BIT610

SAP Workflow Programming

mySAP Technology

Date _____

Training Center _____

Instructors _____

_____

Education Website _____

Participant Handbook

Course Version: 62
Course Duration: 3 Days
Material Number: 50086626

An SAP course - use it to learn, reference it for work

Internal Use SAP Partner Only

Copyright

Trademarks

Disclaimer

Internal Use SAP Partner Only

About This Handbook

This handbook is intended to complement the instructor-led presentation of this course, and serve as a source of reference. It is not suitable for self-study.

Typographic Conventions

American English is the standard used in this handbook. The following typographic conventions are also used.

Type Style	Description
Example text	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths, and options. Also used for cross-references to other documentation both internal (in this documentation) and external (in other locations, such as SAPNet).
Example text	Emphasized words or phrases in body text, titles of graphics, and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example SELECT and INCLUDE.
`Example text`	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, and passages of the source text of a program.
`Example text`	Exact user entry. These are words and characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.

Icons in Body Text

The following icons are used in this handbook.

Icon	Meaning
	For more information, tips, or background
	Note or further explanation of previous point
	Exception or caution
	Procedures
	Indicates that the item is displayed in the instructor's presentation.

Contents

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Course Overview

This course covers customer adaptations and enhancements to workflows that require programming.

Participants learn how to program workflow interfaces at the level of: Objects, events, methods, rules, and attributes. They will also find out about controlling and monitoring the runtime system and about programming exits at workflow and step level.

Target Audience

This course is intended for the following audiences:

- Workflow developers (with ABAP knowledge)
- Workflow consultants (with ABAP knowledge)

Course Prerequisites

Required Knowledge

- SAPTec
- BIT601
- BC400 or equivalent skills

Recommended Knowledge

- Basic knowledge of object-oriented programming

Course Goals

This course will prepare you to:

- Perform workflow programming at all levels
- Manage and monitor the workflow runtime system

Course Objectives

After completing this course, you will be able to:

- Identify all points in the workflow system at which programming is possible or necessary
- Create your own object types and enhance existing object types

Internal Use SAP Partner Only

- Create rule function modules
- Program events
- Create check function modules and receiver type function modules
- Use AP to create work items
- Administrate the workflow runtime system

SAP Software Component Information

The information in this course pertains to the following SAP Software Components and releases:

- SAP NetWeaver 1.0

Unit 1

Introduction:

Unit Overview

The introduction unit covers a brief overview of the uses and tasks of a workflow management system.

It explains the definition architecture and the places within it that can be programmed.

This unit also introduces the example that will be used in the course exercises.

Unit Objectives

After completing this unit, you will be able to:

- Explain the SAP Business Workflow architecture
- Recognize the components of SAP Business Workflow that enable or require programming by the user

Unit Contents

Lesson: Overview of Objectives and Structure of the Workflow Engine

Lesson Overview

This lesson reinforces your knowledge of the underlying architecture of the Workflow Engine, and the associated terminology. It also introduces the example workflow that is created in the course.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the SAP Business Workflow architecture
- Recognize the components of SAP Business Workflow that enable or require programming by the user

Business Example

You have implemented workflows in you company and you now want to program rule function modules, create your own object types, and trigger events in applications yourself.

Workflow Management Structure

What Is Workflow Management?

- A system that delivers work
  - In the correct sequence
  - With all the necessary information
  - At the correct time
  - To the people responsible

Workflow Management is used to link work steps automatically.

It enables the control of connected activities across transactions and across different applications.

Tasks of a Workflow Management System

- Process definition

  "What happens, and in which order?"

  Workflow builder, task definition
- Organizational modeling

  "Who does what?"

  Organization model, rule definition
- Application encapsulation

  "Which objects are required?"

  Business Object Builder, Business Object Repository

Through the process definition, the a workflow management system ensures that "work is delivered **in the correct sequence...**"

Through organization modeling, a workflow management system ensures that "work is delivered **to the people responsible...**"

Through application encapsulation, a workflow management system ensures that "work is delivered with **all the necessary information**".

Tasks of a Workflow Management System

- Support for end users

  "What have I got to do today?"

  Business Workplace
- Process control

  "What happens when?"

  Workflow manager, work item manager
- Evaluation of processing

  "Who did what and when?"

  Reporting and analysis

Through end user support, a workflow management system ensures that "work is delivered **to the people responsible at the correct time**."

Through process control, a workflow management system ensures that “work is delivered **in the correct sequence at the correct time**”.

What Does Workflow Not Do?

- Simplify application transactions:

  Complex application functions remain complex application functions

- Provide efficient business processes automatically:

  The modelers themselves are responsible for ensuring the practicality of the workflow from a business perspective.

Workflow **cannot** influence existing interfaces or menu structures.

After the application is started using the workflow, only the application has control over which actions can be performed.

Definition Structure and Levels of Programming

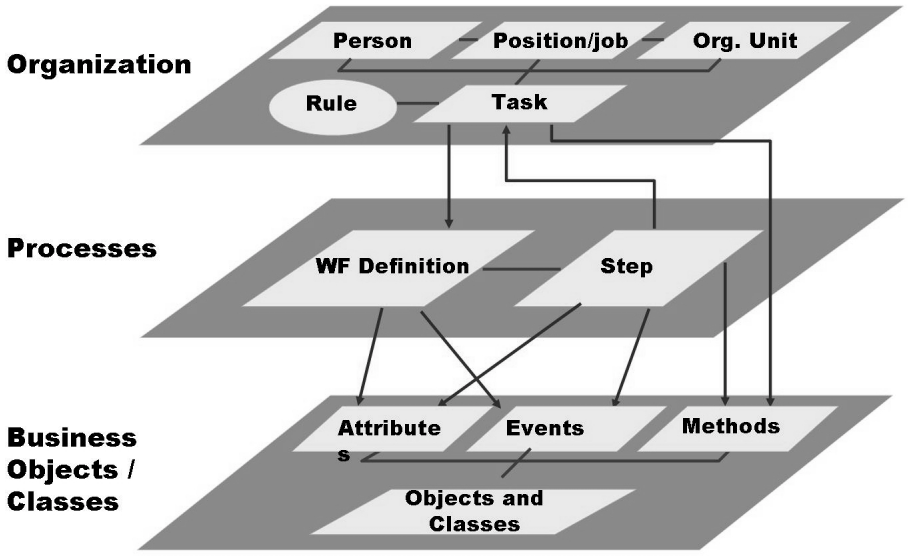


Figure 1: SAP Business Workflow: Definition Architecture

A → B means that A uses B.

Example: A step definition uses attributes (container operation), events (wait steps), methods (secondary methods). A task uses a workflow definition (in multi-step tasks) or methods (in single-step tasks).

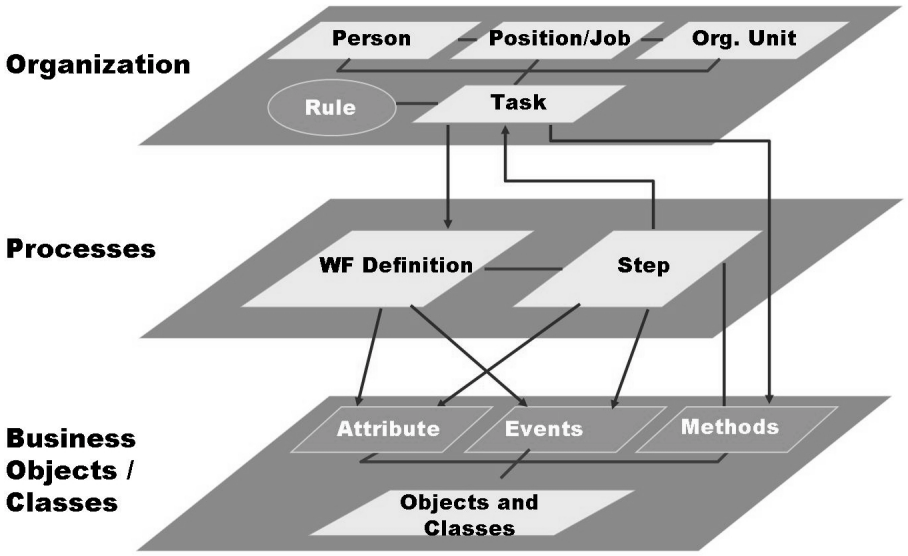


Figure 2: SAP Business Workflow: Programming

Programming is also possible in the following areas:

- Administration
- Reporting
- Creation of work items

The option to use ABAP classes is displayed in the diagrams in the lowest hierarchy level of the workflow architecture.

This technology can be used as of SAP ERP 5.0 (in exceptional cases also with SAP R/3 Enterprise) to carry functions that are to be called in the subsequent architecture levels.

As the use of ABAP classes for workflows is a special case, this course deals exclusively with programming using business object types. In the current release, these business object types provide the main part of the functions from the applications.

An extra day can be added to course BIT610, if required - **covered by course BIT611** - in which participants are given the necessary information to convert application functions using ABAP classes.

Lesson Summary

You should now be able to:

- Explain the SAP Business Workflow architecture
- Recognize the components of SAP Business Workflow that enable or require programming by the user

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Explain the SAP Business Workflow architecture
- Recognize the components of SAP Business Workflow that enable or require programming by the user

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Test Your Knowledge

1. The keyword for the Workflow Engine is: The correct _____ at the correct _____ to the correct _____. Programming is possible in the areas of _____, _____, and _____. The Workflow Engine consists of the _____, _____ and _____ components.

   Fill in the blanks to complete the sentence.

Internal Use SAP Partner Only

Answers

1. The keyword for the Workflow Engine is: The correct work at the correct time to the correct agent. Programming is possible in the areas of object types, rules, and events. The Workflow Engine consists of the Workflow Manager, Work Item Manager and Event Manager components.

   Answer: work, time, agent, object types, rules, events, Workflow Manager, Work Item Manager, Event Manager

Internal Use SAP Partner Only

Unit 2

Object Type Definition and Object Type Implementation

Unit Overview

This unit provides a short description of the underlying concepts of the Business Object Repository (BOR), and explains the definition of an object type and its components in detail.

You will learn how attributes and methods are defined and implemented.

You will also learn how to implement exceptions in methods, and how parameters are called and returned in method processing.

The final part of the unit covers access to attributes and calling methods inside and outside the BOR, and performance recommendations for implementing an object type.

Unit Objectives

After completing this unit, you will be able to:

- Explain the basic technical concepts of the Business Object Repository
- Create an object type.
- Define key fields.
- Define attributes with reference to a database.
- Define virtual attributes.
- Define multiline attributes.
- Describe how object references are accessed in BOR and in workflow.
- Define methods
- Call methods with parameters and return parameters
- Return exceptions from methods
- Describe how the Workflow Engine reacts to exceptions.
- Program an access to the attribute values of an object.
- Call methods of objects directly, for example, from a function module or report.

- Use the new parameter container interface to manipulate the container.
- Explain the different levels of performance of virtual attributes and background methods.

Unit Contents

Internal Use SAP Partner Only

Lesson: Purpose and Underlying Concepts of the Business Object Repository

Lesson Overview

This lesson describes the technical concepts underlying the Business Object Repository. This includes the use of interfaces in object types, settings in the basic data, relationships between object types, and the principle of delegation.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the basic technical concepts of the Business Object Repository

Business Example

You have introduced workflows in your company and you now need to create your own object types, because the standard object types are not sufficient for your requirements.

Object Types in the BOR and Workflow

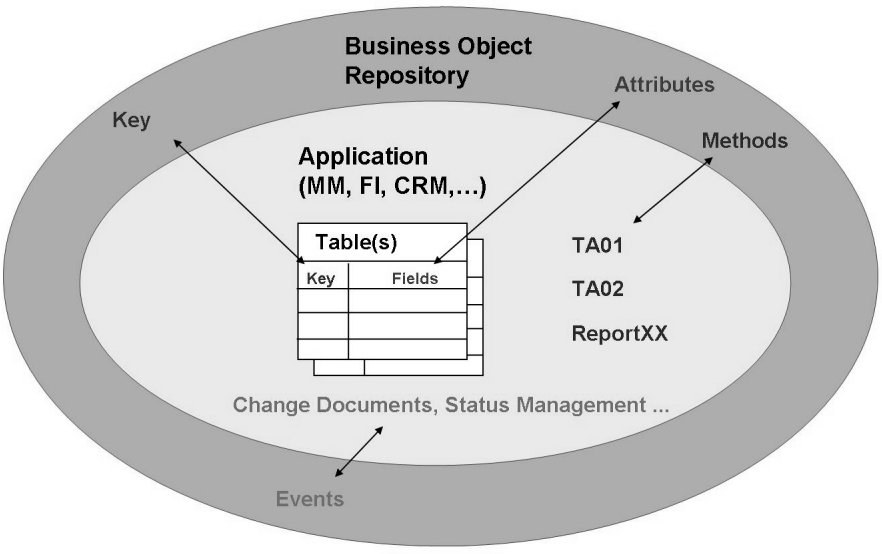


Figure 3: Objects - Purpose

Internal Use SAP Partner Only

In the Business Object Builder, objects encapsulate the application data (tables) and actions that can be performed on this data (such as transactions, reports and function modules).

A business object defines a stable, extendable “business” view. Users of the object are not affected by changes to the underlying database tables, transactions, reports and so on (the object modeler handles these changes by implementing appropriate changes for all of its users). This encapsulation has the following advantages:

- Simple modeling of processes
- Stable processes with a high degree of reusability
- On a technical level: A standard calling interface for the workflow runtime system

Hint: Events are covered in more detail in a later unit.

Objects in the Workflow Modeling

- Methods
  - In tasks: Define the “core” of an activity
  - In steps: Before methods, after methods and secondary methods
- Attributes
  - Conditions: Branches, loops, parallel processing forks, conditions before and after steps
  - Container operation, binding, agent assignment, deadlines
- Events
  - Start workflows, end steps
  - Create event, wait for event

Use of methods in steps (methods can change object data)

- The “main method” of an activity is encapsulated in a task so that it can be reused.
- Before, after and secondary methods define subordinate or parallel activities and are directly specified at a particular step.

Use of attributes (access to object **attributes** in the workflow is always read-only)

- Values are used for process control or transferred as parameters on the method.
- You can query attributes in your own modules in programming exits (check type function modules, programmed binding, receiver type function modules).

Use of events:

- Events are created for communicating between the application and the workflow system or for communicating different workflows.
- The Wait for Event prompt (for example, "Wait until a specific document is released?") is used for synchronizing processes or parts of a process. For example: “Wait until a particular document is released”. The user is then notified of the release by an event.

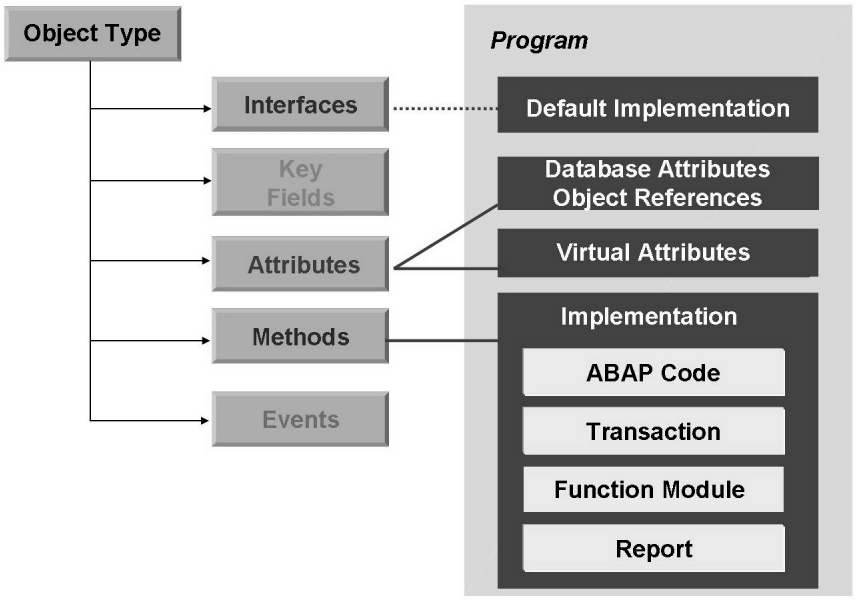


Figure 4: Object Type – Definition in the Business Object Repository

The Business Object Repository (BOR) is a cross-client directory of all defined object types. Each object type is assigned to a development class and therefore indirectly to an application component.

The entire implementation of an object type is contained in a program include (this is an include of your choice when creating the object type).

You can specify a default implementation for interface methods.

Key fields provide a unique reference between an object type and a database table (or a view). Each object instance is clearly displayed in a row in the table specified.

Two types of attribute can be created:

- Database attribute (attribute value is contained in the specified database table)
- Virtual attribute (source text is run to determine the value)

Methods are implemented using transactions, BAPIs, reports and so on. Depending on the implementation, methods are classified according to their processing model (synchronous/asynchronous) or interaction model (dialog/background).

Events are only declared in the Business Object Repository (both their names and parameters are declared). Events are triggered and received using other tools.

Using Interfaces in the Object Type

Interfaces

- Ensure a particular behavior by incorporating the attributes, methods and events defined in the interface
- Allow you to specify a default implementation for interface methods
- Enable hierarchical structuring - interfaces can be nested
- Replacement for multiple inheritance

Implementing an interface using an object type ensures a particular behavior. An object type that implements the interface “IF_FIND” allows all users to access the “Search for an object” function provided in the interface.

The definition of an interface can contain a specified default implementation that is used if the including object type does not perform any implementation of its own.

An object type that implements an interface must implement all the attributes and methods belonging to the interface (unless a default implementation is available).

Interfaces can themselves include further interfaces, thus enabling hierarchical structuring.

Interfaces replace multiple inheritance. They offer the same options as multiple inheritance, but are easier to handle (for example, no conflicts when method names are the same).

Internal Use SAP Partner Only

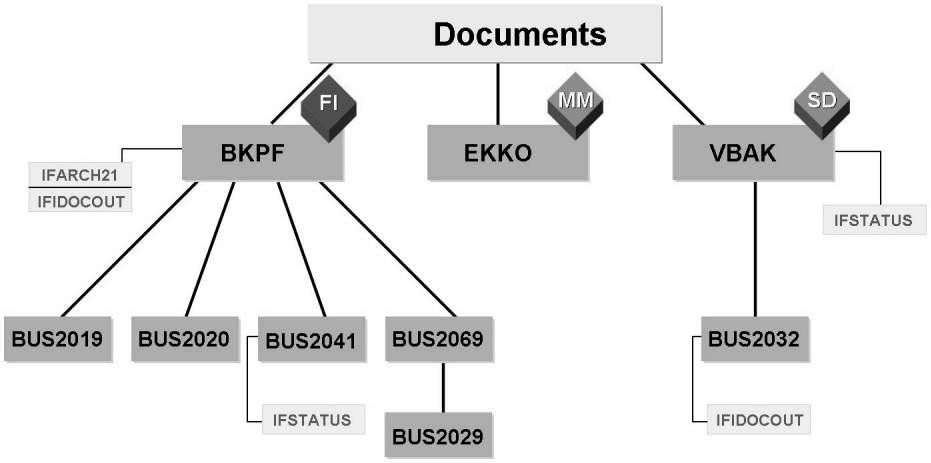


Figure 5: Interfaces - Example

BKPF	Posting document
BUS2019	Customer document
BUS2020	Vendor document
BUS2041	Asset document
BUS2069	G/L account document
BUS2029	Price change
EKKO	Purchasing document
VBAK	Sales document
BUS2032	Sales order

IFSTATUS	Generating events from status management
IFARCH21	ArchiveLink Interface
IFIDOCOUT	IDoc outbound processing

Status and Basic Data of an Object Type

Object Type - Release Status

- Modeled

  Not accessible at runtime.

- Implemented

  Only in tests or internal use, possibly unstable.

- Released

  Released for customer use. Only upward-compatible enhancements possible.

- Obsolete

  The function has been replaced. The old functionality is still supported for two releases.

Object types provide a **stable** interface to the application data that they encapsulate. The object type itself and each of its components can therefore be released for use under controlled conditions.

Components with “modeled” status cannot be displayed outside the Business Object Repository.

Customers should use only objects or components with “released” status.

The release information should be noted when a component's status changes to “obsolete”.

Object Type - Default Settings

- Object name

  Informative name for objects of this type

- Default method

  Display method for an object instance (used in forward navigation)

- Default attribute

  Identifying attribute of an object instance (used in texts)

An object name (“material” instead of “BUS1001”) is used as an element name in the container definition, for example, if container elements of this type are automatically created. The name is not language-specific, you should therefore select it in the project language when working in a project.

A default method is called, for example, when you double-click on objects in the list of object links (detail view in the workflow inbox). You can assume that the default method displays the object.

A default attribute is used, for example, to display objects in body texts or lists, whenever you have to display an object instance as text. SAP recommends that you select a descriptive text in this case (such as a material description or long text instead of a material number). The default attribute selected is often a virtual attribute combining values from key fields with descriptive texts.

Relationships Between Object Types and Delegation

Object Types - Relationships

- Association

  Foreign key relationship - “is related to”

- Composition

  Key enhancement - “is a part of”

- Inheritance

  functional enhancement - “is a type of”

- Delegation

  Supplements the inheritance concept: Enables **modification-free** enhancement of delivered objects

An association is defined as the reference between one object and another object that are closely related. However, both objects can be used independently of each other.

For example: Material *is related to* purchase requisition

In a composition, an object is composed of several objects. The overall object has either limited use or no use at all when separated from its components.

For example: Order *is a part of* order item

In an inheritance, the derived object inherits all of the attributes from the original (“parent”) object, but can then enhance or redefine these attributes in accordance with compatibility rules. In this way, the derived object receives special additional attributes,

For example: The accounting document *inherits data from the* customer document.

Key extensions are not authorized with an inheritance relationship in the Business Object Repository. Interfaces can only be extended in an upward-compatible manner (i.e. it is only possible to add new components or redefine existing ones).

Delegation is an extension concept specific to SAP. Using delegation, you can display all extensions (attributes, methods, events) of a derived object, even on the parent object. Delivered scenarios that use the parent object type can deploy the extended function in new steps (there is no need for conversion to the derived object type).

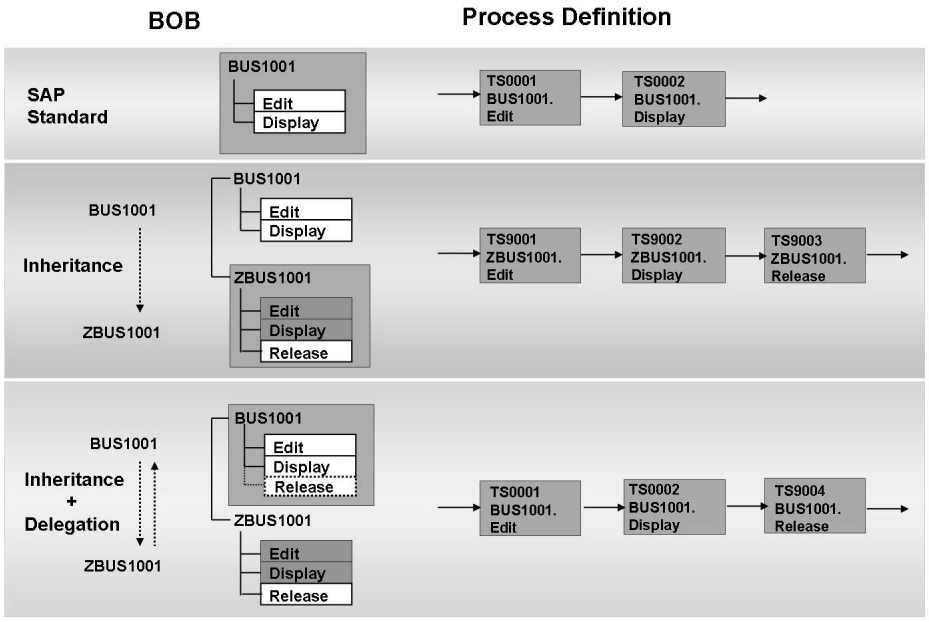


Figure 6: Inheritance and Delegation

Problem:

How can a customer (such as yourself) use your own object extensions with the tasks, events and so on that are supplied by SAP, without having to

- change the application program in which events are triggered,
- redefine existing tasks?

Solution:

You define a subtype and create all of the required extensions (attributes, methods and events). You define a delegation relationship.

With delegation, all of the subtype extensions can be displayed and called on the original supertype (modification-free enhancement). Now only the supertype is used in the modeling (workflows, tasks, events).

Top of the diagram:	Supplied object type and corresponding scenario
Middle of the diagram:	Inheritance - derived object type must be used in all cases
Bottom of the picture:	Inheritance and delegation - The BUS1001 object type behaves in the same way as an object type with the attributes of Z_BUS1001 (that is, additional events, attributes and methods). The delivered object type is used throughout.

In the Business Object Builder itself, from the basic data for a supertype you can see only whether a delegation is defined.

Caution: If problems occur with the object types you created when you convert your systems to Unicode, see Note 563417.

The program SWO_SET_UC_FLAG enables you to set a Unicode indicator that may be missing in the programs generated for your customer-specific object types.

Lesson Summary

You should now be able to:

- Explain the basic technical concepts of the Business Object Repository

Internal Use SAP Partner Only

Lesson: Object Type Definition - Key Fields and Attributes

Lesson Overview

This lesson deals with the creation of an object type and the definition of its key fields and attributes.

Attributes may relate to database fields or may be virtual attributes, for which the content is determined at runtime due to defined source code.

This lesson also shows how the objects should be available for the BOR and the workflow runtime.

The end of the lesson contains a list of macros for the general object handling.

Lesson Objectives

After completing this lesson, you will be able to:

- Create an object type.
- Define key fields.
- Define attributes with reference to a database.
- Define virtual attributes.
- Define multiline attributes.
- Describe how object references are accessed in BOR and in workflow.

Business Example

You have implemented workflows in your company and now have to create specific object types, because object types that you require are missing in the standard system.

First create the object type, define the key fields and the required attributes.

Development Support for Creation of an Object Type

Object Type - Development Support

- The program code for the object type is contained in an include.
- Parts of the object implementation that are automatically generated:
  - Access to database attributes
  - Key fields and attributes: The declaration in the program is automatically adjusted.
  - Methods: Local variables for parameters and data transfer are generated.

When creating the object type, you can choose any name for the program include that you want to use.

The area between BEGIN_DATA object ... END_DATA object. in the program include is automatically adjusted when you make any changes to key fields and attributes.

To make optimum use of automatic source code generation, SAP recommends that you use the following procedure when creating/enhancing an object type:

- The key fields are the first components that you have to create. This links the object type to a database table.
- You must then define and implement at least one database attribute. The database access routine is now generated by the BOB. After this is complete, you should always be able to execute and test the object type.
- You can now create additional database attributes, virtual attributes and methods in any order you like.
- When you create a method, SAP recommends that you first create the entire interface (including all import and export parameters and all exceptions). You can create the method implementation after the interface is complete.

Object definitions and implementations can only be manipulated using the Business Object Builder.

Definition of Key Fields

Key Fields

- The key fields definition determines the application table(s) where the object data is saved.
- One or more key fields can be used.
- The values of all key fields combined provide unique identification for an object instance.

Key fields can refer to a database table or a view. Each row in the table or view saves the data for precisely one object instance, which is assigned to the table row using the key fields.

In the case of client-specific tables *do not* specify the client explicitly as a key field.

Restrictions:

- Key fields must be character-based.
- The concatenated key fields can contain a maximum of 70 characters.

The English ABAP Dictionary names are automatically proposed as key field names when the key fields are created.

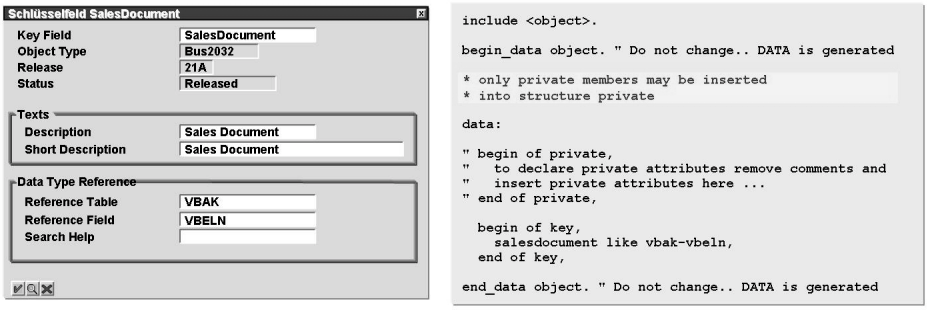


Figure 7: Implementing Key Fields

When the key fields are defined, the Business Object Repository changes the program section provided for handling key fields:

BEGIN_DATA OBJECT

…

BEGIN OF KEY

… key field declarations …

END OF KEY.

…

END_DATA OBJECT

In the example, the variable object-key-salesdocument (or generally, object-key-<name of key>) is declared.

The declaration between BEGIN_DATA and END_DATA is generated entirely from the definition data.

The declaration between BEGIN OF KEY and END OF KEY exactly represents the key field structure that is currently defined.

The macro commands required for programming are incorporated using the include <object>.

Definition of Events

Events

- Parameters can be assigned when events are triggered.
- Definition (Business Object Repository) and implementation (Application/Customizing/Workflow) are separate.
- Suitable documentation is an absolute prerequisite, as event creators and event receivers are entirely separate.

The Business Object Builder contains definitions of events only.

Each event contains default parameters that are automatically filled by the system. The following parameters are always incorporated:

- Triggering object (type, key and object reference)
- Name of event
- Language, date and time
- Triggering user

The include RSWEINCL contains a complete list of the default parameters.

User-defined parameters are useful only for events that are explicitly triggered in user-defined source code. You can transfer the user-defined parameters with the event in this case only. Parameters should not be attributes of the triggering objects, because the event receiver can query these attributes independently.

Definition of Simple Attributes

Attributes

- Database attribute – access to database table
- Virtual attribute – calculation with user's own ABAP code
- Attribute is typed either with the ABAP dictionary field or the object type
- Single-line and multiline attributes may be used.

A database attribute is defined with reference to a column (not a key column) of the related application table. When the database attribute is first accessed, the complete database row (defined by the contents of the key fields) is read and buffered in the object.

A virtual attribute is not saved to the database, but calculated with ABAP code (which you have to define yourself) when you first execute the call.

A multiline attribute corresponds to an internal table used in ABAP. During parameter transfer, you should note that single-line and multiline attributes are handled with different macro commands.

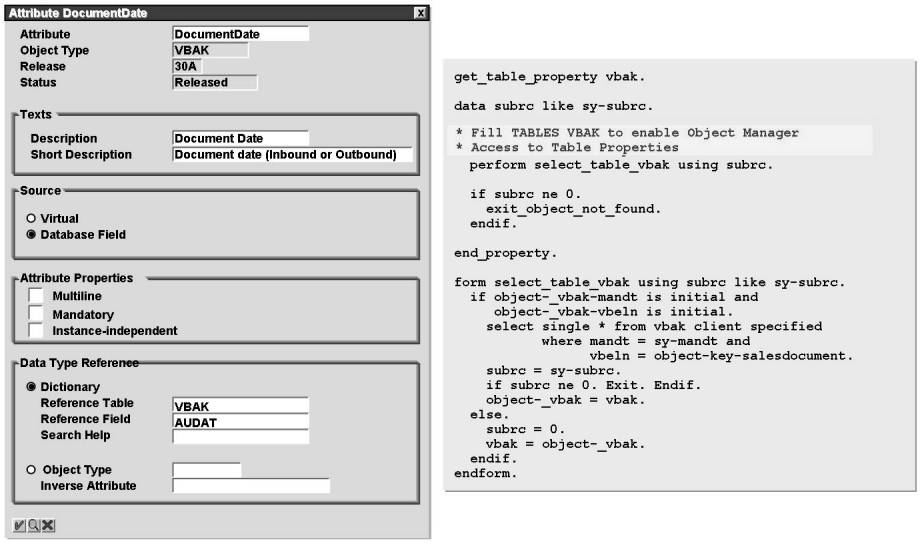


Figure 8: Implementing Database Field Attributes

In the implementation program, source code is automatically generated for database access. This access reads a complete database row, so all database attributes are contained in the buffer after the first access.

The source code between GET_TABLE_PROPERTY and END_PROPERTY is generated from the definition data. The subroutine select_table_<table name> (for the user's own database access) is also generated. A buffer variable is generated for a database row in the data section of the program.

_ <Name of the database table> (In the example: _VBAK like VBAK)

This buffer row is accessed in the ABAP program with the expression ‘object-_vbak’.

If there are no rows found in the database for the existing key field values, the macro EXIT_OBJECT_NOT_FOUND reports the exception “Object not found”, according to T100 message OL 826.

A database attribute can be object-value if the referenced object type has exactly one key field and the user's own database table has a valid key value for the referenced object. The Business Object Repository can automatically generate the required object reference when the attribute is accessed. This is the case if you enter the field containing the key value for the object in addition to the object type in the fields “Reference table” and “Reference field” , when the attribute is defined.

- **Program include contains definition for every key field: object-key-<keyfield>**
- **Program include contains *one* buffer variable for *all* database attributes: object_<tab name>**
- **Program include contains an individual buffer variable for each virtual attribute: object-<vattrib>**

```
begin_data object. " Do not change.. DATA is generated

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,

    _VBAK               like vbak.
    salesanddistribarea  type swc_object,
    companycodecurrency  like t001-waers.

end_data object. " Do not change.. DATA is generated
```

Figure 9: Data Buffer Object Type

Definition of the buffer structure: The “Object” data structure in the program include contains all buffer components. The Business Object Repository automatically adjusts this structure every time a change is made to key fields or attributes.

The Business Object Repository provides a separate buffer row for each object instance.

Filling and using the buffer: When an object instance is (first) created, the Business Object Repository enters the values of the key fields (object-key-<keyfield>) into their buffer.

The first time a database attribute is accessed, the corresponding database row is read and copied to the buffer variable object-_<tab name> (in the example: object-_vbak). Following this, all database attributes are available without any additional database accesses. The database access is contained in the (generated) subroutine “SELECT_TABLE_<tab name>”. You can also call this subroutine yourself, to check that the buffer variable object-_<tab name> is filled.

The value of a virtual attribute should (only) be calculated during the first call and placed in the buffer variable object-<attribute name>.

The macro SWC_REFRESH_OBJECT resets the buffer. You must always call this macro whenever a change (or possible change) is made to the object data on the database and you have to invalidate the buffer contents.

Definition of Virtual Attributes

Virtual Attributes

- Value is calculated or derived from other attributes
- Explicit implementation required
- Calculation upon attribute access
- Value can be buffered in the Business Object Builder

Virtual attributes are used, for example, to provide object references, language-dependent texts, time-dependent values, currency amounts or net/gross values.

The algorithm required for calculating the attribute value must always be explicitly programmed. Following the calculation, the buffer variable object-<attribute name> must be filled. Before the calculation, you must check whether the value is already in the buffer.

The value of the virtual attribute is technically handled in the same way as an export parameter: It must be explicitly transferred (in a data container) to the caller.

A virtual attribute is the functional equivalent of a method without import parameters and with exactly one export parameter. Difference: In workflow modeling, a method is called with a user-defined step, whereas attributes can be queried in conditions, bindings, container operations, agent assignments and so on.

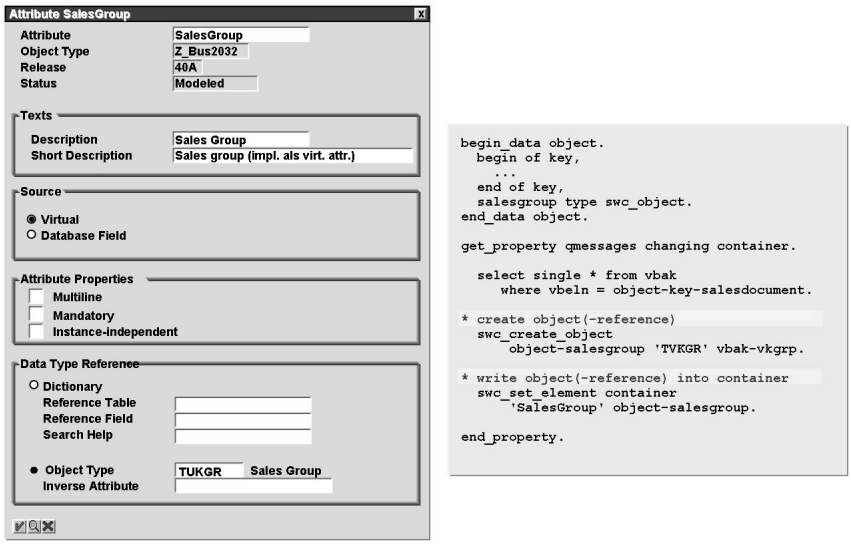


Figure 10: Implementing Virtual Attributes

Internal Use SAP Partner Only

The data declaration is generated from the definition data and is also used as a runtime buffer.

The entire implementation between GET_PROPERTY and END_PROPERTY must be created manually, because no other definition information is available. If the attribute is object-value, you must use the macro SWC_CREATE_OBJECT to explicitly create the object reference to be returned.

The “object-salesgroup” and the associated “SalesGroup” container element used in the Business Object Builder runtime buffer must be completed for data transfer.

Object References in BOR and Workflow

- **Individual objects ("object instances") are called in the program by object references.**
- **Object reference is created using the command SWC_CREATE_OBJECT.**
- **Object reference contains a reference ("pointer") to object data in the Business Object Repository object management:**

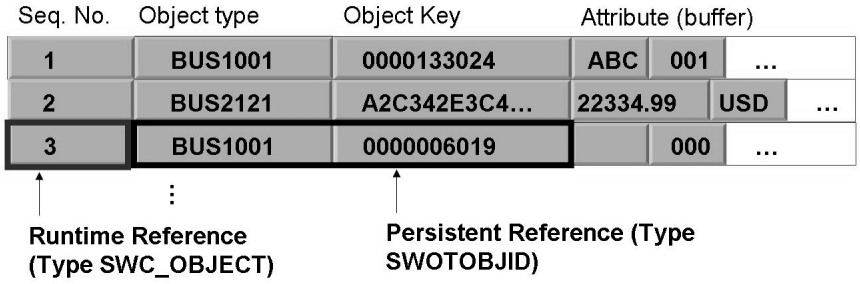


Figure 11: Object References in the Business Object Repository

Object references are represented by SWC_OBJECT variables in the Business Object Repository. This structure (“runtime reference”) indicates an entry in the BOR object management, which is the location where the buffered object data is saved at runtime.

Using an object involves several different stages:

- Creation of a reference variable: DATA: my_variable TYPE SWC_OBJECT.
- A valid object reference is created by specifying the object*type* and *key*. The command SWC_CREATE_OBJECT <reference variable><object type><object key> is used for this reason.
- Using the reference variable, you can now access object attributes (SWC_GET_PROPERTY ...) and methods (SWC_CALL_METHOD ...).

The SELF variable (of the type SWC_OBJECT) is predefined within each object program. This object reference allows you to query “own” attributes and call “own” methods.

To allow you to save object references to the database and to transport them beyond session limits reference variables of the ABAP type “SWOTOBJID” are used ("persistent reference"). This data structure contains an object type and a key. You can use this information at any time to create a runtime reference again with SWC_CREATE_OBJECT.

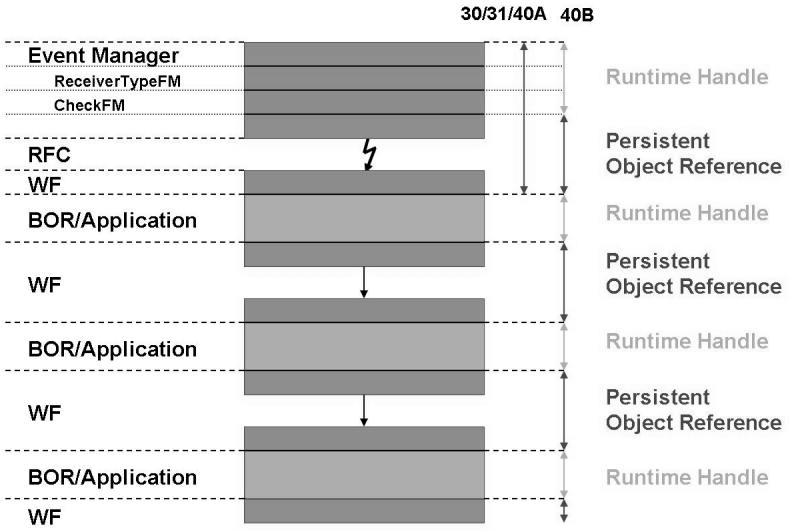


Figure 12: Object References in the Workflow

The workflow system saves object references in the container. These object references must be saved to the database whenever there is a context change (switch from one step to the next). For this purpose, the workflow system converts the object references in the container from runtime presentation (blue areas) to persistent presentation (green areas) and back again, as required.

