The *staff assignments* for each organizational unit are maintained. The organizational unit is assigned *positions* in these staff assignments. A position is derived from a descriptive *job* and assigned to one or more users in your company.

> All of the positions must be linked to jobs. The positions inherit the attributes and properties of the job.

In this unit, you will create the staff assignments for each of your organizational units. To do so, you will:

1. Create positions
2. Classify a position as a chief position.
3. Assign jobs to positions
4. Assign users to positions

Procedure

Choose *Tools* → *Business Workflow* → *Development* → *Definition Tools* → *Organizational Management* → *Organizational Plan* → *Change*. The *Change Organization and Staffing (Workflow)* screen is then displayed.

Create jobs

You must assign every position a job that contains general functions and tasks and which passes on the position. Jobs are normally defined centrally for an organizational plan in the job index. When positions are created, the corresponding job must only be assigned. For this tutorial, you create the positions yourself.

1. Choose *Edit* → *Create job*.

   The dialog box *Create jobs* is displayed. The lower area contains a list of existing jobs and the upper area contains an input table in which you can create new jobs by entering abbreviations and names.

4. In the input table, enter an abbreviation and a name for each of the new jobs.

    Job - head of department:

    Abbreviation: <*ini*_hd_C>

    Name: <Head of department job (*ini*)>

    Job administrator:

    Abbreviation: <*ini*_ad_C>

    Name: <Administrator job (*ini*)>

    ➡

    All new jobs receive a validity period from the current date to the 31.12.9999 as standard.

5. Choose ✓.

Opening staff assignments for an organizational unit

1. If the root organizational unit (*ini*_org) that you created in unit 1 is not open for processing, choose organizational unit in the search area, enter a search term and select your organizational unit from the organizational units found by double-clicking in the selection area.

2. In order to open the staff assignments for an organizational unit, select the organizational unit in the overview area and select the arrow ▤ on the right next to the ▣ and then *Staff assignments (list)*.

    The staff assignments for the organizational unit selected are now displayed in the overview screen.

Create positions

You create positions for your organizational unit in the associated staff assignments.

1. Open the staff assignments for the organizational unit **Research and development (*ini*)**.

    In this organizational unit, assign a position for an administrator and a position for a head of department.

2. Choose ⊞.

    The new position is now displayed in the staff assignments. You can edit all relevant data for the position in the details area.

    ➡

    The details view for positions is available in the key date mode and the periods mode. You can toggle between the two modes with the ▣ and ▣▣ buttons.

3. In the key date mode, enter the abbreviation and name of the new position in the *Position* fields on the *Basic data* tab:

    Abbreviation: <*ini*_ad_S_rd>

    Name: <Administrator position: (*ini*) R&D>

    ➡

    You can change the validity period of the position in periods mode.

4. Repeat steps 2 and 3. Use the following data:

    Abbreviation: <*ini*_hd_S_rd>

    Name: <Head of department job (*ini*) R&D>

5. Set the *Head of own organizational unit* indicator. Confirm the information using the adaptations of the relationship period.

    Chief positions are marked 👤 in the staff assignments.

    ➡

    You must define chief positions if you want to use *roles* to determine recipients in workflows that determine a user's superior.

6. Choose 💾 and then the arrow ▤ on the right next to ⧉ and *Organizational Structure*.

    The organizational unit **Research and Development (*ini*)**is then displayed.

7. Choose 品 to switch to editing of your root organizational unit.

8. Repeat steps 1 - 7 for the organizational unit **sales (*ini*).** Use the following abbreviations and names for the positions:

    Abbreviation: <*ini*`_ad_S_sa`>

    Name: <`Administrator position:` `(`*ini*`)` `Sales`>

    Abbreviation: <*ini*`_hd_S_sa`>

    Name: <`Head of department job` `(`*ini*`)` `Sales`>

    Set the *Head of own organizational unit* indicator for the head of department job sales.

Assign jobs to positions

You assign a job to every position.

1. Display your created position in the selection area. Choose the entry *Position* in the search area and enter the abbreviation you used (`ini`) as the search term.

    All positions created in this unit are displayed in the selection area.

2. Select the **Administrator position (*ini*) R&D** for editing by double-clicking.

    The position and the organizational objects assigned to you (job, organizational unit, user) are displayed in the overview area.

3. Display your created job in the selection area. Choose the entry *job* in the search area and enter the abbreviation you used (`ini`) as the search term.

    All jobs created in this unit are displayed in the selection area.

4. Select the job **Administrator job (*ini*)** in the selection area.

5. Drag the job to the position in the overview area.

6. Choose 💾.

    The job **administrator job (*ini*)** is assigned the position **administrator position (*ini*) R&D**.

7. Repeat steps 1 to 6 for the **head of department job (*ini*) R&D** and assign it the job **head of department job (*ini*)**

8. Repeat steps 1 to 6 for the **head of department job (*ini*) Sales** and assign it the job **head of department job (*ini*).**

9. Repeat steps 1 to 6 for the **administrator position (*ini*) sales** and assign it the job **administrator job (*ini*).**

Assigning users to positions

When you assign users to positions, you define the *holders* of these positions.

➡

This assignment is necessary for workflow applications if you use roles or objects in the Organizational Management to determine recipients.

If you use Personnel Administration (PA), the system recognizes *employees* who are assigned directly to positions. To ensure in this tutorial that the R/3 user can be established as the agent starting from the employee, employees must have a relationship to R/3 users.

If you do not use Personnel Administration, *users* are assigned directly to positions. At workflow runtime, they are established directly as the agents of particular single-step tasks.

You can assign the user to the position in the staff assignments or in the single processing of a position. In this tutorial you assign the users using the single processing of a position.

1. Display your created position in the selection area. Choose the entry *Position* in the search area and enter the abbreviation you used (***ini***) as the search term.

   All positions created in this unit are displayed in the selection area.

2. Select the position in the selection area for which you want to assign a user by double-clicking.

   The position and the organizational objects assigned to you (job, organizational unit, user) are displayed in the overview area.

3. Display the user assigned to in the selection area. Choose the entry *user* in the search area and enter an appropriate search term.

   All of the users that match your search criteria are displayed.

4. Select the user whom you want to assign in the selection area.

5. Drag the user to the position in the overview area.

   ➡

   Users can occupy a position either in full or in part. This depends on the working hours assigned to the position, and on the working capacity of the person or user.

   The staffing percentage refers to the working capacity of a person or user assigned to a position.

6. Choose 💾.

Tutorial: Event Creation During Status Changes

Purpose

This tutorial covers the following topics:

- **Flexible event creation during status changes**

  You can generate an event for a status change in order to respond to a certain business circumstance.

- **Send pre-formulated documents as mail**

  You will learn how to send a text as a message (for example, with a note) to a receiver as a workflow step.

- **Attributes with object reference and multi-level expressions**

The topics are linked to an integrated scenario whose creation you can follow step by step in individual units.

The tutorial is designed for **customers** and **consultants** who want to extend the scenarios provided by SAP. The explanations and working methods are consciously covered in great depth; the fact that existing knowledge is repeated has been taken into account in this case.

⇨

It is of central importance to the scenario (and this tutorial) that the changes you make are logged with a system or user status in an application component. You can only transfer what you learn in this tutorial to other scenarios if you make this setting.

Process Flow

In *Unit 1: Testing the Existing Application Functionality (page 55)*, you will release a production order in the R/3 System.

Identifying and modifying object types

The object type that is important with respect to the scenario must be identified. This is the object type **BUS2005** *(Production order*).

⇨

This object type is supplied as standard in the *Business Object Repository*.

You should use this object type in the manner for which it was intended. For this scenario, however, you have to extend it to include the **released** *event*.

In *Unit 2: Identifying and Extending Object Types (page 57)*, you will extend the definition of the object type.

Creating an event

Setting the status **REL** for the status object type **ORH** should be publicized with the event **released** and therefore made known across the entire system. You must make sure that the event is generated.

The creation of this event is a basic requirement for executing the workflow.

In *Unit 4: Flexible Event Creation (page 62)*, you will link the creation of events to the setting of a specific status.

Agent determination

The recipients of the mail are all employees in an organizational unit. This organizational unit is only created to group together the recipients or agents. It thus takes on the characteristics of a distribution list.

In *Unit 5: Finding Agents - Organizational Plan (page 63)*, you will create the organizational unit and assign users to it.

Defining a workflow

In *Unit 6: Defining a Workflow (page 64)* you will create a workflow for which the event **released** is defined as the triggering event. In order to do so, you define a binding from the event container to the workflow container that provides the reference to the released production order.

Send text as mail

In *Unit 7: Editing Workflow Definitions (page 66)* you will define a task with which an e-mail is automatically sent. To define this task, you will use a wizard.

Unit 1: Testing the Existing Application Functionality

Use

This scenario is based on the existing application functionality of the **PP** component *(Production Planning and Control)*.

Production orders are an essential part of the Production Planning System. They are used to control and monitor production within a plant, as well as being a controlling instrument within cost accounting.

The internal activities are handled via production orders. A production order determines which material should be produced when, where, and how. It also determines which resources must be used and how the order costs are to be offset.

When you release a production order for production, the system status **REL** (*released*) is set for the status object type **ORH** (*PP,PM: Order header)* (=activated).

Procedure

The following steps can be executed as described above within the standard functionality of the application component **PP** in a test or demonstration system (for example, *IDES***)**.

Using the proposed input values, you can follow this example in the IDES system!

Create production order

A production order that is created with reference to an available production order is used as the basis for the scenario.

1. Choose *Logistics* → *Production* → *Production control* → *Order* → *Create* → *With material.*

   The *Production Order Create: Initial Screen* is then displayed.

2. Choose any order in the *Order* field in the *Copy from* box. Use the *F4 input help*.

3. Choose ✓.

   The *Production Order Create: Header* screen is then displayed.

4. For *Basic Dates Finish*, enter a date that is a few weeks in the future.

5. Choose 💾 and note the number displayed. The number is assigned automatically.

   The *Production Order Create: Initial Screen* is then displayed.

Display status information for production order

6. Choose *Order* → *Display* and enter the number of the production order you created in the *Order* field.

7. Click the *Display overview* flag and choose ✓.

   The *Production Order Display: Header* screen then appears.

8. Note the entry in the *Plant* field.

   You will need this entry to release the order later on in this unit.

9. Choose ℹ.

   A list of the system statuses that are set is then displayed. The status **REL** *(released*) is not yet set.

