



# **BC400**

## **ABAP Workbench Foundations**

### **PARTICIPANT HANDBOOK INSTRUCTOR-LED TRAINING**

Course Version: 18  
Course Duration: 5 Day(s)  
e-book Duration: 24 Hours 30 Minutes  
Material Number: 50148766

## SAP Copyrights and Trademarks

© 2018 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. Please see <http://global12.sap.com/corporate-en/legal/copyright/index.epx> for additional trademark information and notices.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors.

National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP SE or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP SE or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon in making purchasing decisions.

# Typographic Conventions

American English is the standard used in this handbook.

The following typographic conventions are also used.

This information is displayed in the instructor's presentation	
Demonstration	
Procedure	
Warning or Caution	
Hint	
Related or Additional Information	
Facilitated Discussion	
User interface control	Example text
Window title	Example text



# Contents

vii	Course Overview
1	Unit 1: Flow of an ABAP Program
2	Lesson: Describing the Processing of ABAP Programs
16	Unit 2: ABAP Workbench
17	Lesson: Introducing the ABAP Workbench
26	Lesson: Organizing ABAP Development Projects
33	Lesson: Developing ABAP Programs
44	Lesson: Finalizing ABAP Development Projects
55	Unit 3: Basic ABAP Language Elements
56	Lesson: Defining Elementary Data Objects
68	Lesson: Using Basic ABAP Statements
78	Lesson: Analyzing Programs with the ABAP Debugger
89	Unit 4: Modularization Techniques in ABAP
91	Lesson: Explaining Modularization
97	Lesson: Defining and Calling Subroutines
109	Lesson: Calling Function Modules
118	Lesson: Creating Function Modules
121	Lesson: Describing Business Application Programming Interfaces (BAPIs)
126	Lesson: Calling Methods of Global Classes
140	Lesson: Creating Global Classes and Static Methods
144	Lesson: Using Local Classes
155	Unit 5: Complex Data Objects
156	Lesson: Using Structured Data Objects
162	Lesson: Using Internal Tables
185	Unit 6: Data Modeling and Data Retrieval
187	Lesson: Explaining Data Models
196	Lesson: Retrieving Single Database Records
203	Lesson: Retrieving Multiple Database Records
207	Lesson: Describing Other Aspects of Database Access
218	Lesson: Implementing Authorization Checks

231	Unit 7:	Classic ABAP Reports
232		Lesson: Implementing ABAP Lists
236		Lesson: Implementing Selection Screens
244		Lesson: Implementing Events of ABAP Reports
254	Unit 8:	Screens
255		Lesson: Creating Screens
274		Lesson: Creating Input and Output Fields
277		Lesson: Implementing Data Transport
287	Unit 9:	SAP List Viewer
288		Lesson: Using the SAP List Viewer
303	Unit 10:	Web Dynpro ABAP
304		Lesson: Describing Web Dynpro ABAP
312		Lesson: Implementing Navigation in Web Dynpro
322		Lesson: Implementing Data Transport in Web Dynpro
333	Unit 11:	Program Analysis Tools
334		Lesson: Improving the Quality of ABAP Code with the Code Inspector
342	Unit 12:	ABAP Development Tools for SAP NetWeaver
343		Lesson: Describing ABAP Development Tools for SAP NetWeaver
346		Lesson: Creating an ABAP Project in Eclipse
352	Unit 13:	SAP Standard Software Adjustments
353		Lesson: Adjusting the SAP Standard Software

# Course Overview

## TARGET AUDIENCE

This course is intended for the following audiences:

- Developer
- Development Consultant
- IT Support
- Project Manager



## UNIT 1

# Flow of an ABAP Program

Lesson 1

Describing the Processing of ABAP Programs

2

### UNIT OBJECTIVES

- Describe the SAP NetWeaver Application Server architecture
- Describe the execution of a dialog program by the ABAP runtime system

## Unit 1

### Lesson 1

# Describing the Processing of ABAP Programs

#### LESSON OVERVIEW

This lesson explains how a simple dialog program is executed by SAP NetWeaver Application Server (AS).

#### Business Example

Your task is to explain the SAP NetWeaver AS architecture and the execution of ABAP programs. For this reason, you require the following knowledge:

- An understanding of the SAP NetWeaver AS architecture
- An understanding of how a simple dialog program is executed by the ABAP runtime system

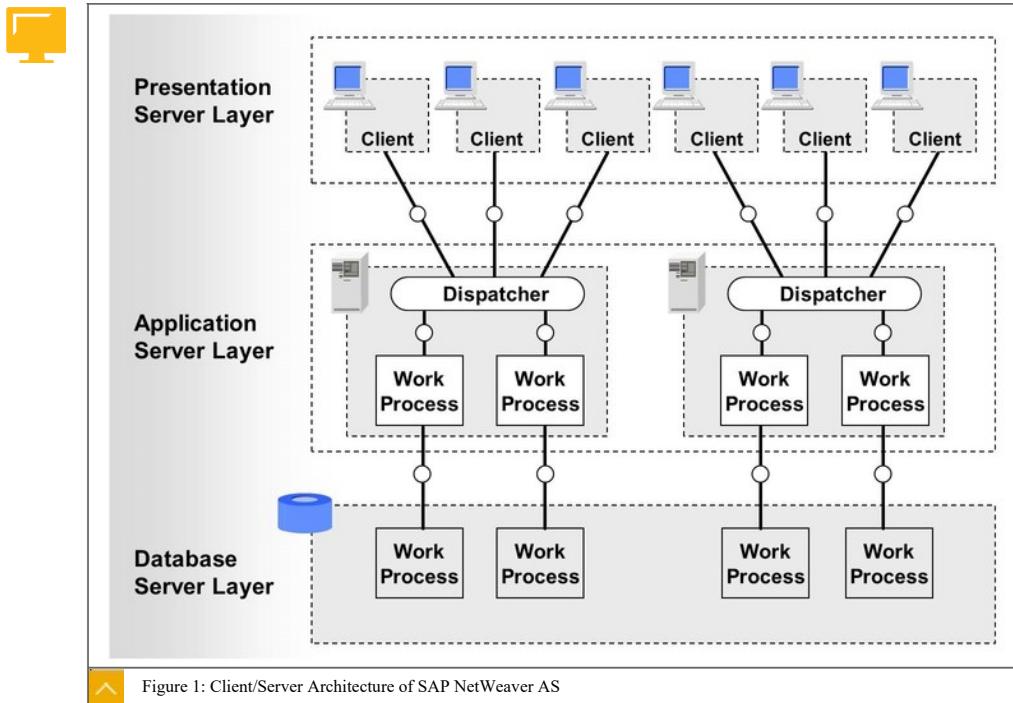


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the SAP NetWeaver Application Server architecture
- Describe the execution of a dialog program by the ABAP runtime system

### ABAP System Architecture



SAP NetWeaver AS has a modular architecture that follows the software-oriented client/server principle.

In SAP NetWeaver AS, presentations, application logic, and data storage can be assigned to different systems. This serves as the basis for the scalability of the system.

#### Architecture of SAP NetWeaver Application Server

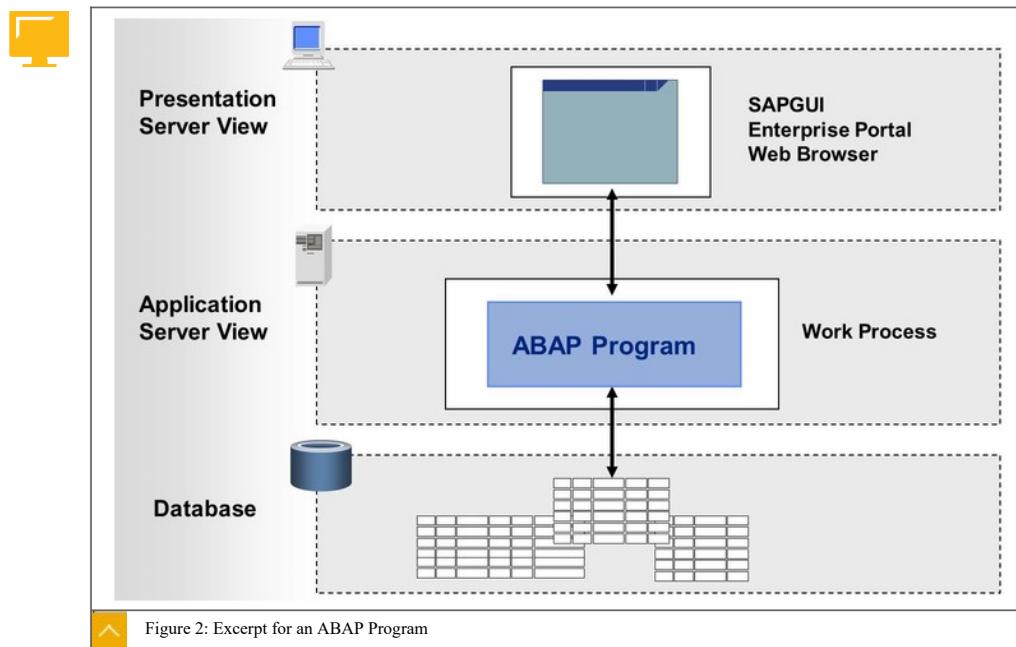
##### Architecture Layers

- The various layers in the architecture of SAP NetWeaver AS are as follows:
  - Database layer** is the lowest layer. At this layer, data is managed with the help of a Relational Database Management System (RDBMS). In addition to application data, it includes programs and the metadata that the SAP system requires for operation.
  - Application Server layer** is the intermediate layer. At this layer, ABAP programs, such as applications that SAP provides and any custom-developed applications, run on the application server. ABAP programs read data from the database, process it, and, if necessary, store new data in the database.
  - Presentation Server layer** is the highest layer. This layer contains the user interface where each user can access the program, enter new data, and receive the results of a work process.

The technical distribution of software is independent of its physical location on the hardware. Vertically, all layers can be installed on top of each other on the same computer or each layer on a separate computer. Horizontally, you can divide the presentation and application server

layer among any number of computers. The horizontal distribution of database components depends on the type of database installed.

Excerpt for an ABAP Program



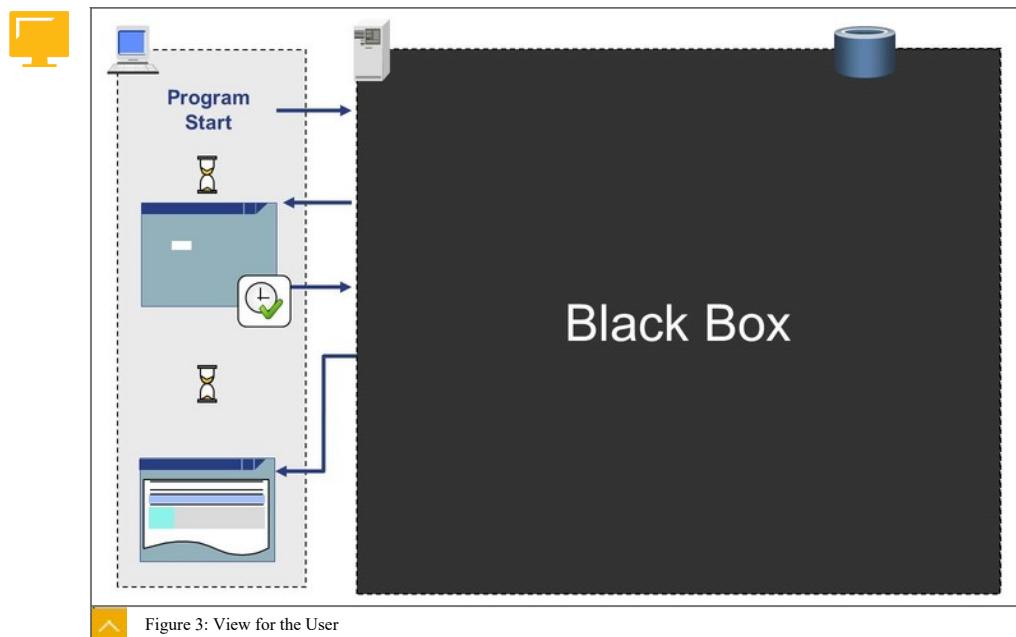
Note:

You can simplify the figure for most of the topics that will be discussed during this course. This course focuses on the interaction between one user and one ABAP program.

Understanding how to write an ABAP program is more important than understanding the exact processes involved in the application server. Therefore, you will be working with a simplified figure that does not explicitly show the dispatcher and the work process. Certain slides will, however, be enhanced to include these details whenever they are relevant to ABAP programming.

ABAP programs are processed on the application server. The design of user dialogs and database accesses is of particular importance when writing application programs.

## Overview of Program Flow



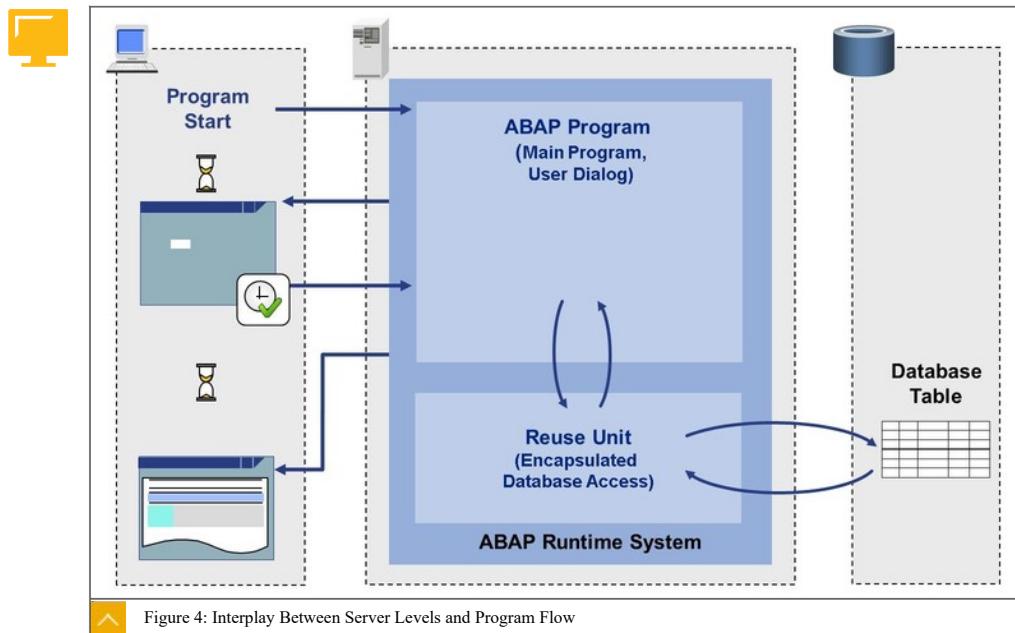
## Note:

This section explains the basic process that starts when an ABAP program is executed. To do this, you use a program with which the user enters an airline ID (for example, LH) and finds details of the airline displayed as a result.

The SAP system is like a black box to an average user. Users do not need to know the precise process flow of an ABAP program. An average user is only interested in the business process and how to enter or display data. The technical aspects of the program are of minor interest to an average user.

In contrast, developers need to understand the interplay between both the server levels and the process flow, when programs are executed, to develop their own programs.

### Interplay Between Server Levels and Program Flow

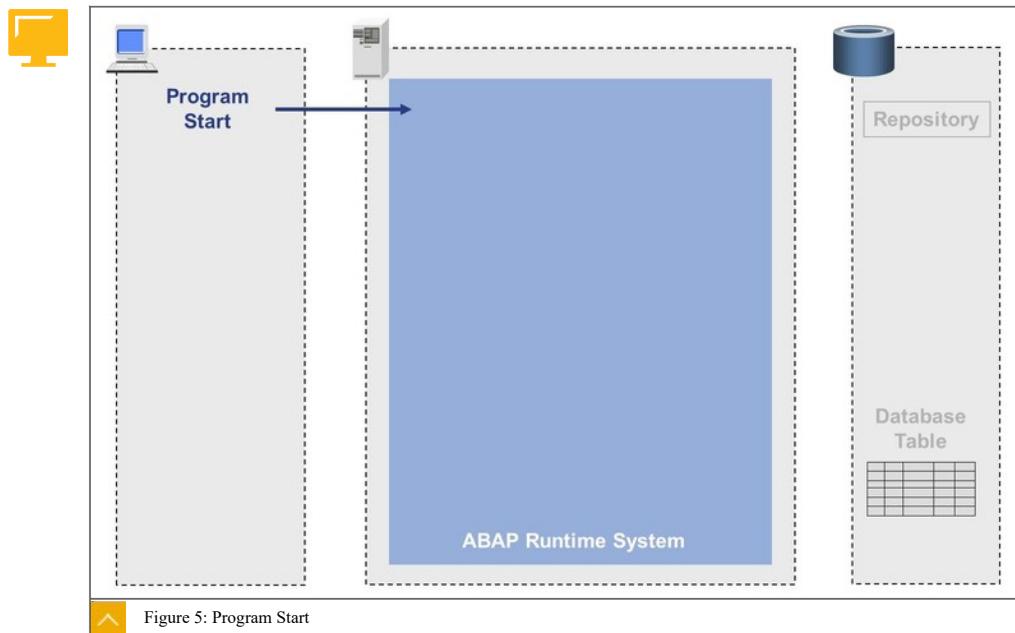


After the user performs an action, such as choosing **Enter**, a function key, a menu function, or a button, control passes from the presentation server to the application server.

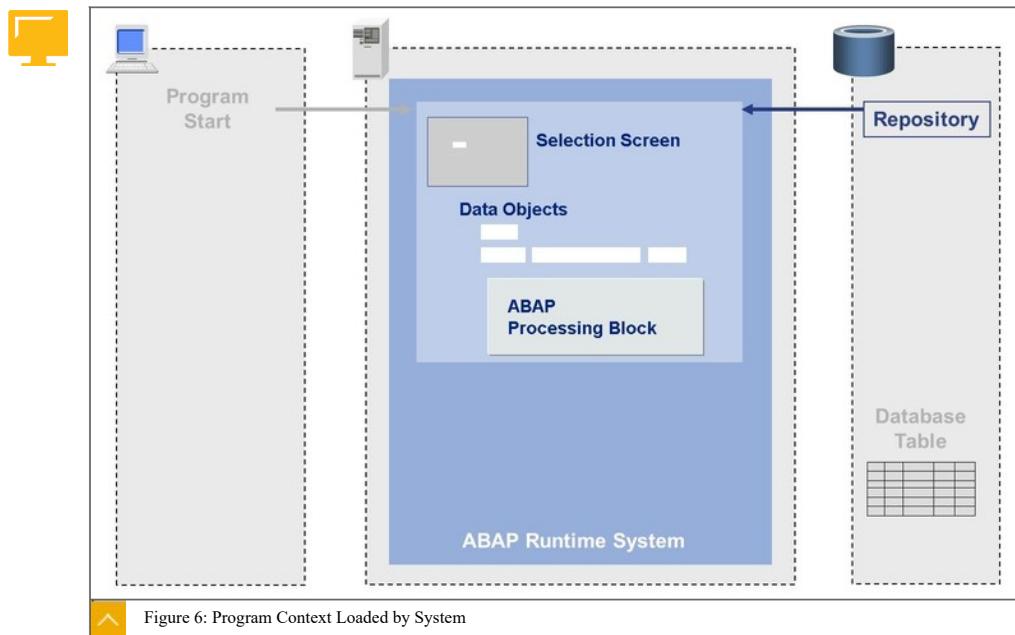
If another user dialog is triggered from the ABAP program, the system transmits the screen, and control is once again passed back to the presentation server.

A program is not made up of a single block, but of several units. This is known as modularization. You can use most of these modularization units in more than one program. This is why they are often termed as reuse units. A good program should have database accesses encapsulated in such reuse units. The reuse unit creates a division between the design of the user dialog and database accesses, making it possible to use the same database accesses for different user dialogs.

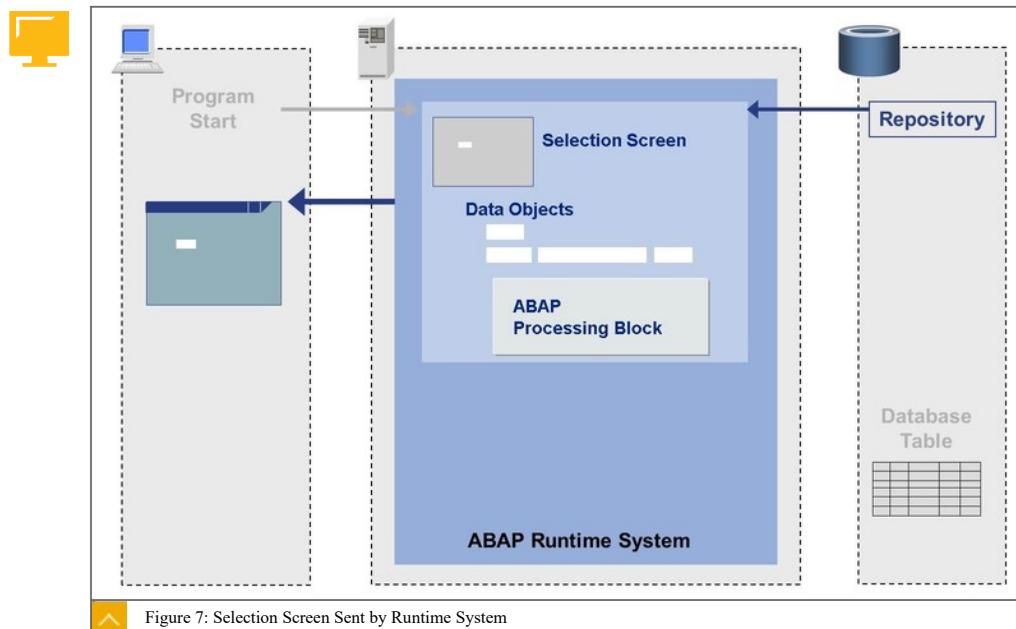
## Program Flow in Detail - Selection Screen and List



Whenever a user logs on to the system, a screen is displayed. From this screen, the user can start an ABAP program using the menu path.

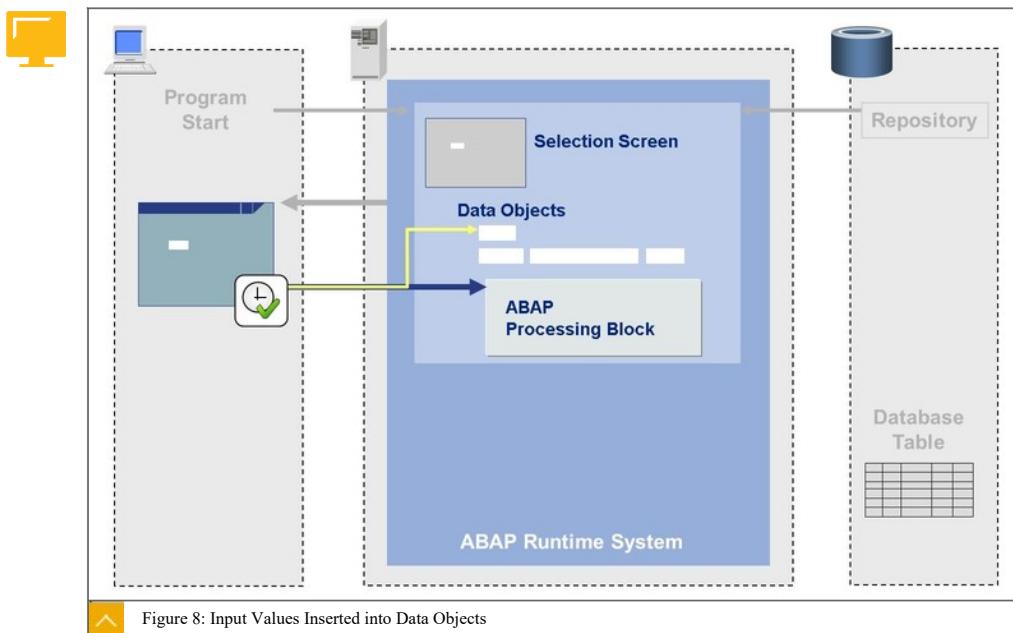


In this case, the system first loads the program context onto the application server. The program context contains memory areas for variables and complex data objects, information on the screens for user dialogs, and ABAP processing blocks. The runtime system gets all these program information from the Repository, which is a special part of the database. The sample program has a selection screen as the user dialog, a variable and a structure as data objects, and one ABAP processing block. The list used to display the data is created dynamically at runtime. The ABAP runtime system controls the subsequent program flow.



Because the program contains a selection screen, the ABAP runtime system sends it to the presentation server. The presentation server controls the program flow for as long as the user has not finished entering data in the input fields.

Selection screens allow users to enter selection criteria required by the program for it to continue.

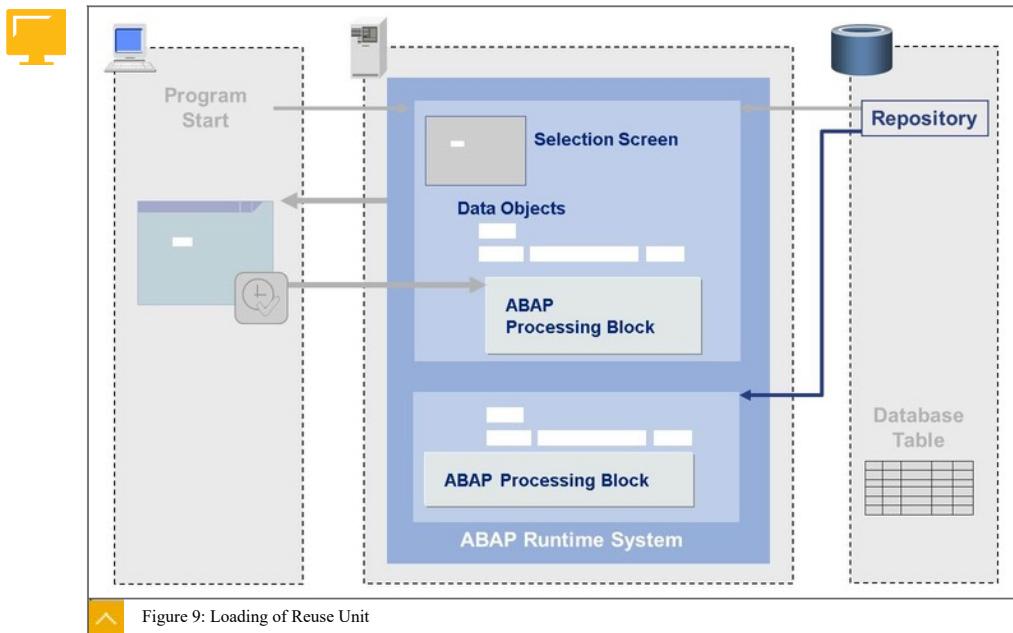


As soon as the user has finished entering data on the selection screen, they can trigger further processing of the program by choosing **Execute**.