If you are using programming exits, this must be taken into account (depending on the release), because the container may contain persistent references that have to be converted to runtime presentation before use. The references that are affected include Check function modules,

Receiver type function modules, modules for programmed binding.

The reference has the wrong format if the macro SWC_GET_ELEMENT returns an error (sy-subrc less than or greater than 0) when reading an object reference from the container, even though the container contains a valid object reference. A call from SWC_CONTAINER_TO_RUNTIME converts all object references

saved in the container to runtime format. Once the container processing is complete, you must perform the conversion back to the persistent format using SWC_CONTAINER_TO_PERSISTENT, so that the original display is available again in the subsequent context. You can also use SWC_OBJECT_FROM_PERSISTENT to convert an individual persistent object reference to a runtime handle.

When a workflow is started directly using the API, the runtime handles created in the program using the Business Object Repository macros must also be converted to persistent object references before SWW_WI_CREATE or SWW_WI_START is called.

Macros for Handling Objects

Macros - Handling Objects

- Object data is manipulated by means of references (pointers).
  - Declare reference variable

    DATA: <obj_ref> TYPE SWC_OBJECT.
  - Create object reference

    SWC_CREATE_OBJECT <obj_ref> <obj_type> <obj_key>.
  - Determine the object type name from the object reference.

    SWC_GET_OBJECT_TYPE <obj_ref> <CHAR10>.
  - Determine the key from the object reference.

    SWC_GET_OBJECT_KEY <obj_ref> <CHAR70>.

SWC_CREATE_OBJECT creates an object reference from the object type and object key. The macros SWC_GET_OBJECT_TYPE and SWC_GET_OBJECT_KEY are the reverse functions: They extract the object type (CHAR10) and the concatenated key fields (CHAR70) from a valid object reference.

When SWC_CREATE_OBJECT is called, the method "ExistenceCheck" of the associated object type is always called implicitly, to prevent references to non-existent objects.

The macro commands SWC_OBJECT_TO_PERSISTENT and SWC_OBJECT_FROM_PERSISTENT convert an individual object reference between its runtime display (SWC_OBJECT) and the persistent display (SWOTOBJID). The macros SWC_CONTAINER_TO_PERSISTENT or SWC_CONTAINER_TO_RUNTIME convert all of the object references of a container.

All (macro) commands are automatically available within the Business Object Repository, that is, in the implementation of attributes and methods. In the user's own programs (function modules, reports), they must be incorporated from the file "<cntn01>". Use this statement:

INCLUDE <cntn01>.

Implementation of Multiline Attributes

Multiline Attributes

- Correspond to internal tables
- Can contain either database fields or object references
- Are almost always virtual attributes

Example:

- An example of multiline attributes is a list of purchase requisitions for a material (BUS1001.PurchaseRequisition).

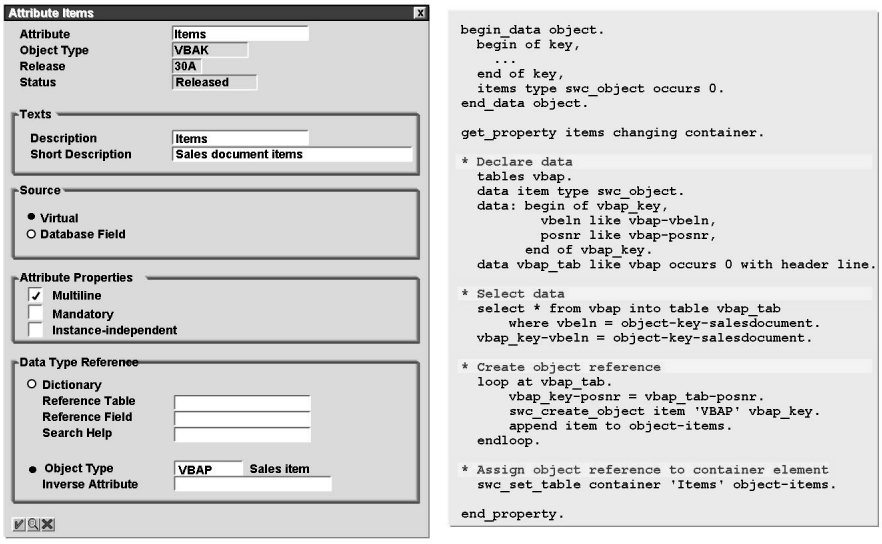


Figure 13: Implementing Multiline Virtual Attributes

The data declaration for a virtual attribute (variable object-<attribute name>) is generated from the definition data and is used as a runtime buffer.

The entire implementation between GET_PROPERTY and END_PROPERTY must be completed manually. The object reference to be returned, in particular, must be created explicitly with the macro SWC_CREATE_OBJECT.

The “object-salesgroup” and the associated “SalesGroup” container element used in the Business Object Builder runtime buffer must be completed for data transfer.

The actual implementation of the attribute also contains a query as to whether the runtime buffer is filled and this prevents unnecessary database accesses.

Internal Use SAP Partner Only

Exercise 1: Create a New Object Type for a Plant.

Exercise Objectives

After completing this exercise, you will be able to:

- Create an object type.
- Define key fields.
- Define attributes.
- Redefine methods.

Business Example

In the material master object type you want to refer to the plants to which the material is assigned.

You notice that there is no object type for a plant in the standard delivery, and for this reason you must create the object type.

Define the Display method, which is contained by interface in the object type, with the Display method from the object type BUS0008.

Task:

Create a new business object type.

1. Create a new business object type **ZPLANT**##, where '##' stands for your group number.

   Complete the basic data of this object type as follows:

Super type:	no entry
Object Type:	**ZPLANT##**
Object name:	**Werk_Group_##**
Name:	**Plant ##**

Continued on next page

Short description:	**Plant for group ##**
Program:	**ZPLANT##**
Application:	**S**

Hint: The name of the object type and the program must be in the customer namespace.

The object type, object name and program name must not contain any blank characters.

Create an object catalog entry as a local object.

2. Define a key field for your new object type. Create the key field with reference to ABAP/4 dictionary field specifications. Use the key of the table **T001W**.
3. Create a new attribute **Name**, which is based on the field **NAME1** of the table **T001W**.

   Do not forget to generate the access to the table T001W in your program.
4. Redefine the **Display** method. Do not define the method with reference to a transaction or a function module. First look at the implementation proposal generated by the Business Object Builder.

   Complete the implementation of this method, by copying the missing rows from the Display method of the object type **BUS0008** , delivered by SAP.

   Before you can execute the method, you must add the view **V_T001W** to the tables assignment of your new object type.

   Change the status of the object type to **Implemented**.

   Generate the object type and test it with plant **1000**.

   Execute the Display method.

Solution 1: Create a New Object Type for a Plant.

Task:

Create a new business object type.

1. Create a new business object type **ZPLANT**##, where '##' stands for your group number.

   Complete the basic data of this object type as follows:

Super type:	no entry
Object Type:	**ZPLANT##**
Object name:	**Werk_Group_##**
Name:	**Plant ##**
Short description:	**Plant for group ##**
Program:	**ZPLANT##**
Application:	**S**

Hint: The name of the object type and the program must be in the customer namespace.

The object type, object name and program name must not contain any blank characters.

Create an object catalog entry as a local object.

a) Create a new business object type.

Solution:

Compare the object type "ZPLANT00" as an example, in the Business Object Repository.

The object type and its definition are under the following path (using SAP Easy Access):

Tools -> Business Workflow -> Development -> Definition tools -> Business Object Builder

Continued on next page

2. Define a key field for your new object type. Create the key field with reference to ABAP/4 dictionary field specifications. Use the key of the table **T001W**.

   a) Compare the object type “ZPLANT00” as an example, in the Business Object Repository.

3. Create a new attribute **Name**, which is based on the field **NAME1** of the table **T001W**.

   Do not forget to generate the access to the table T001W in your program.

   a) Compare the object type “ZPLANT00” as an example, in the Business Object Repository.

4. Redefine the **Display** method. Do not define the method with reference to a transaction or a function module. First look at the implementation proposal generated by the Business Object Builder.

   Complete the implementation of this method, by copying the missing rows from the Display method of the object type **BUS0008** , delivered by SAP.

   Before you can execute the method, you must add the view **V_T001W** to the tables assignment of your new object type.

   Change the status of the object type to **Implemented**.

   Generate the object type and test it with plant **1000**.

   Execute the Display method.

   a) Compare the object type “ZPLANT00” as an example, in the Business Object Repository.

   b) The following section contains the source code for the implementation of the object type.

```
***** Implementation of object type ZPLANT00 *****

INCLUDE <OBJECT>.

BEGIN_DATA OBJECT. " Do not change.. DATA is generated
* only private members may be inserted into structure private
DATA:
```

Continued on next page


```
" begin of private,
" to declare private attributes remove comments and
" insert private attributes here ...
" end of private,
BEGIN OF KEY,
PLANT LIKE T001W-WERKS,
END OF KEY,
_T001W LIKE T001W.
END_DATA OBJECT. " Do not change.. DATA is generated

TABLES: T001W, V_T001W.

GET_TABLE_PROPERTY T001W.
DATA SUBRC LIKE SY-SUBRC.
* Fill TABLES T001W to enable Object Manager Access to Table
Properties
PERFORM SELECT_TABLE_T001W USING SUBRC.
IF SUBRC NE 0.
EXIT_OBJECT_NOT_FOUND.
ENDIF.
END_PROPERTY.

* Use Form also for other Properties to fill TABLES T001W
FORM SELECT_TABLE_T001W USING SUBRC LIKE SY-SUBRC.
IF OBJECT-_T001W-MANDT IS INITIAL AND
OBJECT-_T001W-WERKS IS INITIAL.
" read data from database table, if BOR buffer is empty
SELECT SINGLE * FROM T001W CLIENT SPECIFIED
```

Continued on next page


```
WHERE MANDT = SY-MANDT
AND WERKS = OBJECT-KEY-PLANT.
SUBRC = SY-SUBRC.
IF SUBRC NE 0.  EXIT. ENDIF.
OBJECT-_T001W = T001W.
ELSE. " read data from BOR buffer
SUBRC = 0.
T001W = OBJECT-_T001W.
ENDIF.
ENDFORM.

BEGIN_METHOD DISPLAY CHANGING CONTAINER.
" coding copied from BUS0008
CLEAR V_T001W.
V_T001W-MANDT = SY-MANDT.
V_T001W-WERKS = OBJECT-KEY-PLANT.
CALL FUNCTION 'VIEW_MAINTENANCE_SINGLE_ENTRY'
EXPORTING
ACTION = 'SHOW'
VIEW_NAME = 'V_T001W'
CHANGING
ENTRY = V_T001W.
END_METHOD.
```

Lesson Summary

You should now be able to:

- Create an object type.
- Define key fields.
- Define attributes with reference to a database.
- Define virtual attributes.
- Define multiline attributes.
- Describe how object references are accessed in BOR and in workflow.

Lesson: Object Type Definition - Methods

Lesson Overview

This lesson explains how you can define and implement synchronous and asynchronous methods.

It describes how to call methods with parameters, and how they can return parameters and exceptions.

Exceptions to a method lead to actions in a workflow. The lesson describes how the Workflow Engine reacts to temporary exceptions and to exceptions controlled by the system and applications.

Lesson Objectives

After completing this lesson, you will be able to:

- Define methods
- Call methods with parameters and return parameters
- Return exceptions from methods
- Describe how the Workflow Engine reacts to exceptions.

Business Example

You have created a new customer-specific object type and defined attributes, and now you want to create the methods that you require

You want the methods to be called using parameters and return parameters and exceptions.

Synchronous and Asynchronous Methods

Synchronous/Asynchronous Methods

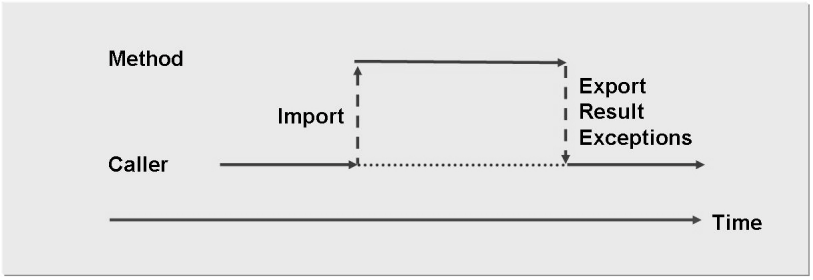
- Synchronous method
  - Export parameters are returned directly to the calling program
  - Exceptions may be triggered
- Asynchronous method
  - Method execution consists of a synchronous part and subsequent update (asynchronous part)
  - Export parameters are not authorized. Values are only returned from the update, as parameters of a terminating event
  - Exceptions are only possible in the synchronous part

Import parameters can be defined for synchronous and asynchronous methods.

Values are returned using export parameters and/or tagged return parameters (synchronous) or using event parameters (asynchronous).

Asynchronous methods are always necessary if the encapsulated application functionality uses updating.

Asynchronous methods can trigger exceptions only in the synchronous or “dialog part”.



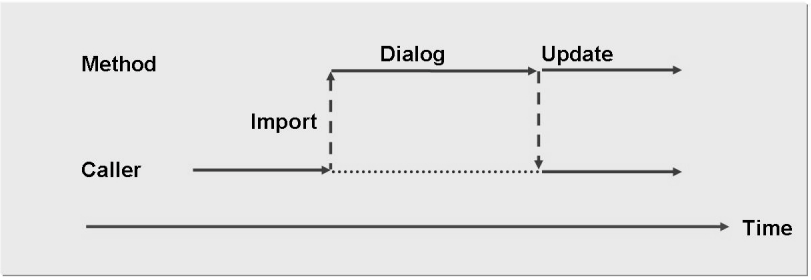
- **All display methods**
- **All methods that do not write their changes to the database using the update task**
- **The caller waits until all of the method's activities are complete**

Figure 14: Call Model For Synchronous Methods

The solid arrows represent the control flow.

The broken arrows represent the transfer of data and control flow.

The dotted line represents the waiting time of the caller.



- **All methods that write their changes to the database using the update task**
- **The caller waits until the dialog part of the method is complete, while the update continues to run asynchronously.**
- **An event notifies you when the update is complete.**

Figure 15: Execution Model For Asynchronous Methods

The solid arrows represent the control flow.

The broken arrows represent the data transfer.

The dotted line represents the waiting time of the caller.

The update task is the only one that runs asynchronously. The source code directly saved in the method is executed synchronously (“dialog part”).

The distinction between synchronous and asynchronous applies to both dialog and background methods.

It is the caller's responsibility to organize any required synchronization.

For example: The workflow system writes a linkage for the expected terminating event before calling an asynchronous method.

It is the method's task to make the synchronization technically possible*that is* an event must be triggered in the update task.

Implementing Synchronous and Asynchronous Methods

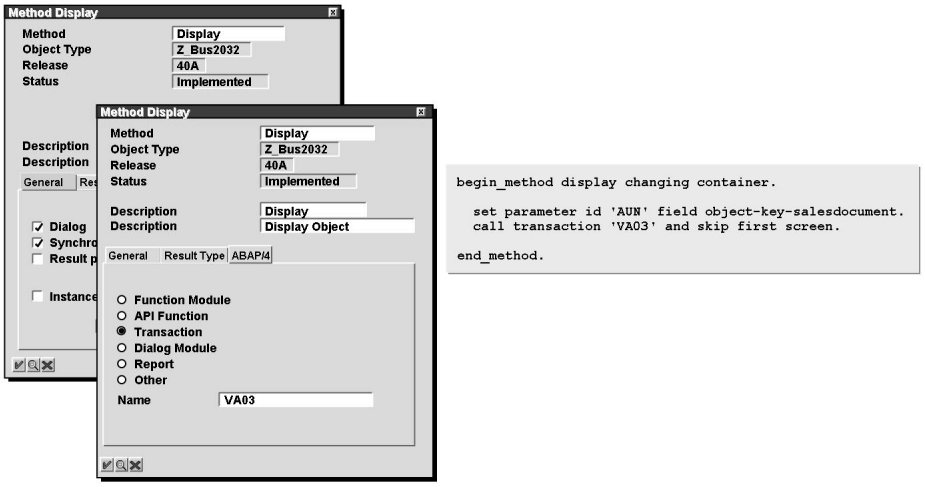


Figure 16: Implementing Synchronous Methods

The implementation between BEGIN_METHOD and END_METHOD is generated from the definition data.

SAP recommends that you define the method parameters before creating the method source code. If the parameters are recognized, additional source code is automatically generated.

The “result” of a method is a special export parameter. If you define the result parameter with reference to a fixed value domain (ABAP data definition refers to domains with fixed values), you get an individual output in the workflow graph in the Workflow Builder for each fixed value for activities that refer to this method.

If you define the method with reference to a transaction, function module, report and so on, the Business Object Repository will generate a suitable call template for you.

You must call the SWC_REFRESH_OBJECT macro at the end of a synchronous method that changes the object attributes. Otherwise, the Business Object Repository will work with the obsolete data in its object buffer.

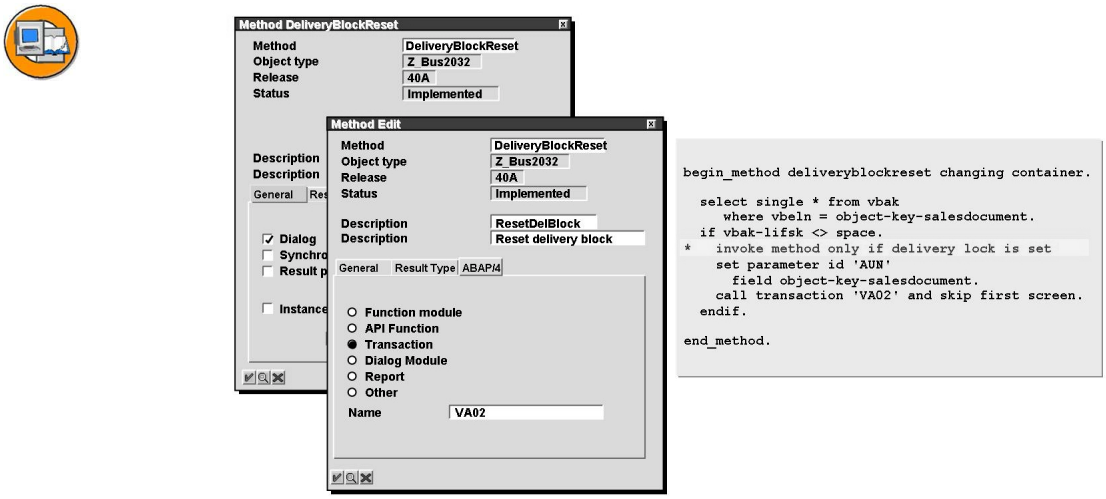


Figure 17: Implementing Asynchronous Methods

The implementation between BEGIN_METHOD and END_METHOD is generated from the definition data. In the example, the generated source code has been extended so that the transaction is called only when a delivery block is set.

The person calling an asynchronous method should always assume that the object data is invalid until he or she has received a terminating event signaling that the update has been successfully completed. After receiving the terminating event, he or she should also call the SWC_REFRESH_OBJECT macro to invalidate the object buffer.

The terminating event must be triggered within the called transaction, that is, from the update task after the data belonging to the object has been written.

A single-step task encapsulating this method requires at least one terminating event.

Methods and Parameter Calls

Communication Methods: Caller - Method

- Communication using method parameters/event
  - Import parameters
  - Export parameters
  - Result (special export parameter)
- Communication via exceptions
  - Temporary error
  - Application error
  - System error
- Communication using events (asynchronous return of values)

Synchronous methods communicate directly with the caller by means of parameters, results and exceptions. The **most important** export parameter should be defined as the “result” of the method so that the special modeling options (automatic branching in the workflow using the result value) can be used.

Asynchronous methods should not define a result or any export parameters. They can only return values indirectly as parameters of an event (triggered in the update task).

When a method returns an object reference, attributes for the same object should not be transferred as additional parameters (the caller can determine attribute values himself).

If object attributes on the database are changed in the method, the object buffer in the Business Object Builder must be invalidated with the SWC_REFRESH_OBJECT macro.

Exceptions are triggered with the EXIT_RETURN macro. There are also two specific macros, EXIT_OBJECT_NOT_FOUND (error in generating an object reference), and EXIT_CANCELLED

(The user has terminated the processing of a transaction. This is handled as a temporary error).

How the workflow system reacts depends on the type of exception:

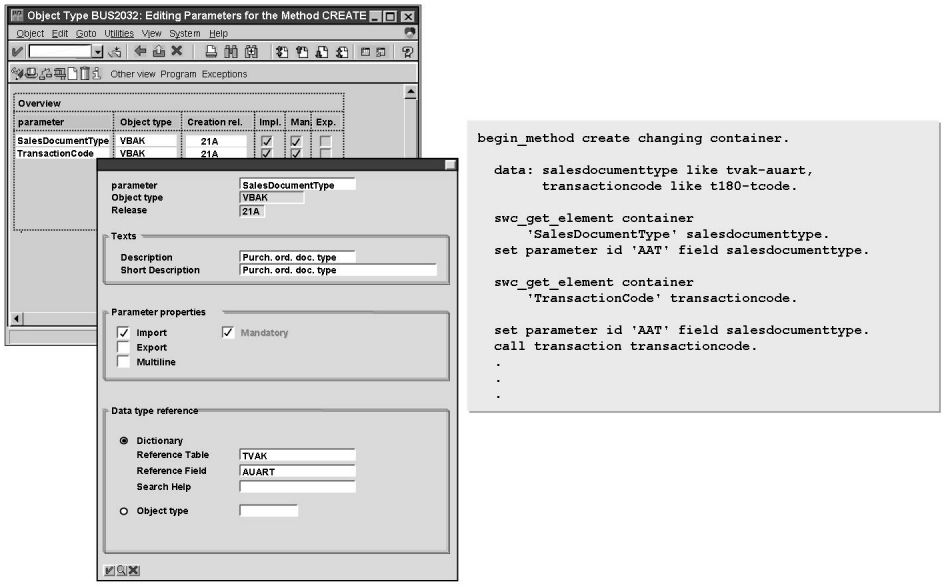


Figure 18: Implementing Parameter Transfer

Parameters of a method are transferred as Name-Value-Pairs in an internal table called “container”.

Single-line input parameters from the container are read using the SWC_GET_ELEMENT macro, whereas export parameters are written to the container with the SWC_SET_ELEMENT macro.

The macros SWC_GET_TABLE and SWC_SET_TABLE are used for multiple-line parameters.

If available, the result parameter is placed in the container under the name “result”, in the same way as an export parameter.

The source code for parameter transfer is generated from the definition data.

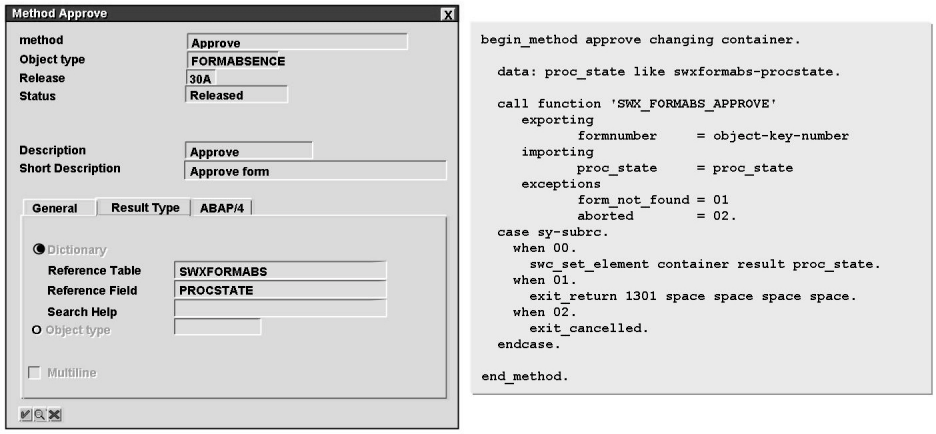


Figure 19: Implementing the Return of Results

The result parameter must be explicitly copied to the container using the SWC_SET_ELEMENT macro (for a multiline result: SWC_SET_TABLE) under the name “result” in the container. The name result is predefined as a fixed name.

Result parameters have an influence on the use of methods in workflow steps: If the result parameter is defined with a data reference on a fixed value domain (such as CHAR1 with the fixed values “A”, “B”, “C”), the modeler receives three outputs with the name “A”, “B”, and “C” in the Workflow Builder for the new step when he or she incorporates a step based on this method. When processing the workflow, the system always specifically selects the output that returns the method as a value for the result parameter.

Macros – Parameter Transfer in the Container

- Parameters are always transferred in a container
  - Reading and setting elements

    SWC_GET(_SET)_ELEMENT <cont> <element> <value>

    SWC_GET(_SET)_TABLE <cont> <element> <value_tab>
  - Initializing container (delete existing elements)

    SWC_CREATE_CONTAINER <cont>
  - Definition of an (additional) container variable

    SWC_CONTAINER <my_cont>

The container is a list with name-value pairs. The predefined variable called "container" must always be used within a method implementation. The method takes the import parameters from this container and copies its export parameters (and the result parameter) to this variable.

An additional container variable is necessary only if, within a method, you want to call another method that requires its own parameters.

Values are always activated using their names (no distinction is made between upper case and lower case). To read values from a container, you require an appropriate variable that can take this value.

Reminder: In the Business Object Repository, all object references are saved in variables of the type SWC_OBJECT ("runtime reference").

Special access commands are available for multiple-line elements (SWC_GET_TABLE or SWC_SET_TABLE).

Exceptions in Methods

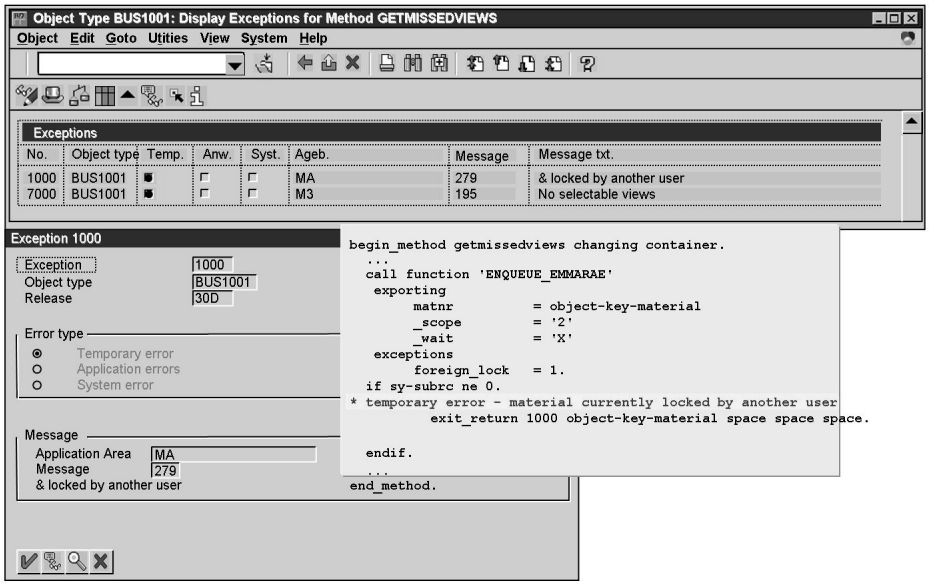


Figure 20: Implementing Exceptions in Methods

An exception is triggered directly for all error types (temporary, application or system errors) using the EXIT_RETURN macro. Source code is not automatically generated for triggering an exception.

Each exception refers to a T100 message. This message is displayed after the exception is triggered in the workflow log.

In the case of application errors, a number in the range 9001 to 9999 must be selected for the definition.

When you call the EXIT_RETURN macro, you must first specify the four-digit number of the exception. This is followed by four additional parameters that correspond to the four placeholders in the T100 message.

Macros: Method Implementation

- Triggering exceptions

  EXIT_RETURN <exception_no> <var1> ... <var4>

  EXIT_OBJECT_NOT_FOUND

  EXIT_CANCELLED

- Resetting the value buffer of an object after database changes

  SWC_REFRESH_OBJECT <obj_ref>

The following macros are used to trigger exceptions:

- EXIT_RETURN - trigger general exceptions (similar to T100 messages).
- EXIT_OBJECT_NOT_FOUND - Indicates that specified object instance not found.
- EXIT_CANCELLED - Indicates that user canceled the action (for example, in methods after BACK). The workflow system treats this macro in the same way as a temporary error.

Customer-defined exceptions must be in the customer namespace (numbers in the range 9001 to 9999).

As the final step, you should use the SWC_REFRESH_OBJECT macro to delete the internal buffer in the method source code of any method that changes object data. The workflow system automatically calls SWC_REFRESH_OBJECT when you execute a work item.

Responses to Exceptions in Workflow

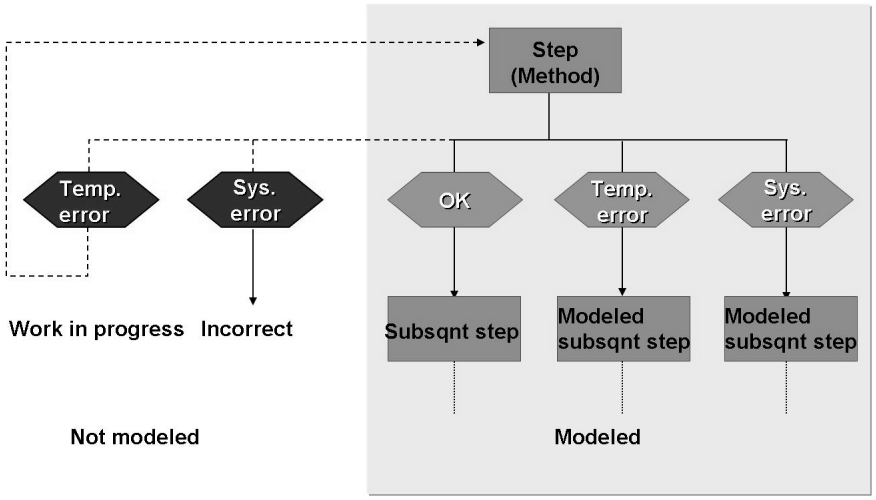


Figure 21: Reaction to Exception - Dialog Step

In the object method, exceptions are defined as temporary exceptions, as application errors, or as system errors. If a subsequent step is modeled in the workflow definition for this exception, this step is executed.

If no subsequent step is modeled, the workflow assumes “error” status for application errors and system errors, and for temporary errors the workflow remains in the status “in process”

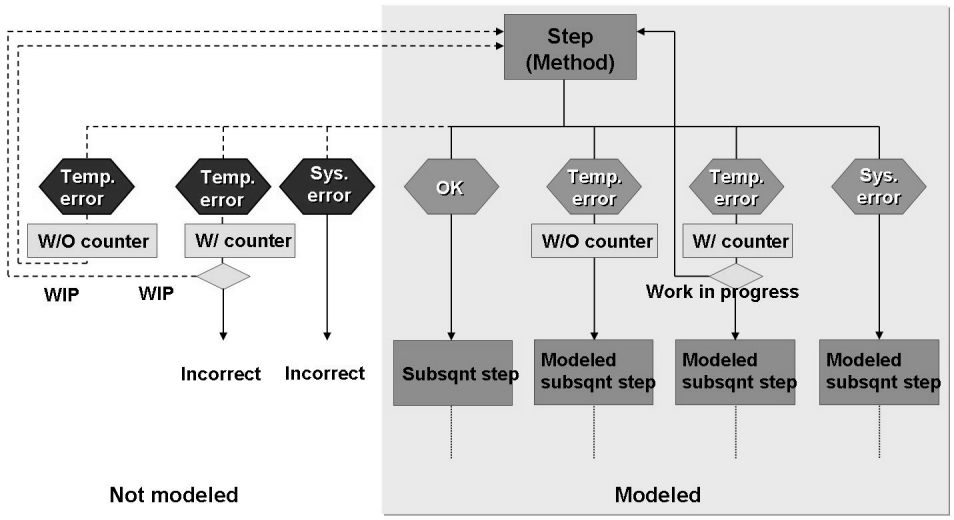


Figure 22: Reaction to Exception - Background Step

Background steps also allow you to model a repeat counter for a temporary error.

If this repeat counter is modeled, the step is automatically repeated for the number of times specified in the definition whenever a temporary error occurs. The step switches permanently to either “error” or “completed” status (followed by the modeled subsequent step) only when this number of repetitions is complete.

The repetition is implemented by a background job (SWWERRE). The background job must be scheduled to allow the repetition to be carried out if temporary errors occur.

Internal Use SAP Partner Only

Exercise 2: Create a subtype ZMARA##A for your supertype Z##MARA.

Exercise Objectives

After completing this exercise, you will be able to:

- Create a subtype.
- Create new methods in a subtype.
- Define virtual attributes.
- Call exceptions from methods.
- Call methods with parameters.

Business Example

When implementing a workflow, you will notice that the object types supplied by SAP do not meet your requirements fully. In these cases, you must enhance an existing object type by creating a customer-specific subtype for it.

Create a subtype ZMARA##A for your supertype Z##MARA (## stands for your group number). This enables you to make enhancements.

Task 1:

Create a subtype of your material object type.

1. Create a subtype ZMARA##A for your object type Z##MARA.

   Complete the basic data of your new object type as follows:

Supertype:	**Z##MARA**
Object Type:	**ZMARA##A**
Object name:	**MARA_##_Version_A**
Name:	**MARA_##_Version_A**
Description:	**Material group ##, version A**
Program:	**ZMARA##A**
Application:	**S**

Continued on next page

Task 2:

Enhance your new object type so that it meets additional requirements.

1. When developing workflows, you require triggering or terminating events for your tasks.

   Create two new events **BasicMaterialChanged** and **OldMaterialChanged** for your object type.

   Change the status of both events to **Implemented**.

2. Implement the two events using event creation from change documents.

   The required change document object is **MATERIAL**.

   Restrict the triggering of the event for **BasicMaterialChanged** to the field **MARA-WRKST** and for **OldMaterialChanged** to **MARA-BISMT**.

   The event **BasicMaterialChanged** should be executed for any change to the Basic material field.

   The event **OldMaterialChanged** is only to be executed when the *Old material number* field is changed to the value **G##**. (## is your group number).

3. For your object type define an additional event **DeadlineTest** with a parameter **LatestEndDate**, which refers to the field SYST-DATUM. This parameter will be used later to transfer an end date for a step within a workflow.

4. Create a new attribute **ChangedBy** based on the field MARA-AENAM. This attribute specifies the user who last changed the underlying material. You will use this attribute at a later stage to filter “your” material.

5. Create a new method **Change** based on transaction MM02.

Hint: Transaction MM02 writes your changes to the database using an update task.

Solution 2: Create a subtype ZMARA##A for your supertype Z##MARA.

Task 1:

Create a subtype of your material object type.

1. Create a subtype ZMARA##A for your object type Z##MARA.

   Complete the basic data of your new object type as follows:

Supertype:	**Z##MARA**
Object Type:	**ZMARA##A**
Object name:	**MARA_##_Version_A**
Name:	**MARA_##_Version_A**
Description:	**Material group ##, version A**
Program:	**ZMARA##A**
Application:	**S**

   a) Call transaction SW01 (Business Object Builder) and use the entries specified above to create a subtype for the object type Z##MARA.

      For an example solution, compare the structure or the relevant sample solution source code for the object type Z00MARA in the Business Object Repository.

Task 2:

Enhance your new object type so that it meets additional requirements.

1. When developing workflows, you require triggering or terminating events for your tasks.

   Create two new events **BasicMaterialChanged** and **OldMaterialChanged** for your object type.

   Change the status of both events to **Implemented**.

   a) Compare the structure with the object type Z00MARA in the Business Object Repository.

2. Implement the two events using event creation from change documents.

Continued on next page

The required change document object is **MATERIAL**.

Restrict the triggering of the event for **BasicMaterialChanged** to the field **MARA-WRKST** and for **OldMaterialChanged** to **MARA-BISMT**.

The event **BasicMaterialChanged** should be executed for any change to the Basic material field.

The event **OldMaterialChanged** is only to be executed when the *Old material number* field is changed to the value **G##**. (## is your group number).

a) Compare the structure with the object type Z00MARA in the Business Object Repository.

3. For your object type define an additional event **DeadlineTest** with a parameter **LatestEndDate**, which refers to the field SYST-DATUM. This parameter will be used later to transfer an end date for a step within a workflow.

   a) Compare the structure with the object type Z00MARA in the Business Object Repository.

4. Create a new attribute **ChangedBy** based on the field MARA-AENAM. This attribute specifies the user who last changed the underlying material. You will use this attribute at a later stage to filter "your" material.

   a) Compare the structure with the object type Z00MARA in the Business Object Repository.

Continued on next page

5. Create a new method **Change** based on transaction MM02.

Hint: Transaction MM02 writes your changes to the database using an update task.

a) Compare the structure with the object type Z00MARA in the Business Object Repository.

b) The following section contains the sample source code of the object type Z00MARA.

******* Implementation of object type Z00MARA *******

```
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,
      MATERIAL LIKE MARA-MATNR,
   END OF KEY,
      PLANT TYPE SWC_OBJECT.
END_DATA OBJECT. " Do not change. DATA is generated

BEGIN_METHOD CHANGE CHANGING CONTAINER.
   SET PARAMETER ID 'MAT' FIELD OBJECT-KEY-MATERIAL
   CALL TRANSACTION 'MM02' AND SKIP FIRST SCREEN.
END_METHOD.
```

Lesson Summary

You should now be able to:

- Define methods
- Call methods with parameters and return parameters
- Return exceptions from methods
- Describe how the Workflow Engine reacts to exceptions.

Lesson: Object Type Definition - Attribute Access and Method Calls

Lesson Overview

This lesson describes how to access attributes and methods of an object.

You may require this information for the implementation of virtual attributes or to use objects outside the Business Object Repository.

Lesson Objectives

After completing this lesson, you will be able to:

- Program an access to the attribute values of an object.
- Call methods of objects directly, for example, from a function module or report.
- Use the new parameter container interface to manipulate the container.
- Explain the different levels of performance of virtual attributes and background methods.

Business Example

You have implemented a new customer-specific object type and want to access attributes and methods from outside the Business Object Repository also.

For this reason you must learn about macros for attribute access and method calls.

Attribute Access - The User's Perspective

Attribute Access: The User's Perspective

- Procedure for querying object attributes:
  - Provide an object reference.
  - Obtain a variable that can take the attribute value.
  - Call the following macro:

    SWC_GET_PROPERTY <obj_ref> <attribute> <variable>

    or (if you have a multiline attribute)

    SWC_GET_TABLE_PROPERTY <obj_ref> <attribute> <int_table>.

You require a valid object reference to execute the attribute access. If you are accessing your own attributes, you can use the predefined SELF reference in the Business Object Repository. If you are accessing attributes from other instances or from outside the Business Object Repository, you might have to create the reference with SWC_CREATE_OBJECT.

Attribute access is carried out for single-line attributes using the macro SWC_GET_PROPERTY and for multiline attributes using SWC_GET_TABLE_PROPERTY with details of the attribute name and a variable to which the value is assigned. The Business Object Repository determines the default attribute if SPACE is transferred as the attribute name.

In your own programs, you must explicitly incorporate these macros with the command include <cntn01>.

Access to attributes is read-only.

You can change object values by calling methods only. This is because changes made to the application data are usually complex operations that can be executed properly only by the relevant application transactions (such as BAPIs or reports).