   However, the release is part of the business transactions that are allowed as the next step. An overview of the possible and permitted order status is shown on the *Business Processes* tab.

10. Choose *Extras* → *Overview.*

    The *Display Status: Overview* dialog box then appears. From the entries in the displayed popup, you can see that the statuses refer to the status object type **ORH** (*production order*). This information is important for creating events.

Release production order

1. Choose *Logistics* → *Production* → *Production control* → *Order* → *Release.*

2. Enter the plant in the *Plant* field and the order number in the *Order* field.
3. Delete the entries in the date fields.
4. Choose .

    The *Release Production Orders: List* screen is then displayed.

5. Select the order and choose *Order.*

    In the *Status* column, you see that the status **REL** is set.

Unit 2: Identifying and Extending Object Types

Object Type BUS2005 *(Production Order*)

Execute the following steps, in order to learn about the object type **BUS2005**

1. Choose *Tools* → *SAP Business Workflow* → *Development* → *Definition tools* → *Business Object Builder.*
2. Enter `BUS2005` in the *Object/interface type* field and choose .

    The screen *Display object type BUS2005* is then displayed.

If the folder symbol in front of one of the entries *Interfaces, Key fields, Attributes, Methods* or *Events* contains a plus sign, elements are available and defined for this object type component.

Open the folders to display an overview of the existing elements.

Interface

In addition to the IFSAP interface, which is supported by every object type, this object type supports three further interfaces.

Key fields

3. Double-click the entry **ProductionOrder.Number** to display detailed information on this key field.

    The *key field* of the object type **BUS2005** is **Number** (*Order number*) and refers to the table field **AUFK-AUFNR**. The key field of an object of the type **BUS2005** identifies it uniquely and enables read access to its attributes.

Attributes

Attributes are defined for the object type **BUS2005**.

The two database field attributes **LastChangedBy** (*name of person who made last change*) and **ChangeDate** (*date of last change*) are used later on, since they are to be included in the request text (along with the key field).

Methods

The **Display** method is used in this scenario.

Enhancements to the Object Type BUS2005

The definition of the object type **BUS2005** is incomplete and cannot be used in this scenario. In other words, this object type must be extended.

You need an *event* to publish the change to the material master data throughout the system. All events which are to be used must be defined beforehand as elements in their object type.

Create Subtype for Available Object Type

Since you **cannot** make any direct changes to the SAP object type **BUS2005**, first create a customer-specific object type as a *sub-type* of this object type. This sub-type **inherits** all attributes and methods of its super type.

1. Choose ⮌.

   You are now in the *Business Object Builder: Initial Screen.* The object type **BUS2005** is still entered.

2. Choose 🗋 *Subtype*.

   The dialog box *Create object type* appears.

3. Enter the following data:

   Object type: `Zini_2005`

   Object name: `<Production order>`

   Name: `<Production order>`

   Description: `<Production order extension BUS2005>`

   Program: `Zini_2005`

   Application: `Z`

4. Choose 💾 and save the subtype as a local object.

   You are now on the *Change Object Type Zini_2005* screen and can edit the new object type you created.

   Ensure that this object type has inherited all methods and attributes from the super-type **BUS2005**. Inherited elements are marked in red.

Creating an Event

The status change should be broadcast with an event with the name **released1**.

This event can only be generated if it is defined for the object type. This is not yet the case.

1. Position the cursor on the *Events* node and choose 🗋.

   The *Change Object Type Zini_2005* dialog box is then displayed.

2. Enter the following data:

   Event: `released1`

   Name: `<Order released>`

   Description: `<Production order released>`

3. Choose ✓.

4. Position the cursor on the event `released1` and choose *Edit* → *Change release status* → *Object type component* → *To implemented.*

   ➡

   Until now, within the object type definition, you have only described the fact that the event `released1` is intended to be used with object type ***Zini*_2005**. You must ensure that the event is actually created. This is described in one of the following chapters.

5. Choose ⚖ to check the object type.

6. Choose ⮌.

Implementing and generating an object type

You are on the screen *Business Object Builder: Initial Screen*. You object type is entered in the *Object/interface type* field.

1. Choose *Object type* → *Change status to* → *Implemented*.
2. Choose to generate the object type.

Delegation

In order to be able to work through this tutorial more than once or with a number of different users at the same time and on the same system, edit the new object type ***Zini*_2005**, which you created as a sub-type of the object type **BUS2005**. This procedure is particularly appropriate for training situations.

For extensions to object types and their use **in productive workflow scenarios**, SAP recommends an extended procedure in which you define the sub-type as the *delegation type* of the object type.

Delegation functionality is not used in this tutorial.

Result

In this unit, you were introduced to the *Business Object Builder* and the object type **BUS2005**. For the object type **BUS2005,** you have created a subtype ***Zini*_2005** and extended this by adding an event.

Unit 3: Adding Further Attributes

Use

In this unit, you will add further attributes that can be used in a workflow definition to the object type.

Procedure

"Last Changed by" as an Object Reference

The inherited attribute `LastChangedBy` (*name of person who made last change*) of the object type `Zini_2005` returns the contents of the database field `AUFK-AENAM`. This is the user name of the person who made the last change.

You can also use the name of the user from the user defaults in task descriptions or work item texts. To do so, you create a new attribute for your object type.

Instead of the user ID **MILLERP,** which is returned as the value of the attribute `LastChangedBy`, the name **Peter Miller** from is be used from the user defaults in the texts.

The name of the user is an attribute of the object type `USR01` (*SAP User*). You must create an attribute for the object type `Zini_2005` (*production order*) with a data type reference to the object type `USR01` (*SAP user*).

1. Open your object type to edit it in the *Business Object Builder*.

   The *Change Object Type Zini_2005* screen is then displayed.

2. Position the cursor on the *Attributes* entry and choose 🗋.

   The query *Create with ABAP Dictionary field proposals?* appears.

3. Choose *No*.
4. Enter the following data:

   Attribute: `ChangedBy`

   Name: `Changed by (object)`

   Description: `Last changed by (object reference)`

   Reference table: `AUFK`

   Reference field: `AENAM`

   Object type: `USR01`

1. Select *Database field* in the *Source* frame.
2. Select *Object type* in the *Data type reference* frame.
3. Choose ✓.
4. Choose 💾.

   ➡

   As this is a database field attribute (source database field) no implementation is carried out for this attribute.

Status of an Object Status as an Attribute

If the status (*set* or *not set*) of an object is to be available as an attribute of an application object, you must add the interface `IFSTATUS` (status management) and implement the attributes `StatusObjNumber` and `StatusObjTyp` that it contains. Then add the `ReleaseStatus` attribute to your object type:

1. Position the cursor on the *Interfaces* entry and choose 🗋.

   The *Enter Interface Type* dialog box appears.

2. Enter the **`IFSTATUS`** interface in the input field and choose ✓.

   This interface provides the `StatusObjNumber` and `StatusObjTyp` attributes.

3. Position the cursor on the `StatusObjType` attribute (*object type of status management*) in the list.
4. Choose .

   The entry changes color. You can now edit the attribute.

5. Position the cursor on the attribute `StatusObjType` and choose *Program.*

   The dialog box "Do you want to have a template generated automatically for the missing section?" is displayed.

6. Select *Yes*.

   The source code, which is generated automatically, is displayed.

7. Make the following changes to the source code:

```
GET_PROPERTY STATUSOBJTYPE CHANGING CONTAINER.
  OBJECT-STATUSOBJTYPE = 'ORH'.
  SWC_SET_ELEMENT CONTAINER
 'StatusObjType' OBJECT-STATUSOBJTYPE.
END_PROPERTY.
```

8. Choose , 💾 and then ⟲.

9. Position the cursor on the `StatusObjNumber` attribute (*object number of the status management*).

10. Choose .

    The entry changes color. You can now edit the attribute.

11. Position the cursor on the attribute `StatusObjNumber` and choose *Program.*

    The dialog box "Do you want to have a template generated automatically for the missing section?" is displayed.

12. Select *Yes*.

    The source code, which is generated automatically, is displayed.

13. Make the following changes to the source code:

```
GET_PROPERTY STATUSOBJNUMBER CHANGING CONTAINER.
  OBJECT-STATUSOBJNUMBER = 'OR'.
  SWC_GET_PROPERTY SELF 'Number' OBJECT-STATUSOBJNUMBER+2.
  SWC_SET_ELEMENT CONTAINER 'StatusObjNumber'
                              OBJECT-STATUSOBJNUMBER.
END_PROPERTY.
```

14. Choose , and then .

15. Position the cursor on the *Attributes* entry and choose .

    The query *Create with ABAP Dictionary field proposals?* appears.

16. Choose *No.*

17. Enter the following data:

    Attribute: `ReleaseStatus`

    Name: `Object status`

    Description: `Status of an object status`

    Object status: `I0002`

18. Select *Object status* in the *Source* frame.

19. Choose ✓.

    The system enters `SWCEDITOR` as the reference table and `OBJSTATUS` as the reference field.

20. Choose .

As this is an object status attribute, no implementation is required for this attribute.

Generate and Test Object Type

1. Choose *Edit* → *Change release status* → *Object type* → *To implemented*.

2. Choose to generate the object type.

3. Position the cursor on the *Object type* entry and choose .

4. Chose FIND in the line. Enter the number of a production order.

   You can enter the number of the production order that you used in Unit 1.

The list displays the values of all attributes for this object type. For the attributes that are created with a data type reference to an object type, you can navigate further to the attributes of the referenced object type.

Unit 4: Flexible Event Creation

Use

You want to establish a relationship between the **status change of an object** (here: release of a production order) and the **subsequent reaction** (here: start a workflow).

Before you define the workflow, you must make sure that an event is created if the status **REL** is set for the status object type **ORH**.

However, you must first ensure that an event is really created when the status **REL** is set for the object type **ORH**, that is, when the status change has taken effect on the object.

Prerequisites

You have defined the `released1` event as an enhancement of the object type `Zini_2005`. This object type was, in turn, created as a sub-type of the SAP object type `BUS2005`, as it is otherwise not possible to make changes to object types delivered by SAP.

This is the precondition for generating this event and using it as a triggering event.

Procedure

Event Creation During Status Change

The event `released1` should be generated if the status **REL** is set.

1. Choose *Tools → Business Workflow → Development → Definition Tools → Event Creation → Status Management.*

   The *Event creation status management* dialog box appears.