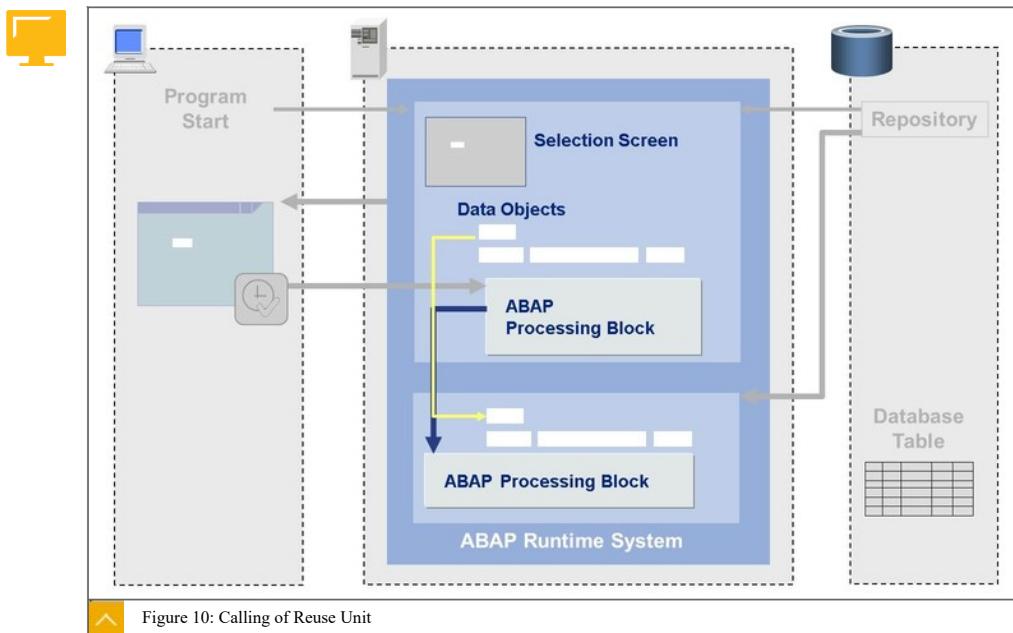
The entered data is automatically placed in its corresponding data objects in the program and the ABAP runtime system resumes control.

In the simple program example, there is only one ABAP processing block. The ABAP runtime system triggers sequential processing of this ABAP processing block.

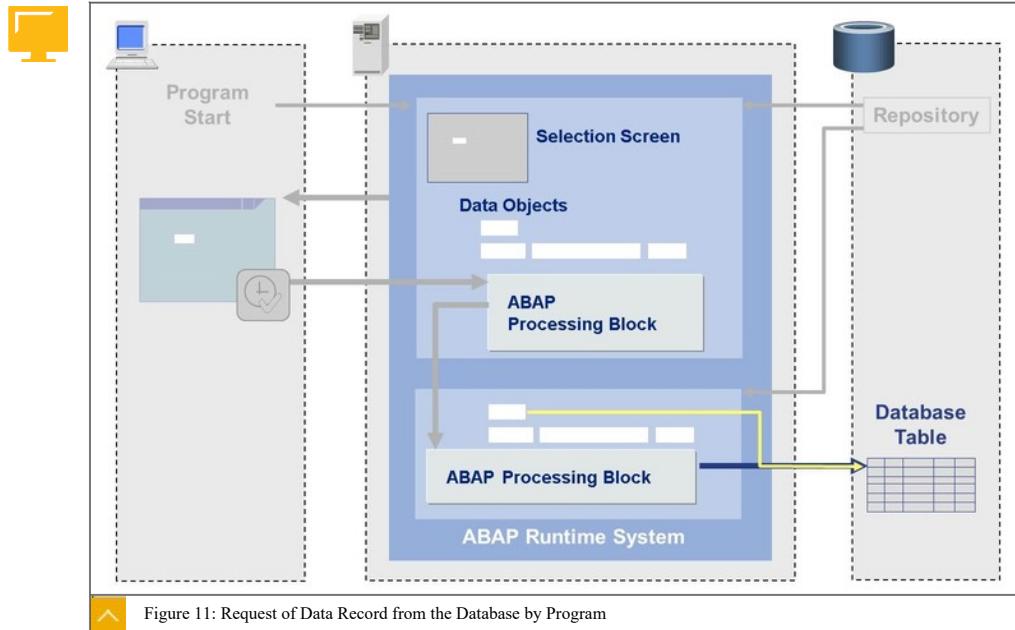
If the entries made by the user do not have the correct type, an error message is automatically triggered. The user must correct their entries.



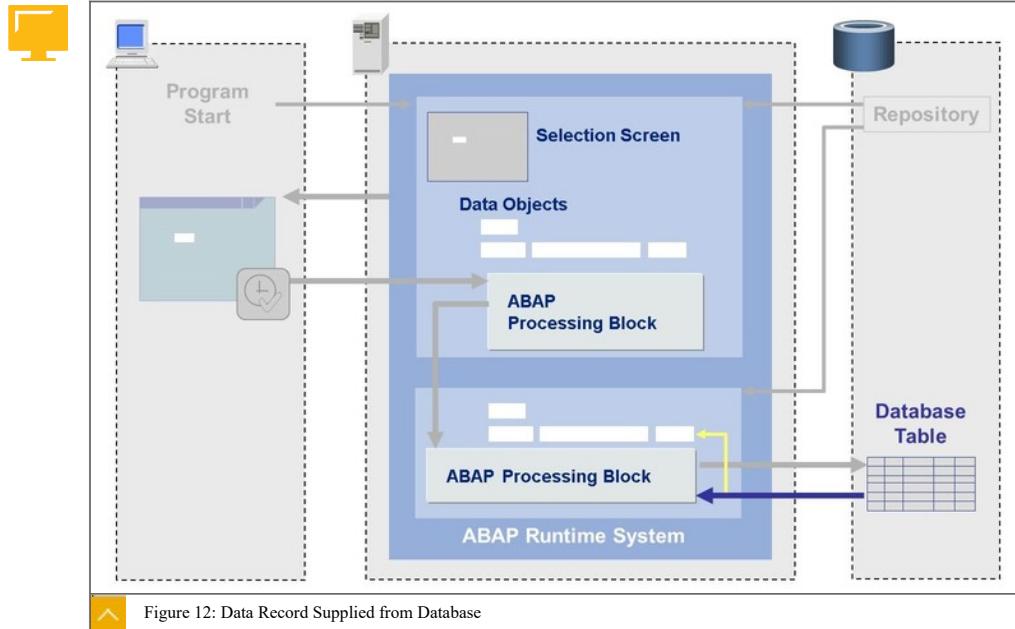
A reusable unit is called in the processing block that encapsulates the database access. The reuse unit is a special processing block in an individual program. The actual example shows a method in a global class. When the reuse unit is called, the program in which it is contained is also read from the Repository and loaded to the application server.



In the call, the required data is transferred to the called program, after which the reuse unit is executed. The execution is synchronous, which means that the calling program waits until the reuse unit has been processed completely.

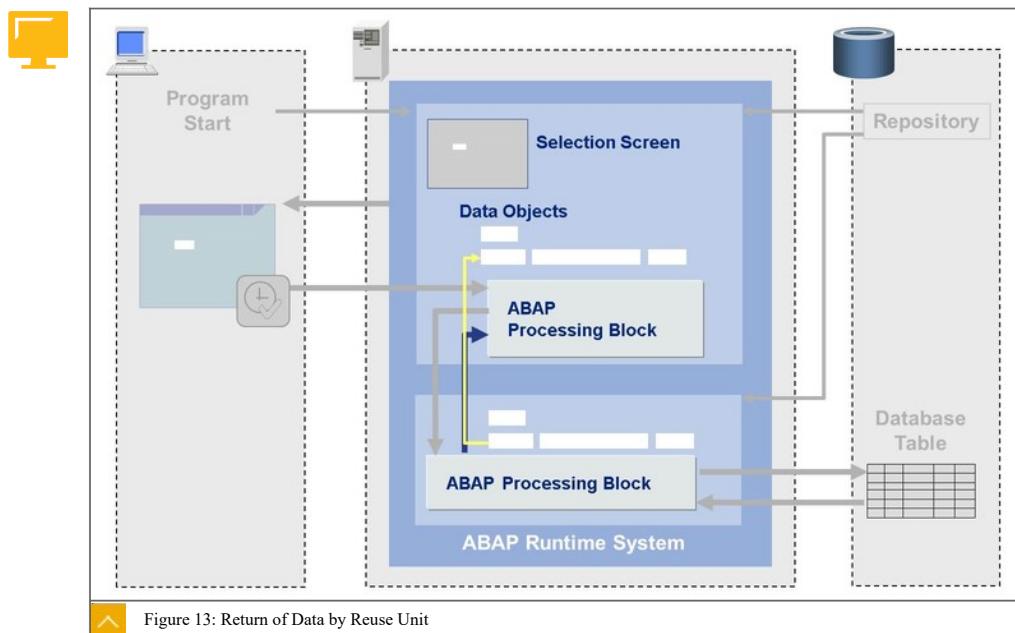


In the example program, read access to the database is programmed in the reuse unit. Correspondingly, information about the database table to be accessed and the row in the table to be read is passed on to the database.

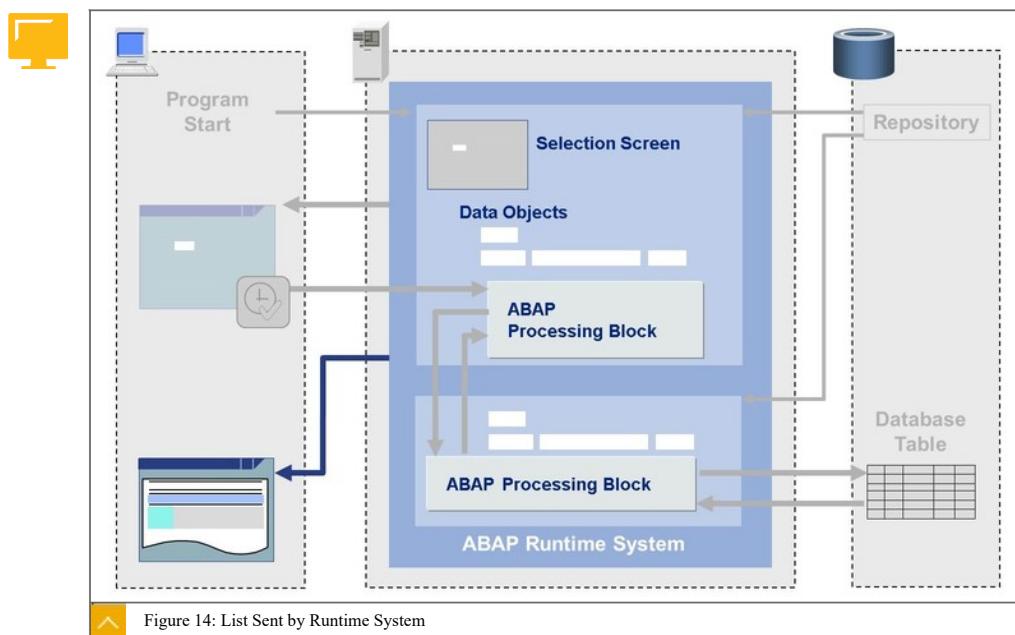


The database returns the requested data record to the program and the runtime system ensures that this data is placed in the appropriate data objects.

If a single record is accessed, this data object is usually a structure that contains relevant components for all the required database fields.



This concludes the processing of the reuse unit and control is returned to the calling program, which resumes immediately after the call. When the system exits the reuse unit, the data that was read from the database is written to a corresponding data object for the calling program, from where it can be processed further.



After the reuse unit has been called, the ABAP processing block receives statements for structuring the list with which the result is to be displayed. After the processing block finishes, the runtime system sends the list as a screen to the presentation server.



#### LESSON SUMMARY

You should now be able to:

- Describe the SAP NetWeaver Application Server architecture
- Describe the execution of a dialog program by the ABAP runtime system

# Unit 1

## Learning Assessment

1. Which of the following layers contains the user interface where each user can access a program, enter new data, and receive the results of a work process?

Choose the correct answer.

- A** Application server layer
- B** Presentation server layer
- C** Application logic layer
- D** SAP NetWeaver Application

2. Which of the following describes the concept of a program made up of several units, instead of a single block?

Choose the correct answer.

- A** Isolation
- B** Modularity
- C** Universality
- D** Multiplicity

# Unit 1

## Learning Assessment - Answers

- Which of the following layers contains the user interface where each user can access a program, enter new data, and receive the results of a work process?

Choose the correct answer.

- A** Application server layer
- B** Presentation server layer
- C** Application logic layer
- D** SAP NetWeaver Application

You are correct! Presentation Server layer is the highest layer. This layer contains the user interface where each user can access the program, enter new data, and receive the results of a work process. Read more in the section, ABAP System Architecture, in the lesson, Describing the Processing of ABAP Programs, in the course BC400 (Unit 1, Lesson 1) or TAW10 Part I (Unit 7, Lesson 1).

- Which of the following describes the concept of a program made up of several units, instead of a single block?

Choose the correct answer.

- A** Isolation
- B** Modularity
- C** Universality
- D** Multiplicity

You are correct! A program is not made up of a single block, but of several units. This is known as modularization. You can use most of these modularization units in more than one program. Therefore, they are often termed as reuse units. Read more in the section, Interplay Between Server Levels and Program Flow, in the lesson, Describing the Processing of ABAP Programs, in the course BC400 (Unit 1, Lesson 1) or TAW10 Part I (Unit 7, Lesson 1).

## UNIT 2

# ABAP Workbench

### Lesson 1

Introducing the ABAP Workbench	17
--------------------------------	----

### Lesson 2

Organizing ABAP Development Projects	26
--------------------------------------	----

### Lesson 3

Developing ABAP Programs	33
--------------------------	----

### Lesson 4

Finalizing ABAP Development Projects	44
--------------------------------------	----

### UNIT OBJECTIVES

- Describe the structure of the ABAP Repository
- Use the search tools of the ABAP Repository
- Display ABAP Repository objects with the Object Navigator
- Describe the ABAP development infrastructure
- Create packages
- Create ABAP programs
- Write ABAP programs with the ABAP Editor
- Activate ABAP programs
- Create transactions
- Release change requests

## Unit 2

### Lesson 1

## Introducing the ABAP Workbench

### LESSON OVERVIEW

This lesson contains an overview of the ABAP Repository and the components of the ABAP Workbench. It presents the Object Navigator as a central development tool.

#### Business Example

You have to describe the structure of the Repository, use suitable tools to search for Repository objects, and analyze their structure.

For this reason, you require the following knowledge:

- An understanding of the structure of the Repository
- An understanding of how to name and use the search tools of the Repository
- An understanding of how to use the Object Navigator for displaying Repository objects

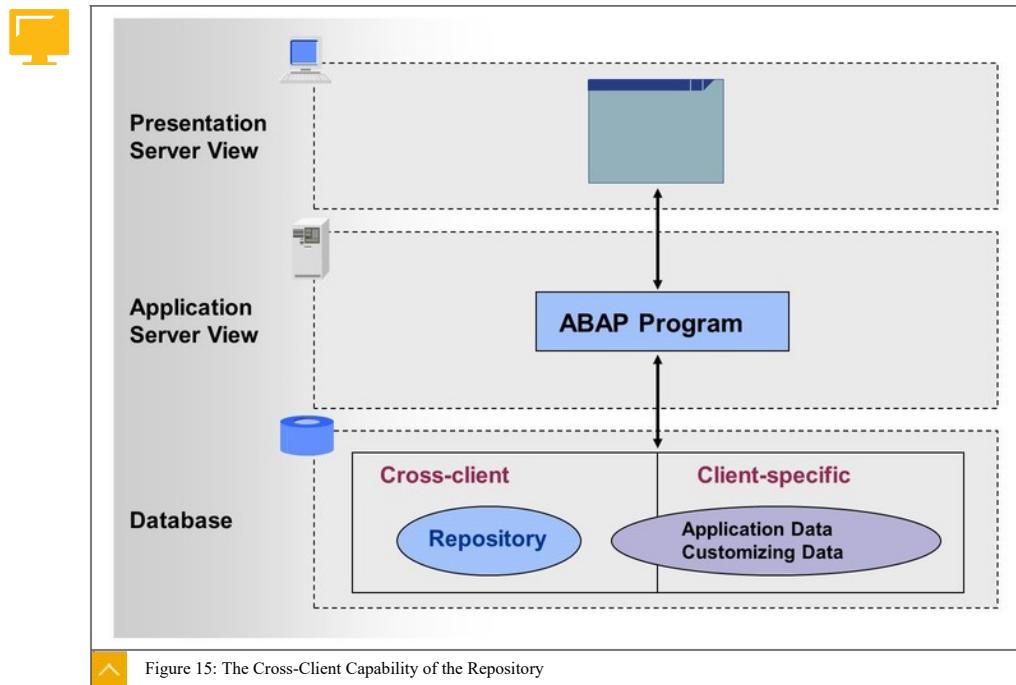


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the structure of the ABAP Repository
- Use the search tools of the ABAP Repository
- Display ABAP Repository objects with the Object Navigator

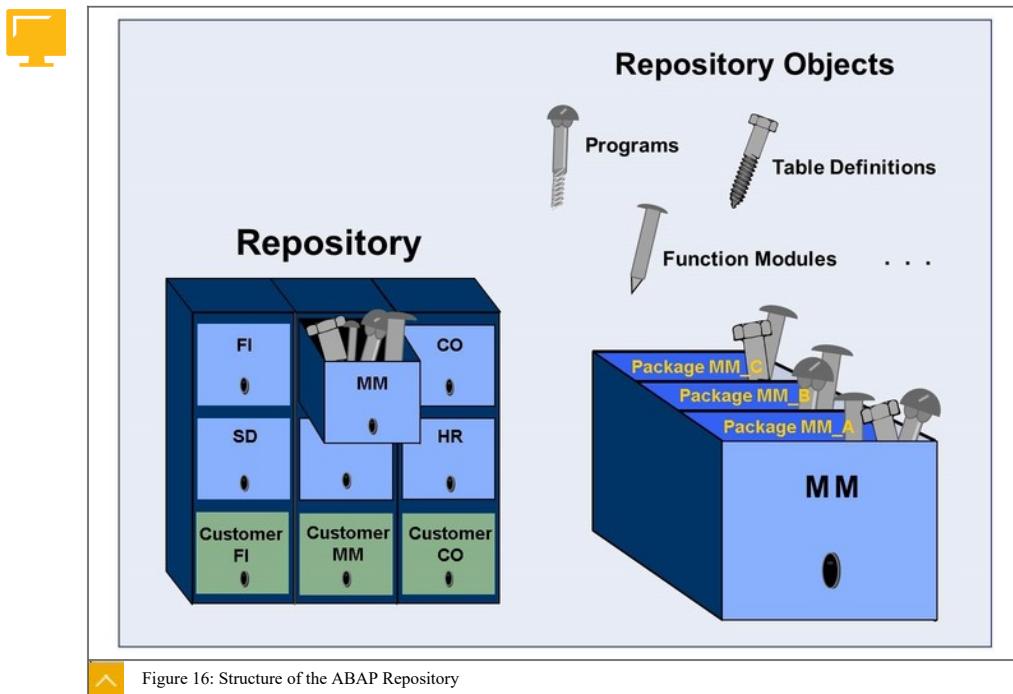
### ABAP Repository



The Repository consists of all system development objects, such as programs, function modules, definitions of database tables, and so on. The Repository contains objects delivered by SAP as well as those defined by the customer. The Repository is in the database and is independent of the client. This means that a Repository object can be accessed from any client and looks the same in each client in the system.

The database also contains application and customizing data, which is normally client-dependent. This means that every data record is assigned to a particular client and can only be read and changed by users who have logged on to that particular client.

## Structure of the ABAP Repository



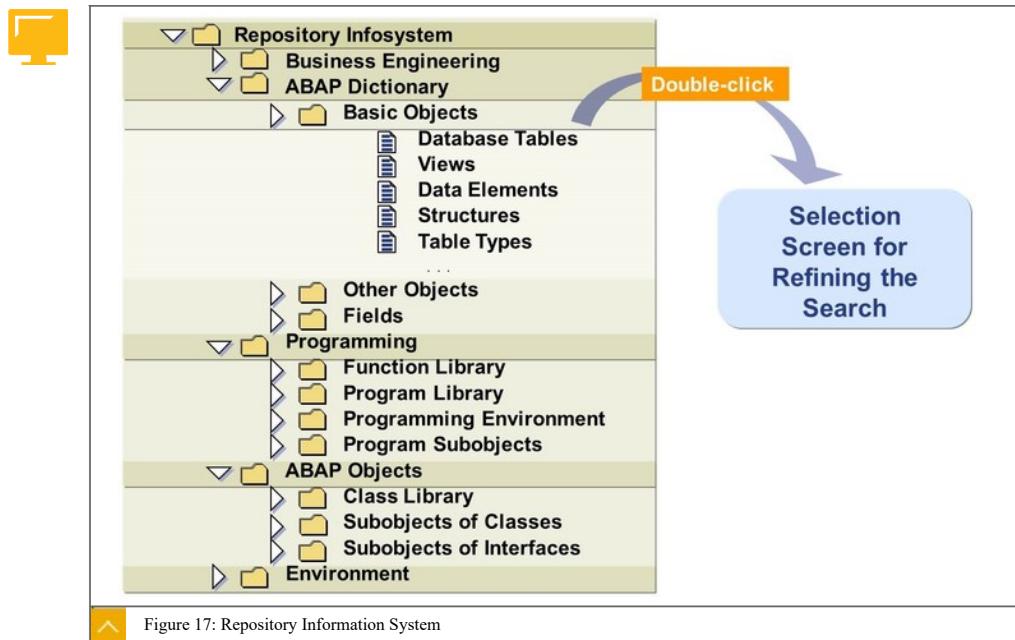
The system subdivides the Repository according to application components.

An application component, for example, Materials Management (MM), consists of several packages containing relevant objects for a more detailed logical subdivision.

Whenever a Repository object is created, it must be assigned to a package.

Some Repository objects can have subobjects that are themselves Repository objects. Furthermore, Repository objects can reference other Repository objects.

## Search Tools of the ABAP Repository



The Repository Information System is suitable for random (that is, not application-specific) searches of Repository objects, for example, all programs by a certain developer or all function modules that were changed after a certain date.

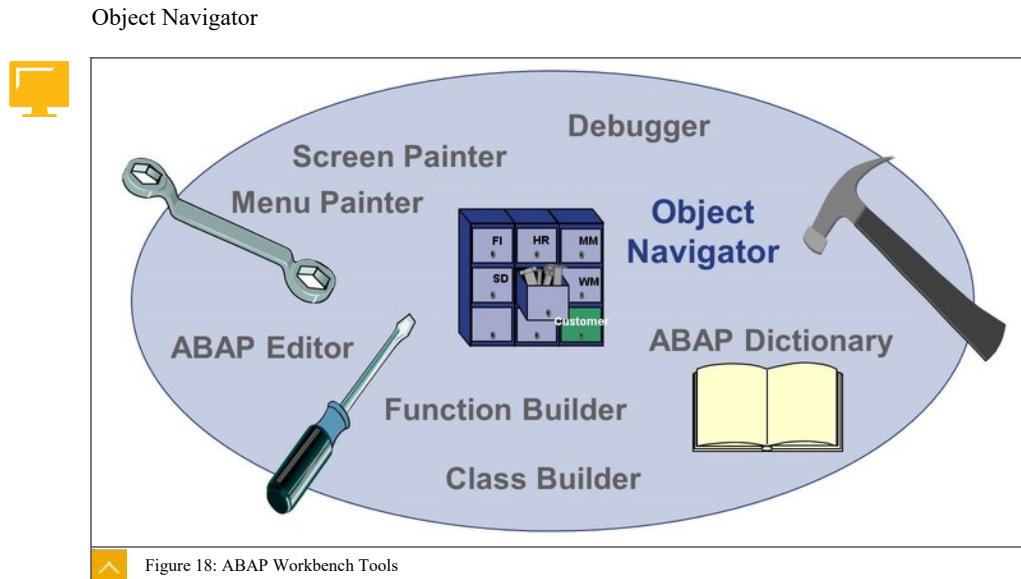
## ABAP Repository Information System Access Method

- To access the ABAP Repository Information System, perform the following steps:

- In the SAP Easy Access menu, choose Tools → ABAP Workbench → Overview → Information System .

Alternatively, you can run transaction SE84.

- Double-click an object type. A selection screen appears allowing you to limit your search.



The ABAP Workbench includes all tools required for creating and editing Repository objects. These tools cover the entire software development cycle.

#### ABAP Workbench Tools

- Some of the ABAP Workbench tools are as follows:
  - The ABAP Editor is used for editing the source code.
  - The ABAP Dictionary is used for editing database table definitions, data types, and other entities.
  - The Screen Painter is used for configuring screens together with functions for user dialogs.
  - The Menu Painter is used for designing user interface components: menu bar, standard toolbar, application toolbar, and function key settings.
  - The Function Builder is used for maintaining function modules.
  - The Class Builder is used for maintaining global classes and interfaces.

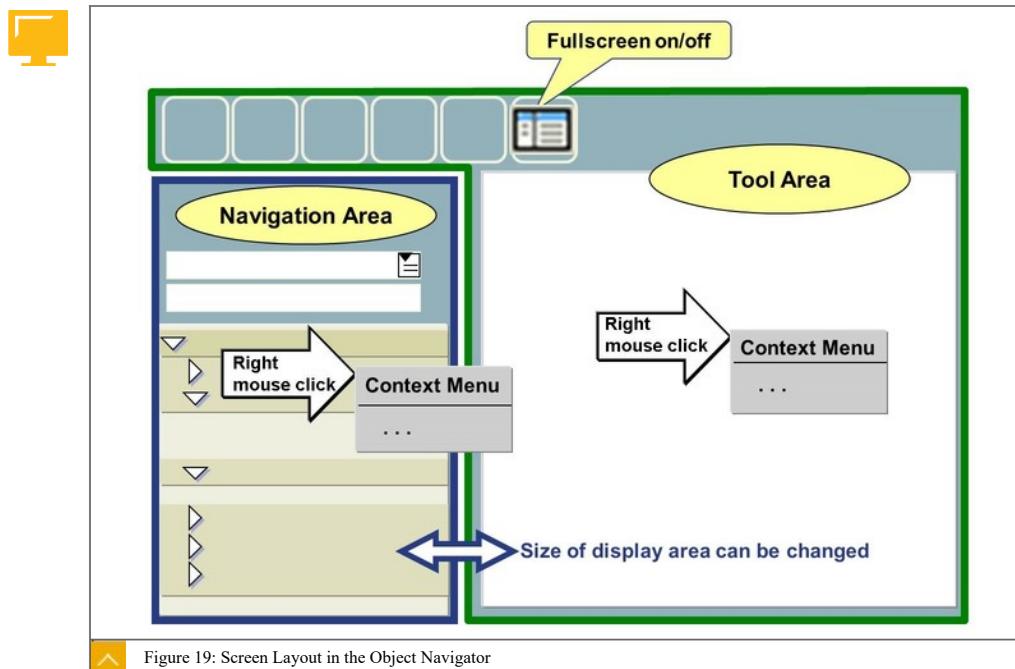
You can call each of these tools explicitly and then load a Repository object for processing, but it is more convenient to access them using the Object Navigator.