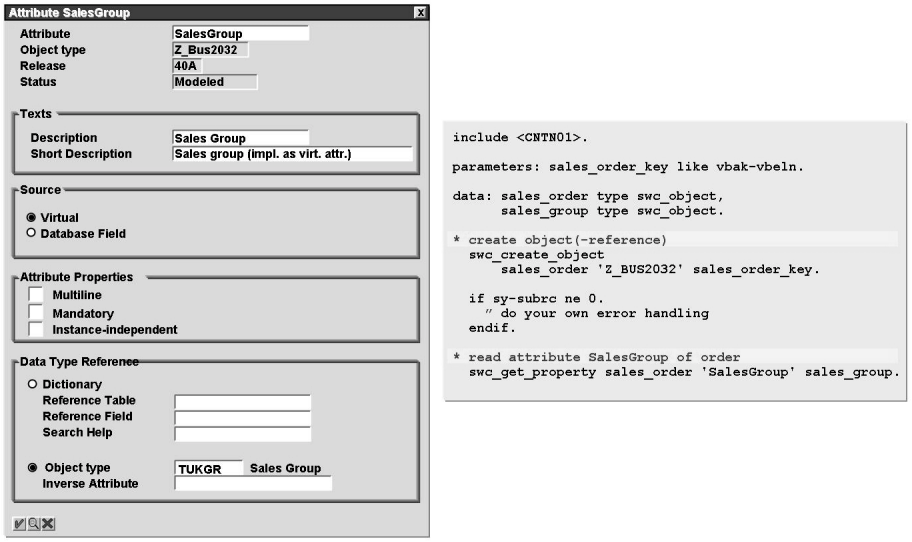


Figure 23: Implementing Attribute Access

In the example here, the object reference required to access the attribute is created on the object that you want to query by calling the macro SWC_CREATE_OBJECT. If the object reference cannot be created, attribute access is not possible and error handling measures must be put in place.

After the macro is called, the value of the SalesGroup attribute is available in the “sales_group” local variable, (if the call is completed with sy-subrc = 0).

The example displays a user-defined report. You must incorporate the include <cntn01> in the calling program so that you can use the object macros outside the Business Object Repository.

Method Calls: The User's Perspective

Method Call: The User's Perspective

- Procedure for calling methods:
  - Provide an object reference

    DATA <obj_ref> TYPE SWC_OBJECT.
  - Obtain a container to hold all parameters

    SWC_CONTAINER <cont>
  - Fill the container with all important import parameters:

    SWC_SET_ELEMENT <cont> <imort_parameter> <value>. and so on.
  - Call the method:

    SWC_CALL_METHOD <obj_ref> <Method> <cont>
  - Retrieve the export parameters from the container:

    SWC_GET_ELEMENT <cont> <export_parameter> <value>. and so on.

You require a valid object reference that allows you to call methods. In the Business Object Repository, you can use the predefined SELF reference when calling your own methods. If you are calling attributes from other instances or from outside the Business Object Repository, you may have to create the reference with SWC_CREATE_OBJECT.

When copying values to the container or when reading values from the container, note that single-line elements must be processed with the command SWC_GET(_SET)_ELEMENT, while multiline elements must be processed with the command SWC_GET(_SET)_TABLE. In your own programs, you must explicitly incorporate these macros with the include <cntn01>.

Methods are called with the macro SWC_CALL_METHOD with a specified method name and parameter container. The Business Object Repository calls the default method if SPACE is transferred as the method name.

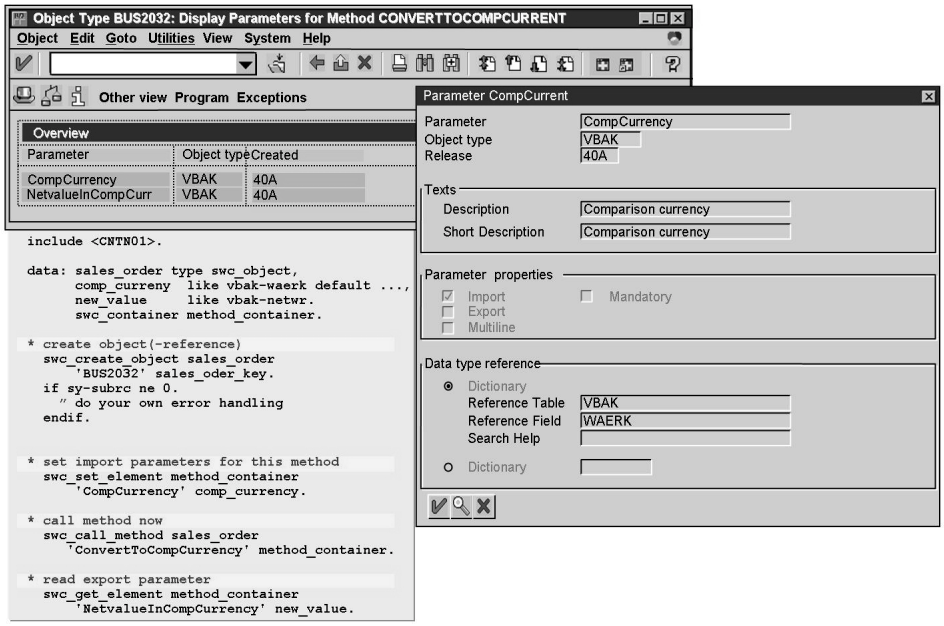


Figure 24: Implementing Method Calls

An object of type BUS2032 (sales order) is created in the example. If this is unsuccessful (because the key is invalid, for example), error handling measures must be put in place (macro EXIT_OBJECT_NOT_FOUND) because no methods can be called.

Before the method call, a container variable must be provided and filled with the import parameters of the method.

The actual method call is implemented with the macro SWC_CALL_METHOD.

When the call is successfully completed (confirmation prompt: SY-SUBRC = 0), you can read the export parameter or the result parameter from the container.

The example shows a user-defined report where the macros must be explicitly incorporated with the include <cntn01>.

Macros for Attribute Access and Method Call

Macros: Attribute Access and Method Call

- Attribute access
  - SWC_GET_[TABLE_]PROPERTY <obj_ref> <attribute> <value>
- Set method parameters
  - SWC_SET_ELEMENT/_TABLE <container> <element> <value>
- Method Call
  - SWC_CALL_METHOD <obj_ref> <method> <container>
- Read method parameters
  - SWC_GET_ELEMENT/_TABLE <container> <element> <value>

An object reference must be created before an object macro is called (for example, for attribute access or method call).

These macros are not usually used in object implementation, but are required in the program that the object uses.

If the <method> is SPACE for SWC_CALL_METHOD, the default method is executed.

Parameter Container Interface

Up to Basis Release 4.6C if you worked with the container, you influenced internal tables of the structure SWCONT using the container macros referred to above. To avoid the restricted options of the old container implementation, a new ABAP object-based programming model was introduced.

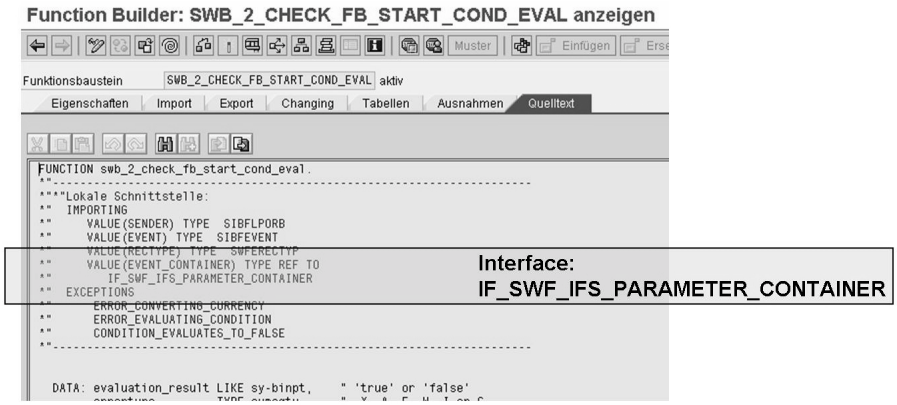
The container is displayed as an ABAP object reference of the type IF_SWF_IFS_PARAMETER_CONTAINER in this programming model.

- The old container is displayed as an internal table, that is, the complicated container macros were previously used to manipulate it
- The new container implementation is based on ABAP objects
- The (released) API IF_SWF_IFS_PARAMETER_CONTAINER is supplied to read and manipulate container data
- You can currently use the following methods:

Methods	Description
CLEAR	Reset an element to its type-related initial value
GET	Delivers the value of an element
GET_TYPE	Delivers the data type of an element
GET_VALUE_REF	Delivers a reference to the value of an element
LIST_NAMES	Delivers a list of names of all elements
SET	Sets the value of an element

Figure 25: Methods of the Parameter Container Interface

The purpose of the new container interface is to simplify the use of containers in check function modules when triggering events and so on.



- Program Examples for the New Programming Interface:
  - FM sww_wi_create_via_event_ibf (receiver FM for standard workflows)
  - FM swb_2_check_fb_start_cond_eval (check function module)
  - FM swe_event_create (function module to trigger an event)

Figure 26: An Example of a Parameter Container Interface

SWCONT has not completely disappeared, it is still used in particular for programming in BOR methods. However, some APIs have already been converted for use in the interface referred to above. Sooner or later additional APIs will follow.

Notes on Performance in the Area of Object Implementation

Performance 1: Buffering

- Database attributes are automatically buffered by the Business Object Repository.
- Buffering of virtual attributes must be programmed explicitly.
- You can buffer additional values in the 'PRIVATE' area of the program include.
- The buffer must be invalidated as soon as object data is changed in the database.
- Do not use buffering if synchronization cannot be guaranteed (buffer invalidation).

The more frequently attributes are queried and the more time-consuming the procedure for data retrieval, the greater the system performance benefits that can be derived from buffering.

If you have to buffer additional values for each object instance, but do not want these values to be displayed (like attributes) externally, you can use the area between BEGIN OF PRIVATE ... ...END OF PRIVATE. You must comment out the area in the program include and declare the required variables. The normal buffer control logic of the Business Object Repository applies to these variables, because they are part of the object structure.

If changes are made to object data on the database, you must delete the buffer by calling SWC_REFRESH_OBJECT. All of the changing methods of an object should carry out this deletion.

In some rare cases, changes are made in the one session to object data for which you cannot ensure a SWC_REFRESH_OBJECT call. If attributes are also required to have the “latest database status” in this, there is no point in carrying out buffering in the Business Object Repository.

Virtual Attributes	Background Methods
Read or calculate individual values or a list of homogenous values	Read or calculate a list of heterogeneous values
	Calculate values that are not dependent on other attributes or system values Change values
Evaluation and execution in Workflow Manager	Evaluation in Workflow Manager, execution in Work Item Manager
Buffering of the results in Business Object Builder	No buffering of results
Fixed communication with the caller	All communication methods possible

Figure 27: Performance 2: Virtual Attribute vs. Method

The return of the value of a virtual attribute to the caller corresponds, in principle, to a result value of a method.

Buffering of a virtual attribute must be programmed explicitly in the implementation of the attribute.

Performance 3: Recommendations

- Implementation in the Business Object Repository:
  - Always use macros to query your own attributes.

    SWC_GET_PROPERTY SELF <attribute name> ... .
  - Reuse the functions of your own methods.

    SWC_CALL METHOD SELF <method name> ... .
  - Do not use complicated checks in the existence-check-method.
- Use business objects outside the Business Object Repository and workflow also.

The buffer is refreshed (if necessary) when you call SWC_GET_PROPERTY. If you address a buffer variable object <attribute> directly, there is a risk that the status of the buffer will be undefined.

Virtual attributes and methods that retrieve data from the database can benefit greatly from Business Object Repository buffering.

The ExistenceCheck method of the object type is first called for every SWC_CREATE_OBJECT. For this reason, only essential source code should be run in this method.

Business objects encapsulate business logic that is used over and over again and that you do not want to have to reprogram every time. Using this business logic in your own projects (use include <cntn01>) reduces development time, avoids errors, and makes it easier to maintain the source code created.

Internal Use SAP Partner Only

Exercise 3: Create Virtual Attributes and Methods with Parameters and Exceptions

Exercise Objectives

After completing this exercise, you will be able to:

- Access attributes in the programming of virtual attributes.
- Define methods with parameter calls and exceptions.

Business Example

You want to create a method, which calls a material with a specific, pre-defined view.

Task 1:

Enhance your new object type, so that it meets additional requirements.

1. Create a new **DisplayView** method.

   This method displays a material under a particular view. The required view is transferred using the **MaterialView** input parameter.

   You must define the **MaterialView** input parameter with reference to the database field T132T-STATM.

   In your implementation, ensure that the "Basic data 1" view is displayed as the standard view. The value for the "Basic data 1' view is **K**. You must transfer this value if no other value has been transferred.

   You can use transaction MM03 to display the actual view.

   Transaction MM03 recognizes a **SET/GET Parameter MXX**, which you can use to control the view where the transaction is started. Assign the value of your input parameter to this SET/GET parameter.

2. A material master can be assigned to several different plants.

   Create a **Plant** virtual attribute that refers to your object type ZPLANT##.

   In this attribute, display one of the plants to which the material is assigned.

   Create the table MARC to find the particular plants that a material is assigned to. The key is the field WERKS.

Continued on next page

Cancel the application after the first record is found, and create an object reference of **the type ZPLANT##** for this plant. This object reference is returned in the Plant attribute.

Task 2:

Optional Enhancements for your Object Type

1. Create an **IndustryName** (virtual database field) attribute for the long text for the value of the existing **Industry** attribute.

   This industry name can be read from the field **MBBEZ** of the table **T137T** in the relevant languages.

   (Select * from T137T where spras = sy-langu and mbrsh = <value of industry for the current material>

   For this access you require the contents of the Industry attribute from the current material (use the SELF variable).

2. Create a (virtual, multiline and object-value) **QMessages** attribute, which lists all quality notifications, in which the current material is used.

   The object type **BUS2078** which is delivered, stands for quality notifications.

   The table **QMEL** contains the quality notifications and the field Matnr in QMEL contains the notifications that refer to a specific material.

   You can test the attribute with the material **200-200**.

3. Create a new method **CreateMaterialFromReference**.

   This instance-dependent, asynchronous method creates a new material on the basis of a material number transfered by the input parameter **ReferenceMaterialNumber** , for which the material type and the industry are taken from the reference material.

   The input parameter **ReferenceMaterialNumber** must be defined with reference to the database field **MARA-MATNR**.

   Use transaction **MM01** and choose Skip to call the first screen.

   The SET/GET parameter is used to display the material type, industry and material number of the template material.

Continued on next page

1) **MTA** (Set the material type)
2) **MTP** (Set the industry)
3) **RMA** (Set the reference material number)

If no material exists for the transferred material reference number, an exception is triggered with reference to the **T100 message M3 305** ("The material & does not exist or is not activated").

Solution 3: Create Virtual Attributes and Methods with Parameters and Exceptions

Task 1:

Enhance your new object type, so that it meets additional requirements.

1. Create a new **DisplayView** method.

   This method displays a material under a particular view. The required view is transferred using the **MaterialView** input parameter.

   You must define the **MaterialView** input parameter with reference to the database field T132T-STATM.

   In your implementation, ensure that the "Basic data 1" view is displayed as the standard view. The value for the "Basic data 1' view is **K**. You must transfer this value if no other value has been transferred.

   You can use transaction MM03 to display the actual view.

   Transaction MM03 recognizes a **SET/GET Parameter MXX**, which you can use to control the view where the transaction is started. Assign the value of your input parameter to this SET/GET parameter.

   a) As an example solution, compare the sample source code in the object type ZRGMARA in the Business Object Builder.

   b) BEGIN_METHOD DISPLAYVIEW CHANGING CONTAINER.

      DATA:MATERIALVIEW LIKE T132T-STATM.

      SWC_GET_ELEMENT CONTAINER 'MaterialView' MATERIALVIEW.

      IF MATERIALVIEW IS INITIAL.

      MATERIALVIEW = 'K'. " display basic view

      ENDIF.

      SET PARAMETER ID 'MAT' FIELD OBJECT-KEY-MATERIAL.

      SET PARAMETER ID 'MXX' FIELD MATERIALVIEW.

      CALL TRANSACTION 'MM03' AND SKIP FIRST SCREEN.

      END_METHOD.

2. A material master can be assigned to several different plants.

   Create a **Plant** virtual attribute that refers to your object type ZPLANT##.

Continued on next page

In this attribute, display one of the plants to which the material is assigned.

Create the table MARC to find the particular plants that a material is assigned to. The key is the field WERKS.

Cancel the application after the first record is found, and create an object reference of **the type ZPLANT##** for this plant. This object reference is returned in the Plant attribute.

a) As an example solution, compare the sample source code in the object type ZRGMARA in the Business Object Builder.

b) GET_PROPERTY PLANT CHANGING CONTAINER.

```
DATA: FIRST_PLANT LIKE MARC-WERKS OCCURS 0 WITH
HEADER LINE.

IF OBJECT-PLANT IS INITIAL. " check buffer contents.

" find one arbitrary plant related to this material

SELECT WERKS FROM MARC CLIENT SPECIFIED

INTO TABLE FIRST_PLANT

UP TO 1 ROWS

WHERE MANDT = SY-MANDT AND

MATNR = OBJECT-KEY-MATERIAL. " create an object reference using
the found plant key

" if we found no such plant, we'll implicitly create an empty ref

READ TABLE FIRST_PLANT INDEX 1.

SWC_CREATE_OBJECT OBJECT-PLANT 'ZPLANT00'
FIRST_PLANT.

ENDIF.

SWC_SET_ELEMENT CONTAINER 'Plant' OBJECT-PLANT.

END_PROPERTY.
```

Task 2:

Optional Enhancements for your Object Type

1. Create an **IndustryName** (virtual database field) attribute for the long text for the value of the existing **Industry** attribute.

Continued on next page

This industry name can be read from the field **MBBEZ** of the table **T137T** in the relevant languages.

(Select * from T137T where spras = sy-langu and mbrsh = <value of industry for the current material>

For this access you require the contents of the Industry attribute from the current material (use the SELF variable).

a) As an example solution, compare the sample source code in the object type ZRGMARA in the Business Object Builder, which is printed at the end of this solution.

2. Create a (virtual, multiline and object-value) **QMessages** attribute, which lists all quality notifications, in which the current material is used.

   The object type **BUS2078** which is delivered, stands for quality notifications.

   The table **QMEL** contains the quality notifications and the field Matnr in QMEL contains the notifications that refer to a specific material.

   You can test the attribute with the material **200-200**.

   a) As an example solution, compare the sample code in the object type ZRGMARA in the Business Object Builder, which is printed at the end of this solution.

3. Create a new method **CreateMaterialFromReference**.

   This instance-dependent, asynchronous method creates a new material on the basis of a material number transfered by the input parameter **ReferenceMaterialNumber** , for which the material type and the industry are taken from the reference material.

   The input parameter **ReferenceMaterialNumber** must be defined with reference to the database field **MARA-MATNR**.

   Use transaction **MM01** and choose Skip to call the first screen.

   The SET/GET parameter is used to display the material type, industry and material number of the template material.

   1) **MTA** (Set the material type)
   2) **MTP** (Set the industry)
   3) **RMA** (Set the reference material number)

Continued on next page

If no material exists for the transferred material reference number, an exception is triggered with reference to the **T100 message M3 305** ("The material & does not exist or is not activated").

a) As an example solution, compare the sample source code in the object type ZRGMARA in the Business Object Builder, which is printed at the end of this solution.

b) ***** Implementation of object type ZRGMARA *****

```
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,
MATERIAL LIKE MARA-MATNR,
END OF KEY,
QMESSAGES TYPE SWC_OBJECT OCCURS 0,
INDUSTRYNAME LIKE T137T-MBBEZ.
END_DATA OBJECT. " Do not change.. DATA is generated

TABLES: T137T.
TABLES: QMEL.
TABLES: MARA.
TABLES: T006.
```

Continued on next page


```
GET_PROPERTY INDUSTRYNAME CHANGING CONTAINER.
DATA: GETINDUSTRYNAME LIKE T137T-MBBEZ.
DATA: INDUSTRY_KEY LIKE MARA-MBRSH.
IF OBJECT-INDUSTRYNAME EQ SPACE. " check buffer contents
SWC_GET_PROPERTY SELF 'Industry' INDUSTRY_KEY.
SELECT SINGLE * FROM T137T
WHERE SPRAS = SY-LANGU AND
MBRSH = INDUSTRY_KEY.
IF SY-SUBRC EQ 0.
OBJECT-INDUSTRYNAME = T137T-MBBEZ.
ELSE.
OBJECT-INDUSTRYNAME = SPACE.
ENDIF.
ENDIF.
SWC_SET_ELEMENT CONTAINER 'IndustryName'
OBJECT-INDUSTRYNAME.
END_PROPERTY.

BEGIN_METHOD GETINDUSTRYNAME CHANGING CONTAINER.
DATA: GETINDUSTRYNAME LIKE T137T-MBBEZ.
DATA: INDUSTRY_KEY LIKE MARA-MBRSH.
SWC_GET_PROPERTY SELF 'Industry' INDUSTRY_KEY.
SELECT SINGLE * FROM T137T
WHERE SPRAS = SY-LANGU AND
MBRSH = INDUSTRY_KEY.
IF SY-SUBRC EQ 0.
GETINDUSTRYNAME = T137T-MBBEZ.
ELSE.
```

Continued on next page

```
GETINDUSTRYNAME = SPACE.
ENDIF.
SWC_SET_ELEMENT CONTAINER RESULT GETINDUSTRYNAME.
END_METHOD.

GET_PROPERTY QMESSAGES CHANGING CONTAINER.
DATA: LOCAL_QMEL LIKE QMEL OCCURS 0 WITH HEADER LINE.
DATA: THIS_QMEL TYPE SWC_OBJECT.
DATA: LINENO LIKE SY-TFILL.
DESCRIBE TABLE OBJECT-QMESSAGES LINES LINENO. " check
buffer contents
IF LINENO = 0.  " read from database
SELECT * FROM QMEL INTO TABLE LOCAL_QMEL
WHERE MATNR = OBJECT-KEY-MATERIAL.
LOOP AT LOCAL_QMEL.
SWC_CREATE_OBJECT THIS_QMEL 'BUS2078' LOCAL_QMEL-
QMNUM.
IF SY-SUBRC EQ 0.
APPEND THIS_QMEL TO OBJECT-QMESSAGES.
ENDIF.
ENDLOOP.
ENDIF.
SWC_SET_TABLE CONTAINER 'QMessages' OBJECT-QMESSAGES.
END_PROPERTY.

BEGIN_METHOD CREATEMATERIALFROMREFERENCE
CHANGING CONTAINER.
DATA: REFERENCEMATERIALNO LIKE MARA-MATNR,
INDUSTRY LIKE MARA-MBRSH,
```

Continued on next page


```
MAT_TYPE LIKE MARA-MTART.
DATA: REF_MAT TYPE SWC_OBJECT. SWC_GET_ELEMENT
CONTAINER 'ReferenceMaterialNo'
REFERENCEMATERIALNO.
SWC_CREATE_OBJECT REF_MAT 'Z00MARA' REFERENCEMATE-
RIALNO.
IF SY-SUBRC NE 0.
EXIT_RETURN 2000 REFERENCEMATERIALNO SPACE SPACE
SPACE.
ENDIF.
SWC_GET_PROPERTY REF_MAT 'Industry' INDUSTRY.
SWC_GET_PROPERTY REF_MAT 'MaterialType' MAT_TYPE.
SET PARAMETER ID 'RMA' FIELD REFERENCEMATERIALNO.
SET PARAMETER ID 'MTA' FIELD MAT_TYPE.
SET PARAMETER ID 'MTP' FIELD INDUSTRY.
CALL TRANSACTION 'MM01'.
END_METHOD.

BEGIN_METHOD COPYMATERIAL CHANGING CONTAINER.
DATA: INDUSTRY LIKE MARA-MBRSH,
MAT_TYPE LIKE MARA-MTART.
SWC_GET_PROPERTY SELF 'Industry' INDUSTRY.
SWC_GET_PROPERTY SELF 'MaterialType' MAT_TYPE.
SET PARAMETER ID 'RMA' FIELD OBJECT-KEY-MATERIAL.
SET PARAMETER ID 'MTA' FIELD MAT_TYPE.
SET PARAMETER ID 'MTP' FIELD INDUSTRY.
CALL TRANSACTION 'MM01'.
END_METHOD.
```

Lesson Summary

You should now be able to:

- Program an access to the attribute values of an object.
- Call methods of objects directly, for example, from a function module or report.
- Use the new parameter container interface to manipulate the container.
- Explain the different levels of performance of virtual attributes and background methods.

Unit Summary

You should now be able to:

- Explain the basic technical concepts of the Business Object Repository
- Create an object type.
- Define key fields.
- Define attributes with reference to a database.
- Define virtual attributes.
- Define multiline attributes.
- Describe how object references are accessed in BOR and in workflow.
- Define methods
- Call methods with parameters and return parameters
- Return exceptions from methods
- Describe how the Workflow Engine reacts to exceptions.
- Program an access to the attribute values of an object.
- Call methods of objects directly, for example, from a function module or report.
- Use the new parameter container interface to manipulate the container.
- Explain the different levels of performance of virtual attributes and background methods.

Test Your Knowledge

1. _____ enable the definition of applications, methods required for applications, and events, and make these available for all _____. The system calls the _____ of an object type, if a link to an object appears in the system, for example, in the work item preview, and it is unclear which method should be called for the execution. An object type and all its components must always have at least the status _____ before they can be called.

   Fill in the blanks to complete the sentence.

2. _____ connect the object type to a table of the DDIC. The first attribute, which is created in the object type, triggers the generation of a _____, in which the leading table of the object type is read. Virtual attributes must be declared explicitly using the macro _____ or _____.

   Fill in the blanks to complete the sentence.

3. You use the macros _____ or _____ to incorporate parameter values into the method. Exceptions refer to messages in the table _____. Errors can be _____, _____, or _____. A workflow, that contains a step in which an _____ occurs for a dialog method is _____, unless you have modelled an exception handling.

   Fill in the blanks to complete the sentence.

4. You can use the macro _____ to read attribute values. You can use the variable _____ to access the current object. In a method call, as the parameter you must enter the _____ the _____ and the _____, in which the method is to be called.

   Fill in the blanks to complete the sentence.

Answers

1. Interfaces enable the definition of applications, methods required for applications, and events, and make these available for all object types. The system calls the default method of an object type, if a link to an object appears in the system, for example, in the work item preview, and it is unclear which method should be called for the execution. An object type and all its components must always have at least the status implemented before they can be called.

   Answer: Interfaces, object types, default method, implemented

2. Key fields connect the object type to a table of the DDIC. The first attribute, which is created in the object type, triggers the generation of a form routine, in which the leading table of the object type is read. Virtual attributes must be declared explicitly using the macro SWC_SET_ELEMENT or SWC_SET_TABLE.

   Answer: Key fields, form routine, SWC_SET_ELEMENT, SWC_SET_TABLE

3. You use the macros SWC_GET_ELEMENT or SWC_GET_TABLE to incorporate parameter values into the method. Exceptions refer to messages in the table T100. Errors can be system errors, temporary errors, or application errors. A workflow, that contains a step in which an application error occurs for a dialog method is incorrect, unless you have modelled an exception handling.

   Answer: SWC_GET_ELEMENT, SWC_GET_TABLE, T100, system errors, temporary errors, application errors, application error, incorrect

4. You can use the macro SWC_GET_PROPERTY to read attribute values. You can use the variable SELF to access the current object. In a method call, as the parameter you must enter the method the method container and the object reference, in which the method is to be called.

   Answer: SWC_GET_PROPERTY, SELF, method, method container, object reference

Unit 3

Tasks – Binding – Programming Exits

Unit Overview

Tasks act as representatives of business process steps and call methods of a BOR object type.

This unit discusses the definition of tasks with synchronous and asynchronous methods, and the integration of tasks in the workflow template.

It also describes the standard binding between the different containers of a workflow, and explains when an 'alternative' binding is necessary, and how to define one.

Programming exits are new in the SAP Web Application Server. Customers can save data on work item status changes in their own reporting tables for subsequent evaluation.

Unit Objectives

After completing this unit, you will be able to:

- Explain the difference between a single-step task and a multistep task
- Define and start single-step tasks and multistep tasks
- Explain and define bindings within a multistep task
- Describe the existing container persistences and their differences.
- Identify when programmed binding is required.
- Create a programmed binding.
- Use a programmed binding in the workflow step.
- Use programming exits to save data for customer-specific reporting in customer-specific tables.
- Explain what a user-specific work item display is.

Unit Contents

Internal Use SAP Partner Only

Lesson: Using Standard Tasks and Container Handling

Lesson Overview

This lesson explains the concept behind calling synchronous and asynchronous methods in tasks.

It also describes how tasks can be started and integrated into workflows. This includes saving binding specifications, so that you can work correctly with the required objects in the individual business process steps at runtime.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the difference between a single-step task and a multistep task
- Define and start single-step tasks and multistep tasks
- Explain and define bindings within a multistep task
- Describe the existing container persistences and their differences.

Business Example

You want to use the methods and attributes of your new object type in a workflow.

To do this, you need to define tasks.

Definition of Standard Tasks

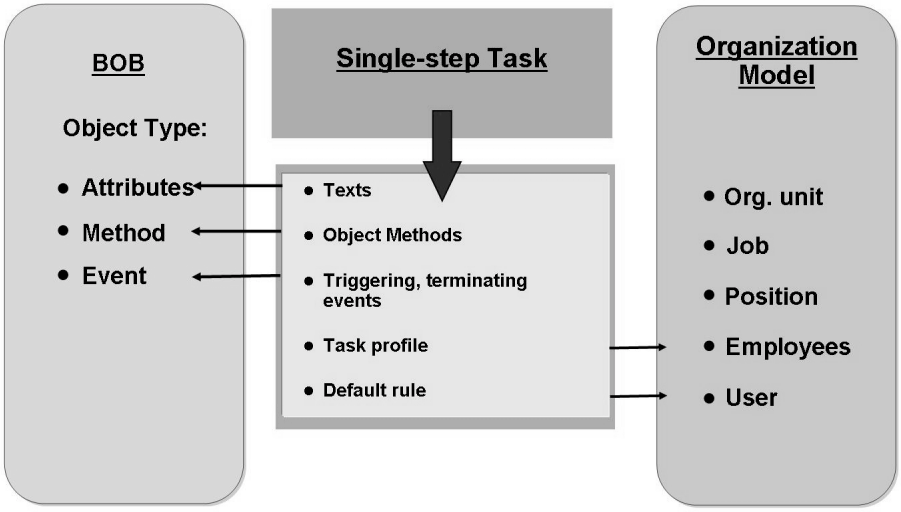


Figure 28: Single-step tasks

The default rule is evaluated if the task is used either outside a workflow or in a step definition without agent assignment.

The following texts can be defined:

- Work item text (workflow inbox in Business Workplace)
- Task description (work item preview in Business Workplace)
- Completion text
- Latest end text
- Latest start text
- Requested end text

Calling Synchronous and Asynchronous Methods

Synchronous/Asynchronous Tasks

- Single-step task referring to synchronous method
  - Single-step task is terminated when method reports back to caller
  - Binding between task container and method container in both directions
- Single-step task referring to asynchronous method
  - Method started by workflow system
  - Report back by terminating event
  - Single-step task is terminated when terminating event occurs

Communication between single-step task and object method:

- Synchronous:

  The method is called, it takes over the process control and, after it has been processed, reports back to the caller.

- Asynchronous:

  The method is called, the synchronous part of the method is processed and ends without returning any export parameters. The asynchronous part (update) runs without any connection to the caller and must report back to the caller with an event.

Note the following about synchronous/asynchronous calling and synchronous/asynchronous reporting back in the step definition of a workflow:

- The task method is always called synchronously (meaning in the same session as the calling program).
- Previous methods and subsequent methods are always called synchronously.
- Secondary methods are always called asynchronously (meaning in a new external session).
- Reports back to the workflow are made only from the task method. Secondary methods are for display only.

Synchronous tasks can also have terminating events. The task is then terminated either when it has been successfully executed or when the terminating event has been received. In this case, the terminating event cannot be triggered within the associated object method.

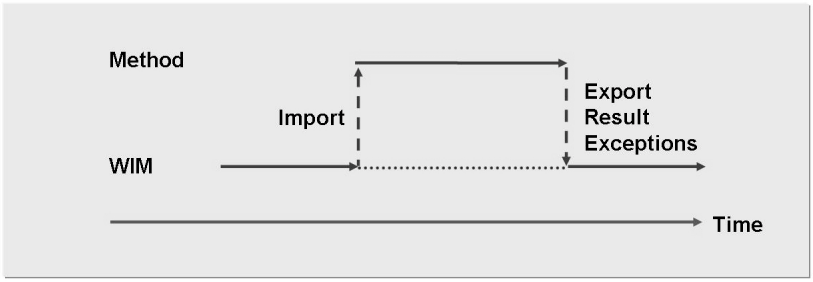


Figure 29: Synchronous Tasks

The task is started via the work item manager (WIM).

The method is started within the task. The work item manager caters for the input parameters of the method.

The work item manager waits until execution of the method is terminated.

After execution of the method is terminated, the work item manager receives the export parameters or exceptions of the method.

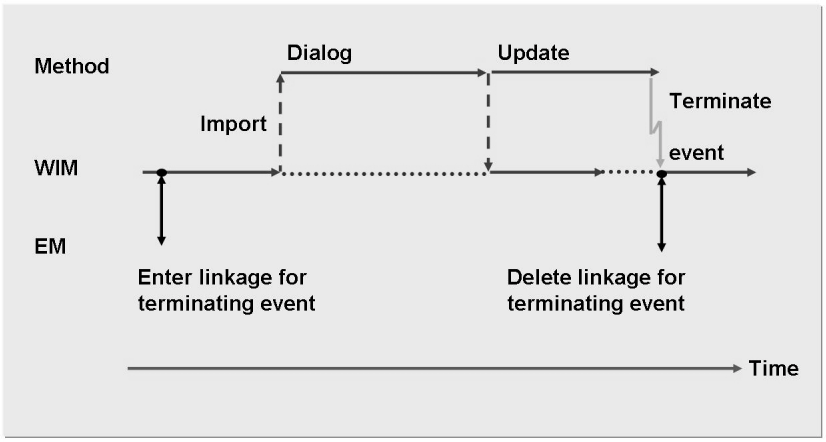


Figure 30: Asynchronous Tasks

The task is started via the work item manager (WIM). When the associated work item is created, the work item manager writes the linkages for the terminating events.

The method is started within the task. The work item manager caters for the input parameters of the method.

The work item manager waits for the end of the synchronous part of the method execution (this part takes place within a dialog process) and can then execute actions that are independent of the terminating event.

After the asynchronous part of the method execution is terminated (usually within an updating process), the work item manager receives a terminating event of the task, deletes all linkages, and then continues.

Options for Starting Tasks

Starting Tasks

- Automatically via the workflow system when you reach a step within a multistep task
- Directly using the workflow development environment
- Program-driven through the API of the work item manager
- Using a triggering event
- Directly using the system menu (for example, using the generic object services - GOS)

Usually, tasks are started automatically through the step definition when a workflow is processed.

For test purposes, tasks are started via the WF development environment. In this case, the current user must be a possible agent of the task.

Starting tasks via the work item manager API is discussed in the unit dealing with the runtime system.

Starting tasks via an event is discussed in the unit dealing with the event manager.

Tasks can be started using the system menu option “Workflow” from the application transaction, if the application has provided an appropriate link.

This link is already implemented in the following applications:

- Accounting (FI and CO)
- Logistics - Purchasing
- Logistics - Master Data
- Plant Maintenance
- Quality Management
- Service Management

Workflow Definition Steps

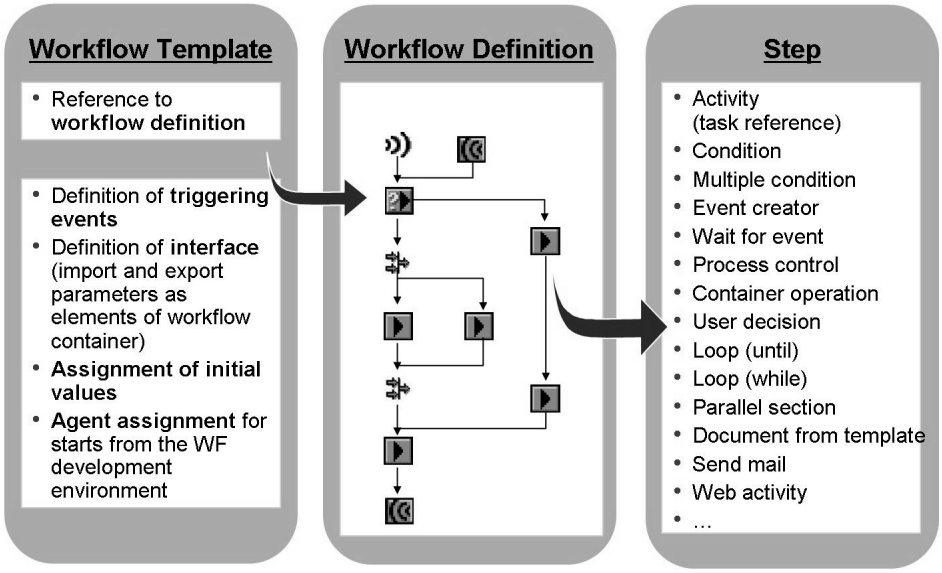


Figure 31: Workflow Definition Steps

All block-oriented modeling constructs such as branches and loops are available. The steps can be arranged in sequence or parallel.

The modeling interface (“Workflow Builder”) contains the relevant graphical operators for handling blocks. Each step is a determining component of its “own” block. Operations such as copying or deleting on individual steps (for example, the “loop” step type) always affect the entire block (the entire loop contents, for example) that depends on this step.

Container and Binding Definition

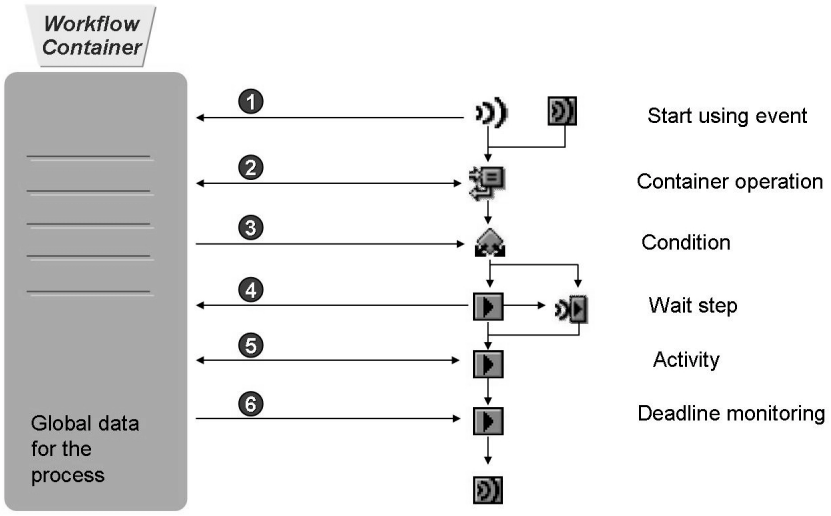


Figure 32: Using the Workflow Container

The workflow container stores the data available throughout the process. It is used to perform the following tasks:

- Specify the import and export parameters for the entire workflow (elements with the property “Import” or “Export”)
- Exchange data between individual steps

(1) When the workflow is started, data from a triggering event is transferred to the workflow container as input parameters.

(2) In a container operation, data in the workflow container can be changed by an allocation or a calculation.

(3) The expressions in a condition refer to data belonging to the workflow container.

(4) In a wait step, data is read from the workflow container (for example, to create the instance linkage for the event being received), whereas the event parameters are written to the workflow container with a binding.

(5)+(6) Data from the workflow container is read in an activity (binding for the task) and written (the output parameters of the task are transferred to the workflow container with binding).

In addition, data is read from the workflow container to resolve the rule used in the step for agent determination and determine any dates (6) defined on the step (such as the deadline) with an expression.