2. Choose *Customer settings*.

   The *View "Display Events for Status/User Status": Initial Screen* is displayed.

3. Choose ✎ to switch to change mode.
4. Choose *New entries*.
5. Enter the following data:

   StatusOT: **`ORH`**

   BusinessOT: **`Zini_2005`**

   Event: **`released1`**

6. Choose 💾.
7. Select your entry and double-click the *Status restrictions* node to go to the associated view.
8. Choose *New entries*.
9. Enter `I0002` in the *System status* field.

   ➡

   The entry `I0002` is the internal representation of the status **REL**. You can also use the F4 possible entries pushbutton to select the status **REL.**

10. Choose 💾 and exit status management.

Test the Event Creation

Test whether the events are created successfully.

1. Check whether the event log is activated. Choose *Tools → Business Workflow → Development → Utilities → Events → Event Trace → Switch Event Trace On/Off* and switch on the event trace.
2. Create a production order and release it. The relevant procedure is described in *Unit 1: Testing the Existing Application Functionality (page 55)*.
3. Choose *Tools → Business Workflow → Development → Utilities → Events → Event Trace → Display Event Trace.*

   The screen *Display Event Trace* is displayed.
4. Restrict the selection by entering values in the *Creation date* and *Creation time* fields so that the event you created appears in the display.

   The list contains the object type you created with the created event. No receiver is entered here.

Unit 5: Finding Agents - Organizational Structure

Use

Certain employees are notified by mail as a response to the release of the production order.

In addition to the existing organizational plan of your enterprise, all employees who are to receive the mail are to be grouped together in a separate, isolated **organizational unit**. The organizational unit is used as a distribution list in this scenario. The organizational unit is assigned existing positions, which are also used in the organizational plan of the enterprise. As a result, the recipient group is determined dynamically from the holders of the positions.

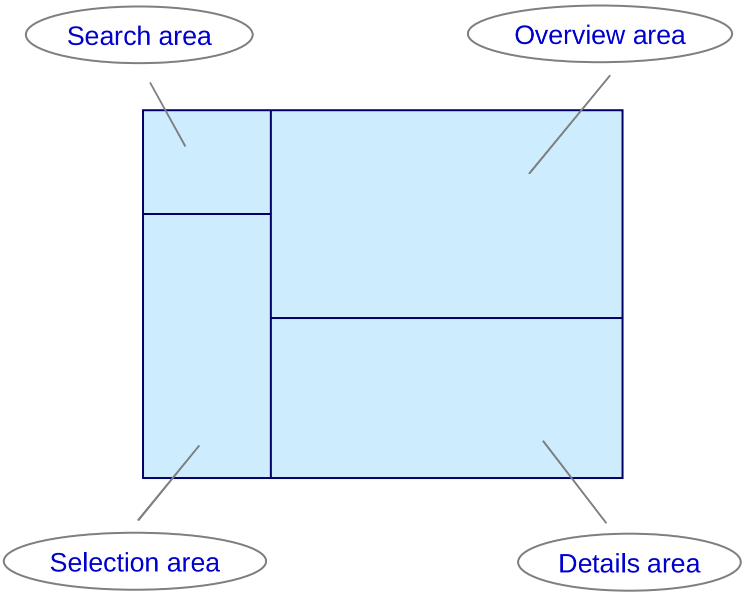
Procedure

You create an organizational unit and assign it positions. These positions are assigned users.

Creating an organizational unit

9. Choose *Tools → Business Workflow → Development → Definition Tools → Organizational Management → Organizational Plan → Create*.
10. Confirm the validity period proposed in the dialog box *Creating a Root Organizational Unit*.

    This takes you to the *Create Organization and Staffing (Workflow)* screen. This user interface is divided into four screen areas:



11. On the Basic Data tab in the details area, enter an abbreviation and a name in the *Organizational unit* input fields.

    Abbreviation: `Zini_OrderRel`

    Name: `Release production order`

1. Choose .

Assigning positions

In the staff assignments, you assign the organizational unit positions that are also used in the organizational plan of your enterprise.

1. Change to the overview area in the *staff assignments* of the organizational unit, in order to assign positions, jobs and holders. Choose the arrow on the right next to the and then the *staff assignments (list)*.

    The staff assignments of the new organizational unit are displayed.

1. Select positions in the find area and enter a search range.

    The positions that were found are shown in the selection area.

2. Select the position that you want to copy to the staff assignments and drag it to the staff assignments table in the overview area. Repeat this procedure until all of the necessary positions have been assigned to the organizational unit.

    Note that you also select a position, that you are assigned as a user, so that you can check the successful execution of the scenario.

3. Choose and exit the screen for editing the organizational plan.

Unit 6: Defining a Workflow

Use

A mail is to be sent once the production order has been released. This is carried out by a workflow which you will define in this unit.

Procedure

1. Choose *Tools* → *Business Workflow* → *Development* → *Definition Tools* → *Workflow Builder.*

   The *Workflow Builder* screen is then displayed. If you are starting the Workflow Builder for the first time, an initial workflow definition is displayed there. Otherwise, the workflow that was processed last is opened.

2. Choose 🗋.

   The *Workflow Builder - Create 'Undefined' [new, not saved]* screen is then displayed. An initial workflow definition is displayed.

Create container element in the workflow container

You must now add a container element to the workflow container that can store an object reference to a production order.

1. Choose <*Double-click to create*> by double-clicking in the *Workflow Container*.

   The dialog box for creating a new container element is displayed.

2. Enter the following data:

   Element: **ProductionOrder**

   Name: **Production order**

   Short Description: **Production order**

   Object type: **Zini_2005**

3. Set the *Import* and *Mandatory* flags.

4. Select *Object type* as the *Data type reference*.

5. Choose ✓.

   The new container element is displayed in the *workflow container*. The symbol ⇇ shows that the new container element is an import parameter of the workflow and thus belongs to the workflow interface.

Saving the workflow

1. Choose 💾.

   The first time a workflow is saved, a multistep task of the type *workflow template* is automatically created.

2. Enter the following data:

   Abbreviation: **Zini_Mail**

   Name: **<Mail for production order>**

   The dialog box *Object directory entry* is displayed.

3. Choose *Local Object*.

Determine Triggering Event

If a workflow is to be started by an event, you must define this as a triggering event. In this scenario, the workflow is to be started as a response to the released1 event of your object type Zini_2005. You must activate the *event linkage* for this event and define a binding.

1. Choose 🎩.

   The basic data of the workflow is displayed.

2. Go to the *Start* tab.

3. Enter the following data:

   Object type: `Zini_2005`

   Event: `released1`

Activate event linkage

4. Click on ◆ in the event column.

   Active event linkages are marked with ●.

Defining the binding between the event container and the workflow container

The event parameter `_Evt_Object` of the event `released1` contains the object reference to the released sales order.

The event parameter `_Evt_Creator` of the event `released1` contains the name of the person whose system user was used to generate the event. The name is stored in the form **US<Name>** in this element.

This information must be transferred by means of a binding between the event container and the workflow container.

The workflow container contains the *production order* (technical name: `ProductionOrder`) element you created as well as the standard *initiator* element (technical name: `_WF_Initiator`).

5. Choose `released1` ▦ in the event line.

   The *Task: Binding for Triggering Events* dialog box is then displayed. The workflow system has automatically assigned the contain element `_Evt_Object` of the event container to the `ProductionOrder` element of the workflow container.

6. Choose ▣ to display all of the container elements in the workflow container to which you can define a binding.

7. Assign the `_Evt_Creator` container element in the event container to the `_WF_Initiator` element in the workflow container. Use the *F4 input help*.

8. Choose ✓.

9. Exit the basic data of the workflow and choose 💾.

Unit 7: Editing Workflow Definitions

Use

You define a mail step in this unit, which when executed, sends a mail to the members of the newly created organizational unit.

For this purpose, use the step type *Send mail*.

Procedure

1. If you are no longer in the Workflow Builder, choose *Tools* → *Business Workflow* → *Development* → *Definition Tools* → *Workflow Builder*. The Workflow Builder is then launched with the workflow that was processed last. If the required workflow is not opened, choose ▣ and enter the abbreviation of your workflow. You can use the F4 input help and enter an abbreviation to search for your workflow. Choose ✓.

   As well as the initial workflow definition, your workflow also contains the defined *triggering event*. If you are not in change mode, choose ✎.

2. Choose *Send mail* in the step type area and click on the *Undefined* step.

   All necessary entries for sending a mail are made on the tab page *Mail*.

3. Specify the person(s) to whom the mail is to be sent. Do not change the presetting *recipient type* `organizational object`. Choose the entry **`organizational unit`** in the area *Recipient* and enter the identification of your organizational unit. You can use the *F4 input help*.

2. Enter the subject of the mail.

   `Production order released`

3. Enter the text for the mail.

   This text should contain text variables that can be replaced at runtime by values taken from the task container. Text variables are surrounded by a & symbol. To insert a text variable, choose . The system fields and the container element of the workflow container are displayed in the new dialog box. Double-click the expression. Choose to toggle between the technical name and the description. When you select a container element, the system creates the necessary container element in the task container and defines a binding definition from the workflow to the workflow container.

   `General information:`

   `The production order &ProductionOrder.Number& was released by &ProductionOrder.ChangedBy.Name& on &ProductionOrder.ChangeDate&.`

4. Set the *Send express* flag.

4. Choose ✓.

   The system automatically creates a task that the sending of the mail executes. You must enter another abbreviation and name for the task:

   Abbreviation: `Zini_Mail`

   Name: **`Mail: Production order released`**

   In the *Create Object Directory Entry* dialog box, choose *Local object*.

   The graphical display is displayed again.

Concluding step definition

Choose .

The workflow definition is checked, saved, and, provided it does not contain any errors, activated. The system displays a message if problems are encountered during the test.

To view the task created by the mail step, carry out the following steps:

1. Open the step definition for the step *Production order released* by double-clicking it.
2. On the tab page *Control,* double-click the task ID to open the task definition.

   The screen *Standard task: Display* is displayed. If you want to test the task (see below), make a note of the task number assigned by the system.

Test the defined task

You can also test the task you defined by starting it in dialog.

1. Choose *Tools → Business Workflow → Development → Runtime Tools → Start Workflow (Test Environment).*

2. In the field *Task*, enter the number of the standard task you defined in the form ***`TS<8-digit number>`*** .

    You can also use the F4 input help pushbutton.