- ABAP Editor: transaction SE38
- ABAP Dictionary (display, change, create): transaction SE11
- ABAP Dictionary (display only): transaction SE12
- Screen Painter: transaction SE51
- Menu Painter: transaction SE41

- Function Builder: transaction SE37
- Class Builder: transaction SE24

You can have the requested Repository objects listed in this central development tool. To edit one of the Repository objects, you double-click it. The corresponding tool is called automatically and includes the selected Repository for displaying or editing.

#### Screen Layout in the Object Navigator



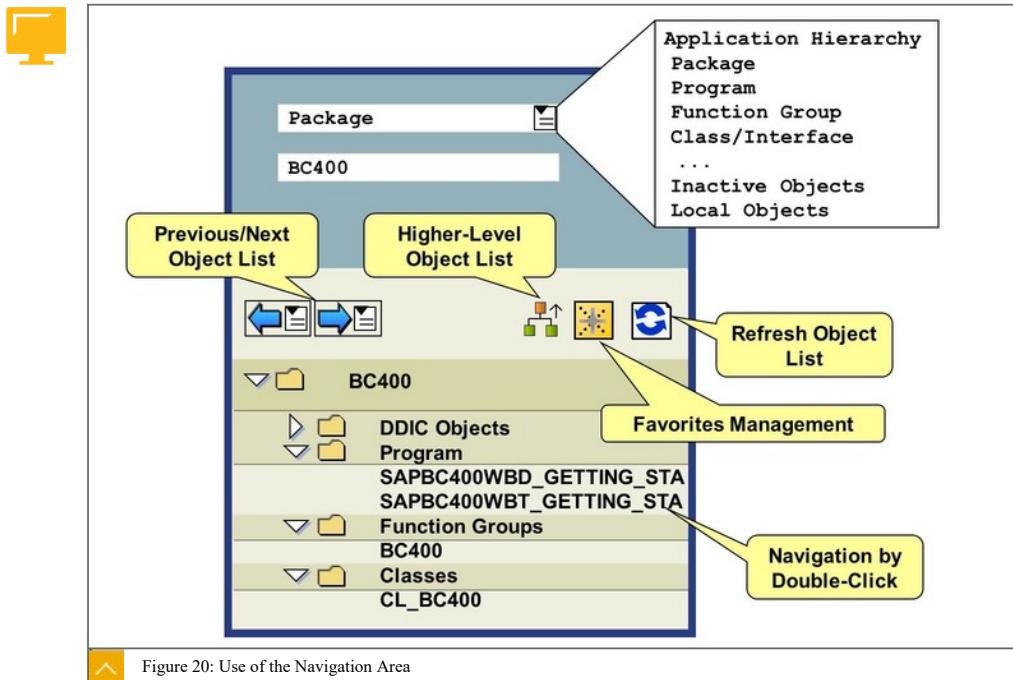
#### The Object Navigator Screen

- The Object Navigator screen is split into the following areas:
  - The navigation area for displaying a hierarchical object list
  - The tool area for displaying and editing a development object using the appropriate tool

You can display or hide the navigation area (      Fullscreen → on/off ).

In both areas, you can choose the functions using a context menu, which you access using the right mouse button. The context menu offers only the functions that have been designed for the respective object.

## Use of the Navigation Area



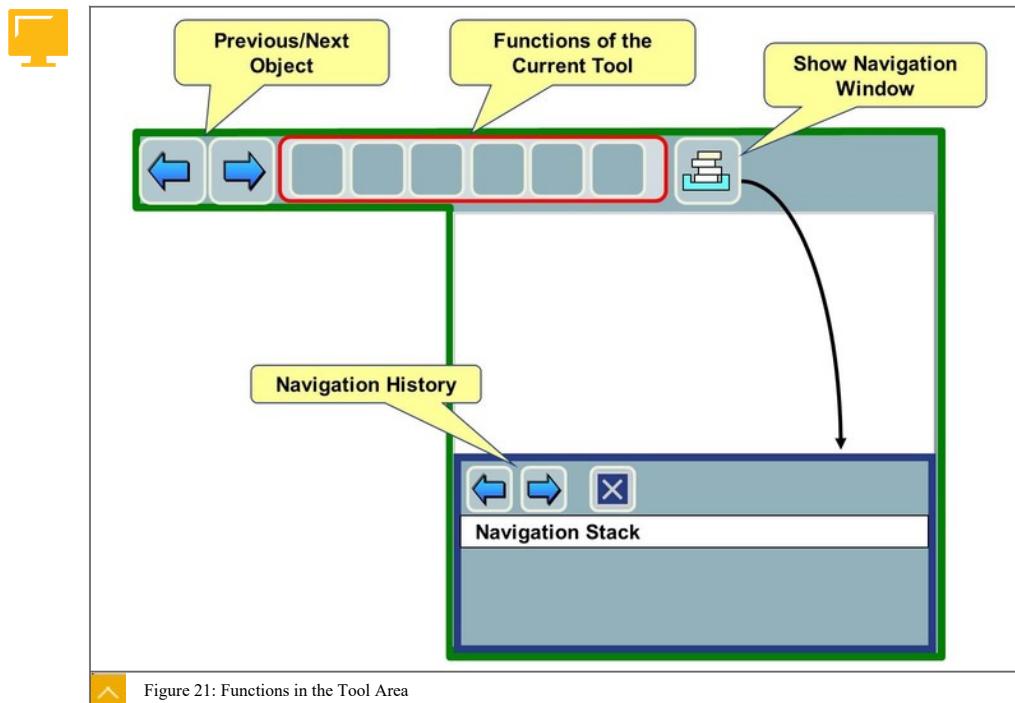
The system displays object lists in the navigation area. For example, if you choose to display a package, all Repository objects belonging to the specified package are listed.

Double-clicking an object allows you to display or edit it.

You can navigate between object lists that have been previously displayed in the current Object Navigator session (blue arrows).

You can add frequently used object lists to your favorites.

Functions in the Tool Area

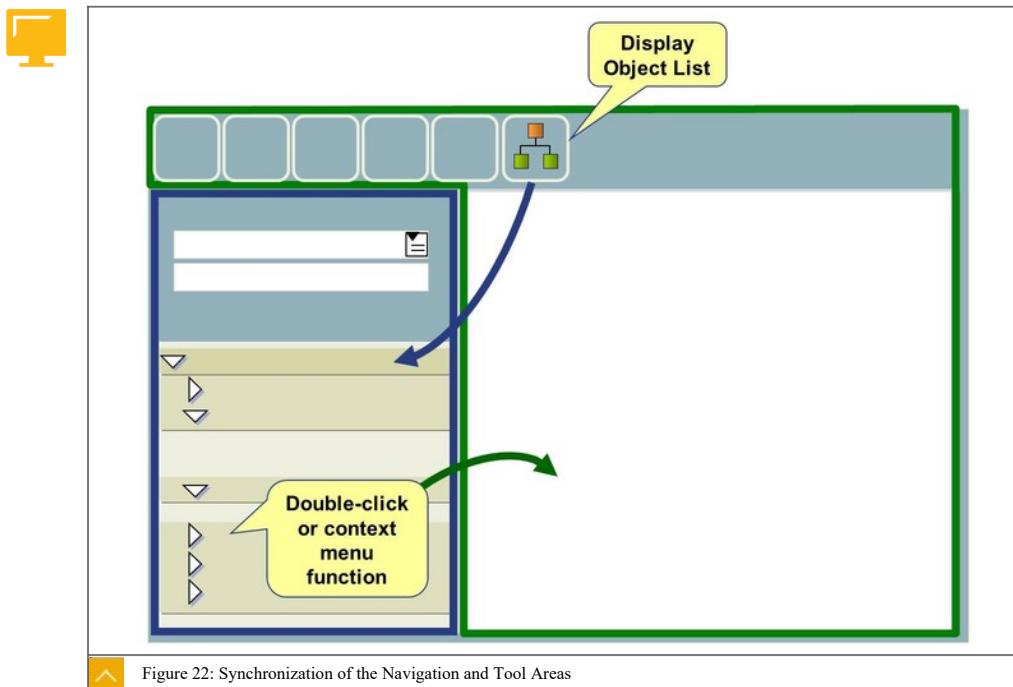


In the tool area, a Repository object is displayed in the corresponding tool.

You can navigate between objects that have been previously displayed in the current Object Navigator session (blue arrows).

You can also display a subwindow with the previous navigation history. When you double-click an object in the navigation history, the object is displayed in the tool area.

## Synchronization of the Navigation and Tool Areas



Navigation in the navigation area is independent of navigation in the tool area. This allows both areas to be used in a flexible manner.

## Synchronization Method

- If necessary, you can synchronize both areas as follows:
  - You can display an object in the tool area by double-clicking in the navigation area or by using the corresponding context menu function of the object.
  - You can display the object list of an object that you are currently editing in the tool area by choosing the **Display Object List** button in the navigation area.

To create a new object, use the context menu for an object type in the corresponding object list. Alternatively, choose **Edit Object** or **Other Object** to create an object.



## LESSON SUMMARY

You should now be able to:

- Describe the structure of the ABAP Repository
- Use the search tools of the ABAP Repository
- Display ABAP Repository objects with the Object Navigator

## Unit 2

### Lesson 2

# Organizing ABAP Development Projects

#### LESSON OVERVIEW

This lesson explains how to create new Repository objects, packages, programs, and transaction codes. This lesson also explains how change requests in the ABAP Workbench are used in the SAP environment to organize development projects and ensure consistent transport of changes to the production systems.

#### Business Example

As part of a development project, you have to create a new package and an ABAP program and make this program available to users by way of a transaction code. For this reason, you require the following knowledge:

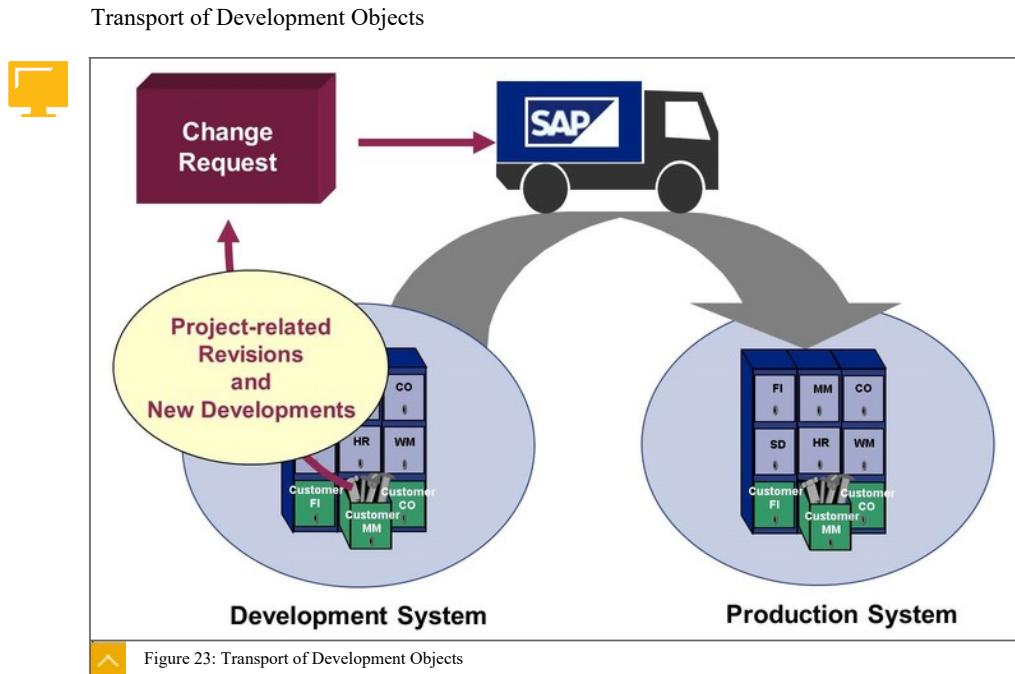
- An understanding of how to name and use utilities for orderly software development
- An understanding of how to create packages, programs, and transactions



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

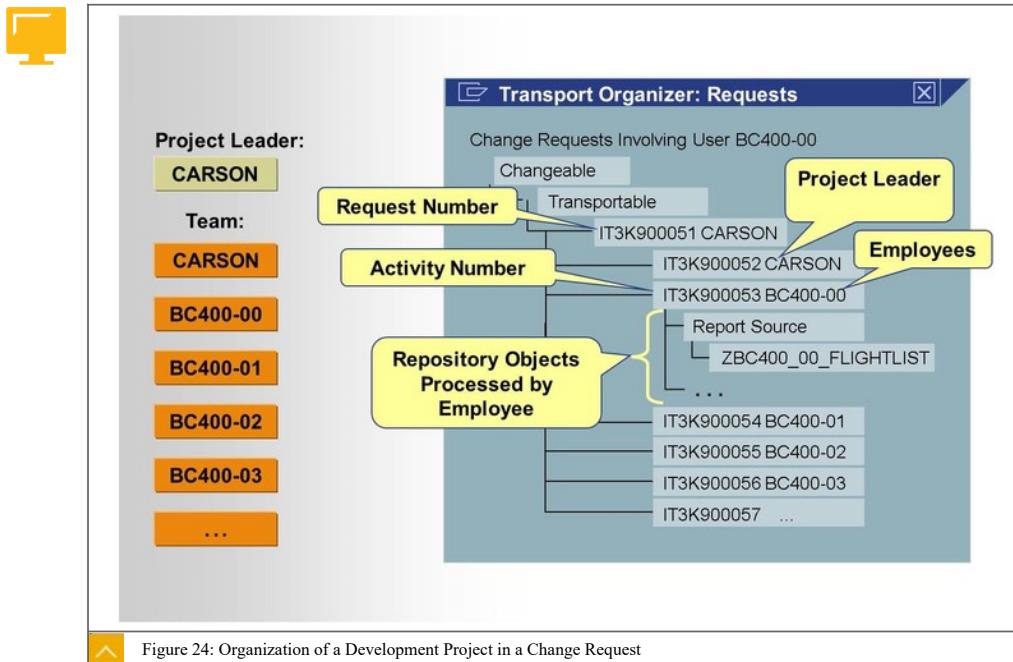
- Describe the ABAP development infrastructure
- Create packages



Development projects are carried out in a development system. The development objects edited or created in a project are transported to subsequent systems (test and/or production system) on project completion. At the start of a development project, the project manager creates a change request in which the project manager names the employees of this project in the Transport Organizer or directly in the ABAP Workbench. The Transport Organizer then creates a task for each project employee in the change request.

When a development object is edited or created, the relevant employee assigns this to the change request. The object is entered into the task of the employee. All Repository objects that an employee works on during a development project are collected within the employee's task.

### Organization of Development Projects



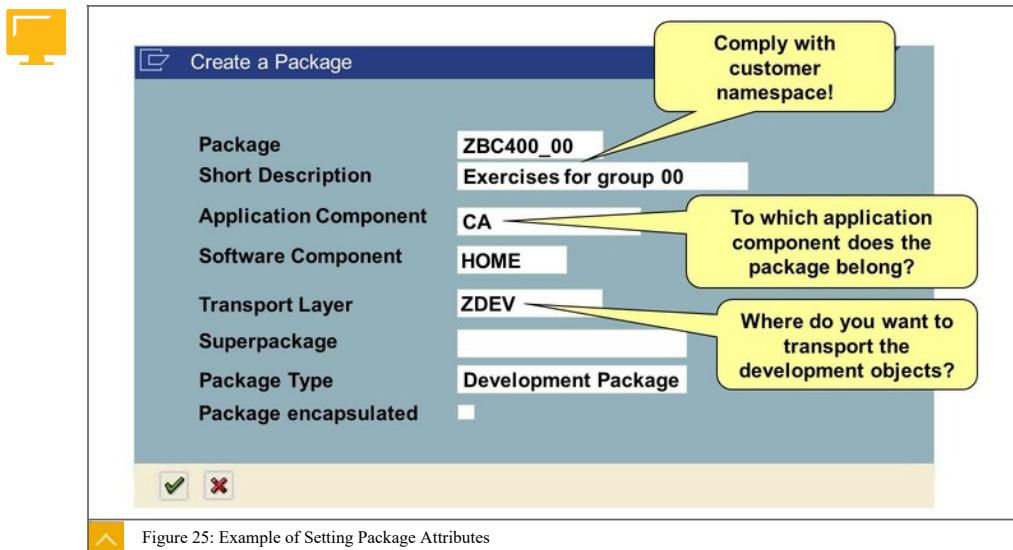
### Organization of a Development Project Using a Change Request

Organizing a development project using a change request offers the following advantages:

- Employees can monitor their project-specific activities.
- Employees involved in the project can process all repository objects assigned to the project. For developers who do not belong to the project team, repository objects remain locked until the project is completed (the change request is released).
- Assigning the objects to the change request ensures that all development objects created or changed during the project are transported automatically at the time of project completion (that is, when the change request is released). The transport route of the packages involved (in which development took place) specifies to which subsequent system they are transported.

Unlike packages, which distinguish between Repository objects in a logical and functional way, change requests are project-related and, therefore, delimit the objects over a period of time. Although a program always belongs to only one package, it can, at different times, belong to different projects.

## Creation of Packages



## Package Attributes

The attributes of a package have the following meaning (detailed information can be obtained by using the field help, F1):

- Application Component

Determine the location of the package within the application hierarchy by specifying the corresponding application component.

- Software Component

For customer developments, enter **HOME** for the software component.

- Transport Layer

The transport layer determines if the objects of this package are to be transported to a subsequent system and, if so, to which system. For customer developments you have to specify the transport layer that your system administrator set up for this purpose.

- Direct Superpackage

The immediate superpackage (or synonym "surrounding package") refers to the package that contains the current package as a direct subpackage based on the package hierarchy. The package hierarchy orders all packages in the form of several trees: Each package has (potentially) a number of subpackages and can, in turn, be the subpackage of a surrounding package.

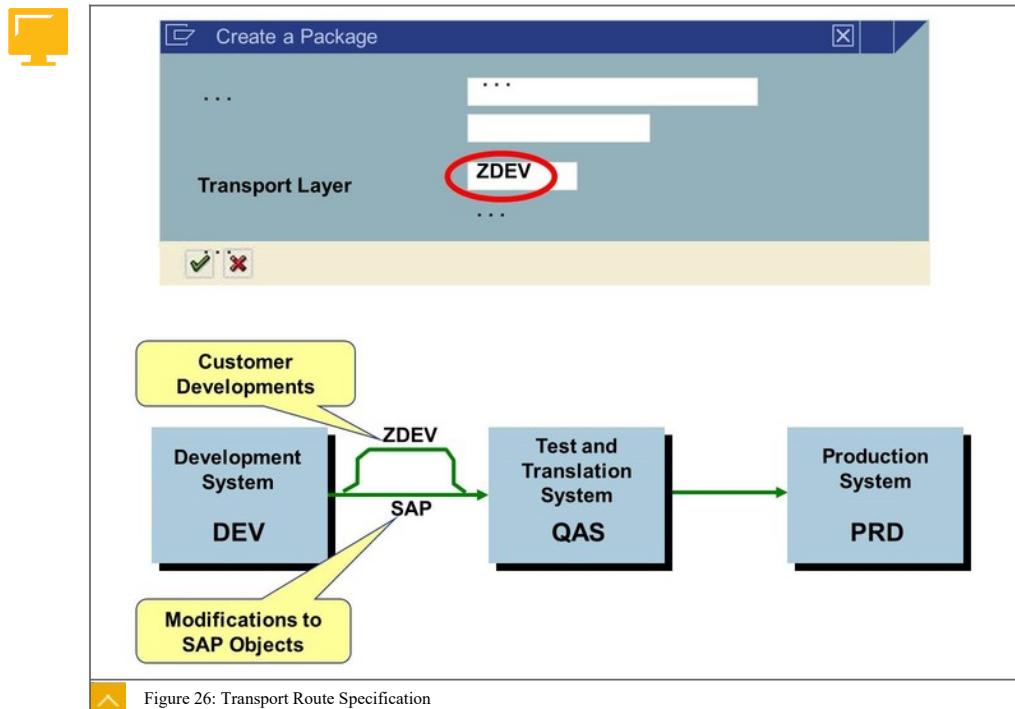
- Package Type

Choose between the three package types: development package (can contain Repository objects and other packages), main package (can only contain other packages), and structure package (can only contain main packages).

- Flag for Package Encapsulation

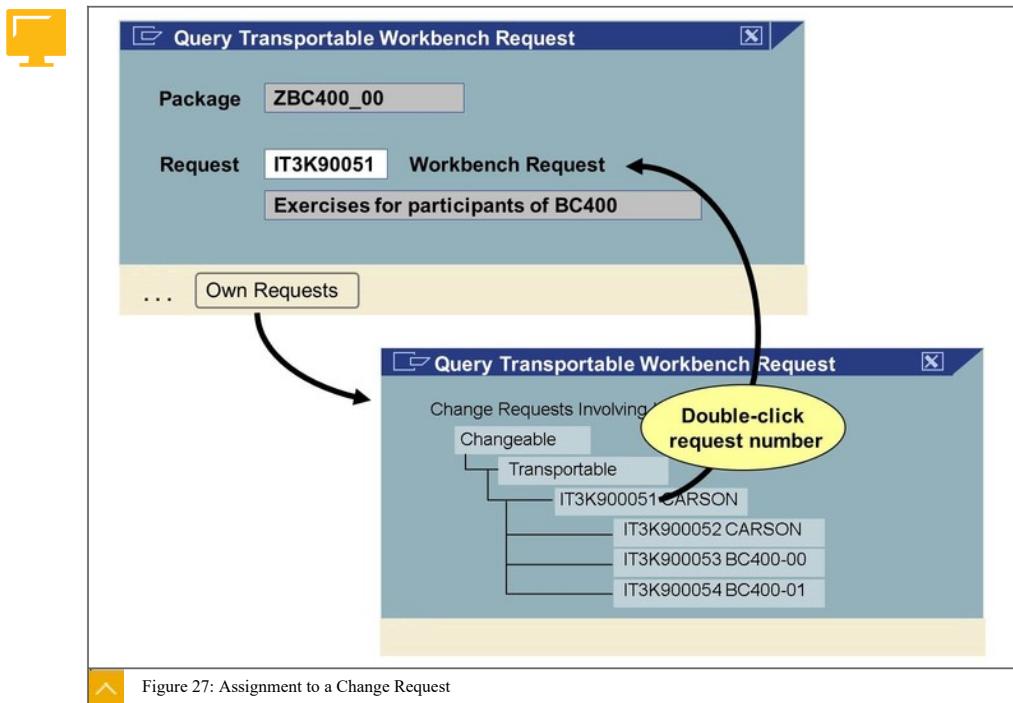
When this package property is activated, the package is encapsulated. The consequence of this is that only the development elements exposed in package interfaces of the package are visible to the outside. Possible client packages need use accesses to those package interfaces that contain the development elements used.

#### Transport Route Specification



Even with a simple system landscape you need at least two different transport layers to distinguish between customer developments and modifications to SAP objects. In a more complex system landscape further transport layers might exist with different target systems.

## Assignment to a Change Request



You can display all change requests in which you have a task using **My Tasks**. To select the relevant request, double-click the transport request.



## Note:

The system takes care of the assignment for your task.



## Hint:

You must assign all Repository objects created or changed to the change request of the respective project.

- 1
- 2
- 3

To Create a Package

1. Navigate to the **Object Navigator**.
2. In the navigation area, choose the **Package** object type and enter the name of the package in the input field below, ensuring that you comply with the customer namespace conventions. Press **Enter**. If the specified package does not already exist, the system will branch off to a dialog to create a package.

Alternatively, you can choose **Edit Object** on the initial screen of the **Object Navigator**. In the dialog box, search for the option of specifying a package and enter the name of the package. Press **F5** to create the object.

3. Set the attributes of the package to be created.
4. Assign the package to a change request.



#### LESSON SUMMARY

You should now be able to:

- Describe the ABAP development infrastructure
- Create packages

## Unit 2

### Lesson 3

# Developing ABAP Programs

#### LESSON OVERVIEW

This lesson explains how to create ABAP programs. It also introduces you to the ABAP programming language and syntax.

#### Business Example

You have to create a new package and a new ABAP program as a part of a development project. You have to make this program available to users with a transaction code. For this reason, you require the following knowledge:

- An understanding of how to create ABAP programs
- An understanding of the ABAP Editor
- An understanding of how to activate ABAP programs

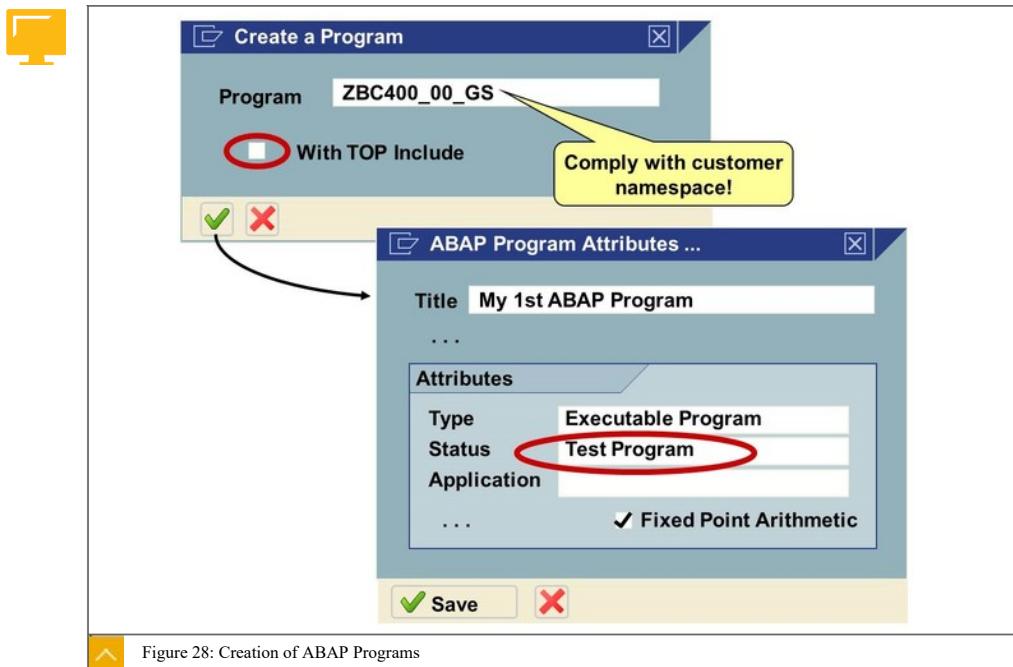


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create ABAP programs
- Write ABAP programs with the ABAP Editor
- Activate ABAP programs

## Creation of ABAP Programs



The figure shows the creation of an ABAP program without a top include and the attributes of the created program.