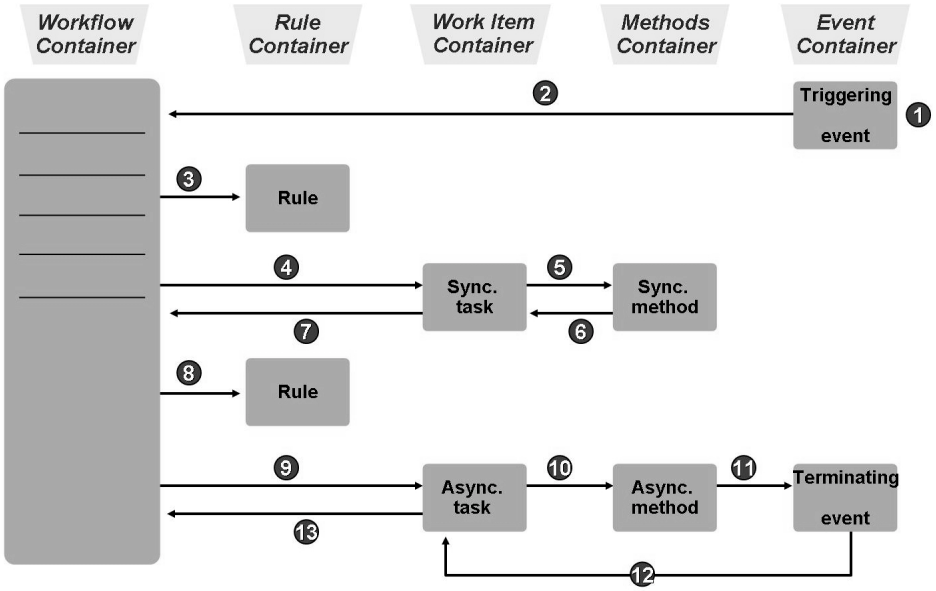


Figure 33: Binding in Workflow Execution

(1) (11)	The event container is filled when the event is created using the container macro (that is, before the event is triggered).
(2)	Binding from the triggering event to the workflow. This binding is absolutely essential; usually, the reference of the triggering object and the user name (or "event initiator") are transferred to the workflow container.
(3) (8)	Binding from the workflow container to the rule: This sets the input parameters for the rule if one is defined for agent determination in the step. If there is a default rule in the task definition, the parameters are supplied from the work item container.
(4) (7) (9) (13)	Binding between workflow container and task container (work item container) for setting the input parameters (during the call) and reading the output parameters of the task (once the task is completed). The Workflow Builder generates a proposal for this binding in the step definition.
(5) (6) (10) (12)	Binding between the task and the method container. There is a binding for synchronous tasks in import (5) and export (6) direction. In asynchronous tasks, the import binding is directly defined (10), whereas the return binding is defined during the terminating event (12). The runtime system copies the entire container if no binding is defined between the task and method.

Changing the Container Persistence

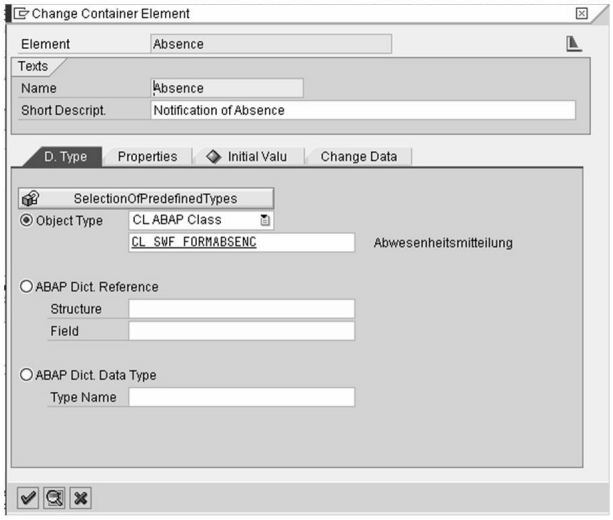


Figure 34: The Container Editor

Depending on the data type reference of the container element, the following entries are made in old releases:

- A structure reference
- An ABAP dictionary field
- An object type reference

The entries are stored at runtime using the old container persistence (the tables Tabellen SWW_CONT and SWW_CONTOB).

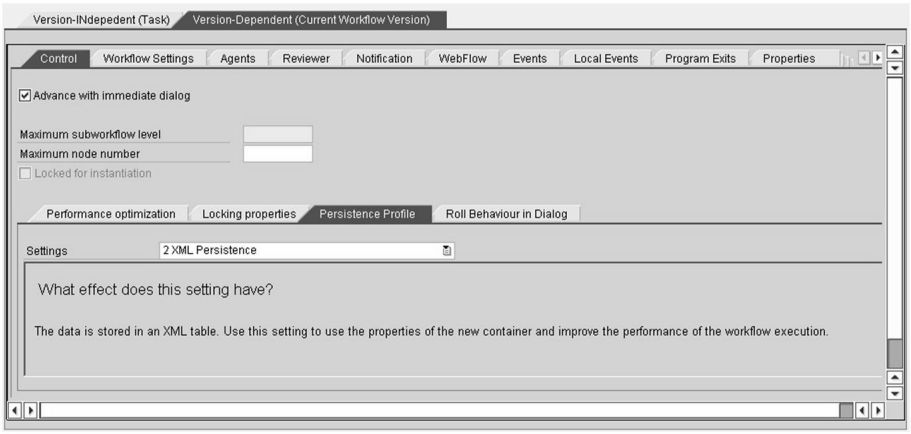


Figure 35: The XML Persistence

In releases as of Web Application Server 6.20 and higher you can now also create elements, which can reference all data types that are possible in ABAP.

These include the following data types:

- ABAP Dictionary types of any length
- Strings with unrestricted length
- Complex structures (not character-based and nested)
- ABAP objects classes

However, you can only use these new data types, if the newly developed **XML persistence** is used. The container saves the data as of 6.20 by default as an XML document.

The changes are available in all tools of the workflow system, that is, in the container editor, in the binding, in conditions and in the workflow steps (tasks).

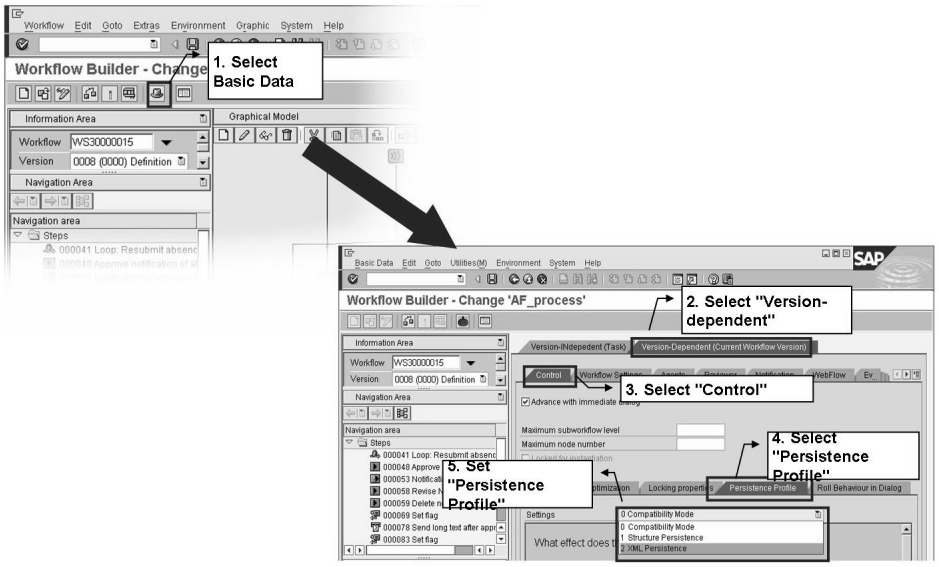


Figure 36: Changing the Container Persistence

The old BOR container cannot process these data types. Therefore, you must consider in particular when changing (old) workflows if the advantages of the new container persistence can be exploited by changing the workflow definition:

- Support of all data types that are used in ABAP
- Saving memory space (in particular for large containers or structures)
- Saving time when saving container data
- No database entry required for each structure field

Disregard the time required to convert (serialize or deserialize) data. The extra time is cancelled out relatively quickly by the faster saving times in the database.

Exercise 4: Creating a Workflow for the Business Process

Exercise Objectives

After completing this exercise, you will be able to:

- Create standard tasks.
- Test standard tasks.
- Implement standard tasks in a workflow template.
- Test a workflow template.

Business Example

You have either found or created all the object types that you want to use in your business process with the associated attributes and methods in the Business Object Builder.

Now you want to create single-step tasks based on the object methods, which you can use later in your workflow definition.

Hint: All tasks are based on the object type **ZMARA##A**.

Create three standard tasks to display or change a material master, or to display it with a pre-set view and test these. Insert the tasks created in a workflow and test them.

Task 1:

Create a new standard task for displaying a material master.

1. Create the **Z##MatDisp** task with the **Display** method.

   Define the task as a **general task**. Enter the following text as the work item text and description: **Display the changed material <Material number from object>)**

   Output the material number in both texts.

2. Test the task by starting it in the workflow development environment using transaction SWUS.

Continued on next page

Task 2:

Create a new standard task for changing a material master.

1. Create the task **Z##MatDisp** with the method **Change**.

   Define the task as a **general task**. Specify a work item text, a text for latest end and a text for completion. In addition to the material number, also issue the number and name of a plant for this material in all texts. Use the following texts:

   *Work item text:***Change material <Material number from object>**

   Latest end text **Material <Material number from object> not changed on time**

   Completion text: **Material < Material number from object> changed**

2. As this task is asynchronous, you must define at least one terminating event. Use the event **BasicMaterialChanged**.

   In this case, the event does not require a binding definition.

3. Test the task in two steps.

   Start the task in the first step as usual, but do not change the *basic material* field in the view *Basic data 2*.

   Instead, exit the transaction again. Then display the instance linkages from the technical work item display.

   To access the instance links, choose *Extras – > Instance link* from the menu.

4. Start the work item again and change the *basic material* field in the *Basic data 2* view as required. (Use the input help).

5. Check the event trace to ensure that the terminating event really was triggered.

   To display the event trace, use transaction **SWEL**.

Task 3:

Create a new standard task for displaying a material master from a predetermined view.

1. Create the task **Z##MatDispV** with the method **DisplayView**. Define the task as a general task. Enter a work item text.

   Text: **Material < Material number from object> changed. Display the material number with view.**

   Display the material number in the work item text.

Continued on next page

Define the view **K** (corresponds to the Basic data 1 view) as the default value for the parameter for controlling the view called for the material.

2. Test the task by starting it in the workflow development environment (transaction SWUS).

Task 4:

Define a workflow with a triggering event.

1. Create a workflow template called **Z##MSIPROC1** for changing a material master.
2. Incorporate the single-step task **Z##MatDisp** into your workflow definition.
3. Define **OldMaterialChanged** as the triggering event for your workflow. Implement the binding between the triggering event and the workflow and activate the event linkage.
4. Check, save, and activate your workflow definition. Test your workflow with your material by starting it from the workflow development environment.
5. Add the single-step task **Z##MatEdit** to your workflow definition as a new first step. Define the workflow initiator as the message recipient when processing is completed.
6. This step should have deadline monitoring. Activate deadline monitoring for the latest end. This is to be triggered if the step has not been fully processed after 5 minutes. Define the workflow initiator as the message recipient if the latest end is missed.
7. Check, save, activate and test your workflow again. When the “Edit” step is complete, check whether you have received a notification of completion in your Business Workplace.

Task 5:

Optional enhancement

Create and change a second workflow.

1. Copy your first workflow template **Z##MSIPROC1** to the new workflow template **Z##MSIPROC2**.
2. Change the message recipient for notification of completion for the “Edit” step. The “manager” of the workflow initiator should now receive the notification of completion.

Continued on next page

Use rule 168 to specify the manager of a particular user.

3. Define another input parameter **LatestEndDate** for your workflow. This parameter is used to determine the latest end for the "Edit" step.
4. Define another input parameter **MaterialView** for your workflow. This parameter is used to control the view in the display step of the process. Define view "K" as the default for this parameter.
5. Define **DeadlineTest** as another triggering event for your new workflow. Define the binding between the event parameters and the input parameters of the workflow. Activate the event linkage.
6. The latest end for the "Edit" step should fall on the date determined in **LatestEndDate**, five minutes after the current time. In your workflow definition, ensure that this parameter never contains a deadline in the past.
7. In the display step, replace the original task **Z##MatDisp** with the task **Z##MatDispV**. Assign the value of the **MaterialView** parameter from the workflow container to the step parameter for controlling the display view.
8. Check, save, activate and test your new workflow both by starting it from the development environment and by triggering the new triggering event explicitly.

Solution 4: Creating a Workflow for the Business Process

Task 1:

Create a new standard task for displaying a material master.

1. Create the **Z##MatDisp** task with the **Display** method.

   Define the task as a **general task**. Enter the following text as the work item text and description: **Display the changed material <Material number from object>)**

   Output the material number in both texts.

   a) Refer to the following task definition as a solution:

      TS30900005 – Z00MatDisp

      Use the following path to display the task (starting from SAP Easy Access):

      Tools → SAP Business Workflow → Development → Definition tools → Tasks/Task groups → Create

2. Test the task by starting it in the workflow development environment using transaction SWUS.

   a) Use transaction SWUS (Runtime - Test Workflow) to test the workflow,

      not forgetting to refresh the organizational environment and transfer input data for the test.

Task 2:

Create a new standard task for changing a material master.

1. Create the task **Z##MatDisp** with the method **Change**.

   Define the task as a **general task**. Specify a work item text, a text for latest end and a text for completion. In addition to the material number, also issue the number and name of a plant for this material in all texts. Use the following texts:

   *Work item text:***Change material <Material number from object>**

   Latest end text **Material <Material number from object> not changed on time**

Continued on next page

Completion text: **Material < Material number from object> changed**

a) Refer to the following task definition as a solution:

TS3090006 – Z00MatEdit

Use the following path to display the task (starting from SAP Easy Access):

Tools → SAP Business Workflow → Development → Definition tools → Tasks/Task groups → Create

2. As this task is asynchronous, you must define at least one terminating event. Use the event **BasicMaterialChanged**.

In this case, the event does not require a binding definition.

a) Enter the terminating event in the task definition.

3. Test the task in two steps.

Start the task in the first step as usual, but do not change the *basic material* field in the view *Basic data 2*.

Instead, exit the transaction again. Then display the instance linkages from the technical work item display.

To access the instance links, choose *Extras – > Instance link* from the menu.

a) Use transaction SWUS to test the workflow,

not forgetting to refresh the organizational environment and transfer input data for the test.

b) Use the work item display from the Business Workplace to display the instance data.

Use transaction SBWP or *Office → Workplace* and then the folders *Inbox* and *Workflow Display*, to display the work item.

The instance data is contained in the *technical work item display* under *Extras → Instance Data*.

4. Start the work item again and change the *basic material* field in the *Basic data 2* view as required. (Use the input help).

a) Change your material in transaction MM02, view *Basic data 2*.

5. Check the event trace to ensure that the terminating event really was triggered.

Continued on next page

To display the event trace, use transaction **SWEL**.

a) Use transaction SWEL to call the event trace and search for events for the object type ZMARA##A.

Task 3:

Create a new standard task for displaying a material master from a predetermined view.

1. Create the task **Z##MatDispV** with the method **DisplayView**. Define the task as a general task. Enter a work item text.

   Text: **Material < Material number from object> changed. Display the material number with view.**

   Display the material number in the work item text.

   Define the view **K** (corresponds to the Basic data 1 view) as the default value for the parameter for controlling the view called for the material.

   a) Refer to the following task definition as a solution:

      TS98000008 – Z00MatDispV

      Use the following path to display the task (starting from SAP Easy Access):

      Tools → SAP Business Workflow → Development → Definition tools → Tasks/Task groups → Create

2. Test the task by starting it in the workflow development environment (transaction SWUS).

   a) ***A prerequisite for the course is that participants know how to create and maintain a task.***

      The solution therefore only refers to created sample tasks.

      The following pages give a brief description of how to avoid problems when creating and maintaining tasks.

      The description follows the tasks.

      Create a standard task.

      1. Call transaction PFTS and choose Create.

      2. Input data on the basic screen:

      ID: As specified in the exercise

      Name: As appropriate for the required task

Continued on next page

Work item text: As specified in the exercise

Enter the material number in the text using the “Insert variables” button underneath the text.

Section object methods:

Object category: BOR object type

Enter the object type ZMARA##A and the method required in the exercise.

Press Enter to confirm. When prompted, transfer elements that are not available from the object method.

3. Enter the task description: Description tab page

Task description text type

Choose the Change button

Text: As specified in the exercise

Enter the material number by choosing the menu option Include - Expression

4. Enter the possible agents:

Possible agents are the users who are authorized to carry out the task in a business context.

Maintenance:

On the Task maintenance basic screen, choose the menu path: Additional Data – Maintain Agent Assignment. Select the task ID and choose the Properties button, then select General Task.

5. Enter texts for missed deadlines/completion: Description tab page

Text type: Required texts (latest end text/completion text)

Text: As specified in the exercise

6. Enter a terminating event: Terminating Events tab page

Entry in the ELEMENT field: Using the F4 help, select the element **_WI_OBJECT_ID**.

Again using the F4 help, select the terminating event **Basicmaterialchanged**.

7. Enter a default value for the input parameter Materialview/Maintenancestatus.

Continued on next page

Internal Use SAP Partner Only

Basic Screen of Task

Container tab page

Double click on the line containing **Materialview/Maintenancestatus**

Initial Value tab page

Enter the value K for the Basic Data **1** view.

8. Test a task

Call transaction SWUS.

Under Task, enter the following: TSxxxxxxxx

Transfer test data:

Click on the line containing the expression _WI_OBJECT_ID

An input line opens on the bottom edge of the screen.

Use the F4 help to select the material that you want to test.

9. Display the instance data

Call the Workplace (transaction SBWP or Office - Workplace)

Open the **Inbox** folder

Click on the **Workflow** folder.

Select the required work item and choose the **Display** icon.

Choose *Goto – Technical Work Item Display*

Choose *Extras – Instance Linkage*

10. Display the event trace

Call transaction SWEL

Enter your object type ZMARA##A

Choose Execute

The triggered terminating event is displayed.

In the detail data (double-click on Event), you can view the number of the work item that was terminated by the event.

Continued on next page

Task 4:

Define a workflow with a triggering event.

1. Create a workflow template called **Z##MSIPROC1** for changing a material master.

   a) ***A prerequisite for the course is that participants know how to create and test a workflow template.***

   Therefore, the following solutions refer to the example processes available in the system only.

   To avoid potential problems, the following pages contain a short description (in bullet points) of the steps to be executed. This follows the tasks.

   Define a workflow with a triggering event.

   Solution:

   WS96000011 – Z00MSIPROC1

   Path (starting from SAP Easy Access):

   Tools → Business Workflow → Development → Definition tools → Workflow Builder → Workflow Builder

   - Choose “Create new workflow”.
   - Choose “Save”.
   - Enter abbreviation “Z##MSIPROC1” and name.
   - Confirm and save your entries.

2. Incorporate the single-step task **Z##MatDisp** into your workflow definition.

   a) See the sample workflow:

   - Double-click the “Undefined Step” in the graphic.
   - Select "Activity".
   - Use the F4 input help to search for the single-step task referred to above.
   - Confirm the dialog to create elements and bindings.
   - Assign agent (for example, using _WF_Initiator)
   - Open the workflow interface (Import indicator)
   - Check and activate your entries.

Continued on next page

3. Define **OldMaterialChanged** as the triggering event for your workflow. Implement the binding between the triggering event and the workflow and activate the event linkage.

   a) See the sample workflow:

   - Go to Basic Data -> VersionINdependent -> Start Events.
   - Enter the event OldMaterialChanged for the business object type ZMARA00A.
   - Choose the “Binding” column to check the binding and confirm this.

     The symbol for the binding should now be displayed in green.

   - Choose the “Active” column to activate it.

     The symbol should no longer be gray, but should be green instead.

   - Save and activate your entries.

4. Check, save, and activate your workflow definition. Test your workflow with your material by starting it from the workflow development environment.

   a) See the sample workflow: Use, for example, the material T-BBD##.

5. Add the single-step task **Z##MatEdit** to your workflow definition as a new first step. Define the workflow initiator as the message recipient when processing is completed.

   a) See the sample workflow:

   - In the context menu right-click the symbol “Workflow started” in the graphic.
   - Choose Create -> Activity.
   - Use the F4 input help to search for the single-step task referred to above.
   - Confirm the dialog to create elements and bindings.
   - Assign agent (for example, using _WF_Initiator)
   - Check and activate your entries.

Continued on next page

6. This step should have deadline monitoring. Activate deadline monitoring for the latest end. This is to be triggered if the step has not been fully processed after 5 minutes. Define the workflow initiator as the message recipient if the latest end is missed.

   a) See the sample workflow:

   - Double-click the symbol for the activity you have just created.
   - Choose the "Latest End" tab.
   - Refer.date/time "Work Item Creation"
   - Set "Recipient of message when latest end missed " to Expression _WF_Initiator.
   - Confirm your entries.

7. Check, save, activate and test your workflow again. When the "Edit" step is complete, check whether you have received a notification of completion in your Business Workplace.

   a) See the sample workflow:

Task 5:

Optional enhancement

Create and change a second workflow.

1. Copy your first workflow template **Z##MSIPROC1** to the new workflow template **Z##MSIPROC2**.

   a) **Solution:**

   WS96000012 – Z00MSIPROC2

   Path (starting from SAP Easy Access):

   Tools → SAP Business Workflow → Development → Definition tools → Tasks/Task groups → Copy

   or

   Display your workflow in the Workflow Builder (SWDD).

   Workflow → Save as

2. Change the message recipient for notification of completion for the "Edit" step. The "manager" of the workflow initiator should now receive the notification of completion.

Continued on next page

Internal Use SAP Partner Only

Use rule 168 to specify the manager of a particular user.

a) See the sample workflow.

3. Define another input parameter **LatestEndDate** for your workflow. This parameter is used to determine the latest end for the "Edit" step.

   a) See the sample workflow.

4. Define another input parameter **MaterialView** for your workflow. This parameter is used to control the view in the display step of the process. Define view "K" as the default for this parameter.

   a) See the sample workflow:

5. Define **DeadlineTest** as another triggering event for your new workflow. Define the binding between the event parameters and the input parameters of the workflow. Activate the event linkage.

   a) See the sample workflow.

6. The latest end for the "Edit" step should fall on the date determined in **LatestEndDate**, five minutes after the current time. In your workflow definition, ensure that this parameter never contains a deadline in the past.

   a) See the sample workflow.

7. In the display step, replace the original task **Z##MatDisp** with the task **Z##MatDispV**. Assign the value of the **MaterialView** parameter from the workflow container to the step parameter for controlling the display view.

   a) See the sample workflow.

8. Check, save, activate and test your new workflow both by starting it from the development environment and by triggering the new triggering event explicitly.

   a) ***The following section provides you with some hints on the procedure for creating processes in the Workflow Builder:***

      1. **Creating a Workflow: Call transaction SWDD**

         Choose the *Save* button.

         Enter the required workflow ID.

         Create the workflow as a local object.

Continued on next page

2. **Integrate the standard tasks as specified in the exercise.**

   Select the undefined step in the Workflow Builder.

   Choose the *Create* step icon.

   Click on the step type *Activity*

   Enter your task

   Select the responsible agents from the possible agents:

   Section *Agents*

   Choose **Users** followed by the required users

   Define recipients for notification of completion:

   Notification tab page.

   Select the **Workflow initiator** expression.

   Define a missed deadline

   Latest End tab page.

   Reference time: *Generate work item*

   Action: *Send text*

   Recipient: Select the **Workflow initiator** expression.

3. **Link a triggering event**

   On the Workflow Builder basic screen: Click on the *Workflow Container* tab page (on the left of the screen).

   Double click on reference variables for your material, on the *Properties* tab page. Select the **Import** indicator.

   On the Workflow Builder basic screen, in the menu choose *Goto* → *Basic Data*, and choose the *Start Events* tab.

   Enter your object type and select the event **oldmaterialchanged**. Activate the event linkage by clicking on the button under A...

Continued on next page

(Event linkage active/inactive)

4. **Testing the Workflow**

   On the Workflow Builder basic screen - click on the **Test** icon.

   The workflow automatically switches to the test transaction SWUS.

   Remember to maintain the test data.

Lesson Summary

You should now be able to:

- Explain the difference between a single-step task and a multistep task
- Define and start single-step tasks and multistep tasks
- Explain and define bindings within a multistep task
- Describe the existing container persistences and their differences.

Lesson: Programmed Binding, Programming Exits and User-defined Work Item Display

Lesson Overview

In certain situations the binding between the tasks and workflow container cannot be processed by the standard binding.

In these cases you must work with programmed binding, which was previously called alternative binding.

in this lesson you will learn when programmed binding is required, how it is programmed and how you report it to the workflow.

You will also learn how to save data from the workflow runtime in customer-specific tables, to evaluate it later with customer-specific reports.

The third subject of the lesson is customer-specific tabs in the business view of work items.

Lesson Objectives

After completing this lesson, you will be able to:

- Identify when programmed binding is required.
- Create a programmed binding.
- Use a programmed binding in the workflow step.
- Use programming exits to save data for customer-specific reporting in customer-specific tables.
- Explain what a user-specific work item display is.

Business Example

You are not sure if the required data can be transferred in a business process step using the standard binding from the workflow container to the task container. You must learn about programmed or alternative binding.

You also want to evaluate runtime data of the workflow, which is not saved in standard tables of the Workflow Engine. You can use the programming exists to save data like this yourself in customer-specific tables for later evaluation.

Finally, you want to define a user-specific tab in one of your workflows in the work item display of a step, and you want to display this.

Internal Use SAP Partner Only

Programmed Binding and Changes in the Binding Editor

As of Basis Release 6.20, programmed binding is no longer required to carry out the following:

- Targeted filling of individual components in a structure: As of 6.20, any expressions can be used as the source and target of bindings.

  Before 6.20, you had to use programmed (alternative) binding if you wanted, for example, to fill specific individual fields in the BAPI structure when calling a BAPI method.

- Executing index accesses on multiline elements: To carry out this action, use the new expression syntax: &Tabelle[&Index&]&.

Expressions support (as of 6.20) index accesses to multiline elements

`&table[index]&`	Access to an entire table row
`&table[index].columnName&`	Access to components of a row
`&table[].columnName&`	Projection of a column
`&customers[1].orders[2]&`	Second purchse order of first customer

Expressions support (as of 6.40) functional method calls

Possible using business objects and classes

The expressions evaluate the target value of the methods

You can transfer parameters

Methods like this are not permitted to have side effects:

"Read only" functions + no database changes

Instance method withoutparameter	`&my_object.get_value( )&`
Instance method with parameters	`&my_object.methodA( param1=&exp1&; param2=17 )&`
Static methods	`%my_class.static_method( param1=&exp2& )%`

Figure 37: Changes in the Binding Editor

You can enter any binding statements in each of the rows in the binding editor. This means that you can use several binding modules consecutively and alternately with "normal" binding allocations.

As of Web Application Server 6.40, you can also call methods of business objects or classes. These methods can have export parameters only and cannot make database changes under any circumstances.

The system only provides methods that:

- Have at least one export or result parameter
- Are defined as synchronous
- Are defined as background method

Programmed Binding - Purpose

Up to and including SAP R/3 4.6c “Programmed binding” is the only option to implement special assignments. The purpose here is to create functions that cannot be modelled using the binding editor:

- User's own calculation functions (such as mean value)
- Transformations that require access to user's database tables (for example, conversion)
- Programming using the function module with fixed interface
- Enables definition of one or more calls for each binding event, even alternately with “normal”binding statements

Before 6.20 you could use a single binding module only and you had to enter it as an alternative to the “normal” binding (in the binding editor with the context menu “for alternative binding”).

- **Interface**

```
function  prog_binding
  importing       dataflow        like swabindef-dataflow
  tables          called_container          structure
swcont
                calling_container    structure  swcont
```

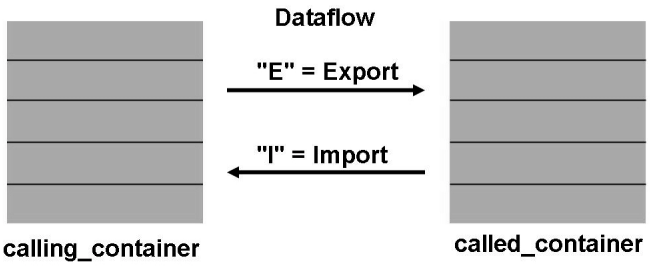


Figure 38: Programmed Binding- Function Module Interface

The variable DATAFLOW specifies the binding time: The value 'E' stands for export, that is, the binding from Calling_Container to Called_Container (for example, the binding from the workflow to the step). The value 'I' stands for import, that is, the binding from Called_Container to Calling_Container (for example, the binding from the step to the workflow container).

Reminder: Object references can be transferred as persistent references in the container (depending on the release). To be release-independent, the function module should apply the macro SWC_CONTAINER_TO_RUNTIME to calling or called containers at the start of processing, so that all object references are converted in the runtime display. The macro commands are incorporated with the include <cntn01>.

Additional restrictions:

- The FM must not trigger any exceptions.
- The FM must not trigger any commits or rollbacks.

Programming Work Items or Programming Exits – Purpose

The Purpose of Work Item Exits

- Exit technology enables user-defined reporting/logging.
- Programmed response to the start and end of a workflow
- Programmed response to any change in the status of a work item

Exit technology allows workflow modelers to save data to their own tables at set times while the workflow data is executed. You can then use this data to create an application-specific reporting function based on unrestricted application-related data, rather than on the technical status of the workflow system.

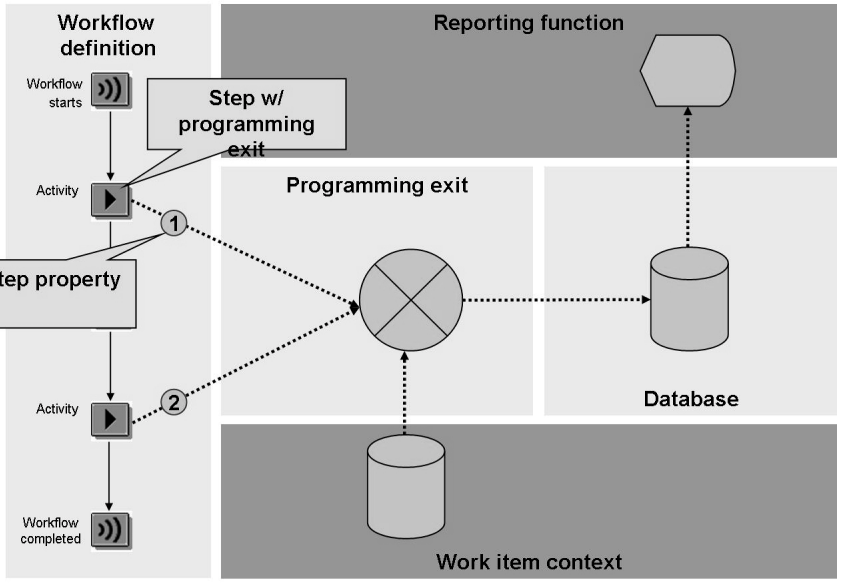


Figure 39: Application-specific Reporting

Calling times:

- Start and end of the workflow
- Creation, just before method execution, following rule resolution, and every time the status of a work item is changed (ready -> in process and so on.)

Programming Exits or Work Item Exits – Implementation

The Technology Behind Work Item Exits

- The exit is created as an ABAP class which implements a predefined interface.
- In the workflow step:
  - The exit class is entered.
  - Parameters are set for the exit class with constants ("Properties").
- The entire workflow context is available when you call the exit class at runtime.
- A user-defined reporting program evaluates the data written in the exit.

A programming exit is an ABAP class that implements the interface IF_SWF_IFS_WORKITEM_EXIT.

The relevant exit class must be entered in the modeling (Workflow Builder) for each of the workflow steps that you want to call for the exit. At this step you can also enter parameters for the call (“Properties”) with constant values. The names of the properties must be globally unambiguous. The prefix “sap.bc.bmt.wfm” is reserved for the SAP Workflow system.

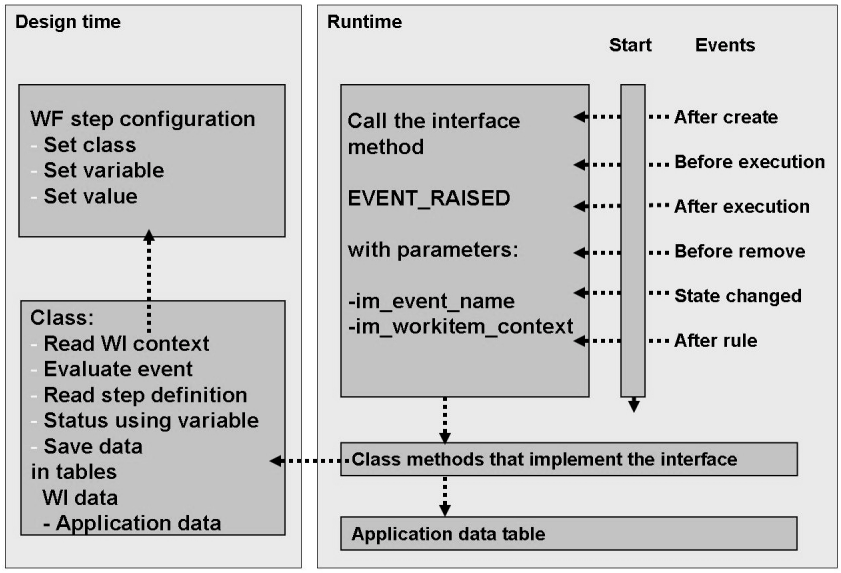


Figure 40: Time-line for the Programming Exit

The exit class is called by the Work Item Manager (WIM) every time the status of the work item is changed. When this happens, it receives the time (symbolic name) and the work item context as parameters. You can also use the context object to query the following:

- The values of the properties saved in the step at the definition time
- The entire work item header
- The workflow container and the container of the current work item

The exit class evaluates this information and fills a database specifically created for this purpose (“Reporting” or “User status”table).

While the workflow is being executed, you can read the status table using a user-defined report.

- Call time is specified
- Context data is determined (WI header or container)
- Call for workflow or for step?
- The required data is written
- Data is evaluated later with a user-defined report.

```
METHOD if_swf_ifs_workitem_exit~event_raised .
    …
    ls_wihead = im_workitem_context->get_header( ).
*-- handle workflow after creation
    IF ls_wihead-wi_type EQ swrco_wi_flow.
      IF im_event_name = swrco_event_after_creation.
        CALL METHOD me->process_started( ).
        …
      ELSE.
*---- handle work item after creation
      IF im_event_name = swrco_event_after_creation.
        CALL METHOD me->process_state_changed( ).
      ENDIF.
    ENDIF.
  …
ENDMETHOD.
```

Figure 41: Implementing Work Item Exits

Currently the interface IF_SWF_IFS_WORKITEM_EXIT contains one method only: "event_raised". This method is called by the workflow system every time the status is changed. Therefore, it must internally separate the possible calling times and respond appropriately to each required time (for example, by calling a suitable private method).

See the class **CL_SWH_WORKITEM_EXIT.** as a sample implementation.

The parameter IM_EVENT_NAME contains the time when the exit class is currently called. Symbolic constants for all possible times are contained in the type pool SWRCO.

The parameter IM_WORKITEM_CONTEXT contains a work item context object that can be used to determine and implement the work item header and work item container. The following selection of query methods is available:

- GET_HEADER (get work item header)
- GET_PROPERTY (query a property defined in the step)
- GET_RULE_RESULT (query result of the rule resolution)
- GET_STATE_TRANSITION (query current status transition (source and target status)
- GET_TASK_ID (query task ID)
- GET_WF_CONTAINER (query workflow container)
- GET_WF_CONTAINER (query work item container)
- ...

User-Defined Work Item Display

User-Defined Work Item Display

- Purpose: The purpose of the display is to provide user-specific functions directly in the workflow inbox.
  - Additional tab in the work item display
  - Subscreen contains its own flow logic
  - Subscreen communicates with superordinate work item display

Process or step-specific data can be displayed on the additional tab. The user's subscreen may also contain processing functions (for example, “Approve” and “Reject”) to allow direct processing from the inbox.

The user-defined tab is never displayed in synchronous processing chains, because the object method on which the step is based is executed immediately.

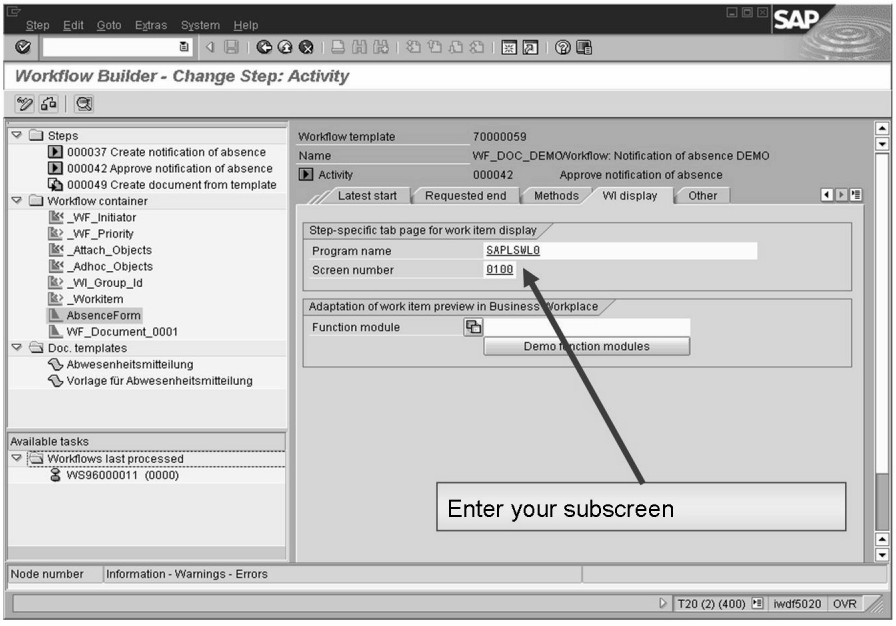


Figure 42: Reference in the Step Definition

The screen entered must be defined as a “subscreen”.

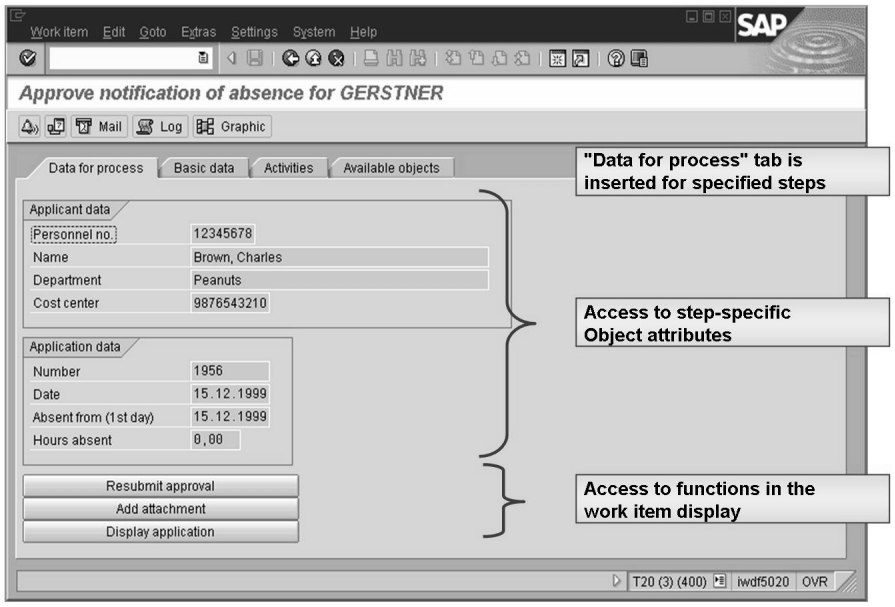


Figure 43: Runtime Display

The “Data for process” tab is always displayed as the first tab.

The “Resubmit Approval” button is mapped to OK Code 0002.

The “Add attachment” button is mapped to OK code 0018.

The “Display application” button is handled directly using the SHOW_FORM form.

Internal Use SAP Partner Only

Exercise 5: (Optional) Create a Programmed Binding

Exercise Objectives

After completing this exercise, you will be able to:

- Create a function module for a programmed binding
- Use the programmed binding in the workflow step

Business Example

You have a workflow running in release 4.6C and want to exchange data between single line and multiple line elements. To enable this, you need a programmed binding.

Create a programmed binding for the step to display your material master with a pre-defined view.

Task:

Optional enhancement

Alternative binding

1. Create a function module.

   Create a function module **Z##_BIND_WF_TO_DISPLAY_STEP** for an alternative binding between the workflow container and the task container of the display step.

   Hint: You only require one binding from the workflow container to the task container of the task **Z##MatDispV**.