3. Choose *Input data* to fill the `ProductionOrder` element in the task container with a value for the production order before execution.
4. In the *Initialize Container* dialog box, assign the number of the production order you created to the `ProductionOrder` container element. To do so, use the *F4* input help. Do not assign a value to the `_WI_Object_ID` container element.
5. Choose ⊞ to display all the container elements.
6. Assign your user name as the email recipient to the `ADDRESSSTRINGS` container element of the task container.
7. Assign an `X` to the `Express` container element.
8. Choose ✓.
9. Choose 🕓 to start the test.

    The mail is sent in the background. Open your Business Workplace to check that the mail has arrived.

Further notes on sending mails

You have followed the above procedure to identify the recipient of the mail by a user name and assigned the task container to the `ADDRESSSTRINGS` container element. This container element is identified as a multiline element and can also store **a list** of mail recipients.

To assign more than one element to the `ADDRESSSTRINGS` container element in the Container Editor, position the cursor in the `ADDRESSSTRINGS` line and choose ⊞.

The `ADDRESSSTRINGS` container element can contain other values in addition to a user name. You assign the recipient type to the `TypeID` container element of the task container.

Unit 8: Executing the Scenario

Use

In this unit you will execute the scenario created in this tutorial.

Prerequisites

You have completed all the units. You have:

- Linked the creation of an event to a status change
- Declared the event as a triggering event for the workflow
- Defined a workflow that executes the step "send mail"

Procedure

Executing the scenario

Release a production order. Follow the description in *Unit 1: Testing the Existing Application Functionality (page 55)*.

Business Workplace

When a production order is released, an event is triggered that starts your workflow.

The express mail `Production order <number> released` is sent to the Business Workplace inboxes of the recipients.

1. Log on to the system under a user name contained in the list of recipients.
2. Choose *Office* → *Workplace* to start the Business Workplace.
3. Check whether the express mail is in your inbox.

Tutorial: Workflow Programming

Purpose

This tutorial provides a step-by-step introduction to defining and implementing object types in the *Business Object Repository* (BOR). You should carry out these steps in the system yourself.

The tutorial is intended for both **customers** and **consultants** who want to modify the scenarios provided by SAP. It is also designed for **developers** within SAP who want to connect their application component to the workflow functionality.

This tutorial is not designed to replace the documentation on the *Business Object Repository*, which you can consult for further details. For further information, refer to *Business Object Repository* in the SAP Business Workflow documentation.

Prerequisites

To be able to work through this tutorial, you must be familiar with the basic ABAP programming principles.

You should have initial practical experience using SAP Business Workflow and have worked through the *Tutorial: Workflow Modeling (page 7)*.

Process Flow

The individual sections of this tutorial provide a step-by-step introduction to working with the *Business Object Builder*, creating, maintaining and implementing object types, and also point out the aspects that require consideration.

These steps are explained using the example of the object type definition *sales order*. **If you do not yet have any experience of object types**, carry out these examples directly in the system. In particular, use the options given to test “your” object type to see whether your work has been successful.

If you want to maintain your own object types in the Business Object Repository, go through this tutorial using the individual sections as a type of “check list”. If you use this tutorial to create your own object types, knowledge of the application environment of the object type to be maintained is absolutely essential. In particular, you should know in which tables the data (*attributes*) of the object type is stored and which transactions or function modules are used to call the main operations (*methods*) for objects of this type.

Tutorial Structure

The tutorial assumes that **no** object type *sales order* **exists** in the system or is to be maintained in the *Business Object Repository*. (However, this object type does already exist. Do not let this irritate you.) Read *Unit 1: Creating Object Types in the Business Object Repository (page 71)* for information on how to create object types.

In the following units of this tutorial, you extend this object type by adding selected attributes, methods and events:

Attributes	
DocumentDate	*Unit 2: Creating Database Field Attributes (page 74)*
SoldToParty	*Unit 3: Creating Object Database Field Attributes (page 76)*
TextWithDocumentNo	*Unit 4: Creating Virtual Attributes (page 77)*

SalesGroup	*Unit 5: Creating Object Virtual Attributes (page 79)*
Items	*Unit 6: Creating Multiline Virtual Attributes (page 81)*
Methods	
Edit	*Unit 7: Creating the Method Edit Without Parameters (page 83)*
ExistenceCheck	*Unit 8: Creating the ExistenceCheck Without Parameters (page 85)*
Display	*Unit 9: Creating the Display Method Without Parameters (page 87)*
Create	*Unit 10: Creating a Method with Parameters (page 89)*
Events	
DelBlockCancelled	*Unit 11: Events and Their Creation (page 91)*

In production operation, you will often encounter situations in which the object type to be maintained already exists in the *Business Object Repository* and is (only) to be extended, because you think customer-specific extensions or adaptations are required in particular areas.

For situations of this kind, refer to *Extending Object Types: Inheritance and Delegation (page 70)* to discover how to derive a “new” object type from an existing object type as a subtype, which can then be extended. Creating an object type *as a sub-type* “saves” you maintaining the basic data and creating key fields. The further maintenance of this object type (creating methods, attributes, and events) *is exactly the same* as for maintaining a completely new object type.

Implementation program for example

Every object type defined in the Business Object Repository has **one** *implementation program* containing the implementation of the key fields, methods and attributes of this object type.

The implementation program of the object type created in this example is listed in the *Appendix: Implementation Program for Object Type Z_BUS2032 (page 93)* of this tutorial. Reference is made in the individual units of this tutorial to the appropriate lines of this program.

Extending Object Types: Inheritance and Delegation

Use

In particular situations you will need to modify an object type in conjunction with delegation rather than create a new object type. This applies whenever you want to add components to an object type which are not provided in the standard version *and* at the same time need to ensure that productive scenarios with the original SAP object type remain operational.

If no modifications are required, you can use any released object type provided by SAP as it is.

Conventions for the other sections of the tutorial

The procedure described below consisting of “create subtype” and “delegation” is always required when you have to extend an SAP object type to meet your requirements.

However, within this tutorial it is always assumed that the object type to be maintained **did not** exist in the Business Object Repository and therefore is created **from scratch**. This section therefore informs you about the procedure involved in certain extensions, but does not really belong to the tutorial.

Procedure

Creating a sub-type

Create a new object type as the *sub-type* of the object type you want to extend. This subtype automatically inherits all the components (methods, attributes, and events) of the original object type including its implementation.

1. Choose *Tools* → *Business Workflow* → *Development* → *Definition Tools* → *Business Object Builder.*
2. In the field *Object type*, enter the object type you want to extend. This is then the supertype in the inheritance hierarchy.
3. Choose 🗋 *Subtype*.
4. For the new object type, enter:
   - A unique name that can be used to identify it
   - An object name
   - A name
   - A short description
5. Enter the name of the implementation program of the object type and assign the code letter of your application.
6. Confirm your entries with ✓ and save the object type created as a local object or with development class in a transport request.

Entering a delegation type

Before you process “your” object type any further, define it as the *delegation type* of the supertype.

1. Exit maintenance of the object type and go back to the *Business Object Builder*. Choose *Settings* → *Delegation* → *System-wide.*
2. Add a new entry to the table. To do this, choose *Table view* → *Display* → *Change*, then *Edit* → *New entries*.
3. Enter the name of the object type (super type) for which you want to specify a delegation type.
4. Enter the name of the sub-type as the delegation type.
5. Choose 💾.

The delegation entered is **client-independent**.

In all SAP Business Workflow definition tools, you must still use the “old” object type (supertype). However, when the definition is read and evaluated (at runtime, for F4 input help, and so on), the definition of the delegation type (subtype) is used.

Unit 1: Creating an Object Type in the Business Object Repository

Use

In this unit, you will learn how to create the object type *sales order* and how to maintain the key fields of this object type. You will see that the system supports you in the selection of key fields and the implementation in the program.

Procedure

Creating an Object Type

1. Choose *Tools → Business Workflow → Development → Definition Tools → Business Object Builder.*
2. Enter the name `Z_BUS2032` of your new object type in the *Object type* field.
3. Choose 🗋 *Create*.

   The dialog box *Create object type* is displayed.
4. Enter the following texts:

   Object name: `Sales order`

   Name: `Sales order (mod.)`

   Short description: `Sales order (modification for tutorial)`
5. Enter `Z_BUS2032` as the *program name.* This is the name of the implementation program of the object type.
6. Enter `Z` as the code letter of your *Application*.
7. Confirm your entries with ✓.
8. Save the object type you just created as a *local object* or with development class in a transport request.

The screen *Change object type* *`Z_BUS2032`* is displayed.

"Analyzing" the object type definition

First take a look at the "initial" object type definition. To do this, expand the hierarchies under the entries *Interfaces*, *Attributes* and *Methods* by clicking on the plus sign.

Interface IFSAP (SAP standard interface)

You see that your new object type already supports the *interface* `IFSAP` (*SAP standard interface*). Every object type supports this interface as standard.

The attribute and the two methods, which are also already in your object type, are defined for the interface `IFSAP` and are inherited from it. (The attribute and methods are therefore only there because the interface `IFSAP` is supported. If you were to delete the interface `IFSAP`, the attribute and the two methods would also "disappear" again.) The following object type components are inherited from the interface:

- Method `ExistenceCheck` (*check existence of an object*)

  Method without dialog, which is called to check whether the object exists.
- Method `Display` (*Display*)

  Method that displays the object in an object-specific way.
- Attribute `ObjectType` (*Object type*)

  Attribute containing the type of the object.

The methods `Display` and `ExistenceCheck` still have to be implemented in the implementation program on an object-specific basis. This can also be seen by the background color. You will also do this in a later unit.

The implementation of the attribute `ObjectType` does not need to be changed.

Maintaining Key Fields

The next thing you have to do is create the *key fields* of your new object type, so that the system knows how to uniquely identify objects of this type.

1. Position the cursor on the entry *Key fields*.
2. Choose 🗋 and answer the query *Create with ABAP Dictionary field proposals?* with *Yes*. This simplifies the entry of the data type reference.

   The dialog box *Create with Data Dictionary field proposals* is displayed.
3. Enter the *table* **VBAK**.

   The table VBAK (*Sales Document: Header Data*) contains the data for a sales order. The only key field of this table, identifying a sales order, is the field VBELN (*sales document*).
4. Choose ✓.

   The only key field VBELN of this table is now displayed.
5. Select this field. (In the case of object types that have several key fields, you should of course select all the key fields.)
6. Choose ✓.