You will learn how to create programs and transactions in the ABAP Workbench.



### To Create an ABAP Program

1. Navigate to the Object Navigator using any of the following options:
  - a) In the navigation area, choose the **Program** object type and enter the name of the program in the input field, ensuring that you comply with the customer namespace conventions. Press ENTER. If the program does not exist, the system goes to the dialog that allows you to create a program.
  - b) Display the package in which you want to create the program. You can branch to the dialog for creating a program using the context menu for the package or the **Programs** node.
  - c) Choose the **Edit Object** button on the initial screen of the **Object Navigator**. In the dialog box, choose the **Program** tab page and enter the name of the program. To create the object, press F5.
2. Leave the **Create with TOP Include** checkbox empty, to prevent your source code from being distributed to several programs.



**Caution:**

In previous releases (before SAP NetWeaver 7.40) this checkbox is selected by default and has to be unchecked manually.

3. Change the title to a self-explanatory short text and, for the purposes of this course, choose Executable Program as the program type. All other program attributes are optional. To access help or get more details, press F1.

### Basic ABAP Principles

#### ABAP Programming Language Features



- The features of the ABAP programming language are as follows:
  - It is typed.
  - It enables multi-language application.
  - It enables SQL access.
  - It is enhanced as an object-oriented language.
  - It is platform-independent.
  - It is upward-compatible.

The ABAP programming language is designed for dialog-based business applications.

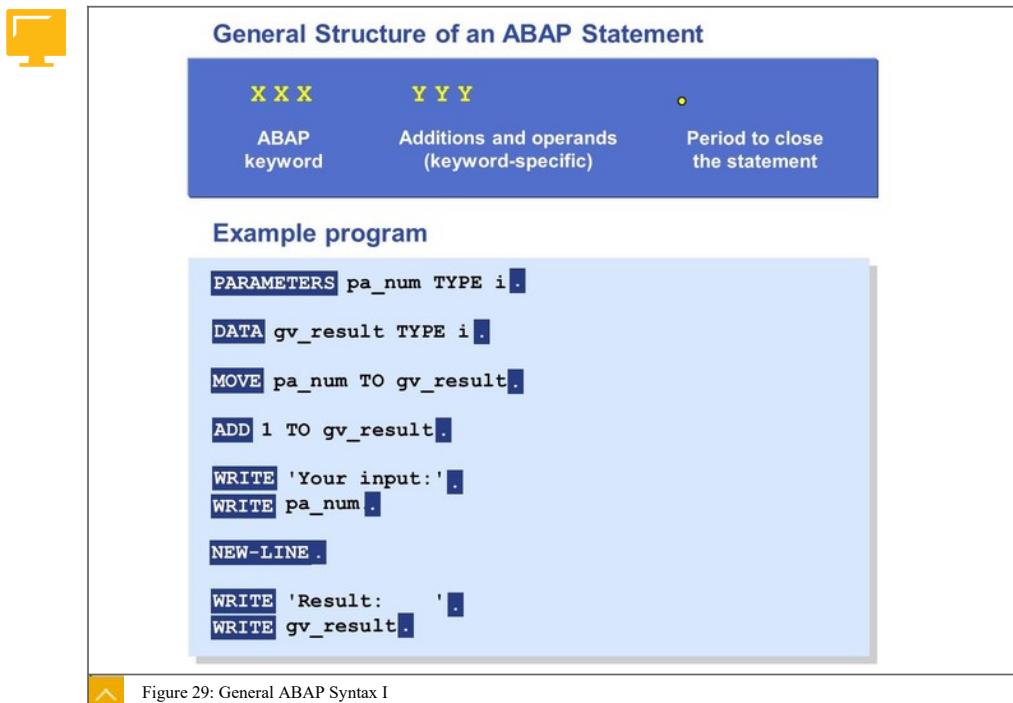
To support the type-specific processing of data, type conversions and type casting are supported.

Using translatable text elements, you can develop multi-language applications.

The Open SQL standard embedded in ABAP allows direct database access.

ABAP Objects is the object-oriented enhancement of the ABAP programming language. ABAP syntax is platform-independent. This means that ABAP syntax always has the same meaning or function, irrespective of the relational database system and operating system for the application and presentation server. Applications implemented in ABAP will also be able to run in future releases because of the upward compatibility of the language.

### General ABAP Syntax I



#### Characteristics of ABAP Syntax

- ABAP programs consist of individual statements.
- The first word in a statement is called an ABAP keyword.
- Each statement ends with a period.
- A space must always separate two words.
- Statements can be indented.
- The ABAP runtime system does not differentiate between upper and lowercase in keywords, additions, and operands.



#### Hint:

Although the ABAP runtime system does not differentiate between upper and lowercase, it has become customary to write keywords and their additions in uppercase letters and operands in lowercase. This course uses this form of representation.

For indentations and for converting uppercase or lowercase letters, use Pretty Printer in the ABAP Editor. Use the following menu path in the Object Navigator to make a user-specific setting for the Pretty Printer: Utilities → Settings → ABAP Editor → Pretty Printer .

## General ABAP Syntax II

The diagram illustrates General ABAP Syntax II with the following code snippet:

```

    * comments ...
    * comments ...
    * comments ...

PARAMETERS pa_num TYPE i.
DATA gv_result TYPE i.

MOVE pa_num
      TO gv_result.

ADD 1 TO gv_result.

WRITE : 'Your input:' , pa_num.
NEW-LINE.
WRITE : 'Result: ' , gv_result.

```

Annotations explain the syntax:

- Comments (whole lines)**: Brackets group three lines starting with an asterisk (\*) as comments.
- Comments (rest of line)**: Brackets group the part of the line after the double quotation mark (" ") as a comment.
- Chained statement**: Brackets group multiple statements separated by commas (,).

Figure 30: General ABAP Syntax II

## Additional Features of ABAP Syntax

- Statements can extend beyond one line.
- Several statements can lie in a single line (although it is not recommended).
- Lines that begin with an asterisk (\*) in the first column are recognized as comment lines by the ABAP runtime system and are ignored.
- A double quotation mark ("") indicates that the rest of a line is a comment.

## Combination of Statements

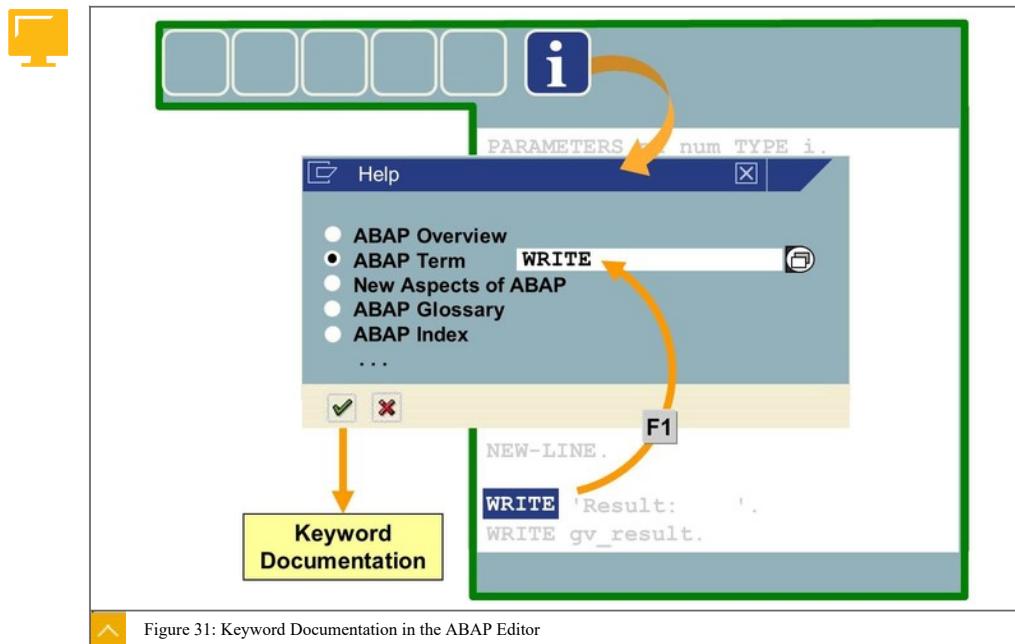
- You can combine consecutive statements with an identical beginning into a chained statement as follows:
  - Write the identical beginning part of the statements, followed by a colon.
  - List the end parts of the statements, separated by commas.
  - Use blank spaces and line breaks before and after separators, that is, colons, commas, and periods.



## Hint:

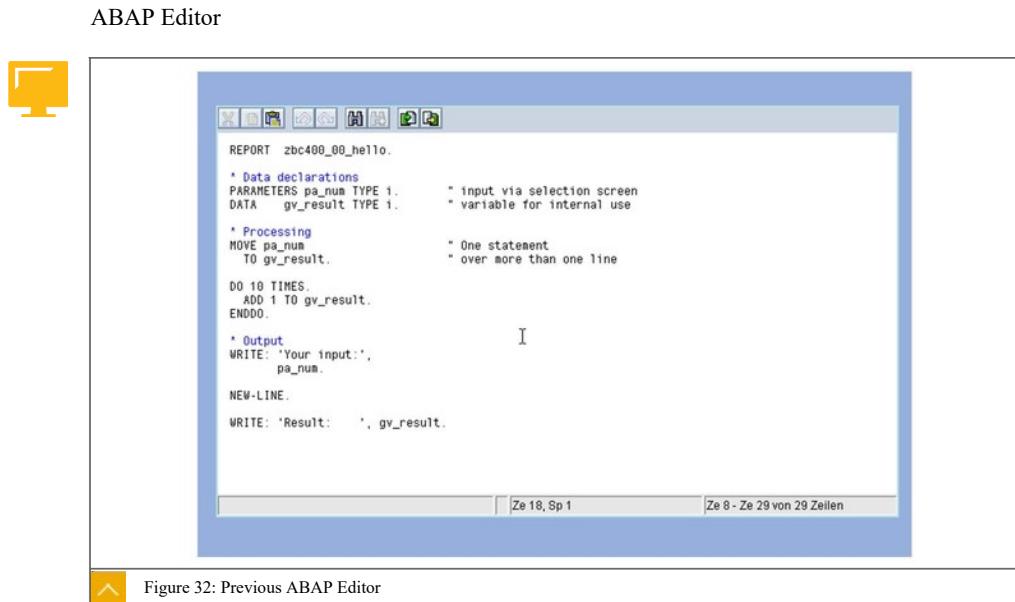
This short form represents a simplified form of syntax and does not offer an improvement in performance. The ABAP runtime system processes each of the individual statements.

Keyword Documentation in the ABAP Editor



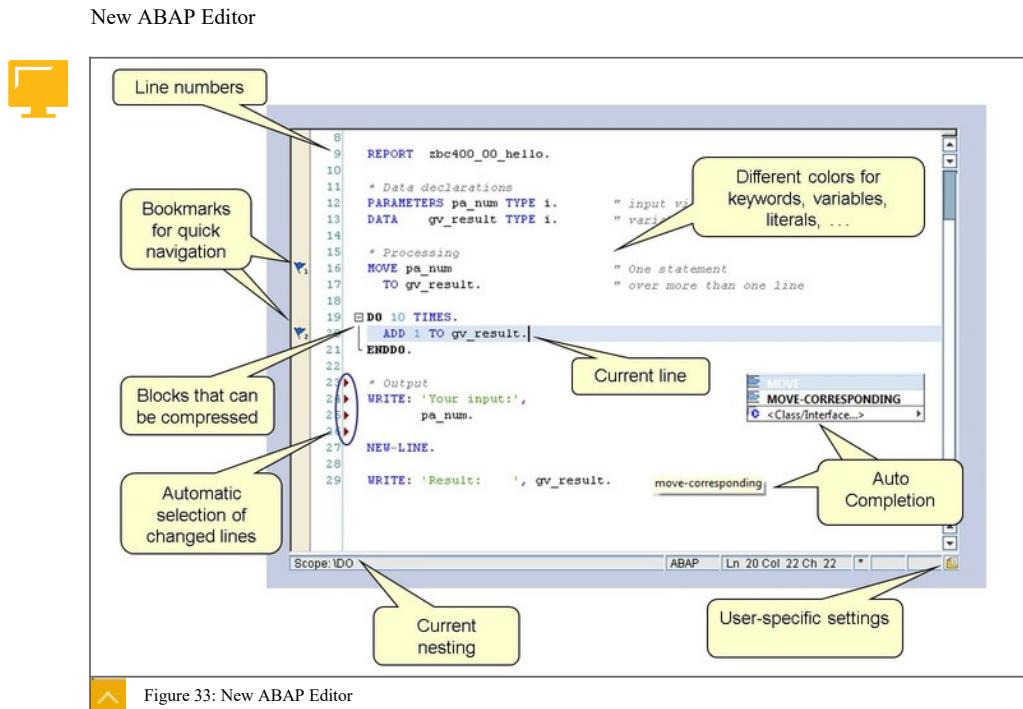
Access to the Documentation for an ABAP Statement

- You can navigate to the documentation for an ABAP statement in the following ways:
  - Use F1 to navigate directly to the documentation for the statement where the cursor is positioned.
  - Choose the **i** button with the description **Help on** to navigate to a dialog box, where you can enter the required ABAP keyword.



You can use several editors to develop ABAP programs, depending on the release and support package level. The choice of the editor is user-specific and can be changed (in the Object Navigator, choose Utilities → Settings → ABAP Editor).

The latest ABAP Editor (new front-end editor) offers a number of new options and easy-to-use additional functions compared to the classic ABAP Editor (old front-end editor). It was developed for SAP NetWeaver 7.0. It can also be used in previous releases that were available after support package levels SAP NetWeaver AS 6.40 (SP18) and SAP Web Application Server 6.20 (SP59).

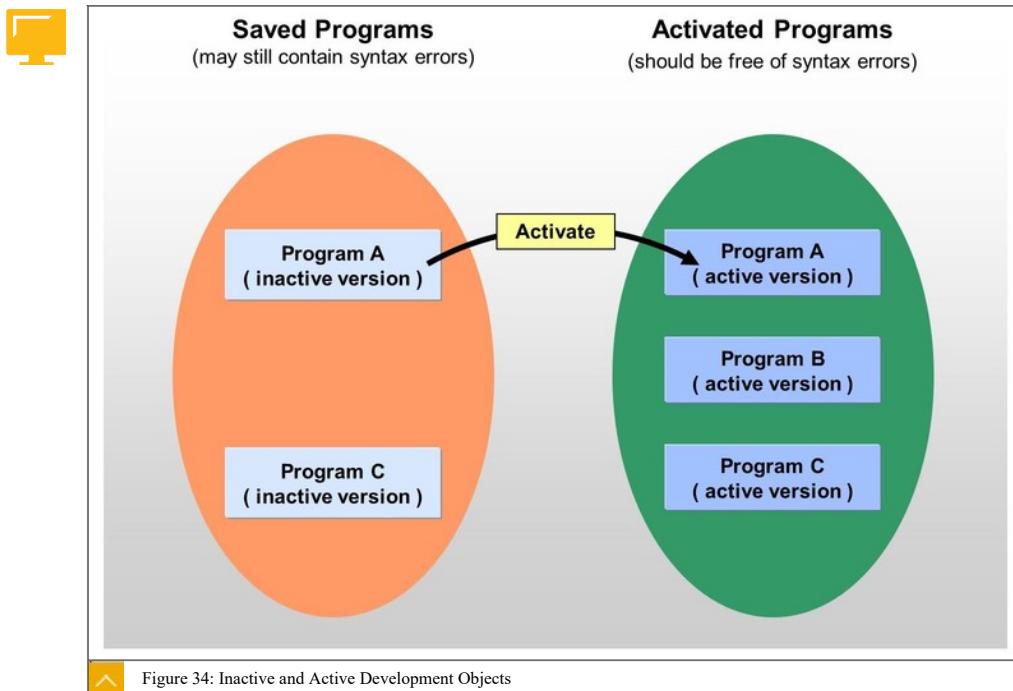


### Options Provided by the New ABAP Editor

Some of the important options provided by the new ABAP Editor are as follows:

- You can choose different display colors for different objects in the source code.
- You can set fonts and font sizes for each individual user.
- You can compress different blocks of source code, such as loops and conditional branches, to provide a better overview.
- You can use bookmarks to find relevant points in the source code faster.
- You can display line numbers and the current nesting to improve orientation.
- You can have complete words suggested by the ABAP Editor when you type the first few characters of ABAP keywords and data objects.
- You can have a small pop-up list generated by the new ABAP Editor with suitable suggestions for the current cursor position (since SAP NetWeaver 7.0 EhP2) when you press Ctrl + Spacebar.

### Activation of Programs



Whenever you create or change a development object and then save it, the system stores an inactive version in the Repository.

You can have an active version and an inactive version of the object. When you complete development of the object, you have to activate the inactive version (editing version) of the object. The inactive version then becomes the new active version of the object.



Note:

The request release and the transport of the developed objects are only possible if all objects in the request have been activated.

If your program is available in both active and inactive versions, you can switch between the displays of these two versions by using the corresponding button in the ABAP Editor.

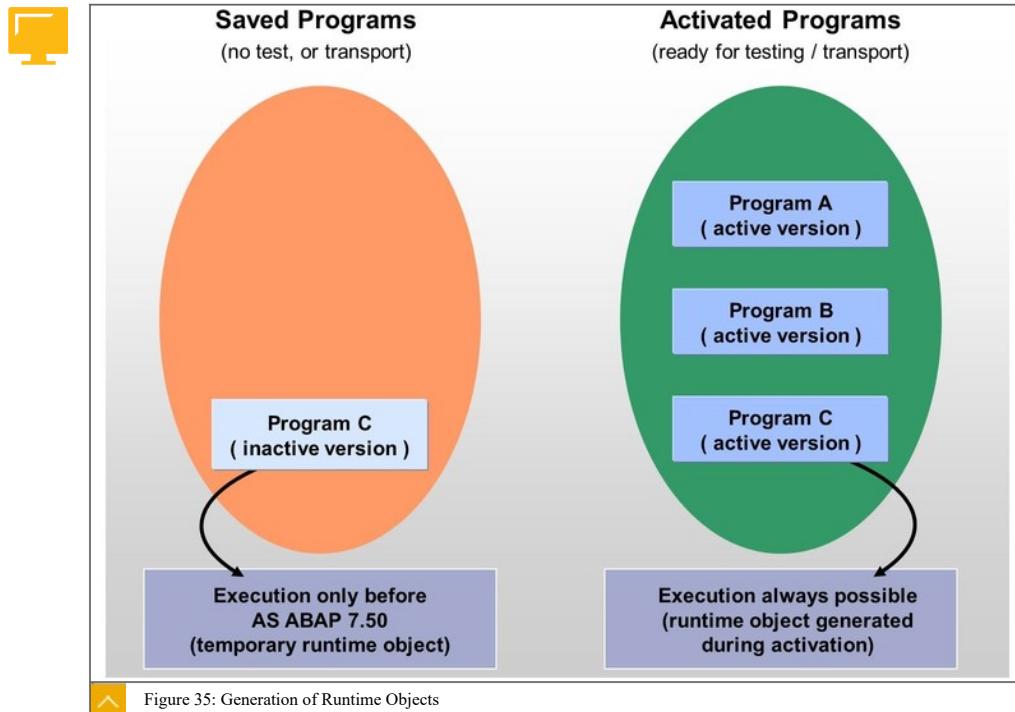
Whenever you activate a program, the system displays a list of all inactive objects that you have processed. This is your work list. Choose those objects that you wish to activate with your current activation transaction.

#### Functions Performed During the Activation of an Object

- The activation of an object includes the following functions:
  - Saving the object as an inactive version

- Checking the syntax or consistency of the inactive version
- Overwriting the previously active version with the inactive version (only after a successful check)
- Generating the relevant runtime object for later execution, if the object is a program

#### Generation of Runtime Objects



When you generate a development object, the system creates a separate runtime object (LOAD compilation) and stores it in the Repository. This generated version is the version that is executed or interpreted at runtime.

#### Use of Active and Inactive Versions

- If you start your program using the context menu of the navigation area, or by executing a transaction, the active version is used. This means that the LOAD generated for the last activation is executed.
- In releases before SAP NetWeaver 7.50, it is also possible to execute inactive version of a program by choosing Direct or by hitting key F8. In this case a temporary runtime object is generated from it and executed.
- Starting with release SAP NetWeaver 7.50 you can only execute your program after you have activated it. An error message displays if you choose Direct or hit the F8 key for an inactive program version.



### LESSON SUMMARY

You should now be able to:

- Create ABAP programs
- Write ABAP programs with the ABAP Editor
- Activate ABAP programs

## Unit 2 Lesson 4

# Finalizing ABAP Development Projects

### LESSON OVERVIEW

This lesson explains how to create transactions and release change requests.

#### Business Example

You need to create transactions and release change requests. For this reason, you require the following knowledge:

- An understanding of how to create transactions
- An understanding of how to release change requests



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create transactions
- Release change requests

#### Creation of Transactions



The screenshot shows the SAP GUI interface for creating a report transaction. The title bar reads 'Create Report Transaction'. In the main area, there is a tree view on the left showing a node 'ZBC400\_00\_GS'. On the right, there are several input fields:

- Transaction Code:** Z00GS
- Package:** ZBC400\_00
- Transaction Text:** ...  
Program: ZBC400\_00\_GS
- Classification:**  Professional User Transaction  
 Easy Web Transaction

You can only include transactions in a role menu and in a user's favorites. If you want to place a program in either a role menu or a user's favorites, you must create a transaction that

represents the program and integrates the transaction into the menu. Alternatively, you can start the program by entering the transaction code in the command field.



To Create a Transaction

1. In the Object Navigator , display the object list for the program.
2. In the navigation area, select your program and choose Create → Transaction from the context menu.
3. Enter the required transaction code. Make sure you comply with the customer namespace conventions.  
Assign a short text and select the Program and Selection Screen (Report Transaction ) radio button.
4. On the next screen, enter the name of the program and choose Professional User Transaction .  
Under GUI enabled , set the SAP GUI for Windows indicator.
5. Save the transaction.
6. Assign the transaction to a package and to a change request on the following screens.



Note:  
Each transaction is a Repository object.

#### Transactions in Personal Favorites

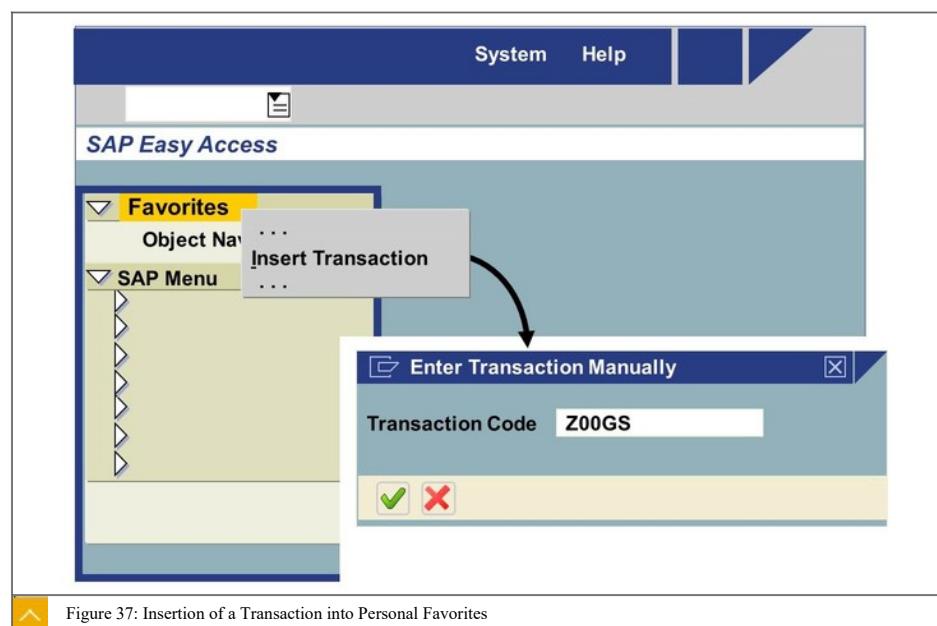


Figure 37: Insertion of a Transaction into Personal Favorites

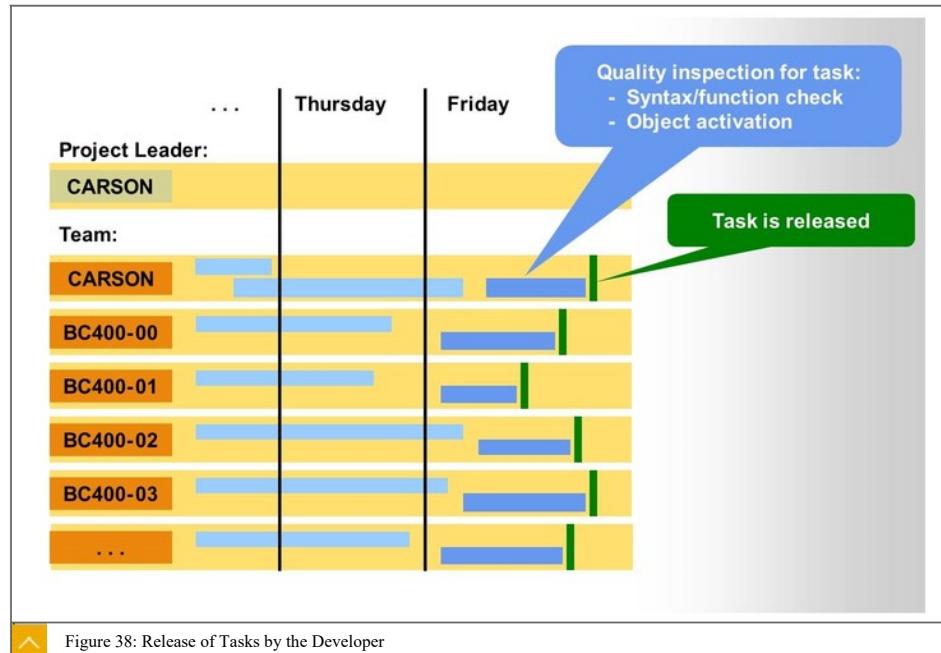


To Add a Transaction to Personal Favorites

1. Navigate to the initial screen, the SAP Easy Access screen.
2. In the Favorites context menu, choose Insert Transaction .
3. In the dialog box, enter the required transaction code.