2. Enter your new function module as an alternative binding for the display step in your workflow definition **Z##MSIPROC2**.

Solution 5: (Optional) Create a Programmed Binding

Task:

Optional enhancement

Alternative binding

1. Create a function module.

   Create a function module **Z##_BIND_WF_TO_DISPLAY_STEP** for an alternative binding between the workflow container and the task container of the display step.

   Hint: You only require one binding from the workflow container to the task container of the task **Z##MatDispV**.

   a) See the coding in the example solution in the function module **Z00_BIND_WF_TO_DISPLAY_STEP**.

   b) The following is the coding for the sample solution:

```
FUNCTION Z00_BIND_WF_TO_DISPLAY_STEP.
*"----------------------------------------------------------------------
*"*"Local interface:
*" IMPORTING
*" VALUE(DATAFLOW) LIKE SWABINDEF-DATAFLOW
*" TABLES
*" CALLED_CONTAINER STRUCTURE SWCONT
*" CALLING_CONTAINER STRUCTURE SWCONT
*"----------------------------------------------------------------------

DATA: MAT_OBJECT TYPE SWC_OBJECT.
```

Continued on next page

Internal Use SAP Partner Only

```
DATA: MAT_VIEW LIKE T132T-STATM.

IF DATAFLOW EQ 'E'.
" only consider export dataflow from wf to wi container
" read data from workflow (= calling) container

SWC_CONTAINER_TO_RUNTIME calling_container.
SWC_CONTAINER_TO_RUNTIME called_container.

SWC_GET_ELEMENT CALLING_CONTAINER 'MaterialMaster'
MAT_OBJECT.
SWC_GET_ELEMENT CALLING_CONTAINER 'MaterialView'
MAT_VIEW.
" write data to work item (= called) container
SWC_SET_ELEMENT CALLED_CONTAINER WI_LEADING_OB-
JECT
MAT_OBJECT.
SWC_SET_ELEMENT CALLED_CONTAINER 'MaterialView'
MAT_VIEW.

SWC_CONTAINER_TO_PERSISTENT called_container.
SWC_CONTAINER_TO_PERSISTENT calling_container.
ENDIF.

ENDFUNCTION.
```

2. Enter your new function module as an alternative binding for the display step in your workflow definition **Z##MSIPROC2**.
   a) Call the binding for the step that displays the material master, and enter the function module **Z00_BIND_WF_TO_DISPLAY_STEP** instead of the standard binding.

Lesson Summary

You should now be able to:

- Identify when programmed binding is required.
- Create a programmed binding.
- Use a programmed binding in the workflow step.
- Use programming exits to save data for customer-specific reporting in customer-specific tables.
- Explain what a user-specific work item display is.

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Explain the difference between a single-step task and a multistep task
- Define and start single-step tasks and multistep tasks
- Explain and define bindings within a multistep task
- Describe the existing container persistences and their differences.
- Identify when programmed binding is required.
- Create a programmed binding.
- Use a programmed binding in the workflow step.
- Use programming exits to save data for customer-specific reporting in customer-specific tables.
- Explain what a user-specific work item display is.

Internal Use SAP Partner Only

Test Your Knowledge

1. Deadline monitoring can refer to the _____, _____ and _____ events. Asynchronous methods can return exceptions as long as the exception is triggered in the _____ of the method. The difference between synchronous and asynchronous methods is not in the _____, but only in the selection of the flag (_____).

   Fill in the blanks to complete the sentence.

2. In Release 4.6C you require programmed bindings, to exchange values between single-line and multiline elements and to edit structures.

   Determine whether this statement is true or false.

   □ True

   □ False

Answers

1. Deadline monitoring can refer to theWork item creation, Workflow creation and Expression events. Asynchronous methods can return exceptions as long as the exception is triggered in the Dialog part of the method. The difference between synchronous and asynchronous methods is not in the coding, but only in the selection of the flag (synchronous/asynchronous).

   Answer: Work item creation, Workflow creation, Expression, Dialog part, coding, synchronous/asynchronous

2. In Release 4.6C you require programmed bindings, to exchange values between single-line and multiline elements and to edit structures.

   Answer: True

   In Release 4.7 programmed binding is no longer required due to the restructuring of bindings. In releases before 4.7 there was no option for assignment in the binding editor.

Unit 4

Rule Function Modules

Unit Overview

The recipients of work items are users who are in the intersection between possible agents (defined in the standard task) and responsible agents (defined in the workflow step).

The responsible agents can be determined using rules.

The rule with "function to be executed", that is, calling a function module at runtime, is one of the existing rule types.

This unit explains how to implement a function module of this type and use it in the workflow.

Unit Objectives

After completing this unit, you will be able to:

- Create a rule function module
- Use a rule function module in a workflow step

Unit Contents

Internal Use SAP Partner Only

Lesson: Agent Determination using Rule Function Modules

Lesson Overview

The responsible agents of a workflow step can be determined using rules.

The rule with "function to be executed" that is, calling a function module at runtime, is one of the existing rule types.

This unit explains how to implement a function module of this type and use it in the workflow.

Lesson Objectives

After completing this lesson, you will be able to:

- Create a rule function module
- Use a rule function module in a workflow step

Business Example

You want to determine the responsible agents in a workflow step using a rule function module. You need to learn how to create and check a module of this type.

Agent Determination in the Workflow

Agent Determination in the Workflow

- First level (task):

  Task profile defines “possible agents”

- Second level (step):

  Determination of “responsible agents” or default rules for the task

- Third level (step):

  Removal of “excluded agents”

On the first two levels, you can use an organizational object (in particular, a rule) to determine the group of agents. In the workflow (in a step definition), you can also use an expression to determine an agent.

The three levels depend on each other hierarchically in the sense that each level further restricts the previous level. Therefore, if the first level has already returned an empty set to agents because the task profile was not maintained, for example, other agents cannot be added in the subsequent levels.

Unlike e-mails, the work items derived from the tasks are not delivered as copies. Instead, only one work item exists for all recipients and they only have one view of it.

The explicit definition of possible agents in the task definition can be replaced by classification as a “general task”. All users known in the system can then process the task.

If a rule is specified both in the step definition and in the task definition, the rule in the step definition overrides the default rule from the task definition.

Users who are not possible agents of the task never see the work item, even if they are the responsible agents as a result of a rule resolution.

The default rule of a task is also evaluated if the agent assignment using a rule in the step definition ends without a result, and the “Terminate if rule resolution has no result” indicator is not set for this rule.

Examples of Rule Resolution

Rule Resolution

- Evaluation using organization model
  - Manager of...

    Manager of the organizational unit

    Department secretary
- Evaluation using master data
  - Person responsible for material

    Customer service operator
- Evaluation using Customizing data
  - Administration clerk

    Person authorized to release

Rules are used to restrict or more precisely define the number of recipients from a business perspective.

Advantages:

- dynamic restriction of the number of possible agents
- The actual organizational plan does not need to be known
- Agents are determined according to current conditions (rule resolution at runtime)

A rule definition contains

- The definition of rules for rule resolution (→ function module) and
- specified runtime-dependent rule parameters required for rule resolution.

You can use the “Manager of...” rule as one example to clarify a rule definition and the use of rules and rule resolution. This rule delivers the superior of the transferred user. The relevant data must be entered in the organizational model.

The rules “Manager” and “sap_org_unit” rules are supplied by SAP.

Addressing with Function Modules

Addressing with Function Modules

- Rule function modules can perform any evaluations
- Rule function modules must be written by the customer
- Interface

```
function role_fm
tables  actor_tab  structure swhactor
ac_container structure swcont
exceptions nobody_found
```

- Container holds the parameters defined for the rule

The rule container is defined using the rule definition.

The rule container must be filled by a binding. If the rule is used in a step, data can be transferred from the workflow container. If the rule is used as the default rule for the task, data can be transferred from the task container.

The function module must comply exactly with the above interface. The table “AC_CONTAINER” represents the container for the rule; the rule parameters are transferred to this container. In the table “ACTOR_TAB” the rule function module sets the determined agent.

When returning agents using the ACTOR_TAB table, you must consider the structure of the table:

- Position 1+2: ACTOR_TAB-Otype

  Represents the object that has been found, for example, US (user), O (organizational unit), S (position), and so on.

- Position 3-14: ACTOR_TAB-Objid

  identifies object that was found, (for example, WF-BC-M-20, 50006543, and so on).

Binding for using a rule of type "function to be executed"

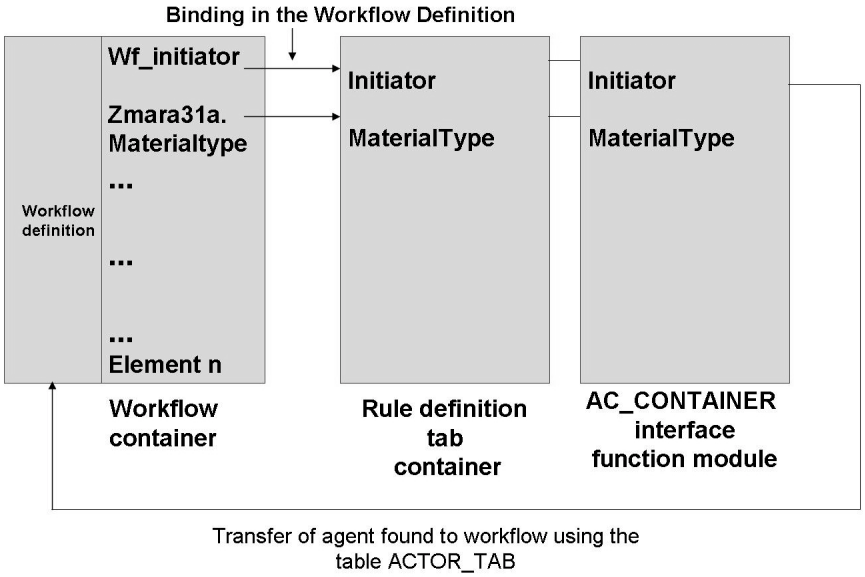


Figure 44: Binding in 'Function to be Executed' Rules

Internal Use SAP Partner Only

Procedure for creating a "function to be executed" rule:

1. In a suitable function group, create a function module with the correct interface, and activate the function module.
2. Create a rule of type “Function to be executed”

   Enter the function module

   On the Container tab page, enter the elements that are required for your check
3. Call the function module

   • From the rule container (AC_CONTAINER), choose the element/s that you have defined

   under

   point 2, in local variables.

   • Create coding for the checks you want to perform

   • Fill the table ACTOR_TAB to inform the workflow of the agents found
4. Enter the rule into the required workflow step

   In the binding for the rule, fill the rule container from the workflow container.

To define a rule, choose:

Tools – Business Workflow – Development – Definition Tools – Rules for Agent Assignment – Create/Change/Display

Structure of a Rule Function Module

- Rule parameters are read from the rule container
- Internal application data is evaluated for agent determination
- Agents determined are returned
- If an agent could not be determined: The exception nobody_found is triggered

```
include <CNTN01>.

function swx_dli_read_for_role.
*"----------------------------------------------------------------------
*"       TABLES
*"              ACTOR_TAB STRUCTURE SWHACTOR
*"              AC_CONTAINER STRUCTURE SWCONT
*"       EXCEPTIONS
*"              NOBODY_FOUND
*"----------------------------------------------------------------------
data:
  dli_name    like soobjinfi1-obj_name,
  dli_entries like sodlienti1 occurs 0 with header line.

clear actor_tab. refresh actor_tab.

swc_get_element ac_container 'DLI_NAME' dli_name.

call function 'SO_DLI_READ_API1'
     exporting    dli_name    = dli_name
     tables       dli_entries = dli_entries
     exceptions   others      = 99.

if sy-subrc ne 0. raise nobody_found. endif.

loop at dli_entries.
  where member_typ eq space.
    actor_tab-otype = 'US'.
    actor_tab-objid = dli_entries-member-nam.
    append actor_tab.
endloop.

if actor_tab[] is initial. raise nobody_found. endif.

endfunction.
```

Figure 45: Structure of a Rule Function Module

The above is only an extract - the actual SWX_DLI_READ_FOR_ROLE function module is more detailed. This function module is used for determining agents on the basis of a distribution list.

The result of a rule resolution does not necessarily have to be a user. It can also be any organizational object (such as an organizational unit, a position, and so on).

Internal Use SAP Partner Only

Exercise 6: Create a rule function module that returns an agent depending on the material type of the object used.

Exercise Objectives

After completing this exercise, you will be able to:

- Create a rule function module
- Use and test a rule function module in the workflow

Business Example

You want the business process step for changing the material to be assigned to different agents depending on the material type of the object used.

If the material type is ROH, the manager of the workflow initiator receives the work item. If the material type is not ROH, the workflow initiator processes the step themselves.

Task:

Create a new rule **Z##MatDispAc** using a rule function module.

1. In the Object Navigator (SE80), create a function group called **`Z610_##`** for your group, and activate it.
2. Create a rule function module in your function group.

   Z##_MM_MGR_OR_USER_GET.

   This function module returns the “manager of” the workflow initiator as the agent if the material type of the material entered has the value “ROH”. Otherwise, the initiator is returned.

   To determine the manager of the workflow initiator, you can call the function module SWX_GET_MANAGER.

   This function module is a rule function module itself and expects data that it has to check in the AC_CONTAINER.

   In the Workflow Builder, display an example of the binding for the “Manager of..” rule, to establish to which particular container element you need to assign the workflow initiator.

Continued on next page

3. Now define the rule using the new function module. In this case, define the parameters for this rule according to the implementation of your function module.

   You define the rule parameters using the rule container.

   The workflow initiator can be defined with reference either to field WFSYST-INITIATOR or to RHOBJECTS-OBJECT.

4. Use the new rule for agent assignment of the Display step in your workflow **Z##MSIPROC2** or **Z##MSIPROC1**.

5. Check, activate and test your workflow Z##MSIPROC2 or Z##MSIPROC1.

Solution 6: Create a rule function module that returns an agent depending on the material type of the object used.

Task:

Create a new rule **Z##MatDispAc** using a rule function module.

1. In the Object Navigator (SE80), create a function group called **`Z610_##`** for your group, and activate it.

   a) Call transaction SE80 (ABAP Workbench) and create a new function group **Z610_##**. Activate the function group using the context menu.

2. Create a rule function module in your function group.

   Z##_MM_MGR_OR_USER_GET.

   This function module returns the "manager of" the workflow initiator as the agent if the material type of the material entered has the value "ROH". Otherwise, the initiator is returned.

   To determine the manager of the workflow initiator, you can call the function module SWX_GET_MANAGER.

   This function module is a rule function module itself and expects data that it has to check in the AC_CONTAINER.

   In the Workflow Builder, display an example of the binding for the "Manager of.." rule, to establish to which particular container element you need to assign the workflow initiator.

   a) In the function group, create a function module with a suitable interface according to the course material.

      As a sample solution, see the example code in the function module *Z00_MM_MGR_OR_USER_GET*.

3. Now define the rule using the new function module. In this case, define the parameters for this rule according to the implementation of your function module.

   You define the rule parameters using the rule container.

Continued on next page

The workflow initiator can be defined with reference either to field WFSYST-INITIATOR or to RHOBJECTS-OBJECT.

a) Call transaction PFAC_INS for creating rules. (Path: *Development → Definition Tools → Rules for Agent Assignment → Create*)

Enter an ID, a name, and under the category *Function to be executed*, enter the name of your function module.

Enter the required values in the suitable container.

b) As a example solution, see the rule *Z00MATDISPAC*.

The example coding is included at the end of this solution.

4. Use the new rule for agent assignment of the Display step in your workflow **Z##MSIPROC2** or **Z##MSIPROC1**.

a) Call the Display step in either of your two workflows.

b) Enter your rule in the *Agents* section, and maintain the binding.

c) Compare the relevant step in workflow Z00MSIPROC2 as a sample solution.

5. Check, activate and test your workflow Z##MSIPROC2 or Z##MSIPROC1.

a) Generate a new runtime version of your workflow and test it using transaction SWUS.

b) In the following example compare the sample source code of the rule function module.

Example solution:

Rule AC30900129 – Z00MatDispAc

Function module Z00_MM_MGR_OR_USER_GET

```
FUNCTION Z00_MM_MGR_OR_USER_GET.
*"----------------------------------------------------------------------
*"*"Local interface:
*" TABLES
*" ACTOR_TAB STRUCTURE SWHACTOR
```

Continued on next page

Internal Use SAP Partner Only

```
*" AC_CONTAINER STRUCTURE SWCONT
*" EXCEPTIONS
*" NOBODY_FOUND
*"----------------------------------------------------------------------
DATA: MATERIAL_TYPE LIKE MARA-MTART. " type of changed material
DATA: WF_INITIATOR LIKE SWHACTOR. " workflow initiator
DATA: COUNTER TYPE I.

REFRESH ACTOR_TAB. CLEAR ACTOR_TAB. * get type of material under consideration

SWC_GET_ELEMENT AC_CONTAINER 'MaterialType' MATERIAL_TYPE.
SWC_GET_ELEMENT AC_CONTAINER 'WorkflowInitiator' WF_INITIATOR.

CASE MATERIAL_TYPE.
WHEN 'ROH'. " get manager of wf_initiator from role
SWC_SET_ELEMENT AC_CONTAINER 'ORG_OBJECT' WF_INITIATOR.
CALL FUNCTION 'SWX_GET_MANAGER'
TABLES
ACTOR_TAB = ACTOR_TAB
AC_CONTAINER = AC_CONTAINER
EXCEPTIONS
NOBODY_FOUND = 1.
IF SY-SUBRC NE 0.
RAISE NOBODY_FOUND.
ENDIF.
```

Continued on next page


```
WHEN OTHERS.
ACTOR_TAB = WF_INITIATOR.
APPEND ACTOR_TAB.
ENDCASE.

* check if we have found any agents at all
DESCRIBE TABLE ACTOR_TAB LINES COUNTER.
IF COUNTER = 0.
RAISE NOBODY_FOUND.
ENDIF.

ENDFUNCTION.
```


Internal Use SAP Partner Only

Lesson Summary

You should now be able to:

- Create a rule function module
- Use a rule function module in a workflow step

Unit Summary

You should now be able to:

- Create a rule function module
- Use a rule function module in a workflow step

Internal Use SAP Partner Only

Test Your Knowledge

1. Rule function modules have a defined interface that must be observed.
   Determine whether this statement is true or false.
   - □ True
   - □ False

2. Agents are returned using the table _____. The container that is transferred is called _____. Determined agents must be returned in a defined way. The first part specifies the object type, for example _____, _____, _____. The second part returns the _____ of the relevant object.
   Fill in the blanks to complete the sentence.

Answers

1. Rule function modules have a defined interface that must be observed.

   Answer: True

   In this interface, the rule container is transferred and the determined agents are returned.

2. Agents are returned using the table ACTOR_TAB. The container that is transferred is called AC_CONTAINER. Determined agents must be returned in a defined way. The first part specifies the object type, for example User, organizational unit, position. The second part returns the ID of the relevant object.

   Answer: ACTOR_TAB, AC_CONTAINER, User, organizational unit, position, ID

Unit 5

Event Definition and Implementation

Unit Overview

Following on from the discussion of the concepts underlying event processing and the event manager, this unit explains how an event is triggered in the application using a function module.

Events trigger actively linked workflows.

You can use check function modules to attach additional conditions to the triggering of workflows.

You can use receiver type function modules to select one workflow from n workflows that should actually be started, based on runtime data.

A receiver type function module is applicable if you do not want start a workflow after an event but, for example, you want to call a report instead.

Using the event queue, receivers of events are not started directly, but can be buffered in a queue and processed in separate portions.

This improves the performance of your system.

Unit Objectives

After completing this unit, you will be able to:

- Explain the basic concepts of using events
- Explain the processing logic of the event manager
- Find type linkages and instance linkages in the system
- Use the event log for documentation and testing of events
- Explain the enhancement of the event concept to process workflow instances.
- Describe the new options of the wait step in the workflow definition.
- Assess the use of “local events” and “correlations”.
- Trigger an event using a function module

- Describe the difference between triggering an event with the same name on a supertype, or on one of it's subtypes.
- Create a check function module
- Create a receiver type function module
- Describe the application of a receiver type function module
- Describe the functions of the event queue
- Activate the event queue
- Activate type linkages for the event queue

Unit Contents

Internal Use SAP Partner Only

Lesson: Event Processing Using the Event Manager

Lesson Overview

This lesson deals with the basic concepts of event processing in the Workflow Engine and how the Event Manager processes events.

The lesson explains type linkages and instance linkages, and how you can call all the events that have taken place in the event log.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the basic concepts of using events
- Explain the processing logic of the event manager
- Find type linkages and instance linkages in the system
- Use the event log for documentation and testing of events

Business Example

Your company wants to start a workflow using events.

You must understand the basic logic of how the Workflow Engine works with events.

Using Events and Basic Concepts

Purposes in SAP Applications

- Why events?
  - Redistribute complexity of communication
  - Increase flexibility
  - Easier to enhance and integrate
  - Increase transparency

The communication between (parts of) the applications is often complex and requires knowledge of several cross-components. To make the whole system more understandable and easier to maintain, it is worthwhile removing this complexity from the individual applications and making them transparent in a central component (event manager) for all participants.

At runtime, you can simply activate and deactivate event linkages and even create new linkages. Consequently, you can integrate additional components (processes, applications, systems, and so on) in existing processes without having to modify the existing applications.

Examples of applications for event communication include:

- Starting workflows from the application (flexible linkage from workflow)
- Terminating steps with updating (integrating asynchronous parts of processes)
- Waiting for concurrent actions (synchronization)

Event linkage: Basic Concepts

- Publish and subscribe
- ECA paradigm
- Static or dynamic linkage
- Sequential and transaction-dependent linkage mode
- Acceptances of delivery times not possible
- Event queue: Facilitates temporary storage and delayed delivery of events

Event linkages work according to the “publish and subscribe” principle whereby the creator publishes the event without knowing whether or how many receivers exist.

Receivers need to “subscribe” to the event manager (EM).

The EM evaluates linkages (object type + event name <-> receiver) according to the ECA paradigm. The **E**vent is triggered, the receivers are determined and checked for relevancy (**C**heck) and the relevant receivers are notified (**A**ction).

In accordance with the OO approach, the inheritance hierarchy (in direction) for the triggering object type is used to find relevant linkages for a triggered event.

User exits can dynamically determine the relevancy of a given linkage (using a check function module) as well as the receiver to be notified (using a function module for determining receiver types).

Default:

Triggered events are delivered immediately and are not saved temporarily. When event queues are used, events are stored temporarily and delivered by a periodic job. You can restrict the maximum number of events delivered per minute.

Events are delivered by remote function call (RFC). Therefore, an event only reaches its recipient(s) following a SAP commit (the ABAP keyword is COMMIT WORK); an implicit database commit is not sufficient. After a ROLLBACK WORK, this means that all triggered events in the same SAP LUW are rejected.

Transactional RFC Basic Design

- Dependent on explicit COMMIT WORK
- New login
- Error handling using own log

Events are (technically) delivered using a tRFC (transactional Remote Function Call). After you trigger events, you must execute the commit work statement explicitly to start the delivery. A “rollback work” causes all events of an SAP LUW that have not been delivered yet to be canceled (transactional property).

A new login is carried out for the event receivers for each (RFC) destination.

You can read the transactional RFC log in transaction SM58.

Events: Creator and Receiver Context

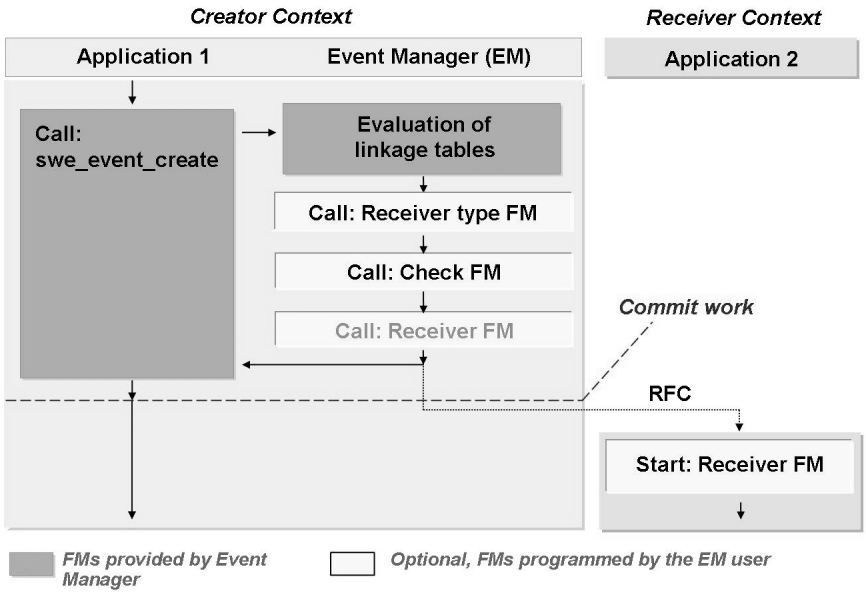


Figure 46: Events: Creator and Receiver Context

Internal Use SAP Partner Only

The application triggers an event. Consequently:

- The event manager (EM) determines all subscribed receivers (“event receivers”) by reading the linkage tables.
- For each linkage found:

  • The receiver can be determined dynamically using a receiver type function module (optional), so you do not have to enter the event receiver permanently in the linkage table.

  • A check can be carried out using a check function module (optional) to see whether a receiver is actually supposed to receive the current event.

  • Finally, a tRFC call requests the delivery of the event. However, delivery (RFC execution) is only made when the next COMMIT WORK is executed in the system.

The receiver type and check function modules are still called in the context of the event creator. This means that only one tRFC (critical system resource) is used in each case for notifying the receivers determined as relevant.

Hint: Since SAP NetWeaver Application Server 7.0 you can influence the delivery of events directly in the event linkage. The “Event delivery” parameter provides three options:

1. “The T-RFC (standard)” is the option used by the system by default (as described above)
2. “The LUW buffer (synchronous)”
3. “An event queue (always)” defines that delivery always occurs with the event queue

“The T-RFC (standard)” is the option used by the system by default (as described above).

Type Linkages and Instance Linkages

Events: Type Linkage

- Linkage at type level: Triggering object type and event receiver (type)
- Typical application: Event starts new workflow instance
- Maintenance of a type linkage explicitly using view maintenance or function modules
- Type linkage can be transported

If you enter and activate a triggering event in a task definition, the workflow system automatically creates the associated type linkage. When you define a start condition, a standard check function module is included in the linkage. If you are using a check function module or receiver type function module you implemented yourself, then you must always enter this manually (as part of the Customizing for the relevant workflow).

Type linkages are client-specific (like all event linkages). In contrast to instance linkages, they are implemented in the linkage table using entries with the property "global". The "active" indicator signifies that the linkage should be included when events are delivered. You can configure the behavior of a linkage if errors occur, for example, to deactivate the linkage (default setting), to mark linkage as having errors or as incorrect, or to not change linkage.

Caution: Be careful when using event-receiver linkages.

When the "Behavior upon error feedback" parameter is set (see properties of a receiver linkage in transaction SWETYPV), the processing of processes in your systems is affected.

By default, this entry is set to **System defaults**.

The property **Deactivation of linkage** must never be set in the parameters of the event queue or in this actual indicator.

If this is the case, "data loss" may occur, (that is, incomplete event-receiver linkages). At this point we would recommend you set the property **Mark linkage as having errors** or the property **Do not change linkage**.

The table entries are usually transported without the activity indicator, but you must then activate the linkages explicitly in the target system. You can transport the activity indicator using the transport connection for the generated view maintenance.

Relevant fields of a type linkage for programming:

- Name of a receiver type function module (dynamic recipient determination)
- Name of a check function module (relevance check for recipients)
- Name of a receiver function module (RFC module for event delivery)

Events: Instance Linkage

- Linkage between triggering object instance and receiving instance
- Typical application: Event terminates a workflow (step)
- Explicit display using workflow development environment or work item display
- Maintenance usually implicit using function modules
- No transport

The instance linkage is implemented in two phases using a combination of the type linkage table and the instance linkage table. An entry in the type linkage table with the property “global” = space is used as an “anchor” for one or several dependent entries in the instance linkage table. The latter table contains additional fields that can be used to identify individual instances both for the trigger (object key) and the recipient (work item ID).

The instance linkage table is only evaluated if the associated type linkage is “active” and not “global”. Maintenance is usually carried out only using function modules. The corresponding workflow-related entries (for example, for terminating events of a step) are entered and then deleted again automatically by the workflow runtime system. For test purposes, you can also enter or change data manually in the instance linkage table.

Check or receiver type function modules are only rarely used for terminating events. If a function module of this type is required, the type linkage must always be entered manually (as part of Customizing for the relevant workflow).

Entries in the instance linkage table are client-specific and are never transported.

Event Manager - Processing Logic

- Finds all active type linkages
- Evaluates dynamic receiver determination
- If global linkage (triggering event)
  - Calls check function module
  - Calls receiver function module
- Otherwise:
  - Finds all instance linkages
  - Calls check function module for each instance linkage
  - Calls receiver function module for each instance linkage

A dynamic receiver is determined if you enter a receiver type function module in the type linkage.

Internal Use SAP Partner Only

Documentation of Events in the Event Log

Events: Event Trace

- All triggered events are logged.
- Entries describe the event delivery. They do not provide information about the behavior of the receiver.
- Trace is activated, deactivated and displayed using the workflow development environment.
- The event trace function should be deactivated in the live system (for performance reasons).

Each event triggered results in an entry in the trace, even if a receiver is not found. A separate entry is made for each receiver. Error messages for the check function module and the receiver type function module are also noted here.

Internally, the trace is regarded as an additional receiver that is always triggered. It is independent of the commit behavior of other (explicit) receivers.

Writing the trace is time-consuming because a separate tRFC is triggered for this. Therefore, you should deactivate the trace in live systems (see note 46358).

You can restrict the writing of the trace to selected object types or events.

Internal Use SAP Partner Only

Exercise 7: Assess the Effect of the Type Linkage Table

Exercise Objectives

After completing this exercise, you will be able to:

- Simulate an event.
- Read which workflows react to which event.
- Use the event log.

Business Example

You have completed your workflow definition and assigned the individual dialog steps to the correct agents. You now want to check whether your workflow starts correctly using events.

Task:

Assess the effect of the entries in the type linkage table.

1. Ensure that your two workflows are correctly linked to the event **OldMaterialChanged** for your object type.
2. Simulate the event **OldMaterialChanged**. Check the output of the simulation tool.
3. Check the *event trace* for entries for the **Oldmaterialchanged** and **Basicmaterialchanged** events for your object type, which you triggered in an earlier task.

Solution 7: Assess the Effect of the Type Linkage Table

Task:

Assess the effect of the entries in the type linkage table.

1. Ensure that your two workflows are correctly linked to the event **OldMaterialChanged** for your object type.

   a) Navigate to the type linkage table and search for the entries for the event **OldMaterialChanged** for your object type.

   Basic menu path:

   Tools → Business Workflow → Development → Definition Tools → Events → Event Linkages → Type Linkages

   You should find two entries that refer to your object type ZMARA##A and to both of your workflows.

2. Simulate the event **OldMaterialChanged**. Check the output of the simulation tool.

   a) Call the *Simulate event* function, which you can use to test the linkage entry.

   Basic menu path:

   Tools → Business Workflow → Development → Utilities → Events → Simulate event

   Enter your object type **`ZMARA##A`** and the event **`OldMaterialChanged`**.

   Choose Execute.

   The system determines and checks the active linkages.

   Success messages as well as error messages are displayed on the screen.

Continued on next page

3. Check the *event trace* for entries for the **Oldmaterialchanged** and **Basicmaterialchanged** events for your object type, which you triggered in an earlier task.

   a) Call the *event trace* and search for the events that have taken place for your object type.

      Path (using SAP Easy Access):

      Tools → Business Workflow → Development → Utilities → Events → Event Trace → Display Event Trace

      Enter your object type ZMARA##A and the system lists the entries found.

Internal Use SAP Partner Only

Lesson Summary

You should now be able to:

- Explain the basic concepts of using events
- Explain the processing logic of the event manager
- Find type linkages and instance linkages in the system
- Use the event log for documentation and testing of events

Lesson: Enhancing the Event Concept in the Workflow Builder

Lesson Overview

This lesson deals with new developments in the event concept of the Workflow Builder. It deals in particular with functions that were created during the development of cross-component business process management (ccBPM) (SAP NetWeaver PI) for SAP NetWeaver Application Server 6.40.

You will learn about a new concept to distribute events to the receiver in a workflow instance and will see the enhancements to the wait step for this purpose in the workflow definition.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the enhancement of the event concept to process workflow instances.
- Describe the new options of the wait step in the workflow definition.
- Assess the use of “local events” and “correlations”.

Business Example

Your company wants to synchronize a workflow with other processes, using events. You must understand the basic logic of events in the Workflow Builder and you must evaluate the new options, which result from Basis Release 6.40.

Internal Use SAP Partner Only

Parked Events

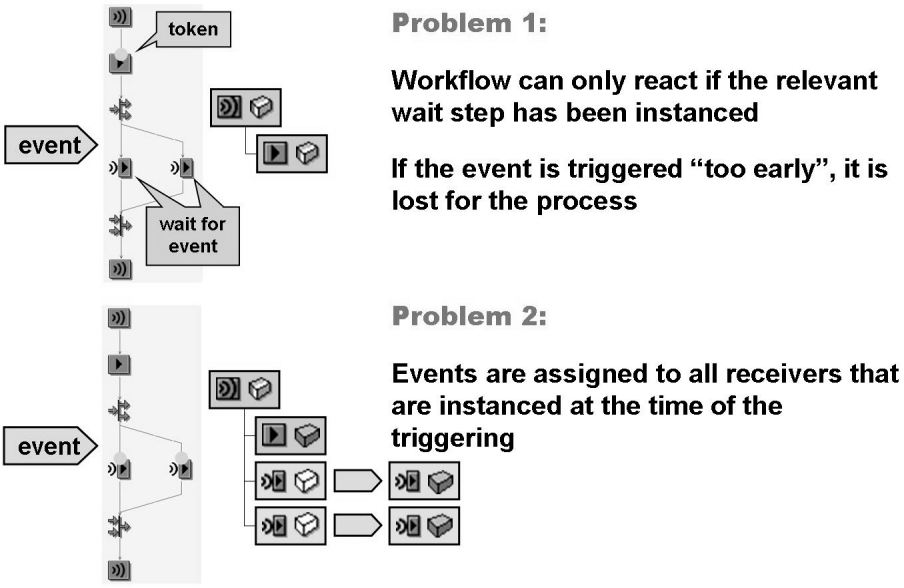


Figure 47: Event-receiver Problems

You can use two step types to synchronize processes: “Wait for Event” and “Event creator” in the Workflow definition

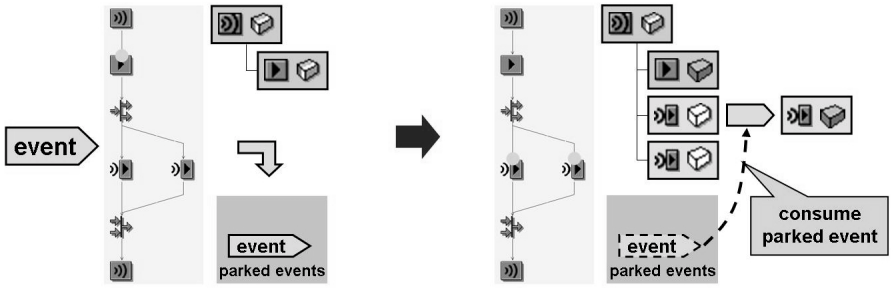
The principle behind this (the standard system response) is that an event can only be assigned to a receiver if this receiver has already been instantiated.

In the past this led to two basic problems with the processing of processes:

1. The event triggered does not find any receiver, because, for example, the workflow instance has not yet reached the point in the workflow definition at which the relevant wait steps are instantiated. This means the event is lost for the process, that is, it disappears and can no longer be used.
2. Several instances are formed by wait steps in a process. In this context all instances are receivers for the same event. If one event is published in the system, all wait steps are executed by the event-receiver linkage.

Both cases can lead to problems with the synchronization of processes in the traditional process design, in a back-end system.

In the BPM functions of SAP NetWeaver PI (Exchange Infrastructure), which are created in the Workflow Engine, this procedure is a bigger problem. This means the message-based processes can no longer be handled without errors.



Process instance can be used as event temporary storage

Figure 48: Solution: Event-receiver Problems

To solve the problem described above, a temporary storage had to be created for the events assigned to the process. As of Basis Release 6.40 the workflow instance itself takes over this task. In addition, the necessary functions were added to the wait step mechanism, as described in a later section. The following rules apply:

- If a wait step exists, the instance first created (this wait step) is linked with the event.
- If no active wait step is instantiated, the event is moved to the temporary storage.
- The first wait step, which is instantiated, uses the event that is saved.
- In each case an event is always used by exactly one wait step.

The following diagram shows how to create the wait step, in conjunction with the technology described above.

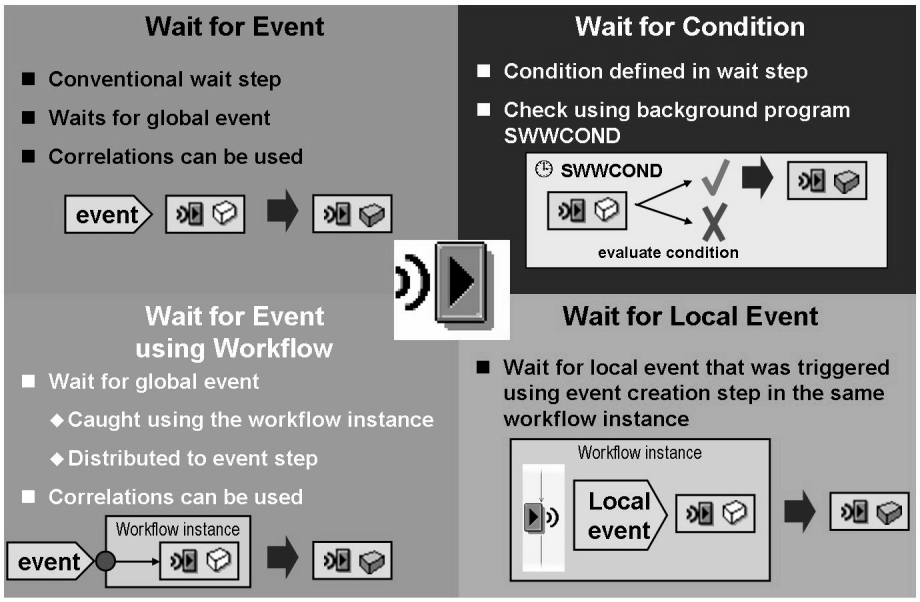


Figure 49: The Enhanced Wait Step

1. Wait for Event

   In the “Wait for Event” step, the wait step terminates if the specified event is reached. The event terminates all wait steps, which wait for this event. The important thing here is that wait step has already been instantiated.

   You can use a wait step like this, for example, for the following purposes:

   - To stop the workflow until the defined event is reached
   - To wait for an event in a parallel processing branch, which makes processing in other branches unnecessary
   - To wait for the result or a message of another workflow, which was started in another system by the WebFlow function
   - To wait for a message of the process, which the workflow started using an Wf-XML message

2. Wait for Condition

   The wait step terminates when you wait for a condition, if the relevant condition is true.

3. Wait for Event using Workflow

   When you wait for an event in the workflow, the event is first caught by the workflow and is saved temporarily. As soon as the wait step has been activated, the event is transferred to the wait step. When you wait using the workflow, an event can terminate a maximum of one wait step. If several wait steps are active, the event terminates the oldest wait step.

4. Wait for Local Event

   When you wait for a local event, the wait step terminates if the local event is reached.

Local Events and Correlations

Conventional event linkages:

- Wait steps are instanced for existing or known object instances
- The object key is used to connect the event with the wait step

Requirement:

- Wait for events of unknown objects
  - Object key is unknown when the wait step is created
- Object key cannot be used to connect the event with the wait step

Solution:

- Use a semantic connection between object instances (correlations)
  - An object instance leads to several object instances that correlate
- Wait step not defined with object key, but with object type and correlation instance instead

Figure 50: Correlations

A correlation is a function that was also created during developments for the ccBPM. The extent to which this function is of use for workflows in the back-end must be considered based on the relevant process.