   You go to the dialog box *Create* with text proposals for the key field to be created.
7. Check the proposed texts. Changes do not have to be made (but they could be made).
8. Choose 🗋. This is final confirmation that you want to transfer this key field into the object type definition.

   The key field is created with the automatically-proposed name SalesDocument for the object type Z_BUS2032.
9. Choose 💾.

Adding interfaces

In addition to the interface IFSAP (*SAP standard interface*), which is automatically supported by the newly created object type, you have the option of assigning the object type additional interfaces with methods, attributes and events, which this object type must then support.

1. Position the cursor on the entry *Interfaces*.
2. Choose 🗋.

   The *Insert Interface Type* dialog box is then displayed.
3. Enter **IFCREATE** as the *Interface type* and choose ✓.
4. Choose 💾.
5. Repeat this procedure for the interface types **IFEDIT** and **IFFIND**.

You have now added three new interface types to the object type Z_BUS2032.

- *Create* interface (IFCREATE)

  This interface adds the method Create to your object type.

  Every object type that provides a function for creating should support the interface *Create*. Using the interface ensures that names are assigned in a standard manner, and makes it possible to set up a generic search help.

- *Edit* interface (IFEDIT)

  This interface adds the method Edit (*change*) to your object type.

Every object type that provides a function for changing should support the interface *Create*. Using the interface ensures that names are assigned in a standard manner, and makes it possible to set up a generic search help.

- *Find* interface (`IFFIND`)

  This interface adds the method `Find` to your object type.

  This method determines an object which is available with this type and returns a reference to the object found.

  Every object type should support the interface *Find*.

The *implementation* of the attributes and methods inherited from the interface is usually incomplete and not adapted to the current conditions of the object type. This will be discussed in a later unit.

Implementing object type

At this point go to the implementation program of the object type you have created.

Choose *Program*.

Since methods and attributes have not yet been implemented, the implementation program only contains the data declaration for the object key. This section is generated automatically from the information you have specified for the key fields of the object type.

Do not change this part of the implementation program!

Key fields in the implementation program

Analyze the implementation program of your object type as it stands. You can use the *implementation program in the appendix (page 93)* as a comparison.

➡

> Note that the implementation program in the appendix already contains the complete implementation of object type `Z_BUS2032`.
>
> Some of the data declarations and program components there have not yet been included in your example, and are not required or created until the examples in the units still to come.

The key fields of the object type are declared in the program between the two macro commands `BEGIN_DATA OBJECT` and `END_DATA OBJECT`.

These macro commands, together with the commands in the implementation program, declare a structure for all key fields, which always starts with `OBJECT-KEY-` and whose fields are derived from the defined key fields of the object type.

The (only) key field of the object is hence available in the variable `OBJECT-KEY-SALESDOCUMENT` in the program.

Unit 2: Creating Database Field Attributes

Use

In this unit you create and implement the *database field attribute document date*. This attribute contains the document date as a field value as it is stored on the database.

The system helps you implement this attribute by generating source text automatically.

Procedure

1. Position the cursor on the entry *Attributes*.

2. Choose 🗋 and answer the query *Create with ABAP Dictionary field proposals?* with *Yes*. This simplifies the entry of the data type reference.

3. Enter **`VBAK`** in the *Table* field.

   All database field attributes of an object type refer either to the table used to define its key fields or to a table with an identical key structure.

4. Select the `AUDAT` field (*document date*) from the table fields displayed.

5. Choose ✓.

   The *Create* dialog box is then displayed.

6. Check the proposed texts. You do not need to make any changes.

7. Choose 🗋.

   The attribute is transferred into the object type definition with the automatically-proposed name `DocumentDate`.

Implementing attributes in the implementation program

Check the definition made so far. To do so, choose ⧉.

The system detects that the database field attribute has not been implemented in the program of the object type and allows you to generate a template automatically for the missing source text.

Make sure you choose this option.

The automatic generation of the implementation program has satisfactory results for database field attributes so that no revision is necessary.

The system then automatically goes to the implementation program of the object type you created.

Here, you can see additional program components that control access to the table with the attributes.

Do not change this part of the implementation!

Database field attribute

Database field attributes are implemented between the macro commands `GET_TABLE_PROPERTY <table name>` and `END_PROPERTY`.

To program one access for all attributes that refer to a table, the tables are buffered initially in an internal table. This internal table is created from the database table in the subroutine `FORM SELECT_TABLE_<table name> USING SUBRC LIKE SY-SUBRC`.

Analyze the implementation program as it stands. You can use the *implementation program in the appendix (page 93)* as a comparison.

- In the data declaration part, the variable `OBJECT-_VBAK` is declared as a structure `VBAK` (line 16).
- The table access is implemented between the two macro command `GET_TABLE_PROPERTY VBAK` and `END_PROPERTY` (lines 21-28).
- The automatically generated subroutine `FORM SELECT_TABLE_VBAK` that builds the internal table for buffering purposes is in lines 31-45.

Result

The object type must first be generated and released before it can be instantiated and tested.

1. Choose ◕.

The system informs you if the object type still contains errors. Try to correct these errors in the error overview (*Goto* → *Error list*).

2. Choose *Edit* → *Change release status* → *Object type* → *To implemented*.

The attributes created up to now can then be tested.

Testing the DocumentDate attribute

1. Choose ▤.

   The *Test Object Type <Object Name>: No Instance* screen appears.

2. Choose 🗋 *Instance*.

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the *F4 input help* if necessary.

3. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Unit 3: Creating Object Database Field Attributes

Use

In this unit you create and implement the object *database field attribute sold-to party*. Database field attributes are attributes that evaluate the content of a database field directly. In this case, the attribute reads the sold-to party from the database and creates from it a reference to an object of the type *customer*.

The system helps you implement this attribute by generating source text automatically.

Procedure

1. Position the cursor on the entry *Attributes*.
2. Choose 🗋 and answer the query *Create with ABAP Dictionary field proposals?* with *Yes*. This simplifies the entry of the data type reference.
3. Enter **`VBAK`** in the *Table* field.
4. Select the field `KUNNR` (*sold-to party*) from the table fields displayed.
5. Choose ✓.
6. Check the proposed texts. You do not need to make any changes.
7. Since the value of the database field is to be formatted and returned as an object reference, enter **`KNA1`** *(customer)* in the *object type* field.

   Make sure that the object type entered here *only has one key field* and can be identified via the database field entered.

8. Choose 🗋.

   The attribute is transferred into the object type definition with the automatically-proposed name *SoldToParty.*

Implementing attributes in the implementation program

Check your object type. To do so, choose ⚖.

If you have completed the previous unit, you will see that no further implementation is required. Database field attributes that refer to fields in the table VBAK are already fully implemented.

Result

Testing the SoldToParty attribute

1. Choose ◑.

   This regenerates your object type.

2. Choose 🖥.

   The *Test Object Type <Object Name>: No Instance* screen appears.

3. Choose 🗋 *Instance.*

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the *F4 input help* if necessary.

4. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Unit 4: Creating Virtual Attributes

Use

In this unit you create and implement the *virtual attribute TextWithDocumentNo*. This attribute returns the text `sales order <order number> of <date>`. This text must be “assembled” in the program of the object type.

A virtual attribute always requires a supplement to the proposed implementation in the implementation program.

Prerequisites

You use the macro commands provided by the system to implement these attributes.

For further information on the macro commands provided, see:

- *Macro Commands for Processing a Container*
- *Macro Commands for Accessing Objects, Attributes and Methods*

Procedure

1. Position the cursor on the entry *Attributes*.
2. Choose 🗋 and answer the query *Create with ABAP Dictionary field proposals?* with *No*.
3. Enter the following texts for the attribute to be created:

Attribute:	**TextWithDocumentNo**
Name:	**Text and number**
Short description:	**Text and number of sales order**

4. Select *Virtual* in the area *Source*.
5. Select *Dictionary* for the area *Data type reference*.
6. Enter **SYST** in the *Reference table* field.
7. Enter **TITLE** in the field *Reference field*.

   (You could also specify any character field with the required length here. Note that it is not the *content* of the field that is of interest, but its data type.)

8. Choose ✓.

Implementing attribute in the implementation program

Check the definition of the object type so far. To do so, choose .

The system detects that the implementation is missing and allows you to generate a template automatically for the missing source text.

Make sure you choose this option.

The source text generated automatically for implementing virtual attributes is always incomplete and restricted to setting the relevant container element. You must make changes here and implement the read procedure in the implementation program, which determines the attribute value by evaluating the database contents at runtime.

The system then automatically goes to the implementation program of the object type you created.

Virtual attributes

A virtual attribute is implemented between the macro commands `GET_PROPERTY <attribute name> CHANGING CONTAINER` and `END_PROPERTY`. The program code here determines how the value of the virtual attribute is derived at runtime.

The value determined is then stored in a container. This container forms the interface of the object type.

When you implement a virtual attribute, you must write the attribute value determined to the container of the object. Use one of the following macros for this purpose:

- `SWC_SET_ELEMENT CONTAINER ‘<ElementName>’ <Attribute>` for single-line attributes.
- `SWC_SET_TABLE CONTAINER ‘<ElementName>’ <Attribute>` for multiline attributes.

These macro commands are already in the automatically-generated source text.

Analyze the implementation program as it stands. You can use the *implementation program in the appendix (page 93)* as a comparison.

- The variable `OBJECT-TEXTWITHDOCUMENTNO` is declared for the attribute (line 13 of the program, generated automatically).
- You carry out the implementation yourself between the two macro commands `GET_PROPERTY TEXTWITHDOCUMENTNO CHANGING CONTAINER` and `END_PROPERTY` (lines 47-57).
  - The text is constructed from the values of the attribute already created and the key field of the object type. These attributes are read with the macro command `SWC_GET_PROPERTY` with reference to the object itself (object reference `SELF`). (Lines 51-52).
  - Assemble the text and assign it to the container element `TextWithDocumentNo` using the macro command `SWC_SET_ELEMENT`. (Lines 53-56).

Result

Testing virtual attributes

In order to test the attribute, it must first be released and then the object type must be generated.

1. Position the cursor on the attribute `TextWithDocumentNo`.
2. Choose *Edit* → *Change release status* → *Object type component* → *To implemented*.
3. Choose .

The system informs you if the object type still contains errors. Try to correct these errors in the error overview (*Goto → Error list*).