The short text of the transaction now appears under the Favorites context menu. You can start the corresponding program by double-clicking it.

### Release of Change Requests

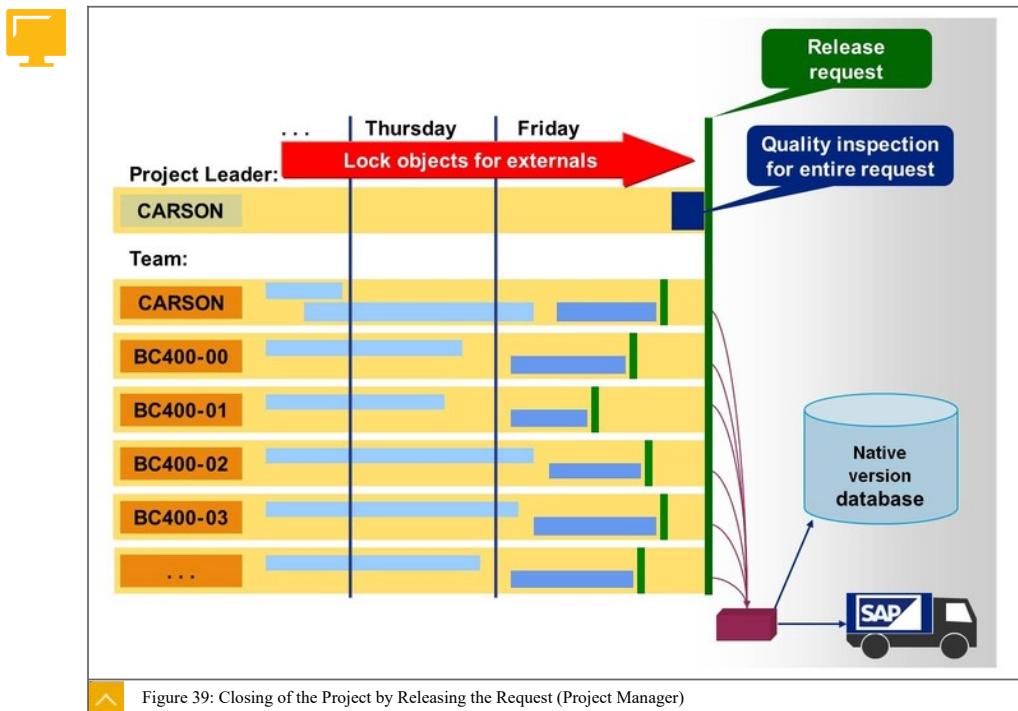


After a project employee completes the required development task, the employee performs a quality check and releases the task within the change request.

The system transfers the corresponding object entries and the change locks of the object for third parties that were automatically set at the start of the project, from the task to the request.

All project employees can still edit these objects. After all the tasks in a change request have been released, the project manager performs the final check for the objects and releases the change request. This concludes the project.

## Closing of the Project by Releasing the Request (Project Manager)



When the request is released, all the object locks that belong to the request are removed.

Copies of the developed objects are exported to the system's own transport directory where they stay until the system administrator imports them to their intended target system.

Another copy of the exported development objects is stored in the system's own version of the database.



## LESSON SUMMARY

You should now be able to:

- Create transactions
- Release change requests

## Unit 2

### Learning Assessment

1. Which of the following system development objects are included in the Repository?

Choose the correct answers.

- A** Programs
- B** Function modules
- C** Object Navigator
- D** Definitions of database tables

2. Which of the following ABAP Workbench tools is used to edit the source code?

Choose the correct answer.

- A** ABAP Editor
- B** ABAP Dictionary
- C** Repository Information System
- D** Menu Painter

3. Which of the following are package types?

Choose the correct answers.

- A** Application component
- B** Development package
- C** Main package
- D** Software component

4. When is the transport of development objects for a development request triggered?

Choose the correct answer.

- A When an object is saved  
 B When an object is activated  
 C When a task is released  
 D When a request is released

5. Whenever you create or change a development object and save it, the system stores two inactive versions in the Repository.

Determine whether this statement is true or false.

- True  
 False

6. Which of the following statements is correct about the ABAP programming language?

Choose the correct answers.

- A It enables multi-language applications.  
 B It enables SQL access.  
 C It is platform-dependent.  
 D It is typed.

7. ABAP Objects is the object-oriented enhancement of the ABAP programming language.

Determine whether this statement is true or false.

- True  
 False

8. How do you include comment lines?

Choose the correct answer.

- A Begin the line with a pound # in the first column.  
 B Begin the line with an asterisk \* in the first column.  
 C End the line with an asterisk \* in the last column.  
 D End the line with a pound # in the last column.

9. After a project employee completes the required development task, they should do the following:

Choose the correct answer.

- A** Transfer the task to the Production System
- B** Release the task within the Change request
- C** Unlock and export the task to the Transport Directory
- D** Export the Project Request

## Unit 2

### Learning Assessment - Answers

1. Which of the following system development objects are included in the Repository?

Choose the correct answers.

- A** Programs
- B** Function modules
- C** Object Navigator
- D** Definitions of database tables

You are correct! The repository consists of all ABAP development objects, such as programs, function modules, and definitions of database tables. It contains objects delivered by SAP as well as those defined by the customer. Note that the terms repository object and development object are often used interchangeably. Read more in the section, ABAP Repository, in the lesson, Introducing the ABAP Workbench, in the course BC400 (Unit 2, Lesson 1) or TAW10 Part I (Unit 8, Lesson 1).

2. Which of the following ABAP Workbench tools is used to edit the source code?

Choose the correct answer.

- A** ABAP Editor
- B** ABAP Dictionary
- C** Repository Information System
- D** Menu Painter

You are correct! The ABAP Editor provides a range of functions, including editing the source code, syntax checks, and options for the capitalization of ABAP keywords. Read more in the section, ABAP Workbench Tools, in the lesson, Introducing the ABAP Workbench, in the course BC400 (Unit 2, Lesson 1) or TAW10 Part I (Unit 8, Lesson 1).

3. Which of the following are package types?

Choose the correct answers.

- A Application component  
 B Development package  
 C Main package  
 D Software component

You are correct! There are three package types: development package (can contain Repository objects and other packages), main package (can only contain other packages), and structure package (can only contain main packages). Read more in the section, Package Attributes, in the lesson, Organizing ABAP Development Projects, in the course BC400 (Unit 2, Lesson 2) or TAW10 Part I (Unit 8, Lesson 2).

4. When is the transport of development objects for a development request triggered?

Choose the correct answer.

- A When an object is saved  
 B When an object is activated  
 C When a task is released  
 D When a request is released

You are correct! When a developer completes the development project, the developer releases the corresponding task. This release transfers the objects in the task to the transport request. Once all team members release their tasks, the development leader can release the transport request, and the transport is triggered. Read more in the section, Closing of the Project by Releasing the Request (Project Manager), in the lesson, Finalizing ABAP Development Projects, in the course BC400 (Unit 2, Lesson 4) or TAW10 Part I (Unit 8, Lesson 4).

5. Whenever you create or change a development object and save it, the system stores two inactive versions in the Repository.

Determine whether this statement is true or false.

- True  
 False

You are correct! Whenever you create or change a development object and then save it, the system stores only one inactive version in the Repository. You have an active version and an inactive version of the object. When you complete object development, you have to activate the inactive version (editing version) of the object. This version becomes the new active version of the object. Read more in the section, Activation of Programs, in the lesson, Developing ABAP Programs, in the course BC400 (Unit 2, Lesson 3) or TAW10 Part I (Unit 8, Lesson 3).

6. Which of the following statements is correct about the ABAP programming language?

Choose the correct answers.

- A** It enables multi-language applications.
- B** It enables SQL access.
- C** It is platform-dependent.
- D** It is typed.

You are correct! The features of the ABAP programming language are as follows: It is typed, It enables multi-language application, It enables SQL access, It is enhanced as an object-oriented language, It is platform-independent, It is upward-compatible. Read more in the section, ABAP Programming Language Features, in the lesson, Developing ABAP Programs, in the course BC400 (Unit 2, Lesson 3) or TAW10 Part I (Unit 8, Lesson 3).

7. ABAP Objects is the object-oriented enhancement of the ABAP programming language.

Determine whether this statement is true or false.

- True
- False

You are correct! ABAP Objects is the object-oriented enhancement of the ABAP programming language. Read more in the lesson, Developing ABAP Programs, Task: ABAP Programming Language Features, in the course BC400 (Unit 2, Lesson 3) or TAW10 Part I (Unit 8, Lesson 3).

8. How do you include comment lines?

Choose the correct answer.

- A** Begin the line with a pound # in the first column.
- B** Begin the line with an asterisk \* in the first column.
- C** End the line with an asterisk \* in the last column.
- D** End the line with a pound # in the last column.

You are correct! Lines that begin with an asterisk (\*) in the first column are recognized as comment lines by the ABAP runtime system and are ignored. Read more in the lesson, Developing ABAP Programs, Task: General ABAP Syntax II, in the course BC400 (Unit 2, Lesson 3) or TAW10 Part I (Unit 8, Lesson 3).

9. After a project employee completes the required development task, they should do the following:

Choose the correct answer.

- A** Transfer the task to the Production System
- B** Release the task within the Change request
- C** Unlock and export the task to the Transport Directory
- D** Export the Project Request

You are correct! When a development object is edited or created, the relevant employee assigns this to the change request. The object is entered into the task of the employee. All Repository objects that an employee works on during a development project are collected within the employee's task. The task will be released by the employee when all assigned work is done. Read more in the lesson, Organizing ABAP Development Projects, section: Transport of Development Objects, in the course BC400 (Unit 2, Lesson 2) or TAW10 Part I (Unit 8, Lesson 2).

## UNIT 3

# Basic ABAP Language Elements

### Lesson 1

Defining Elementary Data Objects	56
----------------------------------	----

### Lesson 2

Using Basic ABAP Statements	68
-----------------------------	----

### Lesson 3

Analyzing Programs with the ABAP Debugger	78
---	----

### UNIT OBJECTIVES

- Compare data types and data objects
- Explain the ABAP syntax for basic programming concepts
- Analyze values of elementary data objects with the ABAP Debugger

## Unit 3

### Lesson 1

## Defining Elementary Data Objects

### LESSON OVERVIEW

This lesson explains the use of elementary data objects and details how they are defined in ABAP programs.

#### Business Example

When developing ABAP programs, you use various data objects to handle data. Therefore, you require the following knowledge:

- An understanding of how types and variables differ and how they are used.
- An understanding of how local and global data types differ and how they are used.
- An understanding of the use of literals, constants, and text symbols.

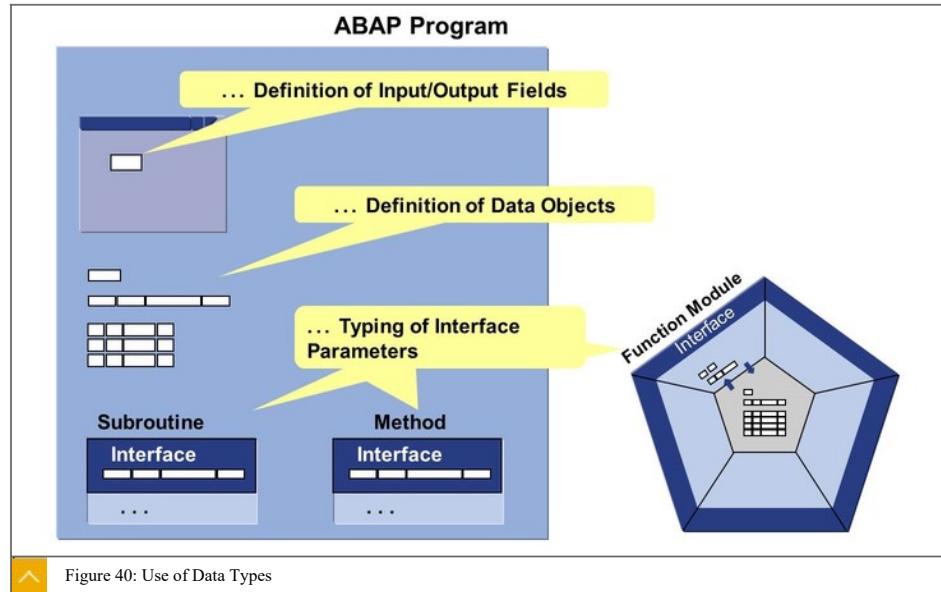


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Compare data types and data objects

### Data Types and Data Objects



A formal variable description is called a data type. A variable or constant that is defined concretely by data type is called a data object.

The figure shows how data types can be used.

#### Use of Data Types

- You can use data types in the following ways:

- **Use data types to define data objects**

The type of a data object defines its technical properties.

- **Use data types to define interface parameters**

The type of an interface parameter determines the type of the actual parameter that is transferred when the modularization unit is called.

- **Use data types to define input/output fields**

The type of an input/output field can provide additional information to the technical characteristics, such as the field and value input help.

This lesson focuses on how to use data types for defining internal program variables.

#### ABAP Standard Data Types

##### Groups of ABAP Standard Data Types

- ABAP standard data types (built-in data types) are divided into two groups:

- Complete

- Incomplete

##### Complete ABAP Standard Data Types

The built-in ABAP standard data types that already contain a type-specific, fixed-length specification are considered complete data types.



Table 1: Complete ABAP Standard Data Types

Standard Types	Description
<b>D</b>	Type for date ( <b>D</b> ), format: <b>YYYYMMDD</b> length 8 (fixed)
<b>T</b>	Type for time ( <b>T</b> ), format: <b>HHMMSS</b> , length 6 (fixed)
<b>I, INT8</b>	Type for integer, either length 4 (fixed) (for <b>I</b> ), or length 8 (fixed) (for <b>INT8</b> )
<b>F</b>	Type for floating point number ( <b>F</b> ), length 8 (fixed)
<b>STRING</b>	Type for dynamic length character string
<b>XSTRING</b>	Type for dynamic length byte sequence (Hexadecimal string)
<b>DECFLOAT16 DECFLOAT34</b>	Types for saving ( <b>DECimal FLOAting point</b> ) numbers with mantissa and exponent, either length 8 bytes with 16 decimal places (fixed) (for <b>DECFLOAT16</b> ) or length 16 bytes with 34 decimal places (fixed) (for <b>DECFLOAT34</b> )

### Incomplete ABAP Standard Data Types

The standard types that do not contain a fixed length are considered incomplete data types. When they are used to define data objects, you need to specify the length of the variable.



Table 2: Incomplete ABAP Standard Data Types

Standard Types	Description
C	Type for character string ( Character) for which the length is to be specified
N	Type for numerical character string ( Numerical character) for which the length is to be specified
X	Type for byte sequence (He Xdecimal string) for which the length is to be specified
P	Type for packed number ( Packed number) for which the length is to be specified (In the definition of a packed number, the number of decimal points might also be specified.)

For more information about predefined ABAP types, refer to the keyword documentation for the TYPES or DATA statement.



Note:

The data types DECFLOAT16 and DECFLOAT34 were introduced in NetWeaver 7.0 EhP2. They combine the advantages of the classic data types P (exact calculation in decimal mode) and F (large value range), while avoiding the drawbacks of both classic data types.

Type INT8 was introduced with SAP AS 7.50. It allows for bigger integer values.

## Local Data Types



## ABAP Program

```

REPORT ... 

Declaration of local data types

TYPES tv_c_type TYPE c LENGTH 8.

TYPES tv_n_type TYPE n LENGTH 5.

TYPES tv_p_type TYPE p LENGTH 3 DECIMALS 2.

```

 Figure 41: Declaration of Local Types

Using standard data types, you can declare local data types in the program that can be more complete or complex than the underlying standard data types. Local data types only exist in the program in question and, therefore, can only be used there. The declaration is made using the TYPES statement.



## Hint:

There is an alternative syntax to specifying the length with the LENGTH addition that you will find in older programs.

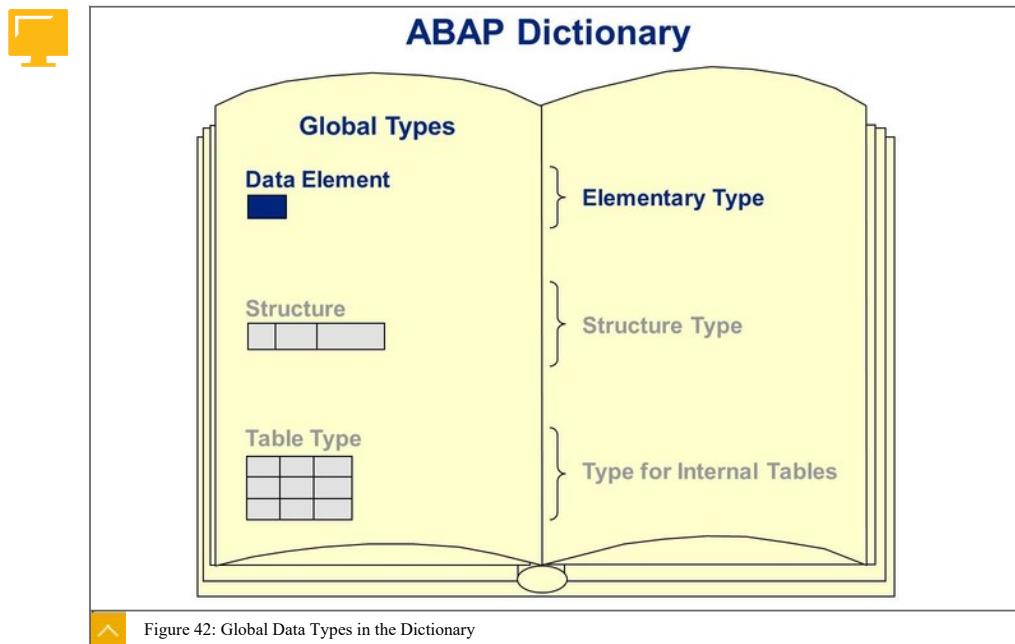
The length is specified in parentheses directly after the name of the type using the following example:

`TYPES tv_c_type(3) TYPE c.`

`TYPES tv_p_type(3) TYPE p DECIMALS 2.`

To improve the readability of your program, you should stop using this obsolete syntax.

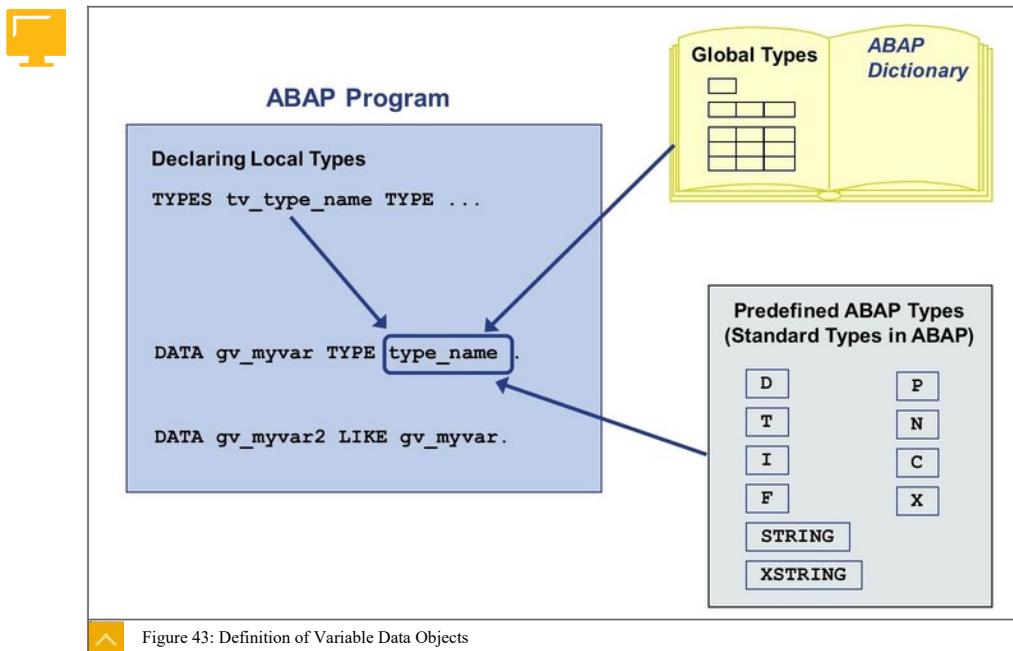
### Global Data Types



A data type defined in the ABAP Dictionary is called global, as it can be used throughout the entire SAP system concerned.

In this lesson, you will only learn how data elements are used as data types for elementary data objects.

## Definition of Variable Data Objects



## Categories of Data Types

- Built-In
- Local
- Global

These types will be used to define variables (data objects).

Data objects are always defined with the DATA keyword. You can use an ABAP standard type, a local type, or a global type to define a data object.

You can refer to a presently defined data object when defining additional variables (LIKE addition).

**Note:**

To increase the readability of your code, SAP recommends that you use naming conventions for names of self-defined types (declared with the TYPES statement) or variables (declared with the DATA statement). Throughout this course we use the following naming convention:

Purpose	Prefix
Program global* type or local** type (no distinction)	tv_
Program global* variable	gv_
Local** variable	lv_

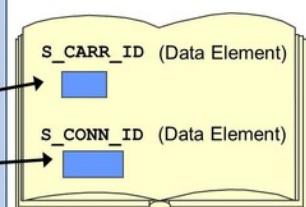
\* Note: Program global in this case means a type or variable that is globally visible within your program. However, it is locally defined in your program. You also can define types as system global (via entries in the Data Dictionary). These type definitions are visible (and usable) within all ABAP programs of your SAP system.

\*\* Note: Local in this case means **local to a subroutine** .

### Examples of the Definition of Elementary Data Objects



```
TYPES tv_percentage  TYPE p LENGTH 3 DECIMALS 2.
DATA: gv_percentage  TYPE tv_percentage,
      gv_number1    TYPE i VALUE 17,
      gv_number2    LIKE gv_number1,
      gv_city       TYPE c LENGTH 15,
      gv_carrid     TYPE s_carr_id,
      gv_connid     TYPE s_conn_id.
```



gv_percentage	0 0 0 0 +
gv_number1	17
gv_number2	0
gv_city	
gv_carrid	
gv_connid	0 0 0 0

Figure 44: Examples of the Definition of Elementary Data Objects

You can use the VALUE addition to pre-assign the value of an elementary data object.

**Hint:**

In the DATA statement, you also have the option of specifying the length in parentheses after the name of the variable, using the following example:

```
DATA gv_myvar(15) TYPE c.  
DATA gv_myvar_p(4) TYPE p DECIMALS 2.
```

However, in order to increase the readability of your program coding, it is recommended that you use the LENGTH addition.

If the length specification is missing from a variable definition, a default length for the (incomplete) standard data type is used (length 1 for types C, N, and X, and length 8 for type P). If the data type is also missing, the standard type C is used. For example, the DATA gv\_myvar statement, having no type and length specification, defines a type C variable with length 1.

**Hint:**

To improve readability, you should always specify the data type and length.

For more details, refer to the keyword documentation for the TYPES or DATA statement.

### Literals, Constants, and Text Symbols



#### Fixed Data Objects Without Label

##### Literals

###### Numeric Literals

Positive Integer : 123  
Negative Integer : -123

###### Text Literals

String : 'Hello'  
Decimal Number : '123.45'  
Floating Point Number : '123.45E01'

#### Fixed Data Objects with Label

##### Constants

```
CONSTANTS gc_myconst TYPE type_name VALUE { literal | IS INITIAL }.
```

Figure 45: Literals and Constants (Fixed Data Objects)

Fixed data objects have a fixed value that is defined when the source text is written and cannot be changed at runtime. Literals and constants belong to the fixed data objects.



Note:

For names of fixed data objects (constants) the following definitions are added to the naming conventions:

Purpose	Prefix
Program global* constant	gc_

\* Note: Program global in this case means a type or variable that is globally visible within your program. However, it is locally defined in your program. You also can define types as system global (via entries in the ABAP Data Dictionary). These type definitions are visible (and usable) within all ABAP programs of your SAP system.

You can use literals to specify fixed values in your programs. There are numeric literals (specified without single quotation marks) and text literals (specified with single quotation marks). In the figure Text Symbols, you find an example for a text literal (the third line of the WRITE statement).

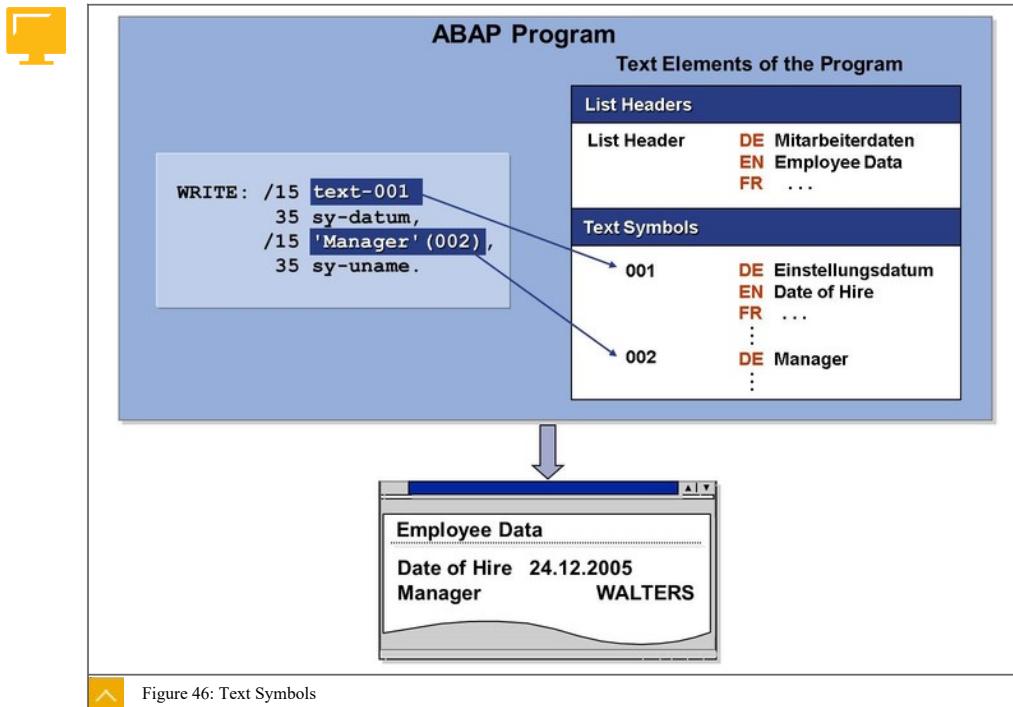
You define constants using the CONSTANTS statement. The TYPE addition is used similarly to the use in the DATA. The VALUE addition is mandatory for constants. With it you define the value of constants.