The specific purpose of this function is to connect messages with the same properties logically and technically in the SAP NetWeaver PI, although the individual instances (messages) are not known by their direct key.

You can use a correlation to identify connected objects, for example, a quotation and the relevant order. The objects are correlated using one or several common elements, for example, using the quotation number. During the definition of a workflow you can determine which object the workflow is waiting for, without having to know the ID of the object.

Actions

You define a correlation with the correlation editor. To ensure the correlation works in the workflow, you must activate it (instantiate). To do this, use the process control.

A subsequent wait step can then use the correlation to determine the object that the wait step is waiting for. You can use a correlation when waiting for an event and when waiting for an event in workflow.

The procedure to create a correlation is described in detail in the SAP Online Help.

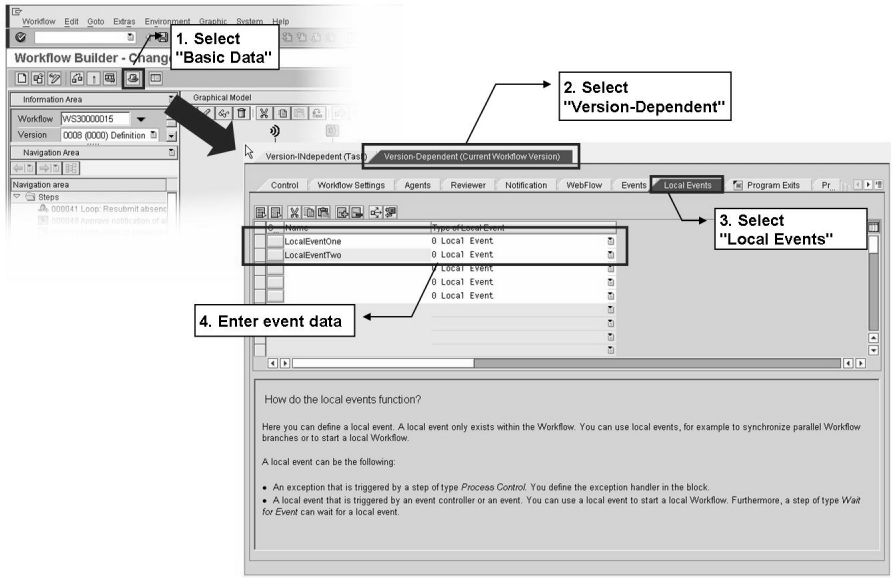


Figure 51: Local Events Creation

As of Basis Release 6.40 you can use the “Local Events” tab in the basic data of your workflow template to define local events also.

Caution: This type of event exists in the workflow only.

You can use local events as a type "signal" to control the workflow, for example, to synchronize parallel workflow branches or to start a local workflow. A local event can also represent:

- An exception, which can be triggered by a step of the process control type You can define the exception handler in the block.
- A local event, which can be triggered by a step of the event creator type or by an event.
- A trigger for a wait step

Lesson Summary

You should now be able to:

- Explain the enhancement of the event concept to process workflow instances.
- Describe the new options of the wait step in the workflow definition.
- Assess the use of “local events” and “correlations”.

Related Information

- Use a URL or a cross-reference tag to point out additional information that the participants may find useful, such as Web sites or White Papers. Delete this if it is not relevant.

Lesson: Using Function Modules to Trigger Events

Lesson Overview

This lesson demonstrates how events can be published in the system by calling a function module.

It includes a programming example for triggering events, and highlights the differences between triggering events with the same name on the supertype or triggering it on one of it's subtypes.

Lesson Objectives

After completing this lesson, you will be able to:

- Trigger an event using a function module
- Describe the difference between triggering an event with the same name on a supertype, or on one of it's subtypes.

Business Example

In the standard system, the application does not trigger the event that starts your workflow.

You are also unable to trigger the event using status management, message control, or by writing change documents.

You now need to explicitly use a function module.

Triggering Events Using Function Modules

Events: Definition/Trigger

- Define events in the Business Object Builder
- Trigger event
  - implicitly using generic tools, for example:

    Change documents

    Status management

    Message control
  - Explicitly using a function module

    SWE_EVENT_CREATE or

    SWE_EVENT_CREATE_FOR_UPD_TASK

Events are defined for the relevant object type in the Business Object Builder. The system is thus notified of the name and the parameters of the event.

Triggering an event using change documents does not require any ABAP source code. Instead, Customizing activities for the triggering application are required. Event parameters cannot be transferred.

Triggering an event using a function module requires the source text of the triggering application to be changed or enhanced. For customers, the call is usually programmed in user exits made available by the application. For user-programmed calls, event parameters can be transferred.

Caution: The function module SWE_EVENT_CREATE can trigger exceptions. If these exceptions are not caught by the triggering application, a runtime error occurs.

The triggering modules themselves do not execute a COMMIT WORK. The onus is on the application or caller to execute a COMMIT WORK at the right time.

SWE_EVENT_CREATE - Triggers event immediately: This can be used only if object data was not changed on the database or if all changes have already been updated.

SWE_EVENT_CREATE_FOR_UPD_TASK – calls SWE_EVENT_CREATE as the update module. This is required if the object that triggers the event is changed, or is only created, in the update task.

SWE_EVENT_CREATE_IN_UPD_TASK – update-capable version of SWE_EVENT_CREATE. This version of SWE_EVENT_CREATE is required if you have to control the update call more accurately than in SWE_EVENT_CREATE_FOR_UPD_TASK.

Programming Example for Triggering an Event

- Event container is filled with event parameters
- Object key is created for triggering object
- swe_event_create is called
- Troubleshooting for FM call
- Event is triggered by explicit commit work

```
include <CNTN01>.

data: begin of asset_key,
           company_code like anla-bukrs,
           asset_no     like anla-anln1,
           sub_number   like anla-anln2,
      end of asset_key.
data: object_key        like sweinstcou-objkey.

swc_container evt_container.

* Write parameters into event container
swc_set_element evt_container 'flag_equi_aendern' 'X'.

* Compose object key
asset_key-company_code = '0001'.
asset_key-asset_no     = '000000123456'.
asset_key-sub_number   = '0100'.
object_key = asset_key.

* Trigger the event
call function 'SWE_EVENT_CREATE'
     exporting
        objtype = 'BUS1022'
        objkey  = object_key
        event   = 'changed'
     tables
        event_container = evt_container
     exceptions
        others  = 01.

if sy-subrc ne 0.
  " do your own error handling
endif.

* start tRFC processing
commit work.
```

Figure 52: Triggering an Event Explicitly

Caution: If the event is triggered in a user exit, a COMMIT WORK should not be programmed here, because this would destroy the COMMIT logic of the surrounding application.

Use the correct version of the triggering module:

If the application writes object data in the update task, you must use SWE_EVENT_CREATE_FOR_UPD_TASK to ensure that the event is also triggered first in the update task (and after the object data has been written).

If parameters were defined for the event in the Business Object Repository, you must set these parameters explicitly in the event container before you call the function module with SWC_SET_ELEMENT.

If required, you can display how the standard parameters of the event container are filled as follows:

1. Enter a new linkage in the type linkage table: Object type: <required object type>event: <required event>receiver type: <user, who should receive a mail>receiver function module: SWE_EVENT_MAIL
2. Set the flag for “Linkage active”
3. As soon as the event is triggered, all actively linked workflows start. In addition, the user mentioned above receives a mail in the workplace in which the values of the event container are listed.

Triggering Events on Supertypes and Subtypes

Object type	Event	Receiver type
BUS1001	Created	WS00005711
Z_BUS1001	Created	WS97000313

```
include <CNTN01>.

data: object_key         like sweinstcou-objkey.
      swc_container            evt_container.

object_key = '000012345678'.

* Trigger the event
call function 'SWE_EVENT_CREATE'
     exporting
        objtype = 'BUS1001'
        objkey  = object_key
        event   = 'created'
     tables
        event_container = evt_container
     exceptions
        others  = 01.

* start tRFC processing
commit work.
```

Figure 53: Inheritance: Triggering - Supertype

When you trigger an event, the linkages for derived object types are **not** taken into account.

Object type	Event	Receiver type
BUS1001	Created	WS00005711
Z_BUS1001	Created	WS97000313

```
include <CNTN01>.

data: object_key         like sweinstcou-objkey.
      swc_container           evt_container.

object_key = '000012345678'.

* Trigger the event
call function 'SWE_EVENT_CREATE'
     exporting
        objtype = 'Z_BUS1001'
        objkey  = object_key
        event   = 'created'
     tables
        event_container = evt_container
     exceptions
        others  = 01.

* start tRFC processing
commit work.
```

Figure 54: Inheritance: Triggering - Subtype

If a subtype is entered, linkages for the object type specified at the time the trigger is executed and all its supertypes are included.

You can use a check function module to prevent this type of behavior.

The object type belonging to the event, for example, BUS1001 (that is, the supertype), is transferred to the check function module.

However, an element _EVT_OBJTYPE containing the object type that actually triggered the event (in our example, Z_BUS1001), is transferred to the check function module at the same time. If both function modules are not the same, the check function module triggers an exception.

Internal Use SAP Partner Only

Exercise 8: Use the Function Module SWE_EVENT_CREATE to Trigger the Event DeadlineTest (ZMARA##A)

Exercise Objectives

After completing this exercise, you will be able to:

- Trigger an event using a function module
- Locate a customer-specific parameter in the event container that is defined in the event

Business Example

The event linkage on your workflow definitions is created correctly. You now want to start workflows using an event that is triggered explicitly in an ABAP program you have written.

Task:

Trigger an event explicitly.

1. Create a report **Z##EVTCR** that calls the function module SWE_EVENT_CREATE to trigger the event **DeadlineTest** for your object type.

   Use a selection parameter to query the object type, material number, event and end date of the parameter.

   Ensure that you have defined the event **DeadlineTest** with a parameter in the Business Object Repository.

   Check the name of the parameter.

   When you call the function module to trigger the event, fill the parameter with the system date, provided the user has not specified a date in the end date selection parameter.

Solution 8: Use the Function Module SWE_EVENT_CREATE to Trigger the Event DeadlineTest (ZMARA##A)

Task:

Trigger an event explicitly.

1. Create a report **Z##EVTCR** that calls the function module SWE_EVENT_CREATE to trigger the event **DeadlineTest** for your object type.

   Use a selection parameter to query the object type, material number, event and end date of the parameter.

   Ensure that you have defined the event **DeadlineTest** with a parameter in the Business Object Repository.

   Check the name of the parameter.

   When you call the function module to trigger the event, fill the parameter with the system date, provided the user has not specified a date in the end date selection parameter.

   a) Compare your solution with the example coding of the following report Z00EVTCR.

      To display the report,

      choose the following path in the basic menu:*Tools > ABAP Workbench > Development > ABAP Editor*

```
*&---------------------------------------------------------------------
*& Explicit triggering of Z00MARA.DEADLINETEST
*&---------------------------------------------------------------------
REPORT Z00EVTCR.

INCLUDE <CNTN01>. " include container macros

DATA: OBJKEY LIKE SWEINSTCOU-OBJKEY,
EVENTID LIKE SWEDUMEVID-EVTID.
```

Continued on next page


```
DATA: EVENT_CONTAINER LIKE SWCONT OCCURS 0 WITH
HEADER LINE.

PARAMETERS:
OBJTYPE LIKE SWETYPECOU-OBJTYPE DEFAULT 'Z00MARA',
MATERIAL LIKE MARA-MATNR,
EVENT LIKE SWETYPECOU-EVENT DEFAULT 'DEADLINETEST',
END_DATE LIKE SYST-DATUM DEFAULT SPACE.

OBJKEY = MATERIAL.

* write parameter into event container
IF EVENT EQ 'DEADLINETEST' AND END_DATE EQ SPACE.
END_DATE = SY-DATUM + 1.
ENDIF.

SWC_SET_ELEMENT EVENT_CONTAINER 'LatestEndDate'
END_DATE.

CALL FUNCTION 'SWE_EVENT_CREATE'
EXPORTING
OBJTYPE = OBJTYPE
OBJKEY = OBJKEY
EVENT = EVENT
IMPORTING
EVENT_ID = EVENTID
TABLES
EVENT_CONTAINER = EVENT_CONTAINER
EXCEPTIONS
```

Continued on next page


```
OBJTYPE_NOT_FOUND = 1.

IF SY-SUBRC NE 0.
WRITE: / 'Object type', OBJTYPE, 'unknown'.
ELSE.
IF EVENTID NE 0.
WRITE: / 'At least one receiver found'.
COMMIT WORK.
ELSE.
WRITE: / 'No receivers found'.
ENDIF.
ENDIF.
```

Lesson Summary

You should now be able to:

- Trigger an event using a function module
- Describe the difference between triggering an event with the same name on a supertype, or on one of it's subtypes.

Internal Use SAP Partner Only

Lesson: Check Function Modules, Receiver Type, and Receiver Function Modules

Lesson Overview

Events start all actively linked workflows.

This lesson shows how you can use check function modules to evaluate additional conditions for starting an event-driven workflow.

It also explains how to use a receiver type function module to start one workflow that is appropriate for the data of the triggered object, from a range of other possible workflows.

Lesson Objectives

After completing this lesson, you will be able to:

- Create a check function module
- Create a receiver type function module
- Describe the application of a receiver type function module

Business Example

You want a workflow to start as soon as the event “Material changed” is triggered in the system. However, the workflow should only start if the material is from the chemical industry.

You also want the workflow to continue processing as soon as a new material is created in the system. You want a different workflow to start depending on which views have already been created for the material. To do this, you use check function modules and receiver type function modules.

Check Function Modules - General Information

Check Function Module

- Test: Is event relevant for the receiver?
- Process still in context of trigger
- Interface

```
function check_fm
importing   objtype   like swetypecou-objtype
            objkey   like  sweinstcou-objkey
            event     like swetypecou-event
            rectype   like swetypecou-rectype
tables      event_container  structure swcont
exceptions  others
```

Check function module:

The check function module is still called in the context of the trigger before an event is delivered. This optional function module checks whether the event is actually relevant for the determined receiver and whether it should be delivered.

The check function module cannot intervene in the COMMIT logic (no COMMIT, no ROLLBACK).

Exceptions (MESSAGE... RAISING...) are recorded in the event trace.

The triggering of events is propagated upward in line with the inheritance hierarchy. With propagated events, the _EVT_OBJTYPE default parameter (in the event container) contains the triggering object type (the subtype), whereas the OBJTYPE interface parameter contains the propagated object type (the supertype).

_EVT_OBJECT **always** contains a reference to the triggering object.

Programming Example of a Check Function Module

- **Event parameters are read from the event container.**
- **Conditions are checked**
- **Exception is triggered if the event should not be delivered to this receiver**

```
function check_object_type_idocappl.
*"----------------------------------------------------
*" importing objtype   like       swetypecou-objtype
*"           objkey   like       sweinstcou-objkey
*"           event    like       swetypecou-event
*"           rectype  like       swetypecou-rectype
*" tables    event_container structure      swcont
*" exceptions check_failed
*"----------------------------------------------------

include <CNTN01>.

constants:
  c_otype_appl like swetypecou-objtype value 'IDOCAPPL',
  c_evt_otype like swcont-element value '_EVT_OBJTYPE'.

data: initiating_otype like swetypecou-objtype.

* Implementation of check fm

* Read originating objtype from event container
swc_get_element event_container
                c_evt_otype initiating_otype .

* check if initiating objtype is desired one
if initiating_otype <> c_otype_appl.
  raise check_failed.
endif.

endfunction.
```

Figure 55: Implementing a Check Function Module

Other typical contents of a check function module:

- Default parameter query (for example, date, time, object type)
- Object attribute query using the SWC_GET_PROPERTY macro
- Additional values written in the event container

If the event is triggered before the object data is written to the database, the object attributes are not available yet for querying because the database is accessed when the attributes are evaluated by the Business Object Builder.

Receiver Type Function Modules - General Information

Receiver type function module

- Dynamic determination of receiver type depending on event data
- Process still in context of trigger
- Interface

```
function rectypeget_fm
importing   objtype  like  swetypecou-objtype
            objkey   like  sweinstcou-objkey
            event    like  swetypecou-event
            generic_rectype like  swetypecou-rectype
exporting   rectype  like  swetypecou-rectype
tables      event_container   structure swcont
exceptions others
```

Receiver type function module:

The event manager calls the receiver type function module in the triggering context to determine the recipient of the event (“receiver type”) dynamically. Benefits: A benefit of using this function module is that several receivers to be activated exclusively can “share” a linkage. For example: Only one of the three recipients should ever receive the event.

When a receiver type function module is used, the type linkage table can also have an entry in the “rectype” field. This entry is transferred to the function module in the “generic_rectype” input parameter and can be used, for example, as the default value for the recipient to be determined.

The check function module cannot intervene in the COMMIT logic (no COMMIT, no ROLLBACK).

Exceptions (MESSAGE... RAISING...) are recorded in the event trace.

The triggering of events is propagated upward in line with the inheritance hierarchy. With propagated events, the _EVT_OBJTYPE default parameter (in the event container) contains the triggering object type (the subtype), whereas the OBJTYPE interface parameter contains the propagated object type (the supertype).

_EVT_OBJECT **always** contains a reference to the triggering object.

Programming Example of a Receiver Type Function Module

- **Event parameters are read from the event container**
- **Receiver type determined based on application-specific settings**
- **Exception is triggered if no suitable receiver can be found**

```
function cvwf_event_rec_dokst_get.
*"----------------------------------------------------------------------
*"*"Local interface:
*" importing objtype              like swetypecou-objtype
*"           objkey               like sweinstcou-objkey
*"           event                like swetypecou-event
*"           generic_rectype      like swetypecou-rectype
*" exporting  rectype              like swetypecou-rectype
*" tables     event_container      structure  swcont
*" exceptions no_rectype
*"----------------------------------------------------------------------
include <CNTN01>.

data: l_dokst like draw-dokst.
data: begin of drawkey,
          dokar like draw-dokar,
          doknr like draw-doknr,
          dokvr like draw-dokvr,
          doktl like draw-doktl,
      end of drawkey.

* Implementation of receiver type fm
* Read document status from event container
swc_get_element event_container
                'DOCUMENTSTATUS' l_dokst.
drawkey = objkey.

* read receiver type from table tdws
select single * from tdws
       where dokar = drawkey-dokar and
             dokst = l_dokst.
If sy-subscr ne 0.
  Raise no_rectype.
Else.
  Rectype = tdws-otype.
  Rectype+2 = tdws-objid.
Endif.

endfunction.
```

Figure 56: Implementing a Receiver Type Function Module

The event receiver can be determined from the following, for example: the event parameters, the default value (“generic_rectype”), a database table created for this purpose, the application server memory or a hard-coded list in the module.

If the event is triggered before the object data is written to the database, the object attributes are not available yet for querying because the database is accessed when the attributes are evaluated by the Business Object Builder.

For a receiver type function module, you must maintain the linkage table in a particular way:

Manually create an entry that does not normally have an entry in the Receiver type field. When you make an entry in this field, this workflow is transferred as a *generic_rectype*.

The *Receiver type* field usually specifies the workflow/task to be started by the event.

Enter the function module you created in the *Receiver type function module* field. This function module determines which workflow should start.

Set the flag for *Linkage active*.

You do not need to enter an event linkage in the corresponding workflow because the receiver type function module explicitly determines which workflow is started.

Receiver Function Modules

Receiver function module

- Process in own context (call using RFC)
- Interface

```
function receiver_fm

importing  objtype like swetypecou-objtype
           objkey  like sweinstcou-objkey
           event  like swetypecou-event
           rectype like swetypecou-rectype
tables     event_container  structure  swcont
```

Receiver function module: is called by RFC to trigger the event receiver.

If possible, the function module should not trigger any exceptions because these exceptions are recorded directly in the tRFC log only. If exceptions occur, workflow receiver function modules send an error message to the workflow system administrator.

An example of a function module is SWW_WI_START_VIA_EVENT.

The triggering of events is propagated upward in line with the inheritance hierarchy. With propagated events, the _EVT_OBJTYPE default parameter (in the event container) contains the triggering object type (the subtype), whereas the OBJTYPE interface parameter contains the propagated object type (the supertype).

_EVT_OBJECT **always** contains a reference to the triggering object.

Internal Use SAP Partner Only

Exercise 9: Create a Check Function Module to Query Special Conditions

Exercise Objectives

After completing this exercise, you will be able to:

- Create a check function module
- Enter the check function module in the type linkage and test it
- Create a receiver type function module if required
- Enter a receiver type function module in the type linkage, and test it

Business Example

You want your workflow to start only if the material was last changed by a particular user and if the material in question belongs to a specific group of materials.

Create a check function module to ensure that your workflow only starts after the event **Oldmaterialchanged** if specific conditions are fulfilled.

Use a check function module.

As an optional enhancement to your scenario, you wan to start one of two existing workflows depending on the material type. Use receiver type function modules to realize this.

Task:

Create a check function module to filter out irrelevant events.

1. Create a check function module.

   Z##_CHECK_MATERIAL_CHANGEDBY.

   The function module should ensure that your workflow is only started if you trigger the event for one of "your" materials (that is, T-BBC##, T-FBC##), and if the material was last changed by the user who represents your manager user.

   In the function module, check whether the transferred material number represents one of your materials and whether the **ChangedBy** object attribute has your manager user as a value.

   If this is not the case, an exception should be triggered.

Continued on next page

2. Insert the check function module into the existing linkages from the OldMaterialChanged event of your object type on your two workflow definitions, Z##MSIPROC1 and Z##MSIPROC2.
3. Now check whether your workflow is only triggered if this concerns one of your materials and if the last person to change the material was your manager user.

Internal Use SAP Partner Only

Solution 9: Create a Check Function Module to Query Special Conditions

Task:

Create a check function module to filter out irrelevant events.

1. Create a check function module.

   Z##_CHECK_MATERIAL_CHANGEDBY.

   The function module should ensure that your workflow is only started if you trigger the event for one of "your" materials (that is, T-BBC##, T-FBC##), and if the material was last changed by the user who represents your manager user.

   In the function module, check whether the transferred material number represents one of your materials and whether the **ChangedBy** object attribute has your manager user as a value.

   If this is not the case, an exception should be triggered.

   a) Compare your solution with the coding of the following example function module: **Z00_CHECK_MATERIAL_CHANGEDBY** , and find the suitable example linkage for the object type Z00MARA.

      Choose the following path (use SAP Easy Access) to create the function module:

      Tools → ABAP Workbench → Development → Function Builder

   b) Create the interface in the same way as the example in the folder.

   c) Example coding for the source text is included at the end of the solution.

2. Insert the check function module into the existing linkages from the OldMaterialChanged event of your object type on your two workflow definitions, Z##MSIPROC1 and Z##MSIPROC2.

   a) Use the following path to maintain the type linkage (starting from SAP Easy Access):

      Tools → Business Workflow → Development → Definition Tools → Events → Event Linkages → Type Linkages

3. Now check whether your workflow is only triggered if this concerns one of your materials and if the last person to change the material was your manager user.

Continued on next page

a) Trigger the workflow for which you have created the linkage, using the relevant test materials and with different users.

b) Example coding of the check function module

```
FUNCTION Z00_CHECK_MATERIAL_CHANGEDBY.
*"----------------------------------------------------------------------
*"*"Local interface:
*" IMPORTING
*" VALUE(OBJTYPE) LIKE SWETYPECOU-OBJTYPE
*" VALUE(OBJKEY) LIKE SWEINSTCOU-OBJKEY
*" VALUE(EVENT) LIKE SWETYPECOU-EVENT
*" VALUE(RECTYPE) LIKE SWETYPECOU-RECTYPE
*" TABLES
*" EVENT_CONTAINER STRUCTURE SWCONT
*" EXCEPTIONS
*" OBJECT_NOT_FOUND
*" INVALID_USER
*" INVALID_MATERIAL
*"----------------------------------------------------------------------

* This check fm checks, if the user who triggered the event (USER) is
* the same person who is related to the sample material and if
* the material changed is really one of my sample materials.

CONSTANTS:
MY_SAMPLE_MATERIAL_ROH LIKE MARA-MATNR VALUE
'200-110',
MY_SAMPLE_MATERIAL_FERT LIKE MARA-MATNR VALUE
'200-101',
```

Continued on next page

```
MY_SAMPLE_MATERIAL_HALB LIKE MARA-MATNR VALUE
'200-100',
MY_OWN_USER LIKE SY-UNAME VALUE 'WF-BC-CLERK'.

DATA:
MY_SAMPLE_MATERIALS LIKE MARA-MATNR
OCCURS 3 WITH HEADER LINE.
DATA: OBJECT TYPE SWC_OBJECT.
DATA: CHANGER LIKE MARA-AENAM,
USER_ACT LIKE SWHACTOR,
USER LIKE SY-UNAME.

* build int table with my sample materials
MOVE MY_SAMPLE_MATERIAL_ROH TO MY_SAMPLE_MATE-
RIALS.
APPEND MY_SAMPLE_MATERIALS.
MOVE MY_SAMPLE_MATERIAL_HALB TO MY_SAMPLE_MA-
TERIALS.
APPEND MY_SAMPLE_MATERIALS.
MOVE MY_SAMPLE_MATERIAL_FERT TO MY_SAMPLE_MA-
TERIALS.
APPEND MY_SAMPLE_MATERIALS.

* check, if the triggering material is one of my materials
READ TABLE MY_SAMPLE_MATERIALS WITH KEY = OBJKEY.
IF SY-SUBRC NE 0.
RAISE INVALID_MATERIAL.
ENDIF.
```

Continued on next page


```
* create the material object for later attribute access
SWC_CREATE_OBJECT OBJECT OBJTYPE OBJKEY.
IF SY-SUBRC NE 0.
RAISE OBJECT_NOT_FOUND.
ENDIF.

* read attribute 'ChangedBy' from material object
SWC_GET_PROPERTY OBJECT 'ChangedBy' CHANGER.
IF SY-SUBRC NE 0.
RAISE OBJECT_NOT_FOUND.
ENDIF.

* check if my sample user and material changer are identical
IF CHANGER NE MY_OWN_USER.
RAISE INVALID_USER.
ENDIF.

ENDFUNCTION.
```

Exercise 10: (Optional) Using a Receiver Type Function Module

Exercise Objectives

After completing this exercise, you will be able to:

- Create a receiver type function module
- Maintain the type linkage in order to use a receiver type function module

Business Example

You want to start different workflows depending on the material type of your changed material master.

Using a receiver type function module, you can choose the most suitable from a range of workflows.

Task:

Optional enhancement

Create a receiver type function module to decide which of your two workflows is to be started, depending on runtime data .

1. Create a receiver type function module

   Z##_RECTYPE_DETERMINE_MATTYPE.

   This function module determines the workflow to be started based on the material type of the triggering object. If the **MaterialType** object attribute has the value **FERT**, the **Z##MSIPROC1** workflow should be started, otherwise start the **Z##MSIPROC2** workflow.

2. Change the type linkage table again so that the workflow to be started is now determined via the receiver type function module for the event OldMaterialChanged of your object type.

3. Now check whether the workflow related to the material type of the triggering material is started.

Solution 10: (Optional) Using a Receiver Type Function Module

Task:

Optional enhancement

Create a receiver type function module to decide which of your two workflows is to be started, depending on runtime data .

1. Create a receiver type function module

   Z##_RECTYPE_DETERMINE_MATTYPE.

   This function module determines the workflow to be started based on the material type of the triggering object. If the **MaterialType** object attribute has the value **FERT**, the **Z##MSIPROC1** workflow should be started, otherwise start the **Z##MSIPROC2** workflow.

   a) To create the function module, choose the following path:

   Tools → ABAP Workbench → Development → Function Builder

   b) Create the function module using the same interface as the example in the documentation.

   c) Example coding for the function module is included at the end of the solution.

2. Change the type linkage table again so that the workflow to be started is now determined via the receiver type function module for the event OldMaterialChanged of your object type.

   a) To maintain the type linkage, choose the following path:

   Tools → Business Workflow → Development → Definition Tools → Events → Event Linkages → Type Linkages

   b) For an example solution, see the example type linkage for the object type Z00MARA, in which there is no *receiver* specified.

3. Now check whether the workflow related to the material type of the triggering material is started.

   a) Test your solution using different materials.

   b) The following is example coding for the receiver type function module.

   FUNCTION Z00_RECTYPE_DETERMINE_MATTYPE.

Continued on next page


```
*"----------------------------------------------------------------------
*"*"Local interface:
*" IMPORTING
*" VALUE(OBJTYPE) LIKE SWETYPECOU-OBJTYPE
*" VALUE(OBJKEY) LIKE SWEINSTCOU-OBJK
*" VALUE(EVENT) LIKE SWETYPECOU-EVENT
*" VALUE(GENERIC_RECTYPE) LIKE SWETYPECOU-RECTYPE
*" EXPORTING
*" VALUE(RECTYPE) LIKE SWEINSTCOU-RECTYPE
*" TABLES
*" EVENT_CONTAINER STRUCTURE SWCONT
*" EXCEPTIONS
*"  OBJECT_NOT_FOUND
*"----------------------------------------------------------------------

* This receiver type fm determines the process to be started by the material
type

DATA: OBJECT TYPE SWC_OBJECT.
DATA: MTYPE LIKE MARA-MTART.

* create the material object
SWC_CREATE_OBJECT OBJECT OBJTYPE OBJKEY.
IF SY-SUBRC NE 0.
RAISE OBJECT_NOT_FOUND.
ENDIF.

* check material typ and set receiver type depending on value
SWC_GET_PROPERTY OBJECT 'MaterialType' MTYPE.
```

Continued on next page


```
IF MTYPE EQ 'FERT'.
RECTYPE = 'WS96000011'.  " Z00MSIPROC1 ELSE.
RECTYPE = 'WS96000012'.  " Z00MSIPROC2
ENDIF.

ENDFUNCTION.
```

Lesson Summary

You should now be able to:

- Create a check function module
- Create a receiver type function module
- Describe the application of a receiver type function module

Internal Use SAP Partner Only

Lesson: Working With the Event Queue

Lesson Overview

In the standard system, workflows that are triggered by events are called immediately if the event is published in the system.

You can use the event queue to buffer when workflows are started, and process them in blocks. This lesson provides information required to do this.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the functions of the event queue
- Activate the event queue
- Activate type linkages for the event queue

Business Example

In your company, the implementation of batch input files during the night triggers events that start a high number of workflows at the same time.

To avoid a bottleneck through the login change of the tRFC, you want to implement the event queue and call the receiver workflows in a buffered sequence.

Event Queue: Purpose and Functions

Event Queue: Purpose

- Event receivers are started by tRFC.
- tRFC processing is a central component of the application server.
- Many events within a short period of time can result in a high tRFC load.
- Result: The performance of the entire application server is affected.
- Solution: The event queue catches the peak loadand facilitates controlled delivery.

A transactional RFC (tRFC) is called for each receiver, that is, the number of events is counted and multiplied by the relevant number of active receivers.

The tRFC mechanism of the application server has its own queue that is typically used only if an overload has occurred and if performance is also impaired.

You can use the event queue specifically for individual event linkages while other (uncritical) events are still delivered without a queue (consequently, these events are delivered sooner).

If the background job of the event queue continues to start the receivers using tRFC (adjustable), then no fewer tRFCs in total are executed than would be executed without the event queue. However, the calls are distributed over a longer period of time so the short-term system load is reduced.

Using the event queue can result in a loading time when events are being delivered, which (for a normal system load) corresponds to the time between two processes for the background job used for the delivery.

Reminder: In event communication, no receiving can (or should) occur via the "runtime" of an event.

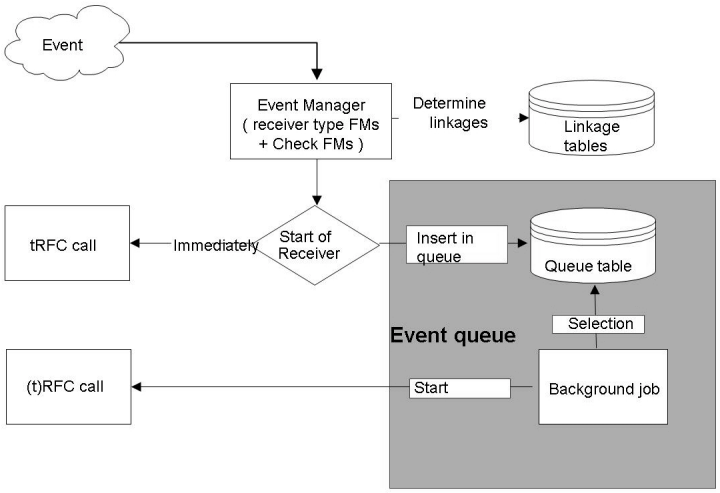


Figure 57: Event Queue: Functions

If an event is triggered, the event manager will initially run normally. Each time a receiver is started, one of the following decisions is made:

- To not use a queue, that is, to call directly using tRFC
- To use a queue, that is, the following happens:
  - The event (including event parameters) is entered in the queue table
  - Events are processed asynchronously from the queue using a background job
  - The job calls the receivers in the background using tRFC

The event queue is used in two phases and is controlled individually for each linkage. The queue is used if the event queue is activated globally AND an indicator in the event type linkage explicitly allows the queue to be used.

(Default: do not use queue).

Possible settings for the background job are:

a) Time between two runs of the job

b) Number of events to process per selection

c) Start receiver via RFC or tRFC (default)

The tRFC is used to remove the link between the receivers called at the same time during the background job.

Event Queue: Implementation in the System

Event Queue: Administration

- Aim of the event queue
  - An event does not start a workflow immediately, instead it buffers the start of the workflow in a queue.
  - The queue is processed in segments, thus balancing the event processing.
- What do you have to do?
  - Activating the event queue
  - Setting the basic data
  - Configuration of the background job and event delivery
  - Specification of the default behavior of linkages in case an error occurs in delivery
  - Activation of the workflow or task linkage for the queue

The event queue alleviates the load on the system in the subsequent processing of events.

The queue must be activated and the background job scheduled, so that the event queue can be used.

To activate the event queue, in the basic menu choose *Tools – Business Workflow – Development - Administration – Event Manager – Event Queue (SWEQADM) –* choose the Activation tab.

You also need to explicitly activate the workflow or task for the queue in the entry in the linkage table. (by selecting the *Enable Event Queue* flag)

For each error in the event delivery a mail is sent to the queue administrator. You can enter a system-wide default value for the behavior of event linkages in case of errors. You can redefine this value individually for every linkage. Possible settings:

- **Deactivation of linkage** The event linkage is deactivated. The event cannot be reprocessed and the linkage is not evaluated for future events.
- **Mark linkage as having errors** Events wait in the queue with the status “waiting due to error”. The affected event can be redelivered (manually), and all future events are placed in the queue.
- **Do not change linkage** The event is entered in the queue with the status “waiting due to errors” and can be redelivered (manually). Future events are processed as usual.

Caution: If an error occurs, this setting can lead to a large number of mails to the administrator.

Event Queue: Browser

- Overview of events in the event queue
- Detailed display of events (including event parameters)
- Events in the queue can be delivered or deleted manually

Based on the following capabilities, the event queue browser is a valuable tool to use for developing event-based communication:

- The queue browser facilitates user-friendly, filtered access to events (waiting for delivery or incorrect) saved in the queue, including their event parameters.
- You can deliver or delete events interactively.
- For debugging purposes, events can also be saved beyond their delivery.
- If the event linkage is set up accordingly (for example by manually setting a linkage to “incorrect”), events can be “stopped” and inspected for debugging purposes.

Lesson Summary

You should now be able to:

- Describe the functions of the event queue
- Activate the event queue
- Activate type linkages for the event queue

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Explain the basic concepts of using events
- Explain the processing logic of the event manager
- Find type linkages and instance linkages in the system
- Use the event log for documentation and testing of events
- Explain the enhancement of the event concept to process workflow instances.
- Describe the new options of the wait step in the workflow definition.
- Assess the use of “local events” and “correlations”.
- Trigger an event using a function module
- Describe the difference between triggering an event with the same name on a supertype, or on one of it's subtypes.
- Create a check function module
- Create a receiver type function module
- Describe the application of a receiver type function module
- Describe the functions of the event queue
- Activate the event queue
- Activate type linkages for the event queue

Internal Use SAP Partner Only

Test Your Knowledge

1. After an application has triggered an event, the event manager searches for active type linkages for the event and evaluates existing check and receiver type function modules. This all takes place in the context of the initiator.

   Determine whether this statement is true or false.

   □ True

   □ False

2. Event triggering using a function module must always end in a COMMIT WORK.

   Determine whether this statement is true or false.

   □ True

   □ False

3. Before you can use a receiver type function module, you need to explicitly maintain the type linkage table. You enter the receiver type function module instead of a receiver.

   Determine whether this statement is true or false.

   □ True

   □ False

4. The event queue must be activated explicitly. You also need to activate type linkages for the event queue.

   Determine whether this statement is true or false.

   □ True

   □ False

Answers

1. After an application has triggered an event, the event manager searches for active type linkages for the event and evaluates existing check and receiver type function modules. This all takes place in the context of the initiator.

   Answer: True

   To avoid consuming unnecessary system performance, the tRFC and therefore the change of context does not take place until it is certain that the workflow is definitely going to start. The checks in the Check and the Receiver type function modules are in the context of the trigger.

2. Event triggering using a function module must always end in a COMMIT WORK.

   Answer: True

   The event is not actually triggered until after the Commit Work.

3. Before you can use a receiver type function module, you need to explicitly maintain the type linkage table. You enter the receiver type function module instead of a receiver.

   Answer: True

   Because the receiver type function module selects one workflow from a selection to start after an event, there is normally no entry in the *Receiver* field.

4. The event queue must be activated explicitly. You also need to activate type linkages for the event queue.

   Answer: True

   If you use the event queue, it is not clear when the reaction to an event will take place. It depends on the number of entries in the queue as well as on the number of entries that are processed at one time. This may be undesirable for the event. Therefore, each linkage must be activated explicitly for each queue.

Unit 6

The Workflow Runtime System

Unit Overview

This unit describes the functions of the Workflow Manager and the Work Item Manager, and explains the different work item types and status transitions of a work item.

There is a recurring problem in the SAP Business Workplace area. There are constant performance problems in this area for various reasons. Part of this unit deals with new BAdIs developed for this purpose.

The unit also describes the implementation of deadline monitoring and possible error handling routines.

The final part of the unit covers administration functions such as the archiving, or in exceptional cases deletion of work items. This section also provides performance recommendations for using the Business Workplace, modeling, and administration.

Unit Objectives

After completing this unit, you will be able to:

- Understand how the workflow runtime system works
- Describe status transitions in the workflow
- Use a function module to start a workflow directly, without using an event
- Improve the performance of SAP Business Workplace.
- Manage large lists of work items.
- Reorganize the extensive use of dynamic columns and control the grouping functions in large worklists.
- Describe the different forms of deadline monitoring.
- Explain special situations for deadline monitoring.
- Restart a workflow if an error has occured.
- Archive work items
- Delete work items in test environments

- List the different performance approaches

Unit Contents

Internal Use SAP Partner Only

Lesson: Components and Architecture of the Runtime System

Lesson Overview

This lesson describes the components of the runtime system and the tasks of the Workflow Manager and the Work Item Manager.

It explains the different work item types, and how the system handles the status transitions of work items.

The lesson introduces function modules that you can use to start a workflow directly using a function module.

Lesson Objectives

After completing this lesson, you will be able to:

- Understand how the workflow runtime system works
- Describe status transitions in the workflow
- Use a function module to start a workflow directly, without using an event

Business Example

You want to start a workflow directly using a function module call, without an event.

Workflow Runtime System - Components and Overview

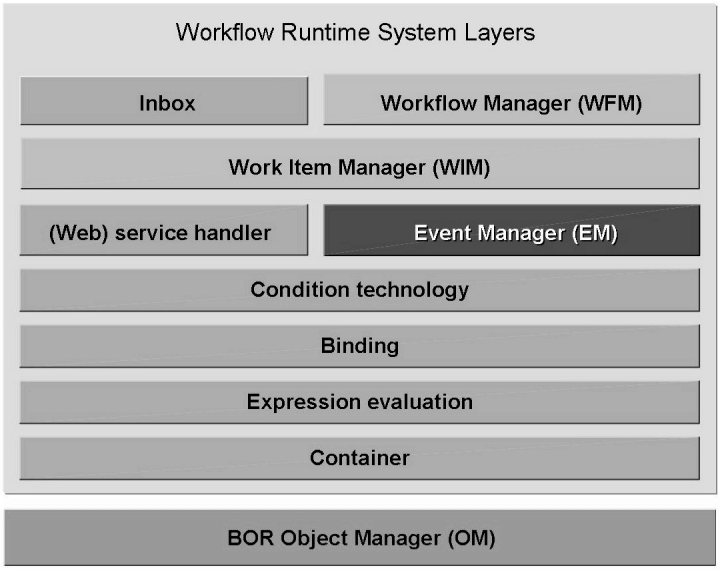


Figure 58: Workflow Runtime System - Components

Together, the workflow and the Business Object Repository (BOR) provide an object-oriented workflow management system.