You can now test the attributes created up to now:

1. Choose 🗔.

   The *Test Object Type <Object Name>: No Instance* screen appears.

2. Choose 🗋 *Instance.*

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the F4 input help if necessary.

3. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Unit 5: Creating Object Virtual Attributes

Use

In this unit you create and implement the object *virtual attribute sales group*. This attribute returns the sales group of the sales order as an object reference.

➡

> Objects of the type *sales group* have only one key field. Therefore, it is not actually necessary to implement this attribute as a virtual attribute. You could also create it as an object database field attribute.
>
> This basic example is simply intended to demonstrate how a virtual attribute is implemented.

A virtual attribute always requires an implementation in the implementation program.

Prerequisites

You use the macro commands provided by the system to implement these attributes.

For further information on the macro commands provided, see:

- *Macro Commands for Processing a Container*
- *Macro Commands for Accessing Objects, Attributes and Methods*

Procedure

1. Position the cursor on the entry *Attributes*.
2. Choose 🗋 and answer the query *Create with ABAP Dictionary field proposals?* with *No*.
3. Enter the following texts for the attribute to be created:

   Attribute: `SalesGroup`
   Name: `Sales group`
   Short description: `Sales group`

4. Select *Virtual* in the area *Source*.
5. Select *Object type* in the *Data type reference* area.
6. Enter `TVKGR` in the *Object type* field.
7. Choose ✓.

Implementing attributes in the implementation program

Check the definition of the object type so far. To do so, choose .

The system detects that the implementation is missing and allows you to generate a template automatically for the missing source text.

Make sure you choose this option.

The source text generated automatically for implementing virtual attributes is always incomplete and restricted to setting the relevant container element. You must make changes here and implement the read procedure in the program of the object type, which determines the attribute value by evaluating the database contents at runtime.

The system then goes to the program of the object type you created.

Virtual attribute

A virtual attribute is implemented between the macro commands `GET_PROPERTY <attribute name>` `CHANGING CONTAINER` and `END_PROPERTY`. The program code here determines how the value of the virtual attribute is derived at runtime.

The value determined is then stored in a container. This container forms the interface of the object type.

When you implement a virtual attribute, you must create an object reference. To create an object reference with given key fields and a known object type, use the following macro command:

- `SWC_CREATE_OBJECT <Object> <ObjectType> <ObjectKey>.`

The variable `<Object>` must first be declared with:

- `DATA: <Object> TYPE SWC_OBJECT.`

This data declaration is already in the automatically-generated source text.

Analyze the implementation program as it stands. You can use the *implementation program in the appendix (page 93)* as a comparison.

- Since the attribute is to return an *object reference*, a variable `OBJECT-SALESGROUP` with type `SWC_OBJECT` is declared automatically.
- The actual implementation must be carried out manually. It must be between the two macro commands `GET_PROPERTY SALESGROUP CHANGING CONTAINER` and `END_PROPERTY`. The system inserts the macro command between these macro commands, with which the object reference to be created is written to the container.

  Add the following two lines to the program:

  ```
  SELECT SINGLE * FROM VBAK WHERE VBELN = OBJECT-KEY-
  SALESDOCUMENT.
  SWC_CREATE_OBJECT OBJECT-SALESGROUP 'TVKGR' VBAK-VKGRP.
  ```
- Save your program.

Result

Testing the virtual attribute SalesGroup

1. Position the cursor on the attribute just defined.
2. Choose *Edit* → *Change release status* → *Object type component* → *To implemented*.
3. Choose .

   The system informs you if the object type still contains errors. Try to correct these errors in the error overview (*Goto* → *Error list*).

Test the attributes created so far.

1. Choose .

   The *Test Object Type <Object Name>: No Instance* screen appears.

2. Choose 🗋 *Instance.*

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the F4 input help if necessary.

3. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Unit 6: Creating Multiline Virtual Attributes

Use

In this unit you create and implement the multiline object *virtual attribute sales document positions*. This attribute evaluates the items in the sales order, and returns a list with references to objects of the type VBAP (*sales document item*).

A virtual attribute always requires an implementation in the implementation program.

Prerequisites

You use the macro commands provided by the system to implement these attributes.

For further information on the macro commands provided, see:

- *Macro Commands for Processing a Container*
- *Macro Commands for Accessing Objects, Attributes and Methods*

Procedure

1. Position the cursor on the entry *Attributes*.
2. Choose 🗋 and answer the query *Create with ABAP Dictionary field proposals?* with *No*.
3. Enter the following texts for the attribute to be created:

   Attribute: `Items`
   Name: `Order items`
   Short description: `Sales order items`

4. Select *Virtual* in the area *Source*.
5. Select *multiline* in the area *attribute property*.
6. Select *Object type* in the *Data type reference* area.
7. Enter `VBAP` in the *Object type* field.
8. Choose ✓.

Implementing an attribute in the program of the object type

Check the definition of the object type so far. To do so, choose .

The system detects that the implementation is missing and allows you to generate a template automatically for the missing source text.

Make sure you choose this option.

The source text generated automatically for implementing virtual attributes is always incomplete and restricted to setting the relevant container element. You must make changes here and implement the read procedure in the program of the object type, which determines the attribute value by evaluating the database contents at runtime.

The system then goes to the implementation program of the object type you created.

Virtual attribute

A virtual attribute is implemented between the macro commands `GET_PROPERTY <attribute name>` `CHANGING CONTAINER` and `END_PROPERTY`. The program code here determines how the value of the virtual attribute is derived at runtime.

Analyze the implementation program as it stands. You can use the *implementation program in the appendix (page 93)* as a comparison.

- As the attribute is to return a *multiline* list of *object references*, an internal table `OBJECT-ITEMS` with type `SWC_OBJECT` is declared (line 14 of the program, generated automatically).
- The actual implementation must be carried out manually. It must be done between the two macro commands `GET_PROPERTY ITEMS CHANGING CONTAINER and END_PROPERTY` (lines 65-96). The system inserts the macro command between these macro commands, with which the object reference to be created is written to the container.
- Data declaration (lines 69-78)
  - Refresh table `OBJECT-ITEMS`
  - Auxiliary variable `ITEM` with type `SWC_OBJECT` to hold *one* object reference
  - Structure VBAP_KEY with the two fields `VBELN` and `POSNR`. These fields correspond to the key fields of the table VBAP (*sales document: item data).*
  - Internal table VBAP_TAB with structure VBAP. The items for an order are “collected” in this table.
- Data selection (lines 81-91)
  - All items for the sales order identified by the key field of the object type are read from table VBAP and written to table VBAP_TAB (lines 81-82).
  - The entries in table VBAP_TAB are used in a loop (lines 87-91) as follows:

    The structure VBAP_KEY is filled

    An object reference `ITEM` is created

    `ITEM` is appended to table `OBJECT-ITEMS`.
- Assignment (line 94)
  - The internal table `OBJECT-ITEMS` with the list of object references is assigned to the container element Items with the container macro `SWC_SET_TABLE`.

Result

Testing the virtual attribute items

1. Position the cursor on the attribute just defined.
2. Choose *Edit* → *Change Release Status* → *Object Type Component* → *To Implemented*.
3. Choose .

   The system informs you if the object type still contains errors. Try to correct these errors in the error overview (*Goto* → *Error list*).

The attributes created up to now can then be tested.

1. Choose .

   The *Test Object Type <Object Name>: No Instance* screen appears.

2. Choose 🗋 *Instance.*

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the F4 input help if necessary.

3. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Note: Object type VBAP (sales document item)

The object type VBAP describes the individual items of a sales document. The key fields of the object type VBAP are:

- `SalesDocumentNo` (*sales document*)
- `ItemNo` (*sales document item*)

These key fields reference the key fields VBELN and POSNR in the table VBAP (*sales document: item data).*

Alternative implementation

An alternative implementation without an additional internal table is shown below.

SYN

```
get_property items changing container.

  tables vbap.
  refresh object-items. clear object-items.

  data item type swc_object.
  data: begin of vbap_key,
          vbeln like vbap-vbeln,
          posnr like vbap-posnr,
        end of vbap_key.

  vbap_key-vbeln = object-key-salesdocument.

  select * from vbap where vbeln = object-key-salesdocument.
    vbap_key-posnr = vbap-posnr.
    swc_create_object item ’VBAP’ vbap_key.
    append item to object-items.
  endselect.

  swc_set_table container ’Items’ object-items.
end_property.
```

Unit 7: Creating the Method Edit Without Parameters

Use

In this unit, you will create and implement the method `Edit` as a method without a parameter.

You **must** implement this method if your object type supports the `IFEDIT` interface.

Prerequisites

The `Edit` method enables you to edit the object. This is one of the methods that your object type inherits from the `IFEDIT` interface.

Although this method is defined for the interface, it is not yet implemented. In other words, the program of your object type must be extended accordingly. To do this, the object-specific transaction (or function module) is called in the implementation of the method `Edit`, with which the object can be edited. For sales orders, this is transaction `VA02`.

Inherited Object Methods

If you want to edit a method that your object type has inherited from an interface or a supertype, note the following:

- An inherited method can only be edited if you release this method for overriding.
- You cannot change the name of the inherited method any longer.
- You cannot delete any parameters. You can only add non-mandatory parameters to the interface.
- When you maintain inherited methods, it is important that the entries you make for the ABAP functionality, call attributes, and result type are correct, since automatic program generation depends on this.

The following properties and exceptions of the method `Edit` are inherited from the interface `IFEDIT`:

- Method properties

  The method `Edit` is a *synchronous method*, which runs with dialog, is instance-dependent and does not return a result. (The task in which this method is referenced will usually have the property *Confirm end of processing*.)

Procedure

1. Position the cursor on the method `Edit`.
2. Choose .

   The color of the entry changes.
3. Double-click the method name to open the method definition.
4. On the tab page *ABAP/4*, enter the transaction **`VA02`** in the field *Name* and select *Transaction*.
5. Choose ✓.
6. Position the cursor on the method `Edit`.
7. Choose *Program*. Let the system create the template for the missing method.

   An (almost) usable implementation is created from the specifications on the ABAP function type:

   The method is implemented between the macro instructions `BEGIN_METHOD <method name> CHANGING CONTAINER` and `END_METHOD`.