Hint:

If possible, avoid literals completely when using statements. Use constants and text symbols instead. This makes it considerably easier to maintain your program.

## Text Symbols



A very important principle in ABAP development is to include multilingual capability. This means that the logon language of the current user is taken into account when texts are displayed on the user interface. The use of text literals in this context is critical, because the literal exists in only one version in the source code and is language independent.

Only create language-dependent texts as text literals for testing purposes. For production programs that are executable with various logon languages, the ABAP programming language provides the text symbols.

Each text symbol is used in the particular program to which it belongs. These symbols are stored outside the source code in their own Repository object, the text pool for the program. Text symbols can be translated into various languages and each of them is stored with a language indicator in the text pool, as shown in the figure. If the program accesses a text symbol when it is executed, the system takes account of the logon language of the user and supplies the text in this language.

A text symbol is identified by means of a three-character alphanumeric ID, '-xxx'.

To use a text symbol in your program, you need to address it as `TEXT-xxx`, where `xxx` stands for the three-character text symbol ID.

To make specifying a text symbol more intuitive, you can also use the following syntax instead of `TEXT-xxx`: `'...'(xxx)`. In this case, `'...'` should be the text of the text symbol in the original language of the program.

### Definition of Text Symbols for Programs

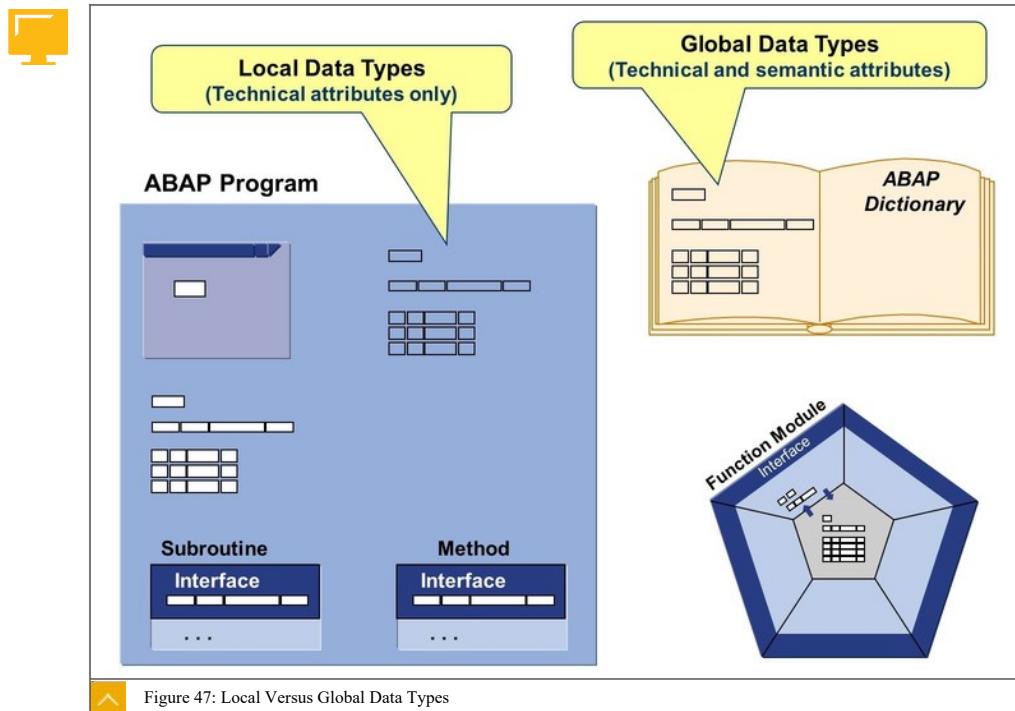
#### Options for Defining Text Symbols for Programs

- In the ABAP Editor, choose Goto → Text Elements ; tab Text Symbols .
- Address the text symbol in your source code using the syntax TEXT-xxx: '...'!(xxx) and double-click its ID (forward navigation).
- To translate the text symbols of your program, choose Goto → Translation from the menu in the ABAP Editor .



Hint:  
Remember that text elements also have to be activated.

### Comparison – Local and Global Data Types



Local data types can only be used in the program in which they are defined. Global data types, in contrast, can be used throughout the entire system.

#### Advantages of Global Data Types

- Global data types have the following advantages:
  - System-wide use increases the system's consistency and reduces maintenance.

- Use of a where-used list includes repository objects that use the data type in question.
- Semantic information in the data types corresponds to the business descriptions of the objects to be defined.
- Global data types can be used throughout the system, which increases the system's consistency. The fact that they can be reused reduces the amount of maintenance efforts required.
- Global data types can have a where-used list generated in the ABAP Dictionary. The where-used list includes repository objects that use the data type in question.
- Global data types can contain semantic information that corresponds to the business descriptions of the objects to be defined in addition to technical information. They can then also be used for designing screen displays (for example, the label on the left of the input field).

Local data types should be defined if they are only required in one program, and only if the semantic information does not matter for the definition of the corresponding data objects.



#### LESSON SUMMARY

You should now be able to:

- Compare data types and data objects

## Unit 3

### Lesson 2

# Using Basic ABAP Statements

#### LESSON OVERVIEW

This lesson explains how to fill elementary data objects with values, and how to perform calculations in ABAP. You will also be given an introduction to the constructions you can use to control the program flow dependent on the content of data objects.

#### Business Example

You are to create a simple ABAP program for basic calculation types. You must be able to enter values and the arithmetic operator. For this reason, you need to know:

- How to fill elementary data objects with values
- How to perform calculations in ABAP

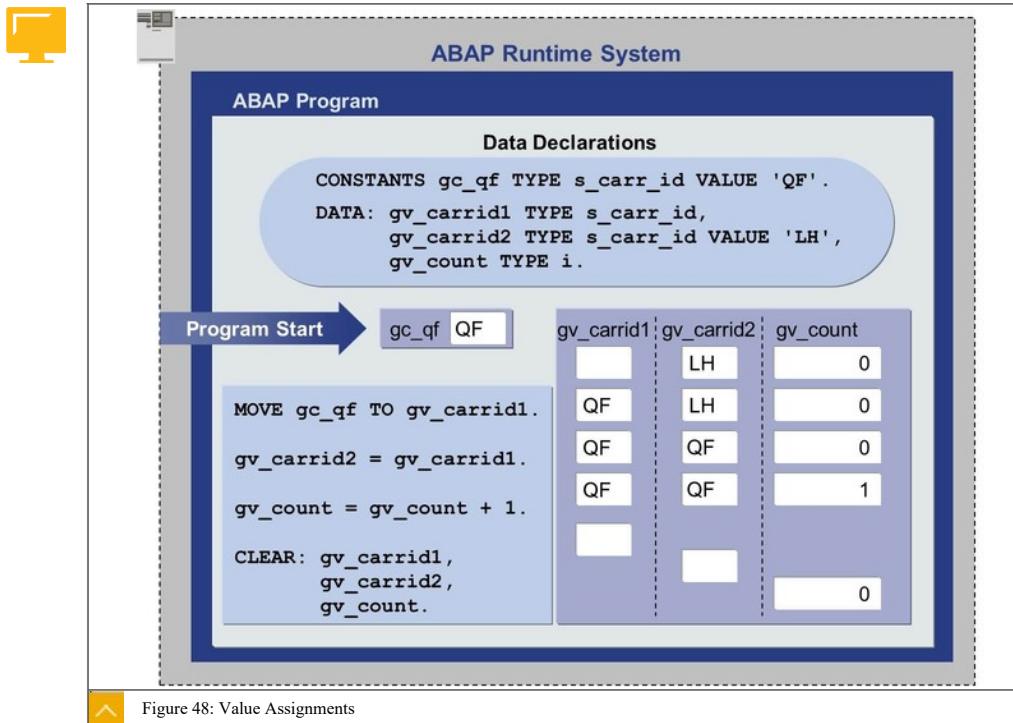


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain the ABAP syntax for basic programming concepts

## Value Assignments



When you start a program, the program context is loaded into a memory area of the application server and memory is made available for the data objects defined in the program. Every elementary data object is pre-assigned the type-specific initial value, except if a different value was set using the VALUE addition.

## Syntax Variants for the MOVE Statement

- You can use the MOVE statement to transfer the contents of a data object to another data object. The following syntax variants have the same effect:

- MOVE gv\_var1 TO gv\_var2.
- gv\_var2 = gv\_var1.

If the two data objects `gv_var1` and `gv_var2` are of different types, there is a type conflict. In this case, a type conversion is carried out automatically if a conversion rule exists. For detailed information about copying and the conversion rules, refer to the keyword documentation.

The CLEAR statement resets the content of a data object to the type-related initial value. For detailed information about the initial values for a particular type, refer to the keyword documentation for the CLEAR statement.

### Calculations and Arithmetic Expressions



### ABAP Program

**Data Declarations**

```
DATA: gv_max      TYPE sbc400focc-seatsmax,
      gv_occ       TYPE sbc400focc-seatsocc,
      gv_percentage TYPE sbc400focc-percentage,
      gv_length     TYPE i,
      gv_string     TYPE string.
```

**Assignment with arithmetic expression:**

```
gv_percentage = gv_occ * 100 / gv_max.
```

**Assignment with result of predefined function:**

```
gv_length = strlen( gv_string ).
```

 Figure 49: Calculations

### Valid Operators in ABAP Programs



- In ABAP, you can program arithmetic expressions nested to any depth.
  - + (Addition)
  - - (Subtraction)
  - \* (Multiplication)
  - / (Division)
  - \*\* (Exponentiation)
  - DIV (Integral division without remainder)
  - MOD (Remainder after integral division)



#### Note:

Parentheses and operators are ABAP keywords; therefore, you must separate them from other words or symbols by at least one space.

Several functions for different data types are predefined in the ABAP runtime environment. For example, the following statement provides the current length of the content of a character variable.

```
gv_length = STRLEN (gd_cityfrom) .
```

In the case of functions, the opening parenthesis is part of the function name. The rest must be separated by at least one space.

Standard algebraic rules apply to the processing sequence. Expressions in the parentheses come first, followed by functions, powers, multiplication and division, and addition and subtraction.

For more information about the available operations and functions, refer to the keyword documentation for the Assignment Operator =.

### Conditional Branches and Logical Expressions

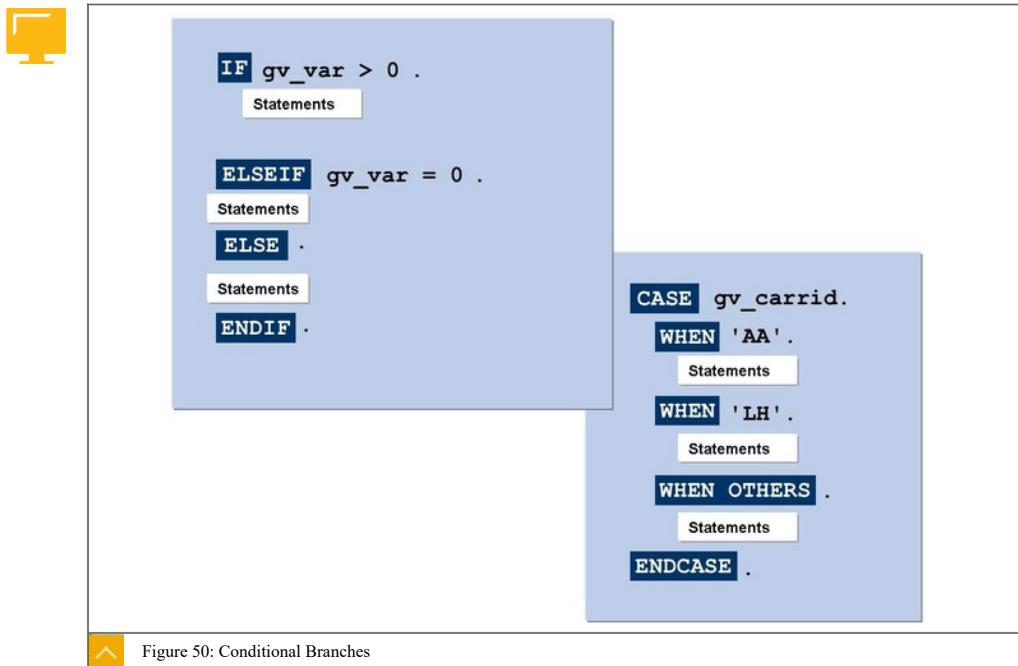


Figure 50: Conditional Branches

#### Execution of Code Based on Conditions

- ABAP offers the following ways to execute different sequences of code depending on conditions:
  - In the **IF construct**, you can define any logical expressions as check conditions.
  - In the **CASE construct**, you can clearly distinguish cases.
- In the **IF construct**, you can define any logical expressions as check conditions. If the condition is met, the system executes the relevant statement block. Otherwise, the condition specified in the next ELSEIF branch (several branches are possible) is checked. If none of the specified conditions are fulfilled, the ELSE branch executes, provided it exists. The ELSEIF and ELSE branches are optional. For more information about formulating a logical expression, refer to the keyword documentation on the IF statement.
- In the **CASE construct**, you can clearly distinguish cases. The content of the field specified in the CASE part is checked against the data objects listed in the WHEN branch to see whether they match. If the field contents match, the respective statement block is

processed. If no comparison is successful, the system executes the OTHERS branch if it is available. Except for the first WHEN branch, all further branches are optional.

In both scenarios, the condition, or match check, happens sequentially from the top down. When the statement block of a branch executes, the system does not process the ENDIF or ENDCASE statements.



Hint:

If you want to implement similarity checks between a field and different values, you should use the CASE construct in preference to the IF statement, because it is more transparent and performs better.

IF Statement



**Negation**

```
IF gv_carrid IS NOT INITIAL.  
  Statements  
ELSE.  
  Statements  
ENDIF.
```

**AND and OR Links with Parentheses**

```
IF ( gv_carrid = 'AA' OR gv_carrid = 'LH' )  
  AND gv_fldate = sy-datum.  
  Statements  
ELSEIF ( gv_carrid = 'UA' OR gv_carrid = 'DL' )  
  AND gv_fldate > sy-datum.  
  Statements  
ENDIF.
```

**Negation Before Logical Conditions**

```
IF NOT ( gv_carrid = 'AA' OR gv_carrid = 'UA' )  
  AND gv_fldate > sy-datum.  
  Statements  
ENDIF.
```

Figure 51: Examples - IF Statement

The figure shows several simple examples (negation, AND and OR links with parentheses, and negation before logical conditions) of using the IF statement.

You can formulate negations by placing the NOT operator before the logical expression. When negating the IS INITIAL query, you can use the special IS NOT INITIAL query.



Hint:

A statement "IF NOT var IS INITIAL" is a valid statement as well (correct syntax). However, to increase the readability of your programming code SAP recommends to not use this statement or syntax.

## Advanced Examples - IF Statement



**Predefined functions**

```
IF abs( gv_val1 ) > gv_val2.  
  Statements  
ELSEIF charlen( gv_str ) BETWEEN gv_lo AND gv_hi.  
  Statements  
ENDIF.
```

**Arithmetic expressions**

```
IF gv_val1 * -2 >= ( gv_val2 - 3 ) * 6.  
  Statements  
ELSEIF sqrt( abs( gv_val1 ) ) / 3 + 7 GT  
  ipow( base = 2 exp = 3 ).  
  Statements  
ENDIF.
```

**String expression**

```
IF 'lit1' && gv_str CA 'abc'.  
  Statements  
ENDIF.
```

 Figure 52: Advanced Examples - IF Statement

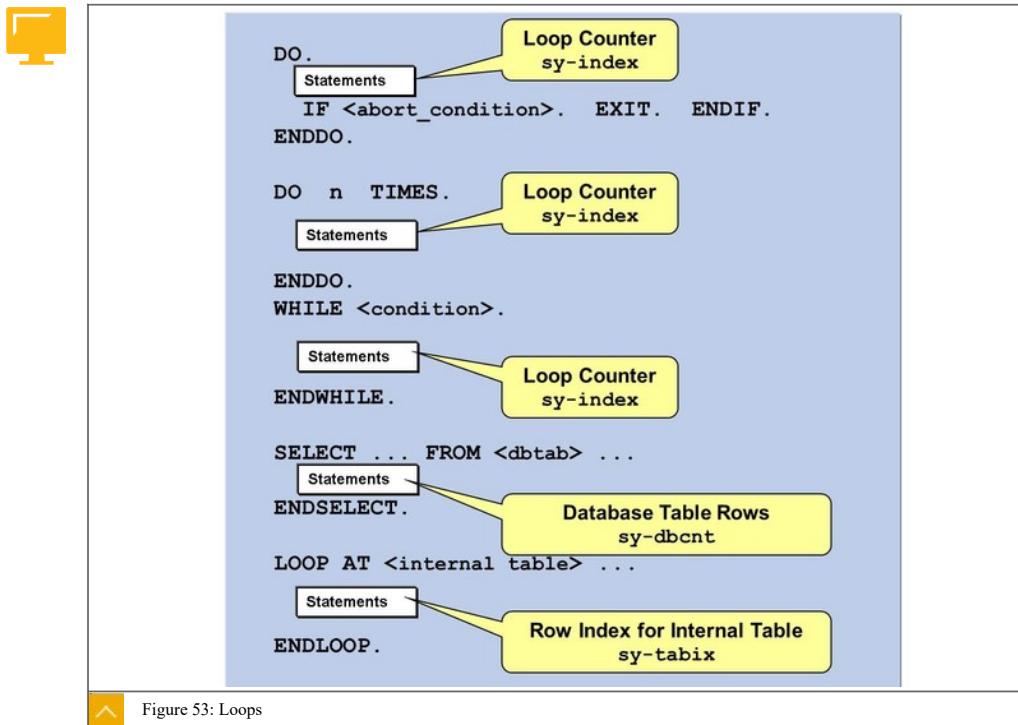
The figure shows some advanced examples (predefined functions, arithmetic expressions, and string expressions) of using the IF statement.

The following can be used (among others) to specify a logical expression: data objects, predefined functions, arithmetic expressions, and string expressions.

You can nest the IF and CASE structures in any way, but make sure that the logic of every structure is correct. Every structure must be closed, for example:

```
IF <log. condition1> . . . .  
IF <log. condition2> . . . .  
ENDIF.  
ENDIF.
```

## Loops



### Loop Constructs

There are four loop constructs in ABAP. The system field will make sense only if queried within a loop. In nested loops, sy-index always contains the loop pass number of the loop in which it is located. In the DO and WHILE loops, the sy-index system field contains the number of the current loop pass. Therefore, you should only query this system field within a loop.

- Unconditional and index-controlled loops

The statement block between DO and ENDDO executes continuously until the loop is left using termination statements such as EXIT. Specify the maximum number of loop passes; otherwise, you might get an endless loop.

- Header-controlled loops

The statement block between WHILE and ENDWHILE continuously executes until the specified condition is no longer met. The condition is always checked before executing the statement block.

- Read loops

The SELECT loop reads several entries of a database table in succession. In an internal table (table variable in the program), the same read function is implemented with the LOOP.

### System Fields (Excerpt)



Table 3: Some Useful System Fields

System Field	Meaning
sy-mandt	Logon Client
sy-uname	Logon Name of the User
sy-langu	Logon Language of the User
sy-datum	Local Date of the ABAP System
sy-uzeit	Local Time of the ABAP System
sy-tcode	Current Transaction Code
sy-repid	Name of the Current ABAP Program
sy-index	Loop Counter at DO and WHILE Loops

In the ABAP source code, you can use several data objects without explicitly declaring them previously (for example, sy-datum and sy-index). The runtime system uses these system fields to provide the application program with information about the actual system status. The table shows several system fields.

You can find a complete list of system fields in the keyword documentation under the term System Fields.

To access system fields in your programs, use read-only access. Write access can result in the loss of important information for program parts that require this information. In addition, the runtime system might change the field content again. Therefore, SAP recommends to only read these fields.

Return Code of an ABAP Statement



```
REPORT ...  
  
PARAMETERS pa_carr TYPE scarr-carrid.  
  
DATA gs_scarr TYPE scarr.  
  
SELECT SINGLE * FROM scarr  
      INTO gs_scarr  
      WHERE carrid = pa_carr.  
  
IF sy-subrc = 0.  
  
  NEW-LINE.  
  
  WRITE: gs_scarr-carrid,  
         gs_scarr-carrname,  
         gs_scarr-url.  
ELSE.  
  
  WRITE 'Sorry, no data found!'.  
  
ENDIF.
```



Figure 54: Return Code of an ABAP Statement

One of the most important system fields is the `sy-subrc` field. With many statements, it is supplied by the ABAP runtime system with the corresponding return code to indicate whether the statement could be executed successfully. The value zero means that the statement was executed successfully. Read the keyword documentation for the respective statements to find out if and how this return value is set in individual cases.

## Dialog Messages



<code>MESSAGE tnnn(message_class) [ WITH v1 [ v2 ] [ v3 ] [ v4 ] ] .</code>			
Type	Meaning	Dialog behavior	Message appears in
i	Info Message	Program continues after breakpoint	Modal dialog box
s	Set Message	Program continues without breakpoint	Status bar *) of next screen
w	Warning	Context-dependent	Status bar *)
e	Error	Context-dependent	Status bar *)
a	Termination	Program terminated	Modal dialog box
x	Short Dump	Runtime error MESSAGE_TYPE_X is triggered	Short dump

\*) Display in modal dialog box also possible through GUI settings



Figure 55: Dialog Messages

Use the MESSAGE statement to send dialog messages to the users of your program. Specify the three-digit message number and the message class when using the MESSAGE statement.

Message number and message class clearly identify the message to be displayed. Use the message type to specify where you want the message to be displayed. You can test the display behavior for various message types by means of the DEMO\_MESSAGES demonstration program that is delivered with the SAP standard system.

If the specified message contains placeholders, supply them with values from your program by using the WITH addition. Instead of the placeholders, the transferred values appear in the displayed message text.

For further information about syntactical alternatives to the MESSAGE statement, refer to the keyword documentation.



## LESSON SUMMARY

You should now be able to:

- Explain the ABAP syntax for basic programming concepts

## Unit 3

### Lesson 3

## Analyzing Programs with the ABAP Debugger

### LESSON OVERVIEW

This lesson explains the workings of the ABAP Debugger for elementary data objects.

#### Business Example

As an ABAP programmer, you develop programs on ABAP. You debug the programs that you develop. For this reason, you require the following knowledge:

- An understanding of the workings of the ABAP Debugger for elementary data objects

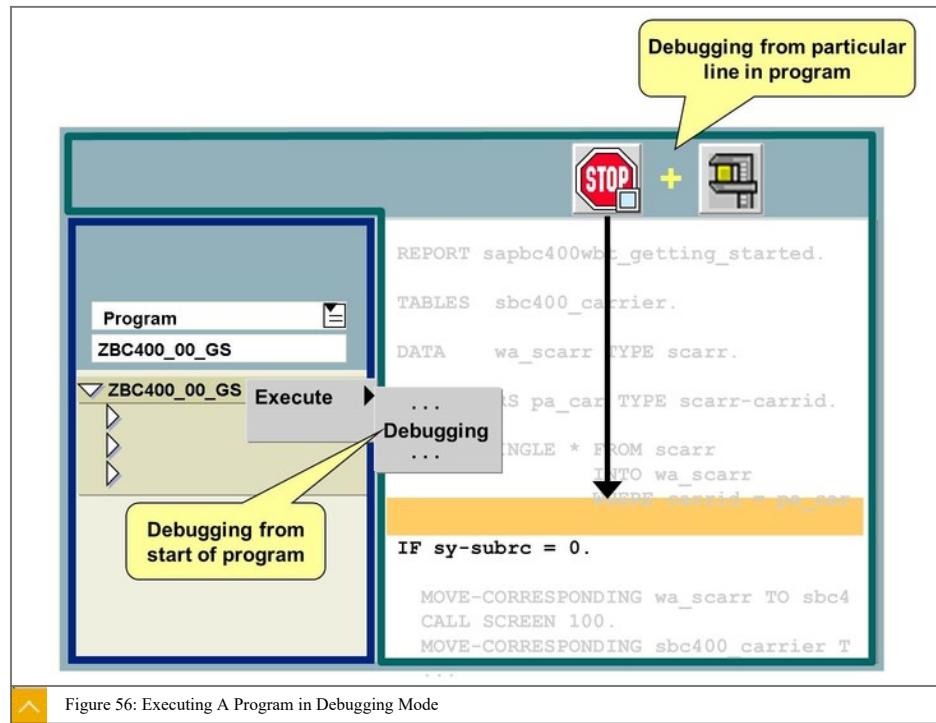


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Analyze values of elementary data objects with the ABAP Debugger

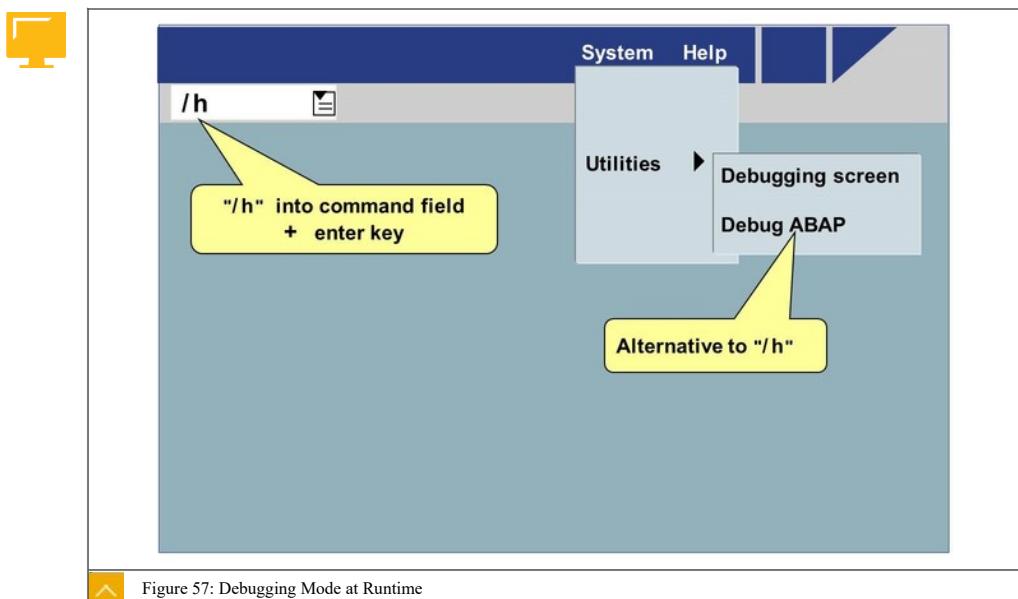
### ABAP Debugger



### Methods for Starting a Program in Debugging Mode

- The two methods used to start a program in debugging mode from the Object Navigator are as follows:
  - In the navigation area, open the context menu for the selected program, and choose Execute → Debugging.
  - In the ABAP Editor area, select the requested program line from which you want to debug. Choose the Set or Delete breakpoint button. Start the program by pressing F8 or by opening the context menu in the navigation area and choosing Execute → Direct. The setting of a breakpoint in the ABAP Editor is only possible for active source texts.