The workflow manager (WFM) controls the execution of a workflow definition.

The work item manager (WIM) controls the execution of individual activities.

The Inbox service provides users on all front-end instances (SAP GUI for HTML, portal, and so on) with a list of their work items for processing.

The (Web) Service Handler controls the execution of functions that are not accessed using classic SAP-specific logs such as RFC (see workflow course BIT603).

The event manager controls the event communication (see the unit “Event Definition and Implementation”).

The workflow runtime is based on basic services or condition evaluations (IF, loops, preconditions and postconditions of steps, expression evaluation and container service).

The BOR runtime (object manager) manages all aspects of object access, for example, loading object data as required, accessing attributes, and calling methods.

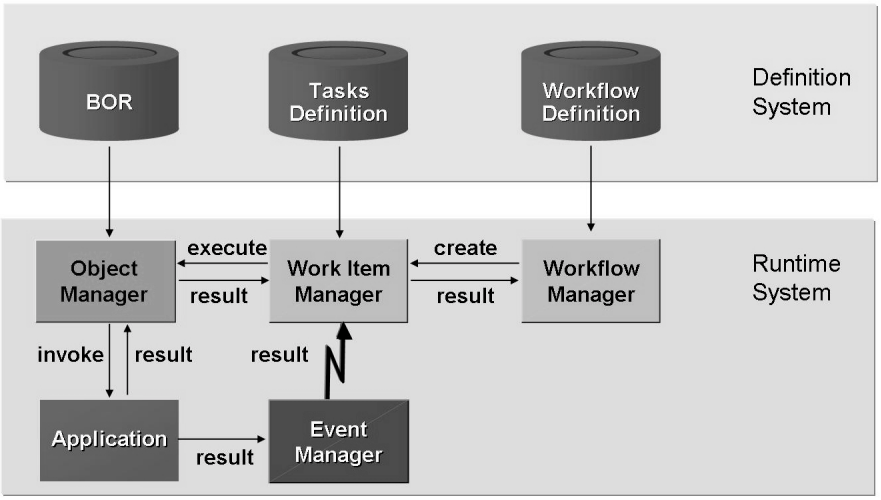


Figure 59: Workflow Runtime System - Overview

The workflow manager (WFM) reads the workflow definition and the step definition and creates one work item (WI) for each step using the work item manager (WIM). The workflow manager is then finished.

The work item manager manages the work item and triggers the application using the object manager (OM).

If the method used is synchronous, the result and the export parameters of the method are returned to the work item manager using the object manager.

If the method is asynchronous, the result of the method is returned to the work item manager using an event triggered in the event manager (EM). You must define this event as a “terminating event” for the step.

After receiving the return parameters, the work item manager checks the return code and, if successful, returns the data to the workflow manager, which determines the next step to be executed.

This loop continues until the end of the workflow is reached.

Event creator, conditions, process controls and container operations are executed temporarily in the workflow manager (that is, a work item is not created).

Tasks of the Workflow Manager and the Work Item Manager

Workflow Manager: Tasks

- Controls the execution of the workflow definition
- Transfers data between individual steps of a workflow definition
- Logs the workflow process
- Calculates deadlines for steps in a workflow definition
- Executes rule resolution at step level
- Executes container operations, process controls and event creators
- Evaluates conditions

The workflow manager component (WFM) advances from one workflow step to its subsequent steps. The workflow manager controls all operations for which the predecessor successor relationship between steps must be known. This also includes canceling a running workflow if an error occurs, for example, (determining and canceling all running work item instances) and the reverse situation of restarting a workflow after correcting an error.

Work Item Manager: Tasks

- Controls processing of individual steps in a workflow definition
- Monitors deadlines for individual steps in a workflow definition
- Assigns agents for individual steps in a workflow definition
- Logs all activities executed on a step in a workflow definition
- Resolves default rule

A work item is created for each activity in a workflow and executed under the control of the work item manager. When the work item is executed, the work item manager is linked to the number of possible work item statuses and the allowed status transitions: The processing of the work item is logged by changing the work item status, for example, from “Ready” to “In process”. During the status transitions, the work item manager locks the work item to guarantee exclusive processing.

Deadlines are monitored using the background job SWWDHEX or the report RSWWDHEX (see below).

The work item manager evaluates the agent assignment of a dialog step: Intersection of “possible” and “selected agents” minus the “excluded agents” (for details, see the unit on “Rule function modules”).

All the activities executed by a user are logged for the work item. A complete processing history is therefore available for each work item.

When the work item is executed, all secondary methods are started in addition to the (main) method from the task. Secondary methods essentially run in a separate session.

Before the method from the task is executed, the 'before' methods are executed (in no particular sequence). After the main method is executed, the 'after' methods are executed (in no particular sequence).

Work Item Manager: Asynchronous Communication

- Required for transition from dialog to background processing
- Use of transactional RFC (tRFC)
- Customizing: Maintaining the Logical Destination

  WORKFLOW_LOCAL _<Clnt> - required

tRFC is a Basis component (CALL FUNCTION ... IN BACKGROUND TASK). The function module called is triggered after the next SAP Commit (see the unit “Event Definition and Implementation”).

The logical destination WORKFLOW_LOCAL<Clnt> is maintained for the workflow runtime system within Customizing using transaction SWUB. The user entered there must have all authorizations and should therefore (for security reasons) be a background user.

Caution: If the password for this user is changed within user administration, it must also be changed in transaction SWUB.

Caution: You cannot maintain the logical destination WORKFLOW_LOCAL<Clnt> in transaction SM59.

The tRFCs issued are logged until they are executed. If the execution is successful, these entries are deleted. If an error occurs during execution, this is also recorded in the log.

Only up to 3.1G: You can view the log in transaction SM58. You can access it using the name of the user who called the tRFC. For the workflow system, this is usually the user entered in the WORKFLOW_LOCAL_<Clnt> destination.

Work Item Types

Work Item Types

- A - Work queue work item
- B - Background work item
- C - Container anchor work item
- D - Missed deadline work item
- E - Wait step work item
- F - Workflow work item
- R - Web work item
- W - Dialog work item

A work queue work item is a “logical unit” of objects and tasks to be executed together. Both the objects and the tasks can differ.

A background work item is used to execute tasks that do not require dialogs. They are executed automatically by the workflow runtime system and do not appear in the Business Workplace.

A container anchor work item is used as a logical relationship between several associated objects. It corresponds to the assignment table from WF21. A container anchor does not have any semantics of its own and never appears in the Business Workplace.

A missed deadline work item is created by default when the workflow system detects that a monitored deadline has been missed.

A wait step work item receives events created externally or by other work items. It does not have any processing logic and does not appear in the Business Workplace.

A workflow work item represents a workflow to be processed, whose definition is then processed by the workflow manager.

A Web work item represents a Web activity.

A dialog work item represents a workflow step to be executed by a user.

Work Item Status

Work Item Status

- WAITING - - waiting
- READY - - ready
- SELECTED - - reserved
- STARTED - In process
- COMMITTED - - executed
- ERROR - - error
- COMPLETED - - completed
- CANCELLED - - logically deleted

A work item has WAITING status if its requested start has not yet been reached. With this status, it cannot be viewed by any user.

A work item with READY status appears in the recipients' Business Workplace. Work items running in the background with this status are executed immediately.

The SELECTED status applies only to dialog and work queue work items. This status means that the work item has been reserved by one of the recipients and therefore disappears from the other recipients' Business Workplace.

As soon as the method used is started but is not yet complete, the work item has STARTED status. The work item remains in the Business Workplace of the current agent.

If the method used has been executed successfully but the work item must be set manually to 'done' , it has COMMITTED status.

If a return code that is not modeled in the related workflow was returned when the method was executed or if irreparable damage occurred in the work item manager, the work item gets ERROR status.

COMPLETED status means the work item was completed successfully.

If the work item is no longer required within the flow logic of the workflow, it is assigned CANCELLED status.

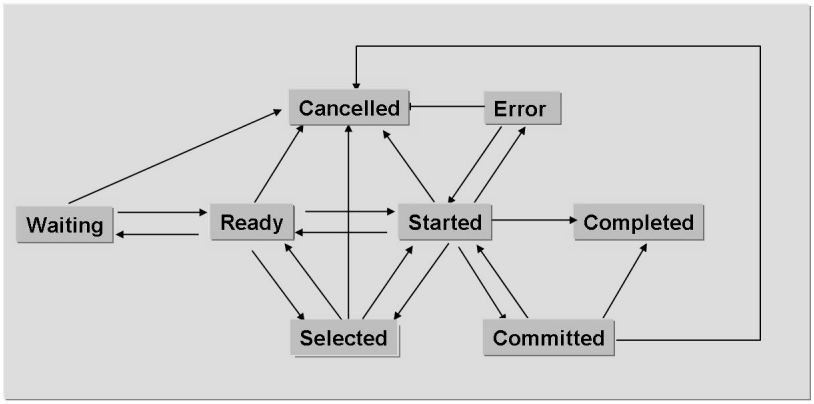


Figure 60: Status Transitions

Not all work item types can assume all statuses (for example, SELECTED applies only to dialog and work queue work items, COMMITTED applies only to dialog work items).

The start statuses are WAITING or READY.

The end statuses are CANCELLED or COMPLETED.

The change from one status to another always occurs implicitly on the basis of an executed activity (for example, “reserve” in the Business Workplace).

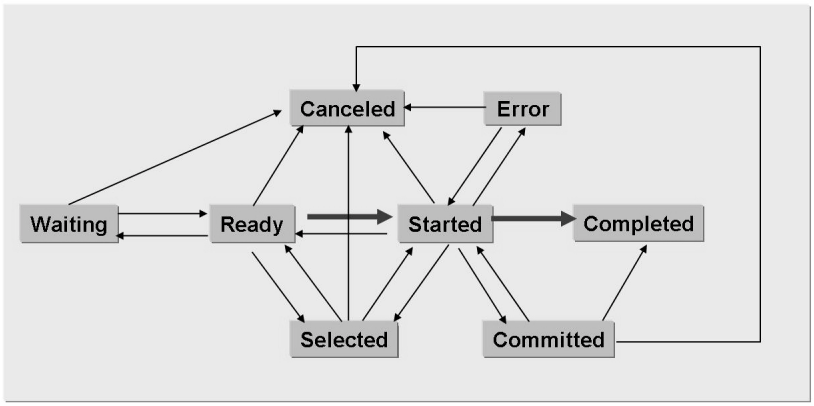


Figure 61: Status Transitions - Background Work Item

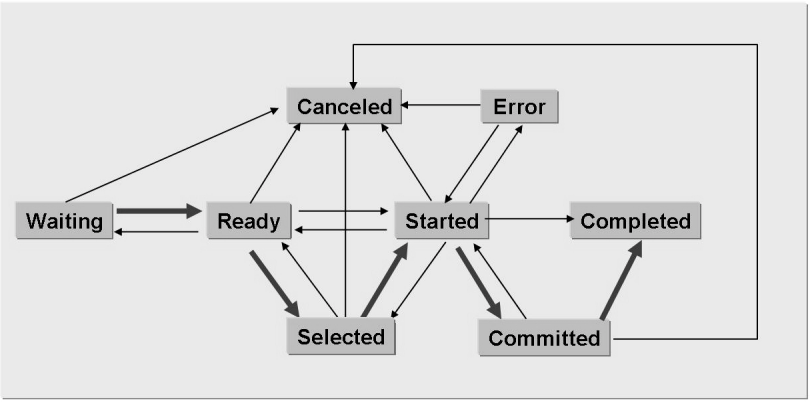


Figure 62: Status Transitions - Dialog Work Item

SAP WAPI Function Modules

SAP WAPI Function Modules

- Function Group SWRC

  Function modules for processing the worklist of a user

- Function Group SWRI

  Function modules for selecting work items according to various criteria (e.g. for reporting and analysis)

- Function Group SWRR

  Function modules for starting a workflow or for triggering an event

SAP WAPI is an SAP-specific API based on the interface definitions of the Workflow Management Coalition (WfMC).

The SAP WAPI function modules call the function modules internally from SWWA. In contrast to the internal modules, the SAP WAPI function modules form a guaranteed interface.

You must always use the function modules from SWRI for work item selection in analysis and reporting situations. You should avoid accessing tables directly (for example, access to the SWWWIDEADL or SWWWIRET tables) at all costs.

WIM API - Function Modules

WIM API - Function Modules

- Creating a work item

  SWW_WI_CREATE(_SIMPLE)
- Executing a work item

  SWW_WI_WITHOUT_WLC_EXECUTE
- Starting a work item (create and execute immediately)

  SWW_WI_START(_SIMPLE)

To access the workflow systems, you should always use SAP WAPI interface function modules (naming convention: SAP_WAPI_*). These function modules are contained in the SWRC and SWRR function groups. The specified modules continue to be supported for downward compatibility reasons only.

When a work item is created or started, the caller must first have filled the task container with input parameters for the task, in accordance with the task definition. The caller must also implement error handling measures if the work item cannot be created or started.

Before a work item is executed, the caller must check whether all the prerequisites for executing work items have been fulfilled (for example, is the current user even permitted to execute the work item). The released WORKITEM_ARCHIVE_OBJECT module for archiving a work item is contained in the SWWA function group.

Work Item Manager Tables

Work Item Manager Tables

- Released header data

  View SWWVPUBLIC
- Deadline data

  Table SWWWIDEADL
- Return data for method execution

  Table SWWWIRET
- Container tables depend on the selected persistence profile

As of Basis Release 6.10, work item containers are saved in the database in accordance with the persistence profile specified in the Workflow Builder. As of Release 6.10, the tables SWW_CONT and SWW_CONTOB therefore still contain only the data of those work item containers that are written with the persistence profile for **compatibility mode**. When you use the XML persistence (allows strings and other structures in the container), other tables are used and other individual elements can no longer be accessed directly. Therefore, access to container data must always be changed to the released read modules (SAP_WAPI_CONTAINER_READ).

The persistence profile is set on the 'Control' tab in the version-dependent basic data of the workflow.

The SWWVPUBLIC view contains the header data of work items available for general selection. Changes to the structure of the SWWVPUBLIC view are upward compatible only. The structures of the remaining tables (and others) are not guaranteed. If you want the data of a known work item to be read, you should always use the SAP_WAPI_* selection modules for this.

Changes to work items must never be made by changing tables directly. You should always use the SAP_WAPI_* function modules instead.

Internal Use SAP Partner Only

Exercise 11: Start a Workflow Directly using a Function Module, Instead of Using a Triggering Event

Exercise Objectives

After completing this exercise, you will be able to:

- Start a workflow directly from a function module without using a triggering event.

Business Example

Rather than using an event, you want to use the API of the work item manager to start a workflow directly (you may want to do this, for example, for performance reasons).

Task:

Start a workflow using the work item manager API.

1. Create a report called **Z##START** that starts your workflow **Z##MSIPROC1** by calling the function module SWW_WI_START_SIMPLE API.

Solution 11: Start a Workflow Directly using a Function Module, Instead of Using a Triggering Event

Task:

Start a workflow using the work item manager API.

1. Create a report called **Z##START** that starts your workflow **Z##MSIPROC1** by calling the function module SWW_WI_START_SIMPLE API.

   a) For an example solution, see the report Z00START in transaction SE38.

   Path (starting from SAP Easy Access):

   Tools → ABAP Workbench → Development → ABAP Editor

```
REPORT Z00START.
<INCLUDE CNTN01>. " definition of container and object macros
PARAMETERS: OBJKEY LIKE MARA-MATNR DEFAULT '200-100'.
DATA:TASK LIKE SWWVPUBLIC-WI_RH_TASK VALUE
'WS96000011',
WI_ID LIKE SWWVPUBLIC-WI_ID,
OBJECT TYPE SWC_OBJECT.
DATA:AGENTS LIKE SWHACTOR OCCURS 0 WITH HEADER LINE,
WI_CONTAINER LIKE SWCONT OCCURS 0 WITH HEADER LINE.
* set the import parameter for flow Z00MSIPROC1
* first: create object reference (runtime handle)
SWC_CREATE_OBJECT OBJECT 'Z00MARA' OBJKEY.
* second: write reference to work item container using parameter def
SWC_SET_ELEMENT WI_CONTAINER 'MaterialMaster' OBJECT.
* third: convert references in container from runtime handle to
* persistent object reference
SWC_CONTAINER_TO_PERSISTENT WI_CONTAINER.
* start the workflow via the function API
```

Continued on next page

Internal Use SAP Partner Only

```
CALL FUNCTION 'SWW_WI_START_SIMPLE'
EXPORTING
* CREATOR = ' '
* PRIORITY = NO_PRIO
TASK = TASK
* CALLED_IN_BACKGROUND = ' '
* DEADLINE_DATA = ' '
IMPORTING
WI_ID = WI_ID
* WI_HEADER =
* RETURN =
* WI_RESULT =
TABLES
AGENTS = AGENTS " not needed here
* DEADLINE_AGENTS =
* DESIRED_END_AGENTS =
* LATEST_START_AGENTS =
* EXCLUDED_AGENTS =
* NOTIFICATION_AGENTS =
* SECONDARY_METHODS =
WI_CONTAINER = WI_CONTAINER
EXCEPTIONS
ID_NOT_CREATED = 1
READ_FAILED = 2
IMMEDIATE_START_NOT_POSSIBLE = 3
EXECUTION_FAILED = 4
INVALID_STATUS = 5
OTHERS = 6.
```

Continued on next page


```
IF SY-SUBRC NE 0.  " error message
MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ELSE. " success message
MESSAGE ID 'WZ' TYPE 'S' NUMBER '703' WITH WI_ID.
ENDIF.
```

Lesson Summary

You should now be able to:

- Understand how the workflow runtime system works
- Describe status transitions in the workflow
- Use a function module to start a workflow directly, without using an event

Lesson: Using BAdIs to Manipulate SAP Business Workplace

Lesson Overview

This lesson explains how to use various BAdis to improve the performance of SAP Business Workplace.

In this lesson you will also learn how to control large quantities of work items and the related use of container accesses in various areas of the display.

Lesson Objectives

After completing this lesson, you will be able to:

- Improve the performance of SAP Business Workplace.
- Manage large lists of work items.
- Reorganize the extensive use of dynamic columns and control the grouping functions in large worklists.

Business Example

A call center works with SAP CRM software. Many processes are implemented with SAP Business Workflow. Some call center employees have a large number of work items in their inboxes (sometimes more than 1000 tasks).

The IT department of your company must reduce the number of tasks in a workplace to improve performance. You should assess the new options to do so and work out the correct procedure.

Identification of Central Problems

Various BAdIs have emerged from customer projects, to solve frequent problems for the SAP Business Workplace, for example, if many employees work on large, joint task lists (call center scenario):

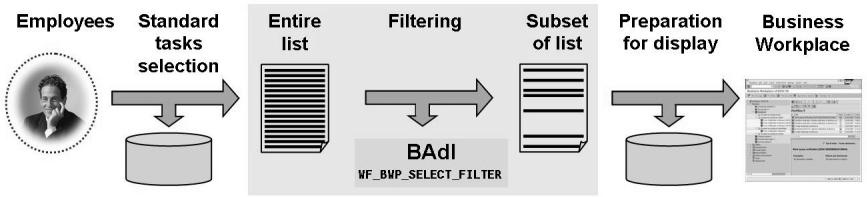
- Large, slow worklists
- Worklists must be refreshed often to prevent tasks overlapping.

The extensive use of dynamic columns in conjunction with large worklists is also a widespread problem.

In this context, we must also refer to the grouping of tasks in large lists, in which container data is accessed.

- Grouped according to contents
- Grouped according to content type
- Grouped according to sort key

The BAdI SBWP: Filtering Worklists



- **The BAdI WF_BWP_SELECT_FILTER reduces the number of work items that are displayed in the workplace**
- **Customer-specific filter algorithms can be defined**
  - **Example implementation available based on random numbers**
- **Performance-critical operations executed in reduced worklist**
  - **Default attributes, dynamic columns, work item texts in various languages, …**

Figure 63: Filtering Task Lists

You can use the BAdI WF_BWP_SELECT_FILTER to reduce the number of work items that are displayed in a user inbox.

The BAdI imports the complete list (work item header information) and exports a (reduced) worklist, which can then be displayed in the inbox.

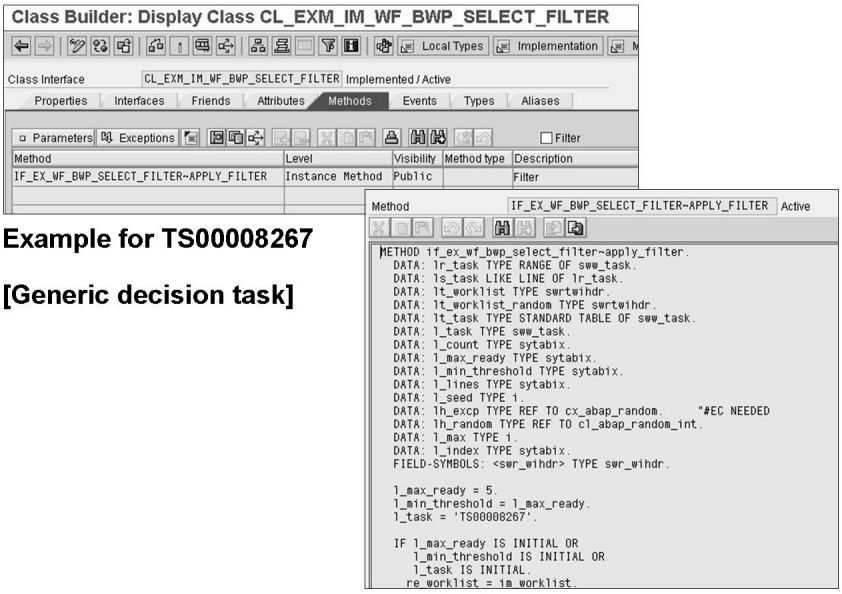
If the employee assignment is not selective enough, you can implement additional algorithms at this point.

- Disjoint or intersecting distribution of work to different agents
- Stable or unstable with respect to inbox refresh

Task selection is not a performance-critical operation (as it involves a database access). The preparation of the selected data for display in the Business Workplace is a more expensive operation.

The preparation is called after the BAdI has run, and is only executed for a reduced number of items for this reason.

The BAdI referred to is available for Web AS 6.20 Support Package 44, Web AS 640 Support Package 9 (see also Note 7657830).

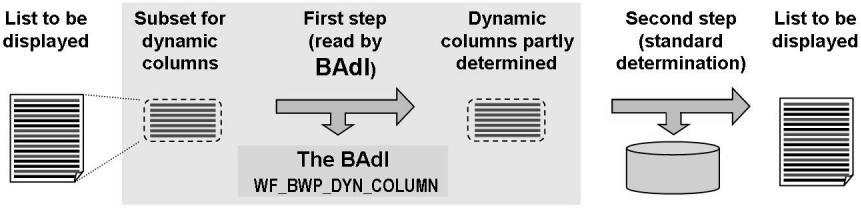


Example for TS00008267

[Generic decision task]

Figure 64: Example for WF_BWP_SELECT_FILTER

The BAdI SBWP: Accelerating Dynamic Columns



- **Dynamic columns can be defined at task level**
- **The use of dynamic columns is performance-critical**
  - **Containers are read, objects are instantiated and object attributes are read for every individual work item (container and BOR cannot be used for mass processing)**

`WF_BWP_DYN_COLUMN` can help to improve performance

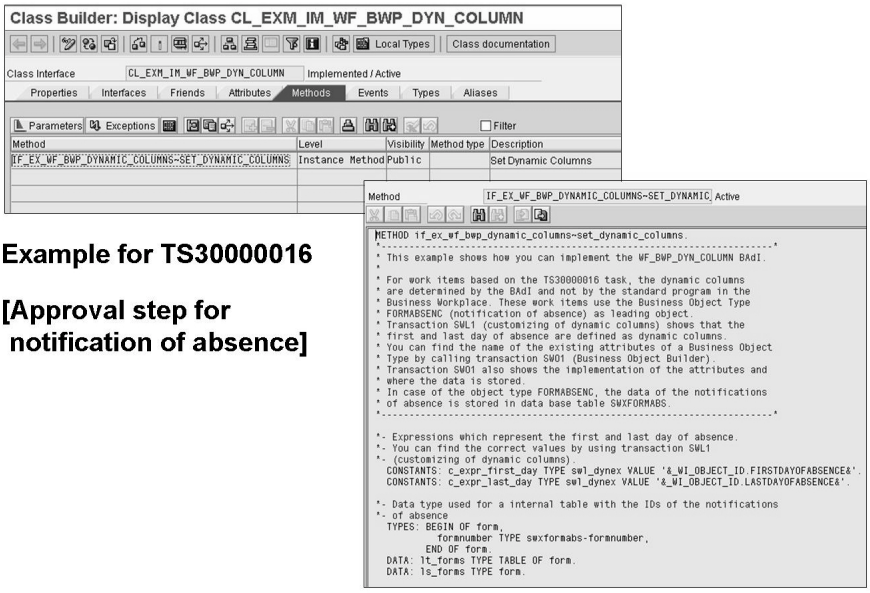
Figure 65: Accelerating Dynamic Columns

You should use dynamic columns very carefully, because they can cause large lists to become incredibly slow. Business objects must be instantiated and attributes must be read:

- Mass processing cannot be carried out in the container or in the BOR, and for this reason they are critical in terms of performance and throughput.

You can use the BAdI WF_BWP_DYN_COLUMN to avoid the standard selection for dynamic columns. This enables direct access to the database en masse and the existing storage mechanisms.

Dynamic columns for which the contents cannot be determined by the BAdI, run in the old inefficient way.

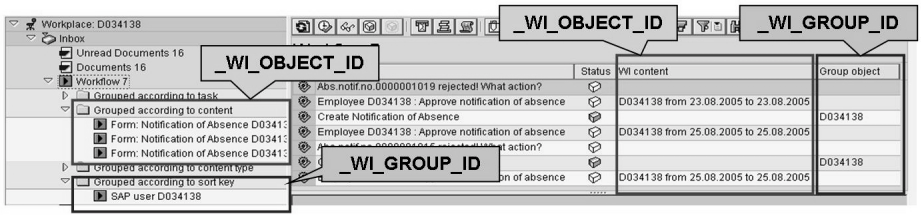


Example for TS30000016

[Approval step for notification of absence]

Figure 66: Example for WF_BWP_DYN_COLUMN

The BAdI SBWP: Evaluating Default Attributes



- **Default (object) attributes displayed as object representation**
- **The handling of these object attributes is critical for the performance**
  - **Cannot be used for mass processing and very critical for large worklists**
- **The BAdI `WF_BWP_OBJ_ATTRIBUTE` can help to improve performance**
  - **Enables customers to define programming for mass processing**

Figure 67: Evaluating Default Attributes

In the display function of SAP Business Workplace, object instances are represented by values of their default attribute. To evaluate these default attributes, information must be read from the task container. This information must also be instantiated and accessed.

This ultimately means there is at least one single access for each object instance. As mass processing cannot be carried out in the container or in the Business Object Repository, the performance of the user interface is seriously affected if there are large worklists.

For the availability of *DYN_COLUMN and *OBJ_ATTRIBUTE: SAPKB62057, SAPKB64015 and SAPKB70006 (see Note: 848382).

Lesson Summary

You should now be able to:

- Improve the performance of SAP Business Workplace.
- Manage large lists of work items.
- Reorganize the extensive use of dynamic columns and control the grouping functions in large worklists.

Related Information

- Additional information is contained in the SAP online help under http://help.sap.com.

Internal Use SAP Partner Only

Lesson: Deadline Monitoring and Error Handling

Lesson Overview

This lesson explains which forms of deadline monitoring are possible, and how they can be modeled technically.

The lesson also explains how to restart workflows with error status.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the different forms of deadline monitoring.
- Explain special situations for deadline monitoring.
- Restart a workflow if an error has occured.

Business Example

You want to monitor deadlines for individual steps in different workflow templates.

In a productive workflow that is already running in your system, workflow instances have gone into ERROR status due to incorrect data. You want to restart the workflows after you have corrected the data.

Deadline Monitoring

Deadline Monitoring

- Four deadlines monitored
  - Requested start (resubmission)
  - Latest start
  - Requested end
  - Latest end
- All deadlines monitored using a background job
  - Systems with Basis Release 6.10 and higher:

    Job regularly runs periodically
  - Systems with Basis releases earlier than 6.10

    Job can be scheduled periodically or as required (default)

All deadlines are monitored using the SWWDHEX background job with job class A. The job calls the RSWWDHEX report.

In systems prior to Release 6.10, you can choose between individual or periodic deadline monitoring. Individual monitoring is active by default.

With individual monitoring, the background job is rescheduled for the next deadline to be monitored and updated if a new deadline is entered for a work item with a deadline that occurs before any other deadlines. Therefore, the background job runs only if it is really required.

If the background job aborts for any reason, you must reschedule it again by starting the above report.

Individual monitoring is not suitable for monitoring many deadlines. If you have to monitor many deadlines, you should switch to periodic deadline monitoring (for example, every three minutes).

You can switch between the two monitoring types at any time using SWWA or the RSWWDHIN report.

As of Basis Release 6.10, the system recognizes only the periodic scheduling of the SWWDHEX job.

Inheritance and Special Circumstances in Deadline Monitoring

Deadline Monitoring Inheritance

- Deadline data and monitoring activity inherited by dependent work items from workflow
- Definition of monitoring activity for dependent work item overwrites inheritance
- Check deadline data in dependent work item against inherited deadlines

Inheritance applies only when you create a subordinate work item. Subsequent changes to data for the superordinate workflow are not transferred to existing dependent work items.

Each change in the data for the dependent work item is checked against the superordinate workflow. You cannot postpone beyond the inherited deadlines.

Deadline Monitoring - Special Situations

- Basis releases before 6.10, and high numbers of deadlines

  → If there is a high number of deadlines switch between single case-based to permanent.
- Multiple instances of SWWDHEX

  → Delete all instances using the job overview.

  → Reschedule
- No instance of SWWDHEX

  → Check selection for job overview.

  → Reschedule

Reschedule SWWDHEX in transaction SWWB.

When many deadlines are to be monitored, a conflict may arise with the SAP_REORG_JOBS background job. In this case, deadline monitoring should be suppressed while SAP_REORG_JOBS is running.

When displaying background jobs in the job overview, remember that this is a cross-client display. You should therefore check whether the background jobs displayed are actually scheduled in the current client.

Error Handling in Workflow

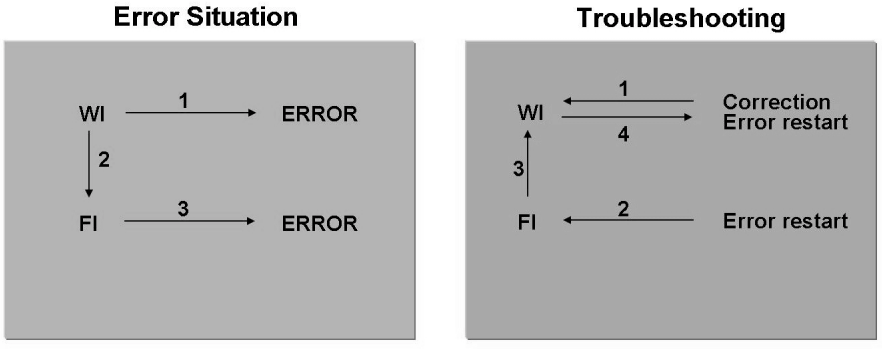


Figure 68: Error Handling

Error situation (example: no agent found for a step)

- The status of the work item changes to ERROR.
- The error is propagated (“up”) to the superordinate workflow.
- The status of the superordinate workflow also changes to ERROR.

Error correction

- The incorrect work item is corrected (by correcting the work item container, for example, or changing the agent assignment of a task).
- A “restart after error” is executed on the *superordinate* workflow.
- The superordinate workflow propagates the restart (“down”) to the work item.
- A “restart after error” is executed implicitly on the work item.

When an error occurs, it is propagated from the “bottom up” to all superordinate workflows.

To correct the error, the work item is first repaired locally.

Then the workflow is restarted from the “top down”, and all work items belonging to the workflow are also restarted. You must always execute the restart at the highest level workflow, and never from the work item with the error.

You should not use transaction SM58 (earlier than Release 4.0A) to restart work items with errors because this might cause steps that have already been executed to be executed a second time.

All correction actions on work items should be performed with great caution to avoid side effects.

Behavior with Non-Modeled Errors

	Temporary error	Non-temporary Error
Background work items Workflow work items	STARTED status, restart with SWWERRE	ERROR status, sends an error mail to the system administrator for workflow
Dialog work items	STARTED status, restart by the user	ERROR status, WI remains in Business Workplace of the user

Figure 69: Behavior with Non-Modeled Errors

The following happens when non-modeled errors occur:

The method definition determines whether an error is temporary or not.

The RSWWERRE job monitors the background activities.

The status of background work items and workflow work items changes to ERROR after the nth unsuccessful execution.

Comparison of Jobs SWWERRE and SWWDHEX

	SWWERRE	SWWDHEX
Trigger	Work item status	Deadline data
Action	Predefined action, triggered by tRFC	Predefined or self-defined action, triggered by tRFC
Job	Job class C, periodic job	Job class A, self-scheduling job
Spacing	Configurable	Configurable or dependent on deadlines
Authorization	Normal authorization check	Authorization check deactivated
Scheduling	Customizing or auto-Customizing	Customizing or automatic, auto-Customizing

Figure 70: Error Handling - Deadline Monitoring

Neither job allows real-time control because background jobs that are running are not interrupted.

Internal Use SAP Partner Only

Exercise 12: Create a report that lists all workflow instances that have not terminated and that started more than a week ago. The List Should Display the Task ID, the Status and the Creation Date.

Exercise Objectives

After completing this exercise, you will be able to:

- Create a report that selects data from the runtime tables of the WebFlow Engine.

Business Example

Some of your workflows are still not complete after what seems like a very long period of time.

You want to extract detailed information on these workflow instances.

Task:

Task

1. Create a report **Z##WEEKL** that lists all workflow instances that are not yet complete and that started more than a week ago. The list should display the task ID, the status and the creation date for each instance of this kind.

Solution 12: Create a report that lists all workflow instances that have not terminated and that started more than a week ago. The List Should Display the Task ID, the Status and the Creation Date.

Task:

Task

1. Create a report **Z##WEEKL** that lists all workflow instances that are not yet complete and that started more than a week ago. The list should display the task ID, the status and the creation date for each instance of this kind.

   a) As an example solution, see the report Z00WEEKL in transaction SE38.

   Path (starting from SAP Easy Access):

   Tools > ABAP Workbench > Development > ABAP Editor

```
REPORT Z00WEEKL LINE-SIZE 512.

INCLUDE RSWWINCL.

TABLES: Z00VHEACON. " public view on work item header data

PARAMETERS: TASK LIKE SWWWIHEAD-WI_RH_TASK,

DAYS_AGO LIKE SWWWIHEAD-RETRY_CNT DEFAULT '07'.

DATA: CAND_WIS LIKE Z00VHEACON OCCURS 0 WITH HEADER
LINE.

DATA: START_DATE LIKE SY-DATUM.

WRITE: / 'List of flows started longer than ', DAYS_AGO, ' days ago:'.

* determine start date for candidate work flows

START_DATE = SY-DATUM - DAYS_AGO.

* search list of candidate flows

IF TASK NE SPACE. " search for a specific task

SELECT * FROM Z00VHEACON INTO TABLE CAND_WIS

WHERE WI_RH_TASK EQ TASK AND
```

Continued on next page

Internal Use SAP Partner Only

```
WI_CD <= START_DATE AND
WI_CHCKWI EQ SPACE AND " top-level flows only
( WI_STAT NE WI_STATUS_COMPLETED AND
WI_STAT NE WI_STATUS_CANCELLED ).
ELSE. " search for all tasks
SELECT * FROM Z00VHEACON INTO TABLE CAND_WIS
WHERE WI_CD <= START_DATE AND
WI_CHCKWI EQ SPACE AND " top-level flows only
( WI_STAT NE WI_STATUS_COMPLETED AND
WI_STAT NE WI_STATUS_CANCELLED ).
ENDIF.
* display result list
LOOP AT CAND_WIS.
WRITE: / CAND_WIS-WI_ID, CAND_WIS-WI_RH_TASK,
CAND_WIS-WI_STAT,
CAND_WIS-WI_CD, CAND_WIS-VALUE.
ENDLOOP.
```

Lesson Summary

You should now be able to:

- Describe the different forms of deadline monitoring.
- Explain special situations for deadline monitoring.
- Restart a workflow if an error has occured.

Internal Use SAP Partner Only

Lesson: Administration Tasks

Lesson Overview

This lesson describes which administration tasks need to be executed with the highest priority.

In particular, this means archiving work items that have status completed or canceled.

The lesson also provides an overview of different approaches for performance optimization in the environment of the WebFlow Engine.

Lesson Objectives

After completing this lesson, you will be able to:

- Archive work items
- Delete work items in test environments
- List the different performance approaches

Business Example

You have been using the WebFlow Engine for a long time and the relevant tables are full. You must archive data. You also want to find an option to delete work items from the database, which were created in the test.

Work Item Administration

Work Item Administration

- Initiate waiting work items
- Release reserved work items
- Restart a work item after errors
- Logically delete a work item
- Successfully complete a work item explicitly

Manual correction:

- The administrator action always originates from the technical work item display in the Business Workplace.

  You display the work item in change mode and execute the required function.

The functions offered will depend on the work item and its status.

All administrator function modules are checked with regard to authorization. Administrator function modules are used to process a work item in exceptional cases. The required authorizations are contained in the authorization profile S_WF_WF_ADMIN.

When a waiting work item is initiated, its status changes from WAITING to READY. This function can be executed if deadline monitoring is not running due to the background queue overloading or an error in the background system.

The status of the work item is changed from SELECTED or STARTED to READY again. You can use this function to display a work item that is blocked by an absentee to other recipients.

When you restart a work item after an error, the status of the work item changes from ERROR to STARTED and this might automatically execute the method in question.

Logical deletion changes the status of the work item to CANCELLED.

The workflow system administrator can change the status of a work item explicitly to COMPLETED but is then responsible for filling the work item container with the expected return parameters of the method beforehand.

Finding Work Items

- Selection report for finding work items
  - ID
  - Work item type
  - Status
  - Task
- Display and change work items of all types

After the work item is selected, the users can change and/or process the work item if they are assigned to the relevant task in the task profile and the work item is not already reserved by another recipient.

If the work item is not task-based (for example, wait steps, work queues, and so on), the user can change and/or process it if it is not already reserved by another recipient.

Work Item Language

- Each work item has a language.
- The language of a work item is derived from the language of the superordinate workflow.
- You can set the workflow language when you create the work item.
- The language of an event can be set as a default parameter.

The work item language determines the language of the short text and the status texts. The language of the long text is the same as the user's logon language.

When you start a workflow using an event, the language of the event is forwarded to the workflow.

In the Business Workplace, users can use the “personal workflow settings” to determine whether they always want to view work items in the Business Workplace in the logon language (instead of the specified work item language). In this case, all work item texts are retrieved in the user's current logon language when the Business Workplace is set up.

This setting can lead to a considerable delay in calling (or refreshing) the Business Workplace.

Archiving and Deleting Work Items

Archiving and Deleting Work Items

- Delete in test system using RSWWWIDE and RSWWHIDE reports
- Delete in production system only using WORKITEM archiving object

Reports RSWWWIDE and RSWWHIDE are intended for internal use only. The work items are deleted from the database without further query or an authorization check. You can start the reports in the administration menu from the workflow development environment. You should call the reports first for checking purposes with the indicator “Display Only”. You should then carry out the actual deletion in a background job.