   When the calls are implemented, the unique ID of the object is available in the structure of the key fields under the variable name `OBJECT-KEY-SALESDOCUMENT`. Using SET/GET parameters, the input field in the first screen of the transaction is filled from this key field. This first screen is then skipped when the transaction is called (`... AND SKIP FIRST SCREEN`).
8. Add the line **`SWC_REFRESH_OBJECT SELF`** to the implementation of the method `Edit`. You can find the source text in the appendix in lines 98-102.

   The standard buffering of attribute values can give rise to the situation where attribute changes that occur when an object is edited are not "noticed" by the runtime system.

   Therefore, after the execution of methods that could have changed attributes (methods `Edit`, `Update`, `Change`, and so on), call the macro in the form

`SWC_REFRESH_OBJECT SELF` before `END_METHOD`. This means that the attributes will be refreshed before the next read access, and not read from the internal table.

9. Choose 💾 and exit the Program Editor.

Result

Testing a method without parameters

1. The object type must first be generated so that it can be instantiated and tested. Choose .

   The system informs you if the object type still contains errors. Try to correct these errors in the error overview (*Goto* → *Error list*).

2. To test the method, choose .

   The *Test Object Type <Object Name>: No Instance* screen appears.

3. Choose 🗋 *Instance.*

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the F4 input help if necessary.

4. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Notes on the calling and reporting behavior of this method

In order for the method to be used in a workflow, it must be encapsulated in a task. This task is represented at runtime by a work item.

If this work item is executed by one of its *recipients*, the runtime system calls the method and thus, indirectly, the transaction. As the method `Edit` is a synchronous method without results, the work item is completed as soon as the method has been executed once. This system behavior is not generally desirable:

- The *current agent* does not receive any information as to whether anything was changed or not.
- It is not possible to call the method several times for multiple changes.

If you want to be sure that the work item only assumes the status *completed* if changes were actually made to the sales order, the method should not be defined as a synchronous object method. Instead, you must define the method as an *asynchronous method* and provide a terminating event for the task, which is created when a change is made to an object.

➡

This event can be implemented by linking the event creation to the writing of change documents.

If the work item is to be executed several times, you must set the flag *Confirm end of processing* in the task definition.

Unit 8: Creating the ExistenceCheck Method Without Parameters

Use

In this unit, you will create and implement the method `ExistenceCheck` as a method without a parameter.

You **must** implement this method if your object type supports the *SAP standard interface*.

Prerequisites

The `ExistenceCheck` method checks whether the object identified by the key fields actually exists. The `ExistenceCheck` method is one of the methods inherited by every object type from the *SAP standard interface* (technical name: `IFSAP).`

This method is not yet implemented for the interface. The implementation program must be extended correspondingly.

The following properties and exceptions of the method `ExistenceCheck` are inherited from the interface `IFSAP`:

- Method properties

  The `ExistenceCheck` method is a synchronous method, which runs without dialog, is instance-dependent and does not return a result. If the object does not exist, the method is terminated with an exception.

- Exceptions

  The exception *The object does not exist* is defined for the method `ExistenceCheck` with the number `0001`.

This exception must be implemented at the appropriate point in the source text. Use the macro command `EXIT_RETURN 0001 SPACE SPACE SPACE SPACE` for this purpose.

Defining exceptions

Exceptions can be defined for each method. For each exception, you must specify whether a temporary error or application or system error is involved. Each exception is identified by a 4-digit number and linked to a message text that is supplemented by up to four variables at runtime. You can use the following number ranges:

- 1001-7999: Application-specific exceptions (for SAP development)
- 9000-9999: Exceptions defined by customer

Implementing exceptions

You trigger the exception within the implementation of the method, that is within the macro commands `BEGIN_METHOD` and `END_METHOD`, if an error occurs when the method is processed. Use the macro command with the call: `EXIT_RETURN Code Variable1 Variable2 Variable3 Variable4.`

As the code (ABAP Dictionary reference `SWOTINVOKE-CODE`), enter the four-digit number of the exception which was specified when the exception was defined. The four variables (maximum) are determined from the definition of the message that is linked to the exception.

Procedure

1. Position the cursor on the method `ExistenceCheck`.
2. Choose .

   The color of the entry changes.
3. Double-click the method name to open the method definition. You do not need to change anything here for this example. Choose ✓.
4. Position the cursor on the method `ExistenceCheck`.
5. Choose *Program*. Let the system create the template for the missing method.
6. The actual implementation must be carried out manually. It must be done between the two macro commands `BEGIN_METHOD EXISTENCECHECK CHANGING CONTAINER` and `END_METHOD`. There is no additional source text initially.

Add the implementation program of the `ExistenceCheck` method, which you can take from the *implementation program in the appendix (page 93)* (lines 104-111), to the program. The `VBAK` table, which contains the header data of a sales order, is scanned with the key entered. If this was not successful, exception `0001` is triggered.

7. Choose 💾 and exit the Program Editor.

Result

Testing a method without parameters

1. The object type must first be generated so that it can be instantiated and tested. To do so, choose ●.

   The system informs you if the object type still contains errors. Try to correct these errors in the error overview (*Goto* → *Error list*).

2. To test the method, choose ▦.

   The *Test Object Type <Object Name>: No Instance* screen appears.

3. Choose 🗋 *Instance.*

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the F4 input help if necessary.

4. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Unit 9: Creating the *Display* Method Without Parameters

Use

In this unit, you will create and implement the method `Display` as a method without a parameter.

You **must** implement this method if your object type supports the *SAP standard interface*.

Prerequisites

The `Display` method displays the object. It is one of the methods inherited by every object type from the *SAP standard interface* (technical name: `IFSAP`**)**.

The method is already implemented for the interface, but this inherited implementation only displays the key fields of the object. To use the object-specific display functionality for the object from the application, the implementation must be redefined for your object type, and the transaction or function module of the application must be called, with which the object is displayed. For sales orders, this is transaction `VA03`.

The following properties and exceptions of the method `Display` are inherited from the interface:

- Method properties

  The `Display` method is a synchronous method, which runs with dialog, is instance-dependent, and does not return a result.

Procedure

1. Position the cursor on the method `Display`.

2. Choose ⧉.

The color of the entry changes.

3. Double-click the method name to open the method definition.
4. On the *ABAP/4* tab, enter the transaction `VA03` in the *Name* field and select *Transaction*.
5. Choose ✓.
6. Position the cursor on the method `Display`.
7. Choose *Program*. Let the system create the template for the missing method.

   You can find the source text in the appendix in lines 113-116.

8. Choose 💾 and exit the Program Editor.

Result

Testing a method without parameters

1. The object type must first be generated so that it can be instantiated and tested. Choose ●.

   The system informs you if the object type still contains errors. Try to correct these errors in the error overview (*Goto* → *Error list*).

2. To test the method, choose ▤.

   The *Test Object Type <Object Name>: No Instance* screen appears.

3. Choose □ *Instance.*

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the F4 input help if necessary.

4. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Further note (can be ignored during the first read operation)

The exception *The object does not exist* is specified in the method definition. This exception is not implemented automatically. It should not normally occur.

To implement it anyway, the following program lines must be inserted immediately after `BEGIN_METHOD DISPLAY CHANGING CONTAINER`:

Syn

```
SWC_CONTAINER CONT.
SWC_CREATE_CONTAINER CONT.

SWC_CALL_METHOD SELF ’ExistenceCheck’ CONT.
IF SY-SUBRC NE 0.
  EXIT_RETURN 0001 SPACE SPACE SPACE SPACE.
ENDIF.
```

First an empty container is created, which the subsequent macro expects as an input. The `ExistenceCheck` method is then called and evaluated referring to itself (reference to the object `SELF`).

Notes on the calling and reporting behavior of this method

In order for the method to be used in a workflow, it must be encapsulated in a task. This task is represented at runtime by a work item.

If this work item is executed by one of its *recipients*, the runtime system calls the method and thus, indirectly, the transaction. Since the `Display` method is a synchronous method without results, the work item is completed as soon as the method has been executed once. If the work item is to be executed several times, you must set the flag *Confirm end of processing* in the task definition.

Unit 10: Creating a Method with Parameters

Use

In this unit, you will create and implement the method `Create` as a method with parameters. Methods may require *parameters* in order to be executed or return them after they have been executed.

You **must** implement this method if your object type supports the `IFCREATE` interface.

Prerequisites

The method `Create` creates an object of the specified type. This is one of the methods that your object type inherits from the `IFCREATE` (*Create*) interface.

This method is not yet implemented for the interface. The implementation program must be extended correspondingly. To do this, the transaction (or function module) of the application is called in the implementation of the method `Create`, with which the object is edited. For sales orders, this is transaction `VA01`.

The following properties and exceptions of the method `Create` are inherited from the interface `IFCREATE`:

- Method properties

  The method `Create` is a synchronous, instance-independent method, which runs with dialog and does not return a result.

Procedure

Analyzing application functionality

1. Choose *Logistics* → *Sales and Distribution* → *Sales* → *Order* → *Create*.
2. Choose *System* → *Status*.

   You see that you have called transaction `VA01`.
3. Choose ✓.

   The screen *Create Sales Order: Initial Screen* is displayed again.

   The *Order type* is a required entry field. If you want to create a sales order using the method `Create`, you must either carry out dialog or pass the order type as a parameter when calling.
4. Position the cursor in the field.
5. Press *F1*.

   You go to the dialog box with the field help.
6. Choose *Technical info*.

   You can see from the section *Field data*, that the input field involves the table field `VBAK-AUART`.
7. Leave the dialog box and the screen for creating sales orders. Return to the *Business Object Builder* in the processing of your object type.

Defining a method

1. Position the cursor on the method `Create`.
2. Choose .

   The color of the entry changes.
3. Double-click the method name to open the method definition.
4. On the tab page *ABAP*, enter the transaction `VA01` in the field *Name* and select *Transaction*.
5. Choose ✓.

Creating parameters

6. Position the cursor on the method `Create`.
7. Select *Parameters*.

   The method container, which is still empty, is then displayed.
8. Choose 🗋.
9. Answer the query *Create with ABAP Dictionary field proposals?* with *Yes*.
10. Enter `VBAK` in the *Table* field.
11. From the table fields displayed, select the field `AUART` (*sales document type*).
12. Choose ✓.

    The dialog box for *creating* a parameter is displayed.
13. Create the parameter **SalesDocumentType** as proposed by choosing 🗋.
14. Choose ⬅.
15. Position the cursor on the method `Create`.
16. Choose *Program*. Let the system create the template for the missing method.

    The implementation of the method `Create`, which was taken automatically from your specifications, remains unchanged.