### Debugging Mode at Runtime



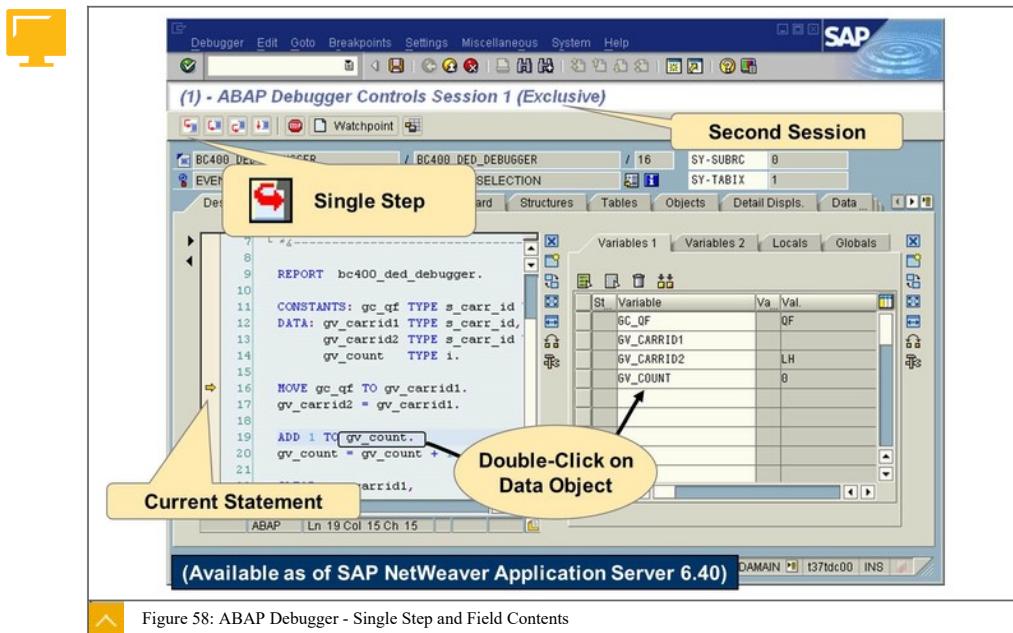
### Ways to Switch to Debugging at Runtime

If you want to debug a certain function of a program, start the program without the ABAP Debugger, and then switch to debug mode immediately before executing the function (for example, button).

#### Ways to Switch to Debugging at Runtime

- Choose System → Utilities → Debugging ABAP (or screen).
- Enter /h in the command field in the standard toolbar, and press Enter .

## ABAP Debugger – Single Step and Field Contents



In the ABAP Debugger, you can choose single-step processing to execute the program statement-by-statement. Moreover, you can display data objects and their current content in the variable display. Simply enter the name of the data object in the variable display. Alternatively you can double-click on the data object name in the source code to transfer the variable name into the debugger field list.



## Note:

In systems using a release previous to SAP NetWeaver AS 6.40, you only have access to the "classic" ABAP Debugger. With more recent release levels, you can use both the new and the classic ABAP Debugger. You can switch easily from the new to the classic ABAP Debugger by choosing **Debugger → Switch to Classic ABAP Debugger** from the menu.

In the classic ABAP Debugger, you can display the contents of up to eight data objects. To do this, proceed as you would with the new ABAP Debugger.

## ABAP Debugger – Breakpoints

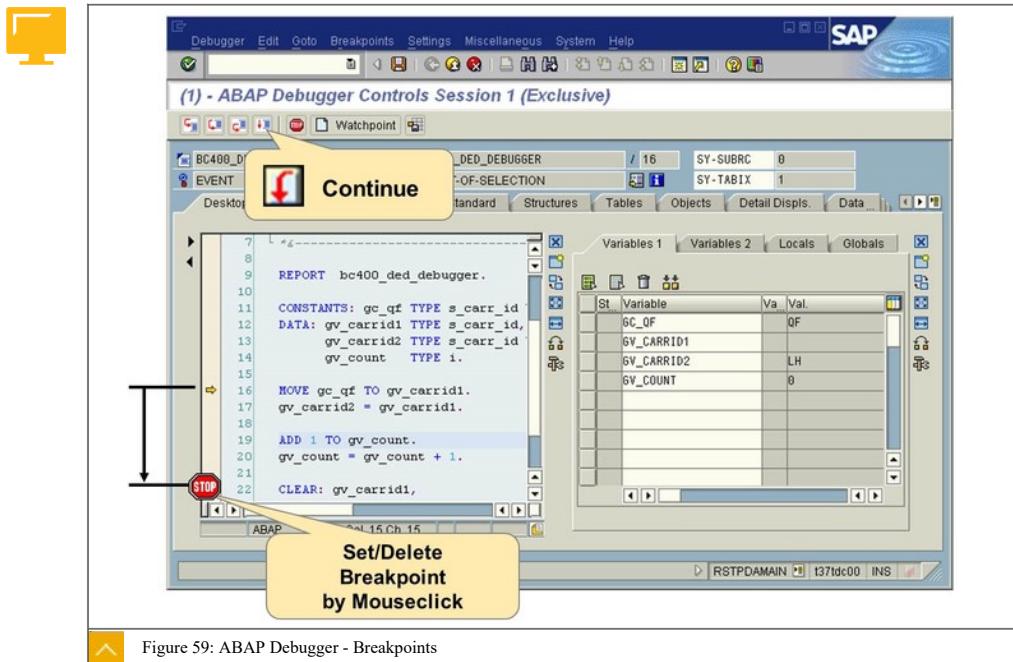


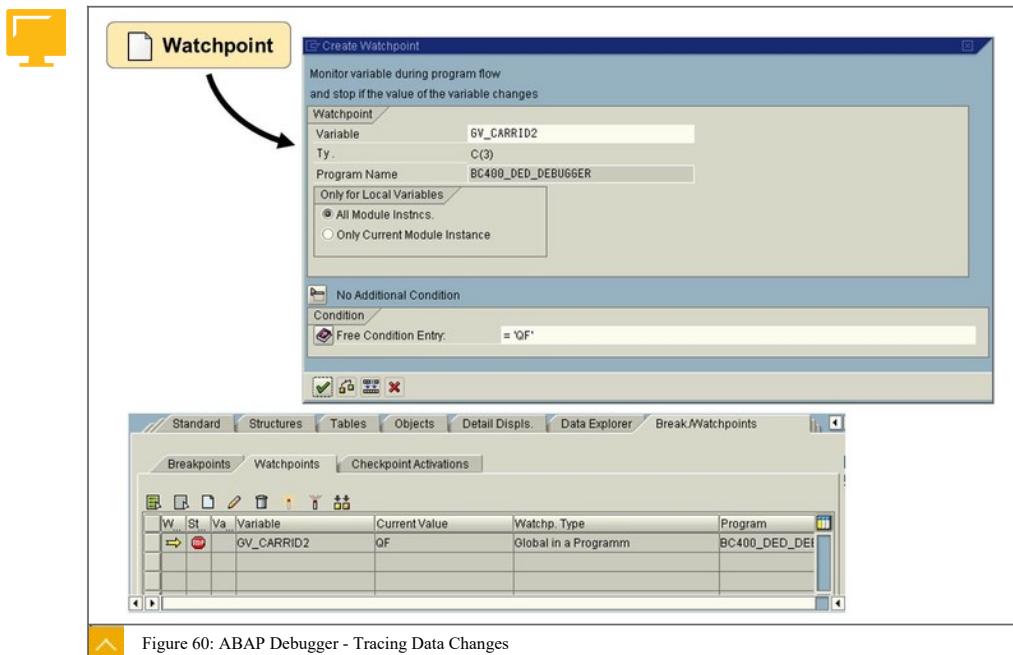
Figure 59: ABAP Debugger - Breakpoints

In the new ABAP Debugger, you can set a breakpoint with a single-click before the line in the source code (in the classic ABAP Debugger, you do this with double-click).

You can also set a breakpoint for specific ABAP statements by choosing Breakpoints → Breakpoint at → Statement . If you choose Continue , the program executes up to the next breakpoint.

The set breakpoints are only valid for the current ABAP Debugger session. However, if you choose Save , the breakpoints will stay in place for the duration of your current SAP session.

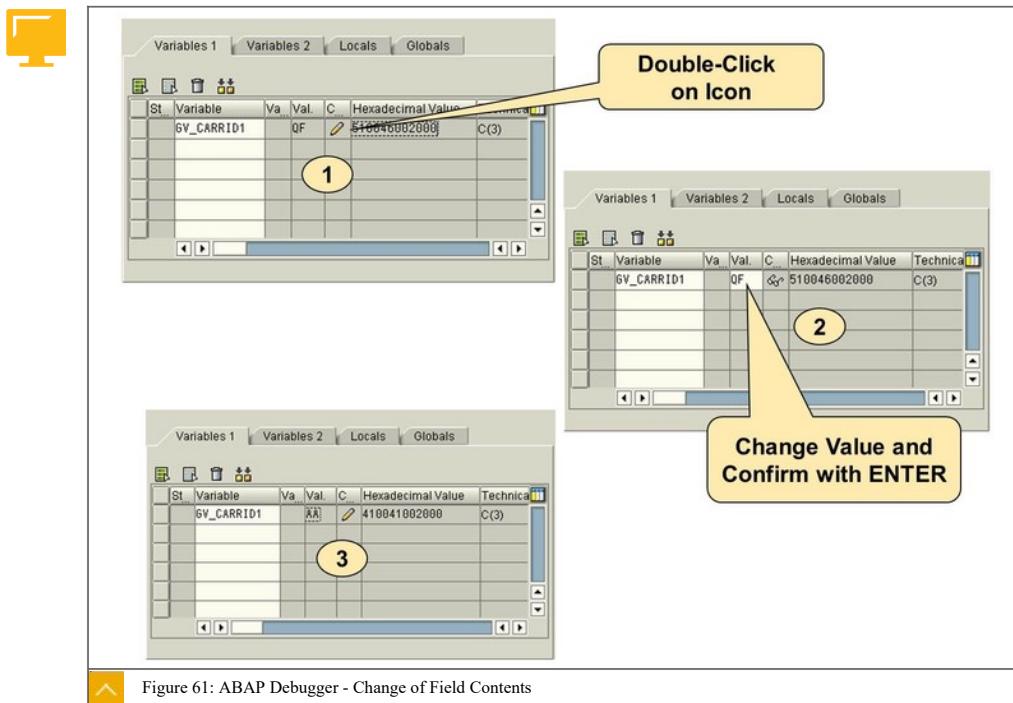
## ABAP Debugger – Tracing Data Changes



## Watchpoints

- When working with breakpoints, consider the following:
  - Watchpoints are breakpoints that depend on the field content.
  - If you set a watchpoint without specifying a relational operator and a comparative value on a field and choose **Continue**, the program executes until the content of the field changes.
  - If you specify the relational operator and the comparative value, the program executes until the specified condition is met.
  - In the classic ABAP Debugger, you can set a maximum of 10 watchpoints. You can link the watchpoints using a logical operator (AND or OR) in the same way as in the new ABAP Debugger.

## ABAP Debugger – Change of Field Contents



If you want to change the content of a field during debugging, double-click the pencil icon in the variable display. The value is then ready for input. Now change the field value and confirm it by pressing ENTER. The value changes while the ABAP Debugger is running.

In the classic ABAP Debugger, you can change the content directly in the field view. You then click the pencil icon to accept the changed value.

## Additional Functions of the ABAP Debugger

## ABAP Debugger: Additional Functions



- The ABAP Debugger includes the following additional functions:
  - ABAP Debugger in second session  
Inputs for the application program are visible in parallel
  - Parallel display options through freely configurable display areas
  - Integration of the new ABAP Editor
  - Quick Info in the source code area for data object value display
  - Watchpoints for internal tables and object references
  - Value comparison of strings, structures, and internal tables
  - Enhanced table tool

- Enhanced Web Dynpro tool

#### Advanced Features of ABAP Debugger

- The ABAP Debugger has the following advanced features:
  - Software-layer aware debugging (SLAD)
  - Automated debugging-debugger scripting
  - Memory consumption analysis



#### LESSON SUMMARY

You should now be able to:

- Analyze values of elementary data objects with the ABAP Debugger

## Unit 3

### Learning Assessment

1. Which of the following are complete ABAP standard types?

Choose the correct answers.

- A T (Time)
- B C (Character)
- C N (Numerical character)
- D INT8 (Integer with Length of 8 bytes)
- E D (Date)

2. Which of the following is the operator used in an IF statement to formulate a negation before logical conditions?

Choose the correct answer.

- A AND
- B OR
- C END
- D NOT

3. Which of the following are required in the syntax of the Message statement?

Choose the correct answers.

- A Message number
- B Message type
- C Message class
- D Message role

4. In nested loops, which of the following contains the loop pass number of the loop in which it is located?

Choose the correct answer.

**A** sy-index

**B** sy-repid

**C** sy-uname

**D** sy-mandt

5. Which of the following is the system command you can enter in the command field of a screen to start the debugger?

Choose the correct answer.

**A** /d

**B** /h

**C** /i

**D** /a

## Unit 3

### Learning Assessment - Answers

1. Which of the following are complete ABAP standard types?

Choose the correct answers.

- A T (Time)
- B C (Character)
- C N (Numerical character)
- D INT8 (Integer with Length of 8 bytes)
- E D (Date)

You are correct! The built-in ABAP standard data types that already contain a type-specific, fixed-length specification are considered complete data types. They are: D, T, I, INT8, F, STRING, XSTRING, DECFLOAT16, DECFLOAT34. Read more in the lesson, Defining Elementary Data Objects, Task: Complete ABAP Standard Data Types, in the course BC400 (Unit 3, Lesson 1) or TAW10 Part I (Unit 9, Lesson 1).

2. Which of the following is the operator used in an IF statement to formulate a negation before logical conditions?

Choose the correct answer.

- A AND
- B OR
- C END
- D NOT

You are correct! You can formulate negations by placing the NOT operator before the logical expression. When negating the IS INITIAL query, you can use the special IS NOT INITIAL query. Read more in the lesson, Using Basic ABAP Statements, Task: IF Statement, in the course BC400 (Unit 3, Lesson 2) or TAW10 Part I (Unit 9, Lesson 2).

3. Which of the following are required in the syntax of the Message statement?

Choose the correct answers.

- A** Message number  
 **B** Message type  
 **C** Message class  
 **D** Message role

You are correct! Specify the three-digit message number and the message class when using the MESSAGE statement. Use the message type to specify where you want the message to be displayed. Read more in the lesson, Using Basic ABAP Statement, Task: Dialog Messages, in the course BC400 (Unit 3, Lesson 2) or TAW10 Part I (Unit 9, Lesson 2).

4. In nested loops, which of the following contains the loop pass number of the loop in which it is located?

Choose the correct answer.

- A** sy-index  
 **B** sy-repid  
 **C** sy-uname  
 **D** sy-mandt

You are correct! In nested loops, sy-index always contains the loop pass number of the loop in which it is located. In the DO and WHILE loops, the sy-index system field contains the number of the current loop pass. Therefore, you should only query this system field within a loop. Read more in the lesson, Using Basic ABAP Statement, Task: Loop Constructs, in the course BC400 (Unit 3, Lesson 2) or TAW10 Part I (Unit 9, Lesson 2).

5. Which of the following is the system command you can enter in the command field of a screen to start the debugger?

Choose the correct answer.

- A** /d  
 **B** /h  
 **C** /i  
 **D** /a

You are correct! To start the debugger from a screen, enter /h in the command field, and press Enter. Read more in the lesson, Analyzing Programs with the ABAP Debugger, Task: Debugging Mode at Runtime, in the course BC400 (Unit 3, Lesson 3) or TAW10 Part I (Unit 9, Lesson 3).

## UNIT 4

# Modularization Techniques in ABAP

### Lesson 1

Explaining Modularization	91
---------------------------	----

### Lesson 2

Defining and Calling Subroutines	97
----------------------------------	----

### Lesson 3

Calling Function Modules	109
--------------------------	-----

### Lesson 4

Creating Function Modules	118
---------------------------	-----

### Lesson 5

Describing Business Application Programming Interfaces (BAPIs)	121
--	-----

### Lesson 6

Calling Methods of Global Classes	126
-----------------------------------	-----

### Lesson 7

Creating Global Classes and Static Methods	140
--	-----

### Lesson 8

Using Local Classes	144
---------------------	-----

### UNIT OBJECTIVES

- Describe techniques of modularization
- Modularize using subroutines
- Use function modules of function groups
- Create function groups

- Create function modules
- Modularize using BAPIs
- Describe object-oriented programming
- Use methods of global classes
- Use instances
- Create simple global classes and static methods
- Use local classes

## Unit 4

### Lesson 1

## Explaining Modularization

### LESSON OVERVIEW

This lesson explains reasons why it could be reasonable to store parts of programs in modularization units. You can also gain an overview of the various modularization options in ABAP programs.

### Business Example

An employee in quality assurance has discovered that many program parts are frequently repeated. Your task is to find out which modularization techniques can be implemented. For this reason, you require the following knowledge:

- An understanding of the basic modularization techniques

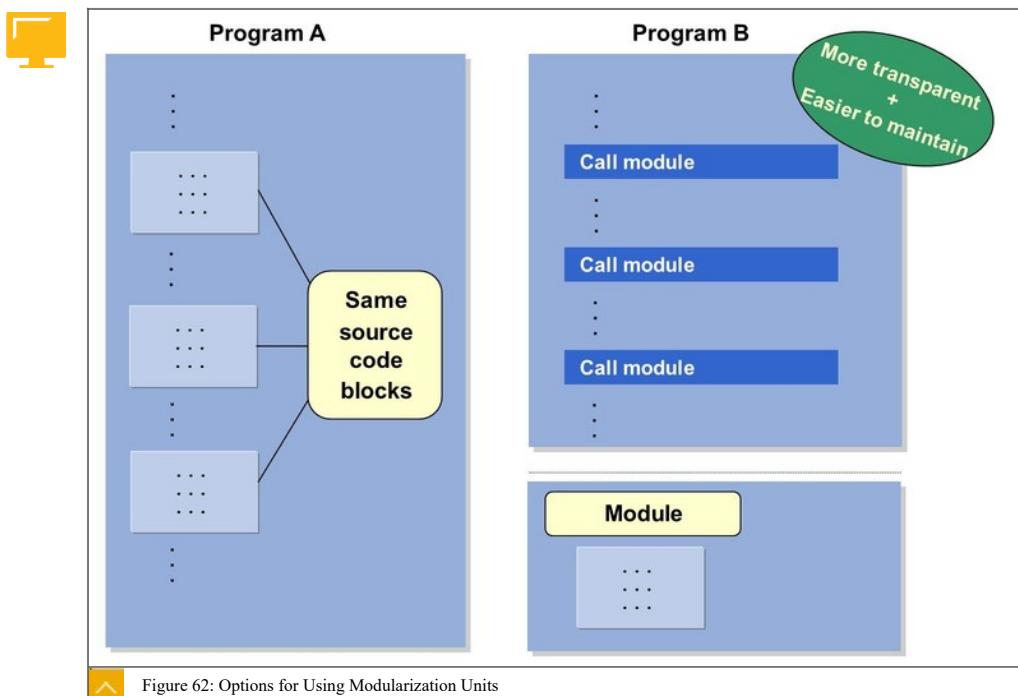


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe techniques of modularization

### Modularization Techniques

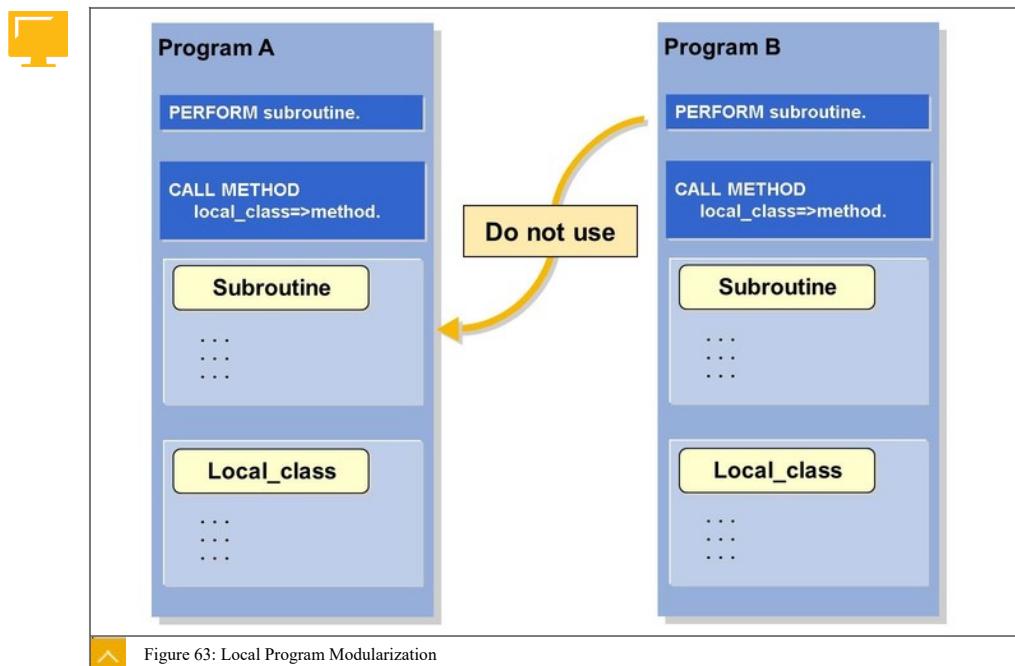


A modularization unit is a part of a program that encapsulates a particular function. You store part of the source code in a module to improve the transparency of the program, as well as to use the corresponding function in the program several times without having to re-implement the entire source code on each occasion (see the figure).

The improvement in transparency is a result of the program becoming more function-oriented. It divides the overall task into sub-functions, which are the responsibility of corresponding modularization units.

Modularization makes it easier to maintain programs, because you only need to make changes to the function or corrections in the respective modularization units, and not at various points in the main program. Furthermore, you can process a call as a unit in the ABAP Debugger while executing your program and then see the result. This makes it easier to find the source of an error.

#### Local Program Modularization



#### Techniques for Local Program Modularization in the ABAP Programming Language

- Subroutines, also known as form routines
- Methods in local classes

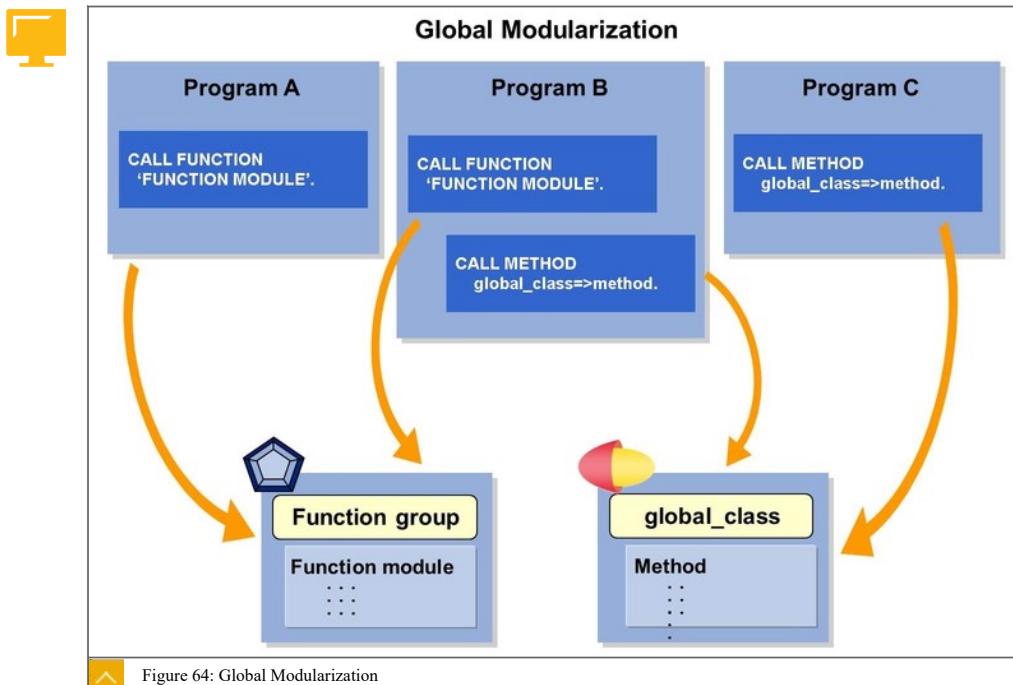
With both local modularization techniques, modularization units are only available in the program in which they were implemented. To call the local module, no other program must be loaded to the user context at runtime. Local classes, methods, and subroutines can have the same name in different programs without producing conflicts. This is because the source code for the programs is handled separately in the main memory of the application server.



## Hint:

It is technically possible to call a subroutine from another program. Do not use this option, however, because this technique contradicts the principle of encapsulation of data and functions.