Workflow work items also automatically delete their dependent work items.

In a production system, you must archive the work items before you delete them (transaction SARA).

For security reasons, archived work items can be read but not reloaded. The read program must be written individually by the customer. A sample program (RSWWARCR) is supplied by SAP.

Other archiving objects can also archive work items using the WORKITEM archiving class. However, for reasons of data consistency, work items cannot be deleted this way.

Container anchor work items should be deleted using the RSWWCIDE report.

Reading Archived Work Items

- RSWWARCR read program
  - Sequential reading of the entire archive
  - Template for own read programs
- RSWWARCP read program
  - Reads a process for a particular object
  - Formats workflow log

RSWWARCR is used as a template for user-defined programs. The user source code begins after the SWW_WI_LIST_ARCHIVED_READ function module is called, which imports the entire archive into internal tables. The tables contain all processes in a depth-first storage format.

RSWWARCP is a special case of a user-defined read program. It searches for the associated process for a particular application object and restructures the workflow log for this process as far as possible.

The archive data cannot be reloaded into the original work item tables because the runtime system reuses the work item numbers and this could cause data to be overwritten during the reloading process.

The Report RSWWWIDE	Archiving Object WORKITEM
Possible in every status	Only poss in COMPLETED status or CANCELLED status
Direct start without authorization check	Authorization check using transaction SARA
Direct deletion from the database	Archiving and deletion two separate steps
Execution in dialog box or in background	Execution in background at a later time
Log data not deleted	Log data also archived and deleted

Figure 71: Comparison Archiving - Deleting

You must never delete work item data directly in tables; use only the SWW_WI_DELETE function module for this.

Large amounts of data should always be deleted in background processing.

You can use the RSWIWADO report to determine the necessity to archive or delete work items.

You can use the RSWWHIDE report if you have to delete only the workflow log.

Performance Considerations

Performance 1 - Business Workplace

- Detailed task assignment in the organization model
- Do not display deadline or grouping data in the standard configuration.
- Display work items in the languages they were created in rather than in the logon language.
- Allow buffering of organizational data.
- Delete work items no longer required.

If possible, general tasks should not be used. However, if general tasks are used, a restrictive agent assignment should be implemented in the step definition, otherwise all system users will be able to display these work items. The more detailed the task assignment, the better the whole system performs.

The following columns lead to a subsequent selection if they are adopted in the standard configuration:

- Object _WI_OBJECT_ID
- Group _WI_GROUP_ID
- Task text WI_RHTEXT (if work item display is in logon language)
- To do by (date) WI_LED
- To do by (time) WI_LET

The buffering of organizational data can be controlled in table T77S0. Follow the instructions in note 98407.

Irrelevant work items or work items created for test purposes only impair performance and result in complicated, long lists in the Business Workplace.

Performance 2 - Modeling

- No workflows that check input data in the first step
- Packaging of input data into smaller units or use of event queue
- Start workflows directly using the work item manager API (no events from report written by user).
- Replace reading/calculating background methods with virtual attributes.
- Group “small” background methods into “larger” units.

Instead of a “check step”, a check function module or a start condition can be implemented in the workflow when the workflow is started using an event. Similarly, restrictive conditions exist that prevent a workflow starting “unnecessarily” when a workflow is started using change document management.

The creation of packages is particularly relevant if large amounts of data are imported into the system (using a batch input, for example) and many events are triggered within a short space of time. The event queue is a generic tool used to achieve this type of packaging. In addition, a destination can be specified for each event receiver in the type linkage table. Consequently, specific events can then be received on a specific application server (to keep the other application servers free, for example).

You can use the work item manager API instead of an event if a workflow is started from a user-defined report scheduled as a background job (each event delivery “costs” a tRFC).

Each activity in the workflow requires different workflow system operations at runtime (binding, and so on) and, in particular, generates a persistent work item with a container. If steps are combined sensibly or if a method call can be replaced with a simple attribute query, this automatically improves the performance.

Performance 3 - Administrator

- Archive regularly.
- Delete work items no longer required.
- Many incorrect work items/many deadlines -> if necessary, report to modeling

With a high system load, the work item header table can contain several million entries. Depending on the database and application server performances, you will need to archive sooner or later.

Many incorrect work items or many deadlines to be monitored can cause a high system load due to monitoring background jobs. You need to clarify with the workflow modeling team whether the workflows used can be further developed to improve the stability of areas prone to errors, and reduce number of deadlines of the processes.

Exercise 13: Archive and delete the work items that your user has created during the 3 days of the course.

Exercise Objectives

After completing this exercise, you will be able to:

- Archive and delete completed work items
- Display the new technical log of archived workflow instances

Business Example

You have been using workflow for some time, and you want to archive and delete completed workflow instances from the runtime tables.

Task:

Archive and delete

1. Archive the work items that your two users completed within three days.

   Display the work items after you archive them.

2. Use the report RSWWWIDE to delete all the work items that are currently in your user's Workplace.

Hint: Set the "Display list only" indicator, and then execute the report. If the list only contains your work items, remove the indicator for the second report run and actually delete the work items.

Once you have deleted the work items, you will not be able to retrieve them.

Solution 13: Archive and delete the work items that your user has created during the 3 days of the course.

Task:

Archive and delete

1. Archive the work items that your two users completed within three days.

   Display the work items after you archive them.

   a) **Note that you must consult with your external auditor and the relevant departments in your company to arrange when exactly you may delete the work items from the database.**

      a. Call SAP Data Archiving:

         Tools → Business Workflow → Development → Administration → Workflow Runtime → Reorganization → Archive Work Item (SWW_SARA)

      b. Choose the menu option “Write”

         Enter the variant name **GRP_##**.

         Choose “Maintain”

         In the **Creation Date** field, enter the date of the first day of the course and then enter your two course users in the **Actual Agent** fields.

         Choose Attributes and enter **GRP_##** in the Description field.

         Enter **GRP_##** in the comments for the “archiving run”.

         Then choose “ Immediately”.

         Choose “Save”.

         Then choose “Back”.

      c. Enter a start time for the archiving program.

         Choose “Start Time”.

         Then choose “ Immediately”.

         Choose “Save”.

Continued on next page

d. Enter the spool parameters

Choose “Spool parameters”.

Select the printer **lp01**.

Choose “Continue” to exit the screen.

The parameters are maintained (green traffic light symbol)

Choose “Execute” to start the archiving run.

e. Choose the “Job” icon to check on progress.

You can see that a write job and a delete job have been scheduled.

You can check the run using the job log and the spool display.

f. Display the archived work item.

Return to the basic screen of transaction SWW_SARA.

Choose “Read”.

Choose “Execute” – The existing archiving runs are displayed.

Select the run with the comment for your group **GRP_##**.

Choose “Continue” to start the Execution.

The content of your archive file is displayed.

2. Use the report RSWWWIDE to delete all the work items that are currently in your user’s Workplace.

Hint: Set the “Display list only” indicator, and then execute the report. If the list only contains your work items, remove the indicator for the second report run and actually delete the work items.

Continued on next page

Once you have deleted the work items, you will not be able to retrieve them.

a) Use the report RSWWWIDE to delete all the work items that are currently in your agent's inbox.

To delete work items directly in a user's Workplace, these work items must have at least the status reserved.

a. Call your Inbox as the processor.

Select the work items with the status *Ready*, and choose *Reserve*.

b. Then go to the transaction for explicitly deleting work items

Tools → Business Workflow → Development → Administration → Workflow Runtime → Reorganization → Delete Work Item (SWWL)

c. Delete all entries.

Under “Agent”, enter your agent user.

Choose “Execute”.

Choose “Select All”

Choose “Delete Work Item”

d. Return to the user’s Workplace and make absolutely sure that all work items have been deleted.

Lesson Summary

You should now be able to:

- Archive work items
- Delete work items in test environments
- List the different performance approaches

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Understand how the workflow runtime system works
- Describe status transitions in the workflow
- Use a function module to start a workflow directly, without using an event
- Improve the performance of SAP Business Workplace.
- Manage large lists of work items.
- Reorganize the extensive use of dynamic columns and control the grouping functions in large worklists.
- Describe the different forms of deadline monitoring.
- Explain special situations for deadline monitoring.
- Restart a workflow if an error has occured.
- Archive work items
- Delete work items in test environments
- List the different performance approaches

Test Your Knowledge

1. If you call workflows directly using function modules, you must enter a workflow container and full it with the leading object.

   Determine whether this statement is true or false.

   □ True
   □ False

2. You use a macro to fill the contained with the leading object, and then you subsequently convert the container into a persistent form.

   Determine whether this statement is true or false.

   □ True
   □ False

3. Errors must always be solved on the level at which they take place. The workflow must be restarted at the top level.

   Determine whether this statement is true or false.

   □ True
   □ False

4. The “Delete work items” function deletes work items without taking into account their status or the consequences for the database.

   Determine whether this statement is true or false.

   □ True
   □ False

5. It is possible to archive work items in any status.

   Determine whether this statement is true or false.

   □ True
   □ False

6. You can display a technical log for archived workflow instances.

   Determine whether this statement is true or false.

   □ True
   □ False

Answers

1. If you call workflows directly using function modules, you must enter a workflow container and full it with the leading object.

   Answer: True

   If a workflow is started using an event, the leading object from the event container is entered in the binding. If the workflow is started using a function module, this binding must also be guaranteed.

2. You use a macro to fill the contained with the leading object, and then you subsequently convert the container into a persistent form.

   Answer: True

   A container must be in persistent form before you can use it in a workflow instance. For filling the container, however, you still need a BOR macro that expects access to the container using a runtime handle.

3. Errors must always be solved on the level at which they take place. The workflow must be restarted at the top level.

   Answer: True

   To ensure the restart is processed correctly, do not start it in the work item in which the error occurred, use the *Restart After Error* administrator function instead.

4. The “Delete work items” function deletes work items without taking into account their status or the consequences for the database.

   Answer: True

   Because deletion requires reorganization of the database, it can only be performed in test systems.

5. It is possible to archive work items in any status.

   Answer: False

   Only work items with the status *ended* or *logically deleted* can be archived. The archiving takes place in the *Archive Development Kit* and the *Work item* archiving object.

6. You can display a technical log for archived workflow instances.

   Answer: True

   As of release 4.6, it is possible to display the workflow log by entering a transaction outside of the workflow application.

Internal Use SAP Partner Only

Course Summary

You should now be able to:

- Identify all points in the workflow system at which programming is possible or necessary
- Create your own object types and enhance existing object types
- Create rule function modules
- Program events
- Create check function modules and receiver type function modules
- Use AP to create work items
- Administrate the workflow runtime system

Appendix 1

Business Scenario

Overview:

This section deals with the business scenario for the example workflow for the course.

Exercise scenario BIT610

Motor Sports International (MIS) works successfully with SAP ERP 6.0. The company now wants to create a material checking process using SAP Business Workflow.

The workflow process at MSI

Triggering:

The process always starts when a user changes the Old Material field in a material master record. Analysis at MSI has shown that the OldMaterial field is not being used, and can therefore be used to notify designers of relevant changes.

The result of the change triggers a workflow event (for example, OldMaterialChanged), by writing a change document. The workflow event then initiates one of two workflow processes.

The definitions of the two workflows are identical, but they use different rule resolutions. The material type determines which workflow starts. The material type 'FERT' triggers the first workflow, all other materials trigger the second workflow.

An additional check ensures that the workflows only run for certain material numbers, and that the last changer of the material master was one of the relevant group users.

In the workflow itself, the material master is displayed in change mode to the user responsible for materials and production. This user adjusts the production process in other areas to suit the change. When the user has completed this, they change the Basic Material field of the corresponding material master. Studies at MSI have also shown that the Basic Material field is otherwise not used.

If the responsible user has not completed the work within 5 minutes, the superior is informed.

This change to the Basic Material field triggers and event that ends the task “Display Material to be Changed” and allows the workflow to continue running.

In the next step, a user receives the material master for inspection, which also notifies them that the material can return to production.

If the material is not a raw material, the workflow forwards the work item to the user who originally changed the material. For a raw material, the superior of the user to make the last change is notified.

Internal Use SAP Partner Only

Appendix 2

“Organizational Data” Rule Type

Overview:

This section provides information about the "Organizational data" rule type.

SAP Organizational Objects

Abbreviation	Long Text	First Mention
API	Application Programming Interface	Unit 3
BI	Background Work Item	Unit 6
BOB	Business Object Builder	Unit 2
BOR	Business Object Repository	Unit 2
ECA	Event Condition Action	Unit 5
EM	Event Manager	Unit 3
FM	Function Module	Unit 2
FG	Function Group	Unit 6
FI	Workflow Work Item	Unit 6
LH	Runtime Handle	Unit 2
LIS	Logistics Information System	Unit 7
LUW	Logical Unit of Work	Unit 5
POR	Persistent Object Reference	Unit 2
RFC	Remote Function Call	Unit 5
tRFC	Transactional RFC	Unit 5
WF	Workflow	Unit 1
WI	Work Item (General)	Unit 6
WIM	Work Item Manager	Unit 3
WIS	Workflow Information System	Unit 7

Figure 72: Abbreviation List

Addressing Using SAP Organizational Objects

- Using object-based attributes for agent determination
- Linking the Business Object Repository and the organizational structure
- No programming required

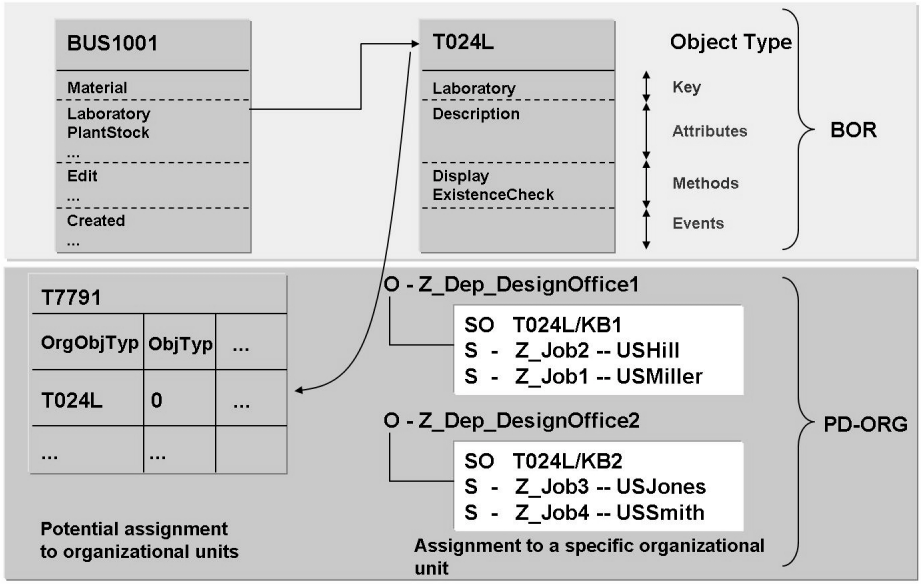


Figure 73: Defining an SAP Organizational Object

1. Definition of the object type to be used subsequently as an SAP organizational object (in this case, T024L) to determine the agent assignment.
2. Definition of an attribute that is mapped on this object type for the object type used in the workflow in which an agent assignment is required (in this case, the laboratory attribute for the object type BUS1001, because the workflow in question works with the BUS1001 object type or one of its subtypes).
3. Definition of the BOR object type from (1) as an SAP organizational object in PD ORG. In this step, the potential links in the organization model are also determined using an entry in table T7791 (in this case, T024L is only to be linked with organizational object type O, that is, organizational units).
4. Creation of a link between an instance of the SAP organizational object and an organizational unit from the organization model (this means linking the KB1 laboratory to the Z_Dep_DesignOffice1 organizational unit and the KB2 laboratory with C_Dep_DesignOffice2).

SAP Organizational Object in Role and Step Definitions

- Role definition with reference to SAP organizational object type
- Step definition with reference to this role
- Binding must write the SAP organizational object to the role container.

Binding in the step definition between the workflow container and the (implicit) role container: The attribute containing the SAP organizational object (in this case, BUS1001.Laboratory) must be written in the role container.

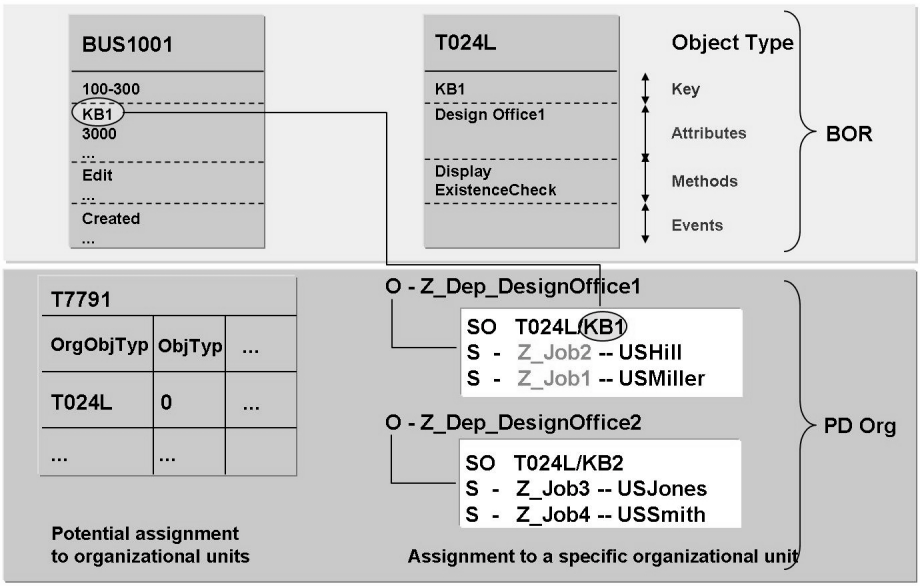


Figure 74: Evaluation of the SAP Organizational Object at Runtime

1. The role evaluation determines the organizational units (in this case, Z_Dep_DesignOffice1) that are linked to the instance of the SAP organizational object (in this case, KB1).
2. All object instances linked to this organizational unit are returned as the result of role evaluation (in this case, Z_Job1 and Z_Job2).

Addressing Using SAP Organizational Objects

- In the Business Object Repository define an object type for the SAP organizational object with a key, the attribute “Description” and the method “ExistenceCheck”.
- Create an attribute for the object type being used, with reference to the defined object type.
- Enter the SAP organizational object in table T7791.
- Include the SAP organizational object in the organizational plan.
- Define a role and use it in the workflow step definition.

Table T7791 is maintained using transaction SM30. In this transaction, you can specify the organizational units to which only one link can be made.

Maintaining the organizational plan from the workflow development environment.

Appendix 3

How to Troubleshoot

Overview:

This section deals with troubleshooting and provides a program example of a customer-specific work item display.

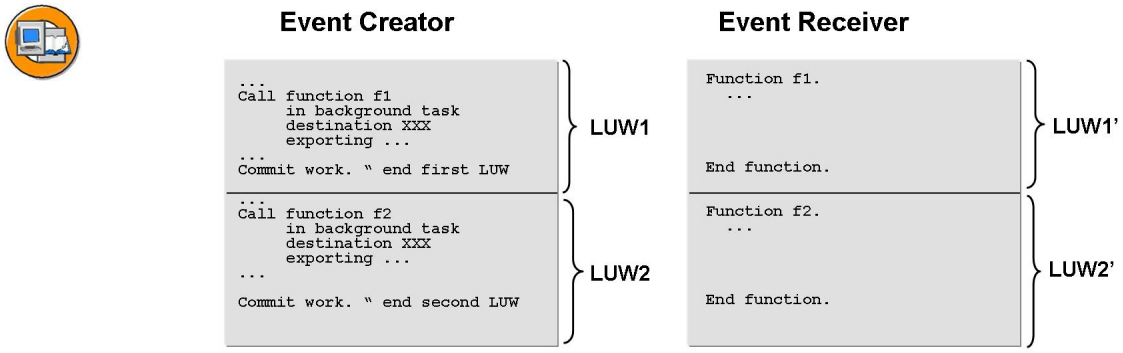


Figure 75: tRFC Packet Processing - Case 1

The event generator has one LUW for every event, and each event has exactly one receiver

=> each event receiver also has its own LUW.

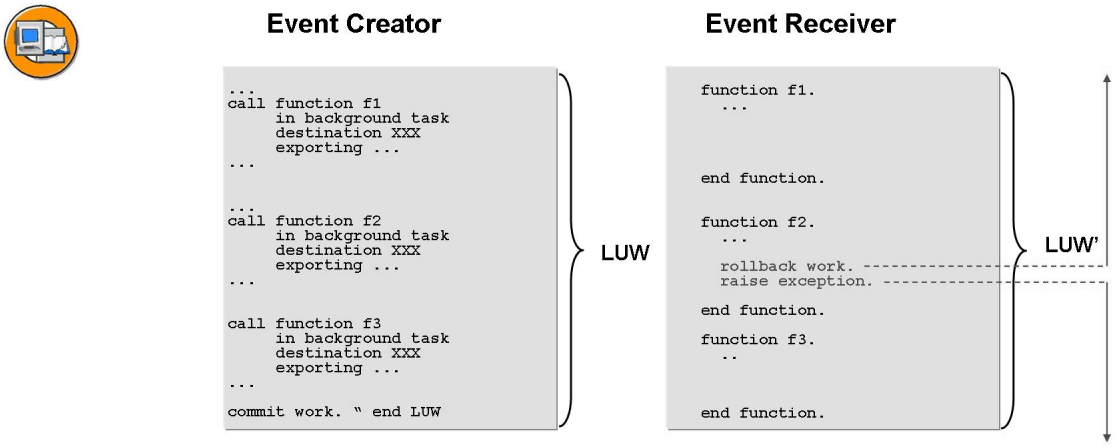


Figure 76: tRFC Packet Processing - Case 2

Multiple events result from one LUW

=> all event receivers are also in one LUW

=> any rollback work or raise events within a receiver also affect all other receivers

.

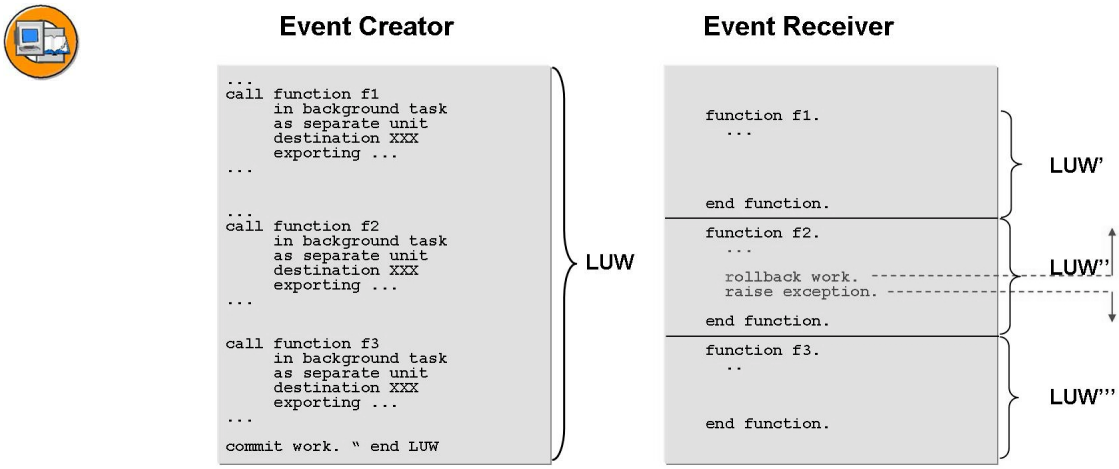


Figure 77: tRFC Packet Processing - Case 3

Multiple events result from one LUW, but the receivers are started with the addition ‘AS SEPARATE UNIT‘.

=> each event receiver has its own LUW

=> Any rollback work or exits only take place locally in the current receiver.

The event manager and package processing

- 3.0F
  - All receiver function modules run independently of each other.
  - Error handling as in 3.0D
- 3.0D
  - Error handling: Deactivate linkage and send an e-mail to the workflow system administrator.
  - Workflow receiver function modules no longer trigger exceptions.
- 3.0C
  - All workflow receiver function modules trigger exceptions.
  - No special error handling.

The addition “AS SEPARATE UNIT” is available as of SAP Business Workflow 3.0F.

Work Queue Definition

- A work queue consists of
  - A list of tuples (object instance, task)
  - A list of agents for the tuples (optional)
  - A default task (optional)
- A specific work item type

The object instances can be of the same or different object types. Semantically, the list is regarded as one set (that is, duplicates are permitted).

A task can be specified for each object instance. If a task is not specified, the default task is used. Only dialog or synchronous background tasks are permitted.

The agents of the object sets are derived either from the agent list or, if this list is empty, the organizational assignment of the relevant task.

The status is handled separately for each individual object set and for the work queue as a whole.

As the work queue is a special work item type, it has all of the properties of a work item, such as:

- Deadline monitoring
- Excluded agents
- Terminating events
- Visibility in the Business Workplace

Criteria for Using Work Queues

- Tasks on different types of objects
- Background task(s) on a set of objects
- Actions both on the set as a whole and on each individual element in the set
- Status monitoring both for the set as a whole and for each individual element in the set
- Detailed error handling

The alternatives for each possible scenario must be examined. Not all the above criteria must be fulfilled to use work queues.

There are certain advantages with background tasks for a set of objects, because the methods belonging to the tasks are executed directly (that is, without creating a background work item).

Background task on a set of objects to be executed together

Actions on the work queue as a whole and on each individual object

Error handling required

For example: Work queues with mass processing in Asset Accounting

Work Queues Outside a Workflow

- Explicit agent assignment required
- Processing in the Business Workplace
- Restricted functions

Because the work queue as such is not assigned to a task, agents must be assigned explicitly when the work queue is created (using the AGENTS table in the SWZ_AI_CREATE function module).

The work queue then appears (as a dialog work item) in the recipients' Business Workplace.

Business Workplace contains some additional menu items to edit work queues, for example, **Release** or **Display object list**. However, the full functions for work queues are available only as an API or as methods of the ARCHLISTWI object type.

Work Queues Within a Workflow

- Use of the object type ARCHLISTWI

  (or a subtype)
- Methods for all work queue actions
- All workflow definitions
- Sophisticated error handling

The basic functions for work queues are incorporated in the ARCHLISTWI object type. Application or customer-specific enhancements must be implemented in a subtype (for example, AM_AI).

Each instance of a work queue can be used only either inside or outside a workflow.

Processing a work queue inside a workflow allows user-defined process definitions (for example, multilevel release of work queue) or sophisticated error handling (for example, handling object sets with errors in a new work queue).

SAP recommends that you use work queues within a workflow.

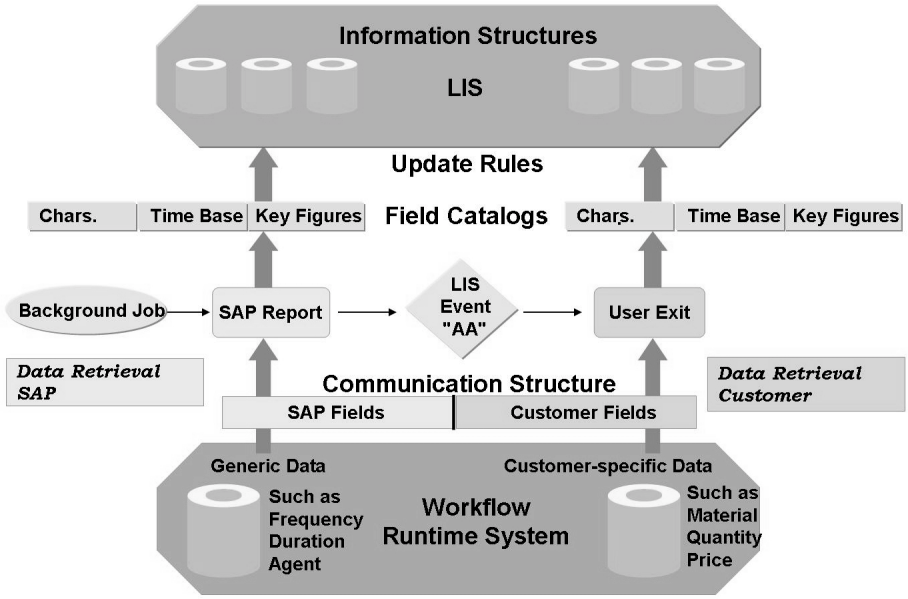


Figure 78: Architecture

The communication structure and Logistics Information System (LIS) event form the LIS inbound interface for external data.

Data retrieval is carried out using the RMCADATA batch job which triggers the LIS event and also fills the communication structure. This background job must be scheduled explicitly. When you schedule the job you must ensure that no data is transferred twice when you set the evaluation period.

The LIS generates a function group for the LIS event. The event is actually triggered by calling a particular function module in this function group and transferring the internal tables previously filled.

The communication structures (for generic data), data retrieval program and function group are supplied by SAP.

A user exit is provided to allow customers to fill their own fields in the communication structure.

Field catalogs, information structures and update rules are defined in LIS Customizing (see online documentation or documentation in the course LO930: Logistics Information System)

You can access the Workflow Information System using the reporting in the workflow development environment or directly using transaction MCA1.

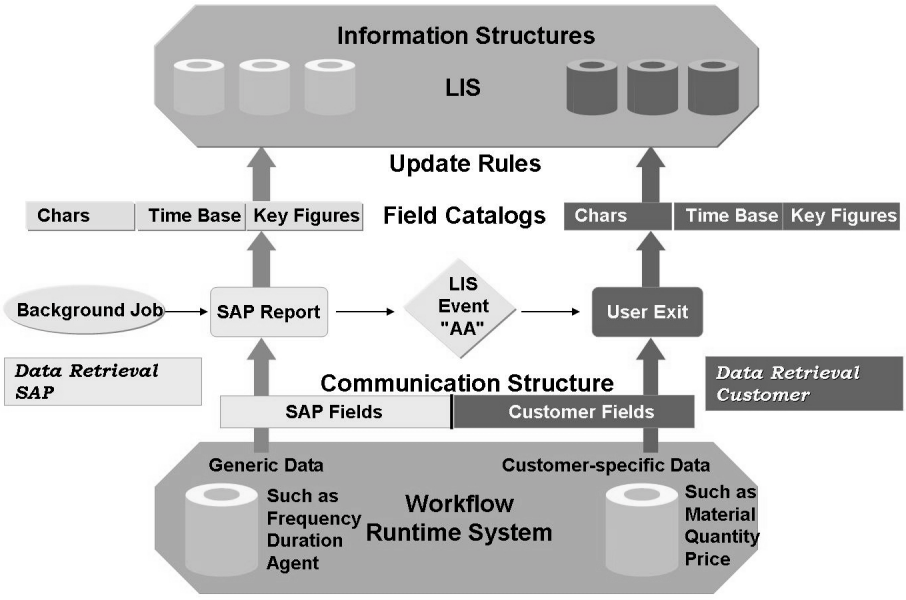


Figure 79: Architecture - Customer Enhancements

Application number 20 is reserved for WIS enhancements within LIS.

Customer Analysis Steps

- Enhancing communication structure
- Programming and activating the user exit
- Creating field catalogs
- Creating info structures
- Creating update rules

These five steps must be performed in sequence.

Work Item Debugging

- Workflow log evaluation
- Workflow diagnosis tool as starting point for active troubleshooting
- Workflow trace evaluation
- Evaluation of event trace, use of event queue

The workflow log is always written and cannot be deactivated. The workflow log is attached to the database commit and writes to the database.

The workflow trace must be activated explicitly. It does not depend on the database commit and writes to the file system of the application server.

The workflow trace is useful if problems occur in background work items or in the binding.

The event trace (must be activated explicitly) and the event queue (must be initiated globally and activated individually for each relevant linkage) can be used to troubleshoot problems with the event communication.

Workflow Diagnosis Tool

- Access to workflow and work item debugging
  - Verification of workflow Customizing
  - Problem: Task not started
  - Problem: Workflow hangs
  - Test environment (special test tools)
- Context-sensitive diagnosis of workflow instances

Access is made either using utilities in the workflow development environment or directly using transaction SWUD.

Transaction SWUD can execute all the tools and procedures, which are described later in detail.

Transaction SWUD can also be used to investigate problems that occur when events are triggered.

Workflow Log

- Record the workflow execution process:
  - Work items: When created/processed/done
  - Process: branches, loops, start/end of forks, subworkflows
  - Incorrect/deleted/aborted steps
- Contains cross-reference and additional information:
  - Agents of all steps
  - Containers of all steps and of the process
  - Jump to work item display
  - Jump to graphical log
- Displays the current status of workflow execution
- First entry point for troubleshooting

The workflow log is the central information source for the process run. The log contains messages for all components (success and error messages) of the workflow systems, in particular, the workflow manager and the work item manager.

Initially, only temporary log entries are written to a buffer to ensure that the commit logic of the caller is not destroyed. At particular points in the flow logic, the SWW_WI_LOG_FLUSH function module is called to write the buffer contents on the database.

If successful, a log entry is written for all activities that can be triggered by a user (including) using the user interface (SWW_WI_LOG_WRITE_SUCCESS function module). If unsuccessful, a log entry is **always** written so that the call hierarchy can be traced to where the error occurred (SWW_WI_LOG_WRITE_EXCEPTION function module).

If work items are created using the work item manager API (or more generally, the work items are controlled using the work item manager API) and the commit is controlled by the creator (that is, the DO_COMMIT parameter was set to SPACE), the creator must call the SWW_WI_LOG_FLUSH function module before the commit work. In all other cases, the workflow runtime system initiates the call.

Workflow Trace

- Detailed description of all technical activities for workflow execution
- Can be restricted to a particular user/workflow definition
- Stored in the file system of the application server

  (independently of the DB commit)
- Must only be activated for immediate troubleshooting (performance/memory space in the file system)
- Enhanced troubleshooting

This is suitable for a very high level of detail for technical support, in particular for detecting binding problems.

Troubleshooting - Starting a Workflow

- Workflow does not start
  - Check the Customizing settings.
  - Check the event trace for initiator.
  - Check the event linkage (exists and active).
  - Check the tRFC log under initiator's login.
  - Check the tRFC log under user's login from WORKFLOW_LOCAL_<Clnt>.
  - Check the workflow definition (active version).

You must carry out a consistency check for Customizing before you start using SAP Business Workflow for the first time.

Hint: Incorrect linkages are deactivated automatically and an e-mail is sent to the workflow system administrator. This behavior can be individually adapted for each linkage. You can use the event queue to adjust the default system behavior when errors occur (for more information, see the unit “Event Definition and Implementation”).

Troubleshooting - Work Item Cannot Be Found

- Work item is not in the Business Workplace
  - Check whether the work item was created correctly (see above).
  - Search for the work item using the work item selection.
  - Check the agent assignment using the task profile and the step definition.
  - Check the rule definition and/or the rule function module.

You might have to refresh the buffer for organizational assignments.

Troubleshooting - Work Item Not in End Status

- Work item cannot be completed
  - Check the workflow log.

    Application or temporary error
  - For a background step, carry out a check using the report RSWWERRE.
  - Check whether the workflow system administrator has received an e-mail.
  - For an asynchronous step, check the event instance linkage.
  - Initiate the test execution.
  - Check the container contents.

A breakpoint should be set in the method implementation before the “test execution”.

If the instance linkage table is filled correctly, for test purposes you can also trigger the terminating event using the workflow development environment.

Troubleshooting - Workflow Hangs

- Work item is completed but workflow does not continue
  - Check the workflow log.

    Preceding step completed correctly?

    Status transition implemented correctly?

    No waiting for event in a parallel branch?

    Work item for next step created correctly?
  - Check whether the step task is locked against instantiation.
  - Check whether the workflow system administrator has received an e-mail.
  - Find the created work item (see above).

Flow Logic of the Screen

- Initialization of data for PBO
- Status-dependent refreshing of the data for PBO

  (optional)
- Reaction to OK Code for PAI
  - Either direct processing on subscreen
  - or OK code transferred to the superordinate WI standard screen

The programming is carried out using macros from the <widisp> include. This include must be integrated into the module pool.

Full documentation on the macros is available in the online documentation.

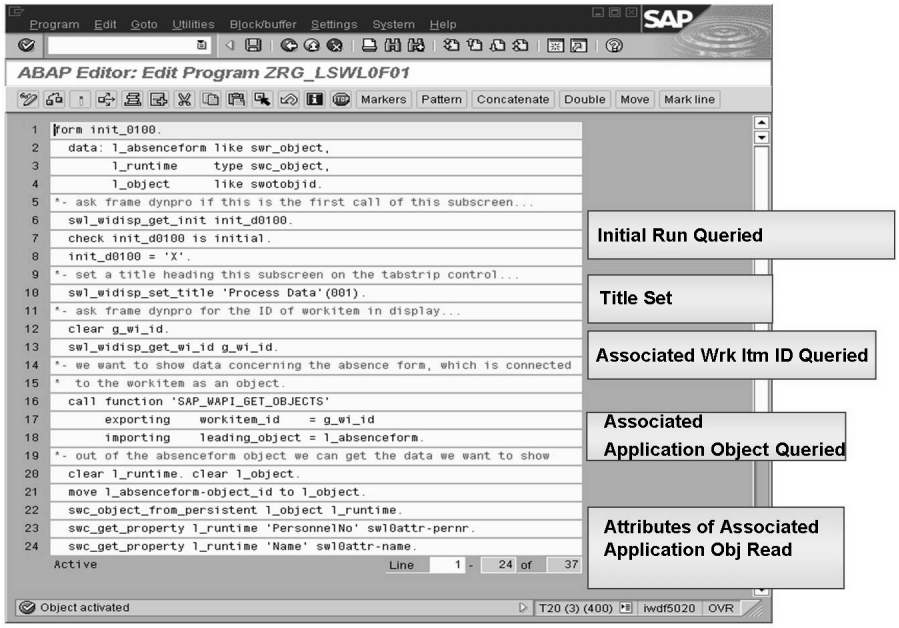


Figure 80: Initializing the Screen for PBO

The SAP_WAPI_* function modules implement the WfMC Interface 2 and are available in the SWRC function group.

SAP_WAPI_GET_OBJECTS returns the leading object as a persistent object reference (POR). Therefore, before the BOR macro is called, a transformation to a runtime handle must take place by calling SWC_OBJECT_FROM_PERSISTENT.

The command include <cntn01> must be incorporated to use the BOR macro.

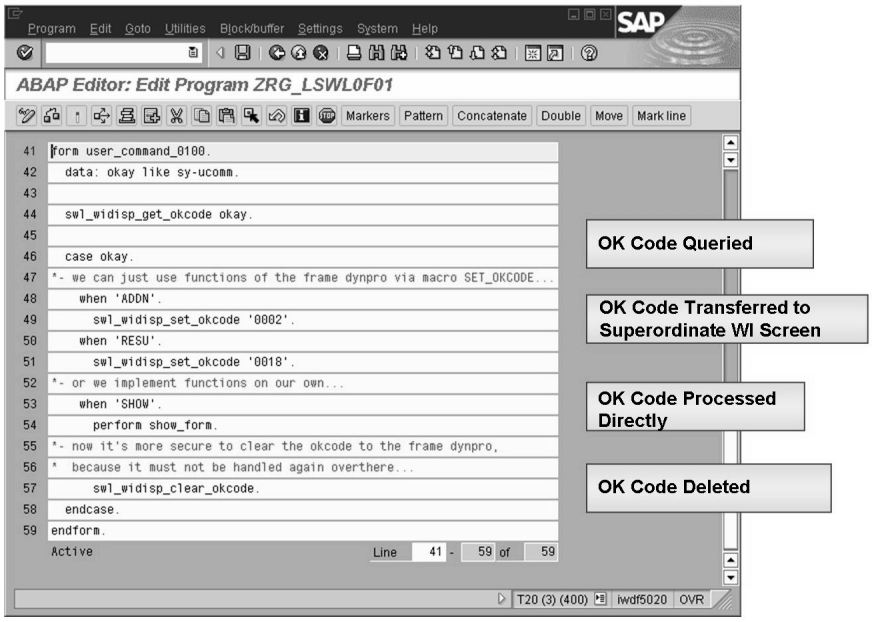


Figure 81: Reaction to OK Code for PAI

The complete list of IDs for the OK code transfer is in the online documentation.

Feedback

SAP AG has made every effort in the preparation of this course to ensure the accuracy and completeness of the materials. If you have any corrections or suggestions for improvement, please record them in the appropriate place in the course evaluation.

Internal Use SAP Partner Only