    Using the provided container macros `SWC_GET_ELEMENT` (for single-line elements) and `SWC_GET_TABLE` (for multiline elements), import parameters of the method are read from the container (`CONTAINER`) and passed to the function module or transaction or its processing parameters ("SET/GET parameters").

    Export parameters are set in the containers `(CONTAINER)` when a method is implemented. To do this, use the container macro `SWC_SET_ELEMENT` for a single-line return parameter or the container macro `SWC_SET_TABLE` for a multiline return parameter.

    You can find the source text in the appendix in lines 118-125.

Result

Testing a method with parameters

1. The object type must first be generated so that it can be instantiated and tested. Choose .

   The system informs you if the object type still contains errors. Try to correct these errors in the error overview (*Goto* → *Error list*).
2. To test the method, choose .

   The *Test Object Type <Object Name>: No Instance* screen appears.

3. Choose 🗋 *Instance.*

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the F4 input help if necessary.

4. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Unit 11: Events and Their Creation

Use

In this unit, you create the *event delivery block cancelled*.

Prerequisites

Events indicate the status changes of an object in the runtime system. Events always belong to an object and are therefore described as components in the object type definition.

➡

Keep in mind that when you create an event for an object type, this is only the first step. You still need to ensure that the event is actually triggered.

Procedure

1. Position the cursor on the entry *Events*.
2. Choose 🗋.
3. Enter the following texts for the event to be created:

   Event: `DelBlockCancelled`
   Name: `no delivery block`
   Short description: `delivery block cancelled`

   ⚠

   Irrespective of whether you use uppercase and lowercase in the event name, the name is always entered in uppercase letters in the event receiver linkage table by the system.

4. Choose ✓.

   The event has now been created and is available in the *Business Object Repository*. You can now use this event in the other definition tools in SAP Business Workflow, it is displayed via the F4 input help.

5. Position the cursor on the new event and choose *Edit → Change release status → Object component → To implemented.*
6. The object type must first be generated so that it can be instantiated and tested. Choose ●.

   The system informs you if the object type still contains errors. Try to correct these errors in the error overview (*Goto → Error list*).

Scenario for this event

The event `DelBlockCancelled` (*delivery block cancelled*) states that an existing delivery block has been deleted.

The event can therefore be used as a *terminating event* of a task, in which the asynchronous method `DeliveryBlockReset` (*reset delivery block*) is referenced. This method does not yet exist, but you could easily define it yourself as a copy of the Edit method. You must ensure that you define an asynchronous method.

At runtime, you can execute the task that encapsulates this method as often as you want and make changes to the sales order. Only when you really cancel the delivery block for your changes, is the event created and the task terminated.

Creating the event

This event can be created using a change document linkage referring to the field `VBAK-FAKSK` (*billing block in SD document*).

In the table for assigning change documents to events, assign the *change document object* `VERKBELEG` to the *object type* `Z_BUS2032` and to the *event* `DelBlockCancelled`.

Also enter a *field restriction* for this assignment, which states that the event is only to be created when a change is made to the field `VBAK-LIFSK` changing it from any value to the initial value.

For further information on creating events, please refer to *Wizards for Creating Events* (see SAP Library).

Unit 12: Basic Data

Use

The basic data contains the general administrative information on the object type.

Procedure

Maintaining Basic Data

1. To maintain the basic data, choose *Goto* → *Basic data* on the screen *Display object type Z_BUS2032*.
2. On the tab page *Defaults*, leave the method `Display` as *default method*.
3. On the tab page *Defaults*, enter the following the attribute **`TextWithDocumentNo`** as the default attribute.
4. Choose 💾.

Unit 13: Final Steps

Use

To be able to use the object type you have created operationally, you need to generate and release it.

Procedure

Generate

To generate the object type and make it available to the runtime system, choose ◐.

➡

An object type can only be generated if all higher-level object types do not contain any errors. If these object types have not yet been generated, they are generated as well. When an object type is generated, all subordinate sub-types are generated as well as long as they do not have any errors.

Release

If you have saved your object type as a temporary object, it can never be released. You can only use it as a “full” object type in your system.

To release an object type, position the cursor on the object type and choose *Edit* → *Change release status* → *Object type* → *To released*.

An object type should only be released if the basic functionality is ensured.

Test/Execute

1. To test the object type, choose .

   The *Test Object Type <Object Name>: No Instance* screen appears.

2. Choose 🗋 *Instance*.

   Identify an object of the type *sales order* by entering the number of a sales order of your choice. Use the F4 input help if necessary.

3. Choose ✓.

   The *Test Object Type <Object Name>* screen then appears, in which you can test your object type (execute methods, check attribute values).

Appendix: Program for Object Type Z_BUS2032

```
*****      Implementation of object type Z_BUS2032         *****
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,
  SALESGROUP TYPE SWC_OBJECT,
..ITEMS TYPE SWC_OBJECT OCCURS 0,
  TEXTWITHDOCUMENTNO LIKE SYST-TITLE,
  _VBAK LIKE VBAK.
END_DATA OBJECT. " Do not change.. DATA is generated

TABLES VBAK.
*
GET_TABLE_PROPERTY VBAK.
DATA SUBRC LIKE SY-SUBRC.
* Fill TABLES VBAK to enable Object Manager Access to Table Properties
```

```
  PERFORM SELECT_TABLE_VBAK USING SUBRC.
  IF SUBRC NE 0.
    EXIT_OBJECT_NOT_FOUND.
  ENDIF.
END_PROPERTY.
*
* Use Form also for other(virtual) Properties to fill TABLES VBAK
FORM SELECT_TABLE_VBAK USING SUBRC LIKE SY-SUBRC.
* Select single * from VBAK, if OBJECT-_VBAK is initial
  IF OBJECT-_VBAK-MANDT IS INITIAL
  AND OBJECT-_VBAK-VBELN IS INITIAL.
    SELECT SINGLE * FROM VBAK CLIENT SPECIFIED
      WHERE MANDT = SY-MANDT
      AND VBELN = OBJECT-KEY-SALESDOCUMENT.
    SUBRC = SY-SUBRC.
    IF SUBRC NE 0. EXIT. ENDIF.
    OBJECT-_VBAK = VBAK.
  ELSE.
    SUBRC = 0.
    VBAK = OBJECT-_VBAK.
  ENDIF.
ENDFORM.

GET_PROPERTY TEXTWITHDOCUMENTNO CHANGING CONTAINER.
  DATA:
    SALESDOCUMENT LIKE VBAK-VBELN,
    DOCUMENTDATE LIKE VBAK-AUDAT.
  SWC_GET_PROPERTY SELF ’SalesDocument’ SALESDOCUMENT.
  SWC_GET_PROPERTY SELF ’DocumentDate’ DOCUMENTDATE.
  CONCATENATE ’sales order’ SALESDOCUMENT ’of’ DOCUMENTDATE
    INTO OBJECT-TEXTWITHDOCUMENTNO SEPARATED BY SPACE.
  SWC_SET_ELEMENT CONTAINER ’TextWithDocumentNo’
    OBJECT-TEXTWITHDOCUMENTNO.
END_PROPERTY.

GET_PROPERTY SALESGROUP CHANGING CONTAINER.
    SELECT SINGLE * FROM VBAK WHERE VBELN = OBJECT-KEY-
SALESDOCUMENT.
  SWC_CREATE_OBJECT OBJECT-SALESGROUP ’TVKGR’ VBAK-VKGRP.
  SWC_SET_ELEMENT CONTAINER ’SalesGroup’ OBJECT-SALESGROUP.
```

```
END_PROPERTY.

GET_PROPERTY ITEMS CHANGING CONTAINER.

* Declare data
  TABLES VBAP.
  REFRESH OBJECT-ITEMS.
  DATA ITEM TYPE SWC_OBJECT.
  DATA:
    BEGIN OF VBAP_KEY,
      VBELN LIKE VBAP-VBELN,
      POSNR LIKE VBAP-POSNR,
    END OF VBAP_KEY.
  DATA BEGIN OF VBAP_TAB OCCURS 0.
    INCLUDE STRUCTURE VBAP.
  DATA END OF VBAP_TAB.

* Select data
  SELECT * FROM VBAP INTO TABLE VBAP_TAB
    WHERE VBELN = OBJECT-KEY-SALESDOCUMENT.

    VBAP_KEY-VBELN = OBJECT-KEY-SALESDOCUMENT.

* Create object reference
    LOOP AT VBAP_TAB.
      VBAP_KEY-POSNR = VBAP_TAB-POSNR.
      SWC_CREATE_OBJECT ITEM ’VBAP’ VBAP_KEY.
      APPEND ITEM TO OBJECT-ITEMS.
    ENDLOOP.

* Assign object reference to container element
  SWC_SET_TABLE CONTAINER ’Items’ OBJECT-ITEMS.

END_PROPERTY.

BEGIN_METHOD EDIT CHANGING CONTAINER.
  SET PARAMETER ID ’AUN’ FIELD OBJECT-KEY-SALESDOCUMENT.
  CALL TRANSACTION ’VA02’ AND SKIP FIRST SCREEN.
  SWC_REFRESH_OBJECT SELF.
END_METHOD.
```

```

BEGIN_METHOD EXISTENCECHECK CHANGING CONTAINER.
  SELECT SINGLE * FROM VBAK
    WHERE VBELN = OBJECT-KEY-SALESDOCUMENT.

    IF SY-SUBRC NE 0.
      EXIT_RETURN 0001 SPACE SPACE SPACE SPACE.
    ENDIF.
END_METHOD.

BEGIN_METHOD DISPLAY CHANGING CONTAINER.
  SET PARAMETER ID ’AUN’ FIELD OBJECT-KEY-SALESDOCUMENT.
  CALL TRANSACTION ’VA03’ AND SKIP FIRST SCREEN.
END_METHOD.

BEGIN_METHOD CREATE CHANGING CONTAINER.
DATA:
     SALESDOCUMENTTYPE LIKE VBAK-AUART.
  SWC_GET_ELEMENT CONTAINER ’SalesDocumentType’ SALESDOCUMENTTYPE.
  SET PARAMETER ID ’AAT’ FIELD SALESDOCUMENTTYPE.
  CALL TRANSACTION ’VA01’ AND SKIP FIRST SCREEN.
  GET PARAMETER ID ’AUN’ FIELD OBJECT-KEY-SALESDOCUMENT.
END_METHOD.
```