## Global Modularization

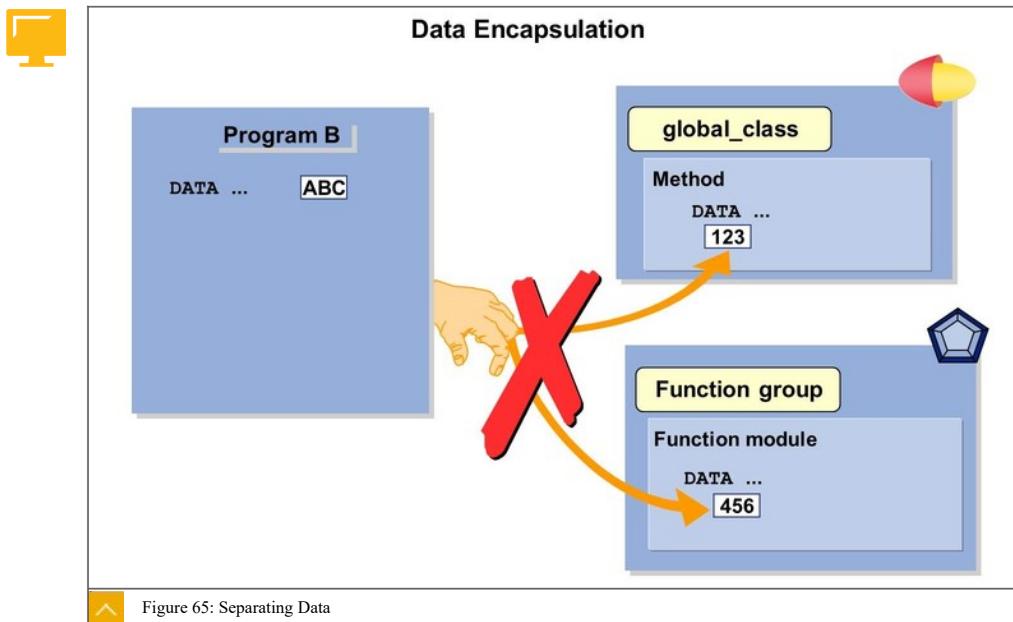


## Techniques for Global Modularization in the ABAP Programming Language

- Function modules organized in function groups
- Methods in global classes

Globally defined modularization units can be used by any number of programs at the same time. These modularization units are stored centrally in the Repository and loaded when called into the context of the calling program.

### Data Encapsulation

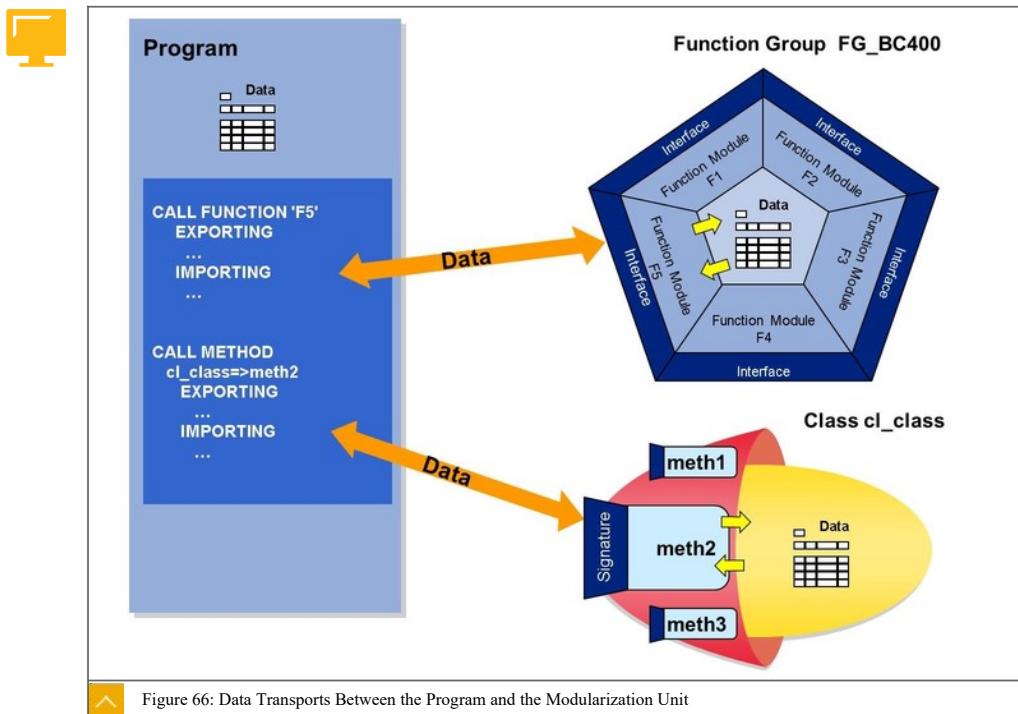


Ideally, the modularization units that are called do not directly use the data objects of the calling program. Conversely, the calling program does not directly change the data in the modularization units. This principle is known as data encapsulation.

Data encapsulation is an important aid in developing transparent, maintainable source codes. Data encapsulation makes it far easier to determine where in the program the contents of data objects were changed. In addition, data encapsulation makes it easier to ensure that data within the modularization units changes consistently when, for example, the contents of several data objects within a modularization unit are mutually dependent.

As an example, consider a modularization unit in which you process a series of invoice documents. Entering different invoice numbers for invoice items that belong to one another can have serious consequences. It can then be difficult for those responsible for the modularization unit to determine the cause of the data inconsistency.

## Data Transports, Parameters, and Interface



Parameters are used to exchange data between the program and the module. The total number of parameters in a modularization unit is referred to as the interface, or signature. The developer who defines the modularization unit also determines the parameters.

## Criteria for Differentiating Parameters

- Parameters are differentiated on the basis of whether they are used to do the following:
  - Pass data to the modularization unit (importing parameters).
  - Return data from the modularization unit to the caller (exporting parameters).
  - Pass data to the modularization unit and return the data after it has been changed (changing parameters).



## Hint:

With subroutines (form routines), only changing parameters and the very specialized using parameters are available, which severely restricts your options for controlling data transport. For this reason, use local classes for local program modularization, where possible.



### LESSON SUMMARY

You should now be able to:

- Describe techniques of modularization

## Unit 4

### Lesson 2

## Defining and Calling Subroutines

### LESSON OVERVIEW

This lesson explains how to employ subroutines in ABAP programs. It also shows how the interface of a subroutine is used to pass parameters, and how various transfer types are used.

#### Business Example

You need to structure your program and encapsulate the source code that is executed several times in a subroutine. For this reason, you require the following knowledge:

- An understanding of how to define and call subroutines
- An understanding of the execution of subroutines in debugging mode



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Modularize using subroutines

### Modularization Using Subroutines within Programs



```
:
PERFORM write_list.

PERFORM write_list.

FORM write_list.
  WRITE 'List of Airlines'.
  SKIP.
  WRITE / 'LH Lufthansa'.
  WRITE / 'AA American Airlines'.
  WRITE / 'UA United Airlines'.
  SKIP. ULINE.

ENDFORM.
```

Main Program  
(Framework Program)

Subroutine

Figure 67: Simple Example of a Subroutine

A subroutine is a modularization unit within a program. For the ABAP interpreter, a subroutine is always part of the main program. No parameters are used in the example shown in the figure, which makes the syntax of the subroutine very simple. But a subroutine normally uses values from data objects and also returns values. The next figure, Parameter Passing – Visibility of Global Variables, illustrates how these variables can be used in the subroutine.

#### Parameter Definition for Subroutines



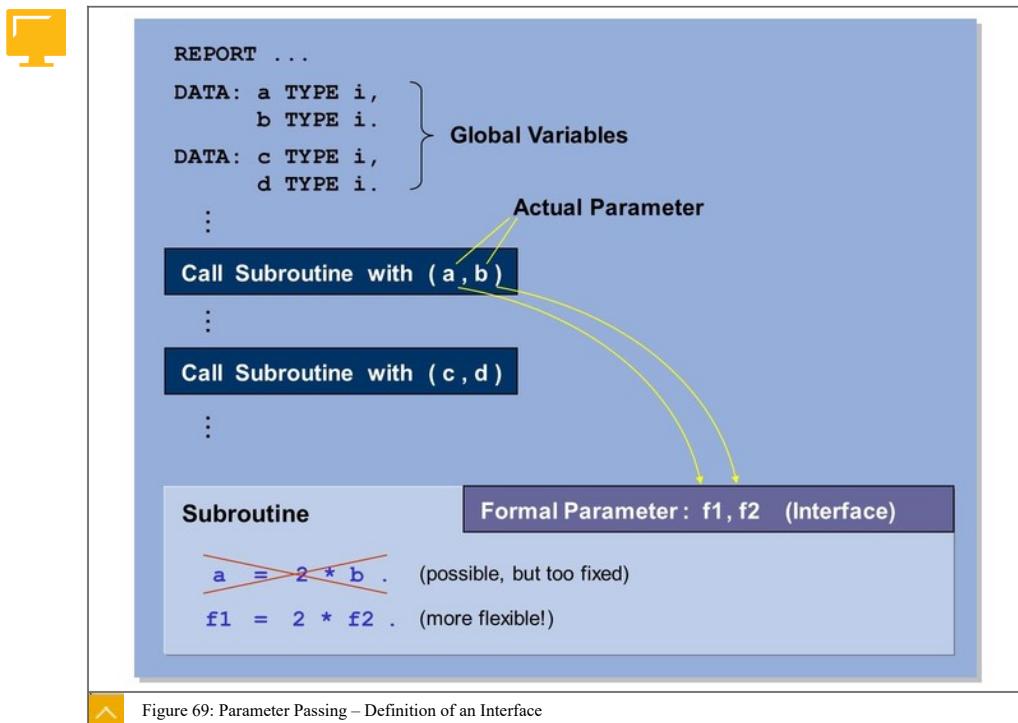
```
REPORT ...  
DATA: a TYPE i,  
      b TYPE i. } Global Variables  
  
:  
Call Subroutine  
:  
Call Subroutine  
:  
Subroutine  
  
a = 2 * b .
```



Figure 68: Parameter Passing – Visibility of Global Variables

Variables defined in the main program are globally visible within the program and can be changed at any point within the program. This means that also subroutines defined within the program can change the variables.

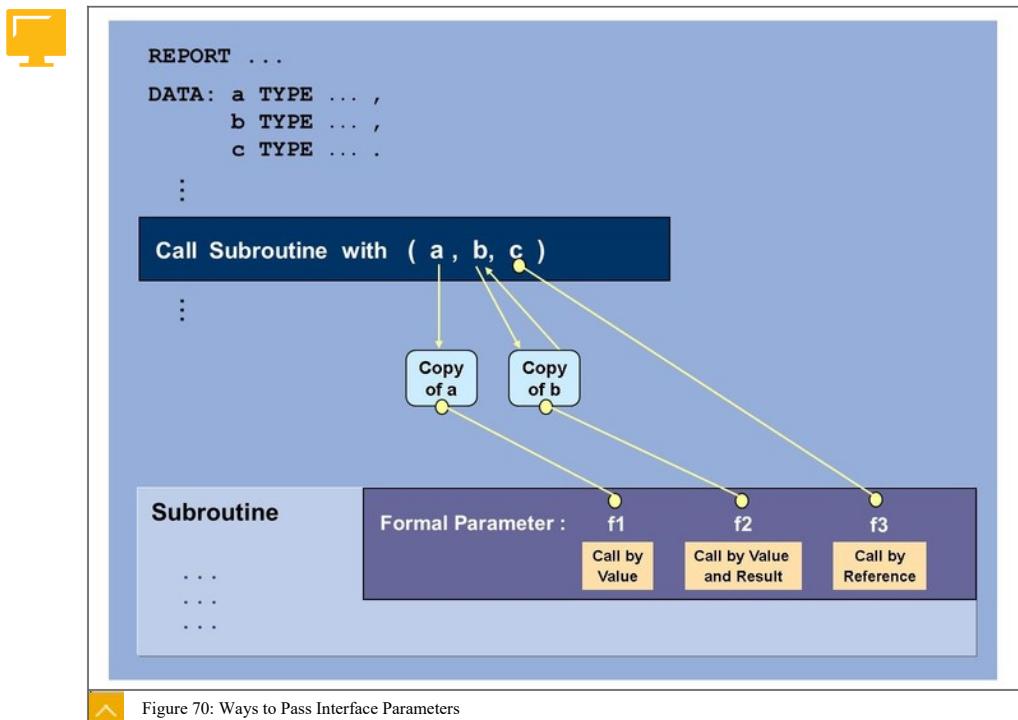
## Parameter Passing – Definition of an Interface



From a subroutine, you can address all global variables defined in the main program. However, to call a subroutine with different data objects for each situation, you must use placeholders instead of global variables. These placeholders are replaced with the required global variables when the subroutine is called. These placeholders are called formal parameters and together they form the subroutine interface. You must declare the interface when you define the subroutine.

When the subroutine is called, formal parameters must be specialized by means of corresponding global variables (actual parameters). This assignment of actual parameters to formal parameters when calling a subroutine is called parameter passing.

## Ways to Pass Interface Parameters



The way these main program variables are passed to the formal parameters of the subroutine is called the pass type and is specified for each parameter in the subroutine interface.

## Pass Types for Subroutines

## Definition of Pass Types for Subroutines

- Call by value

Use this pass type to make the value of a global variable available to the subroutine in the form of a variable copy, without allowing the subroutine to change the respective global variable.

- Call by value and result

Use this pass type to transfer the value of a global variable to the subroutine and to write the fully processed final value of the copy back to the original.

- Call by reference

Use this pass type to run subroutine processing directly on the specified actual parameter.

## Use of Pass Types for Subroutines

- Call by value

The system makes a copy of the actual parameter and assigns this copy to the formal parameter. Any value assignments to the corresponding formal parameter in the subroutine, therefore, refer only to the copy of the actual parameter, not to the original.

Use this pass type to make the value of a global variable available to the subroutine in the form of a variable copy, without allowing the subroutine to change the respective global variable. Creating a copy of the variable value protects the original. However, creating copies, especially for large internal tables, can be time consuming.

- **Call by value and result**

The call by value and result pass type is similar to the call by value pass type. However, at the regular end of the subroutine, the value that was changed in the copy is written back to the original. When the program is prematurely terminated by a STOP statement or a type E user message, the writing back of the value is suppressed.

You use this pass type to transfer the value of a global variable to the subroutine and to write the fully processed final value of the copy back to the original. However, it can be time consuming to create copies and write back values, especially for large internal tables.

- **Call by reference**

The system assigns the actual parameter directly to the formal parameter. This means that value assignments to the formal parameter are carried out directly on the actual parameter.

You use this pass type to run subroutine processing directly on the specified actual parameter. It is a useful way of avoiding the time-consuming creation of copies for large internal tables.

#### Definition and Call of Subroutines

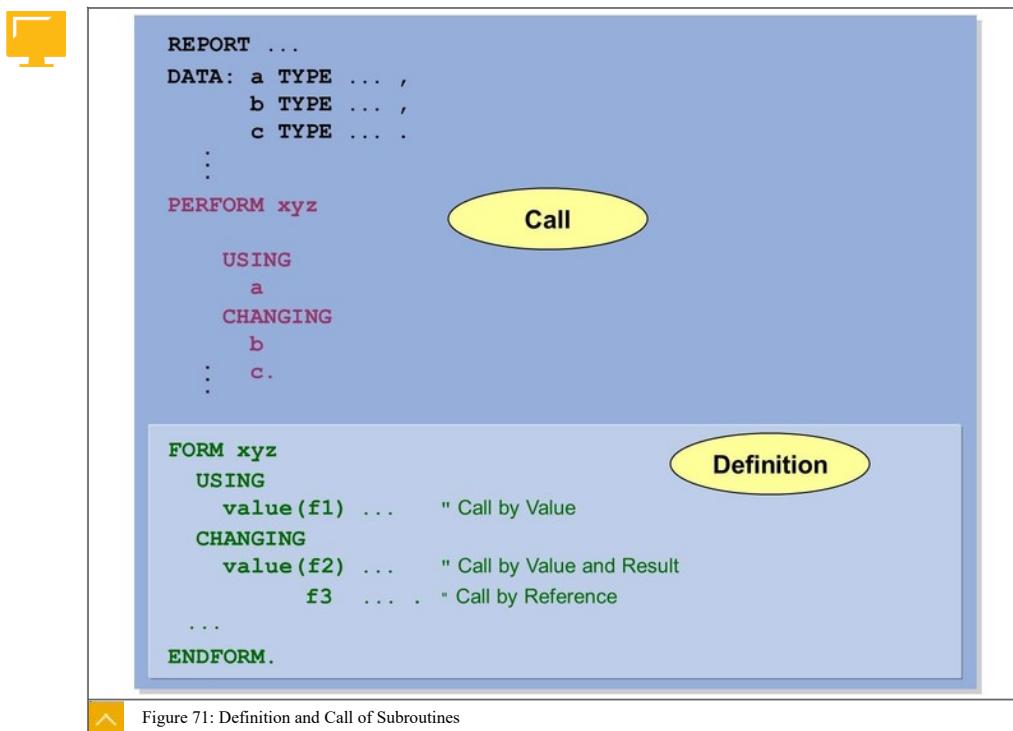


Figure 71: Definition and Call of Subroutines

#### Subroutine Definition Process

- Define a subroutine in the following way:
  1. Introduce a subroutine with FORM.
  2. Specify the name and the interface of the subroutine after FORM. The statements of the subroutine then follow.
  3. Conclude the subroutine with the ENDFORM statement.

In the interface definition, you can list the formal parameters of the subroutine (f1, f2, and f3) and type them.

#### Pass Type Specification

- Specify the required pass type for each parameter as follows:
  - Call by value  
List each of the formal parameters that must have the pass type “call by value” (f1) with the VALUE prefix under USING. Refer to the figure for the syntax.
  - Call by value and result  
List each of the formal parameters that must have the pass type “call by value and result” (f2) with the VALUE prefix under CHANGING. Refer to the figure for the syntax.
  - Call by reference  
List each of the formal parameters that must have the pass type “call by reference” (f3) without the VALUE prefix under CHANGING. Refer to the figure for the syntax.

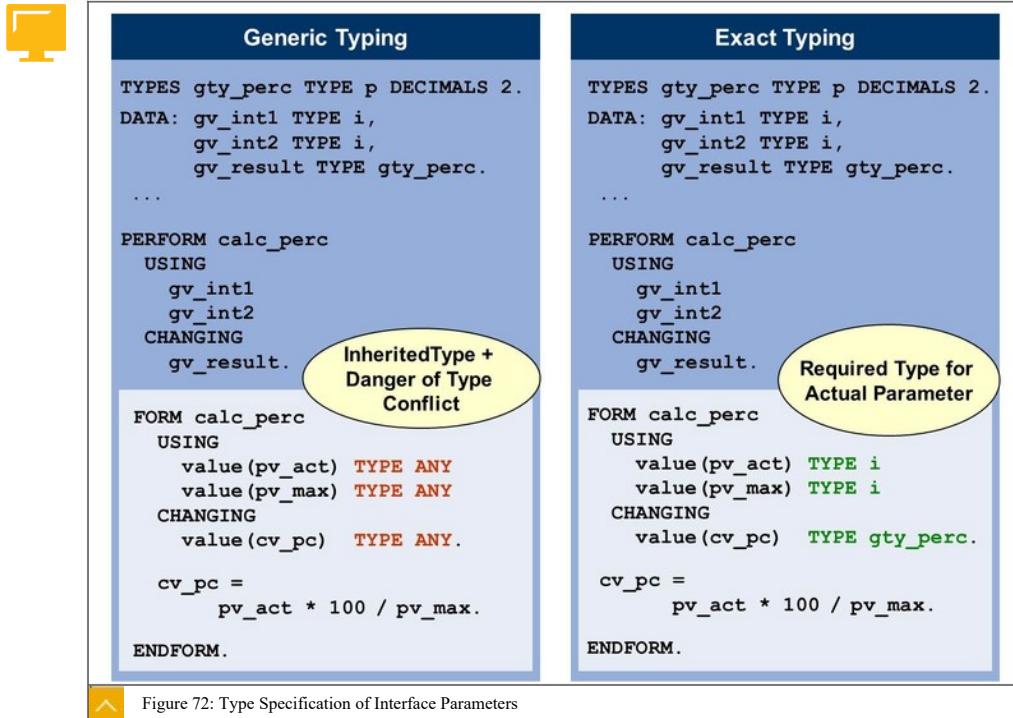


##### Hint:

A parameter placed under USING without the VALUE prefix also has the pass type “call by reference”. However, this declaration syntax only makes sense for formal parameters to which larger internal tables are passed. These formal parameters are not changed in the subroutine (see the documentation for USING), but are passed using “call by reference” to avoid making time-consuming copies.

When the subroutine is called, the actual parameters to be transferred without the VALUE prefix are specified under USING or CHANGING. The order of specification determines their assignment to the formal parameters. In the example shown in the figure, a is passed to f1, b to f2, and c to f3.

## Type Specification of Interface Parameters



A formal parameter is typed generically when it is typed using TYPE ANY, or not typed at all. Actual parameters of any type can be transferred to such a parameter.

At runtime, the type of the actual parameter is determined and assigned to the formal parameter (type inheritance) when the subroutine is called. However, when the statements in the subroutine are not suited to the inherited type, a runtime error can occur (type conflict). Only use generic typing when the type of the actual parameter is not known when the program is created, or when the type of the actual parameter can vary at runtime (dynamic programming).

You implement the concrete typing of a formal parameter by specifying a global or built-in type in the TYPE addition. In doing so, you determine that only actual parameters of the specified type can be passed to the subroutine. A violation of the type consistency between formal and actual parameters is detected in the syntax check. The syntax check increases the stability of your program by preventing type conflicts in statements within the subroutine.

When you type with the standard types P, N, C, or X, the missing characteristic 'field length' is passed from the actual parameter to the formal parameter at runtime. You achieve complete typing with these types (that is, including the field length) by defining and specifying locally defined types.



Hint:

Besides elementary data objects, ABAP also knows structures and internal tables. Formal parameters for such data objects must always be typed in order for their components to be accessed within the subroutine.



Note:

“<scope><kind>\_<name>” -> “<scope>” p = using/passing, “<scope>” c = changing

## Local and Global Data Objects



```
TYPES tv_perc TYPE p DECIMALS 2.  
  
DATA: gv_int1    TYPE i,  
      gv_int2    TYPE i,  
      gv_result  TYPE tv_perc.  
...  
PERFORM calc_perc  
      USING gv_int1  
           gv_int2  
      CHANGING gv_result.  
...  
  
FORM calc_perc  
  USING  
    pv_act TYPE i  
    pv_max TYPE i  
  CHANGING  
    cv_pc  TYPE tv_perc.  
...  
DATA lv_pc TYPE p LENGTH 16 DECIMALS 1. } Local variables  
      (only visible locally)  
...  
ENDFORM.
```

} Global variables

} Formal parameters  
(only visible locally)

Figure 73: Visibility of Global and Local Data Objects

Variables defined in the main program are global data objects. They are visible and can be addressed in the entire main program, as well as in every subroutine called.

Variables defined within a subroutine are called local because they exist in the relevant subroutine in the same way as formal parameters. Memory space for formal parameters and local data objects is allocated when the subroutine is called and is released again after execution.



Note:  
 “<scope><kind>\_<name>” -> “<scope>” l = local

The formal parameters and local data objects of a subroutine cannot have the same names. When there is a global data object with the same name as a formal parameter or a local data object, the formal parameter or local data object is addressed within the subroutine, and the global data object is addressed outside the subroutine. This is the so-called shadow rule: within a subroutine, the local data object shadows the global one with the same name.

To identify your internal program objects uniquely, use the following prefixes for your subroutine objects: **p**... for using parameters, **c**... for changing parameters, and **l**... for local data objects.

#### Syntax Example – Local Auxiliary Variable for Rounding



```

TYPES tv_perc TYPE p DECIMALS 2.

DATA: gv_int1    TYPE i,
      gv_int2    TYPE i,
      gv_result  TYPE tv_perc.

...
PERFORM calc_perc
      USING gv_int1
      gv_int2
      CHANGING gv_result.
...

FORM calc_perc
  USING
    value(pv_act)  TYPE i
    value(pv_max)  TYPE i
  CHANGING
    value(cv_pc)   TYPE tv_perc.

  DATA lv_pc TYPE p LENGTH 16 DECIMALS 1.

  lv_pc = pv_act * 100 / pv_max.
  cv_pc = lv_pc.

ENDFORM.

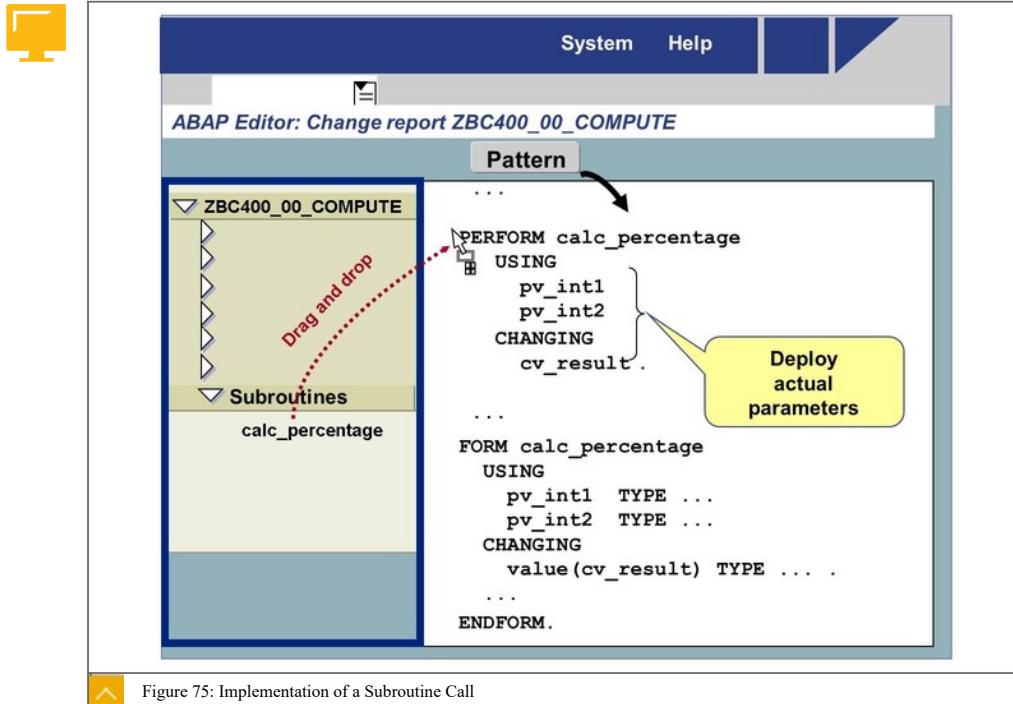
```

Figure 74: Syntax Example – Local Auxiliary Variables for Rounding

In the syntax example illustrated in the figure, the result of the percentage calculation must be rounded internally to have one decimal place, but, nonetheless, return with two decimal places.

To do this, you create a local variable with only one decimal place and store the result of the calculation there first. The runtime system rounds the result to the nearest whole number in accordance with the available number of decimal places. When you copy the result to the return parameter, a zero is added for the second decimal place.

## Call of Subroutines

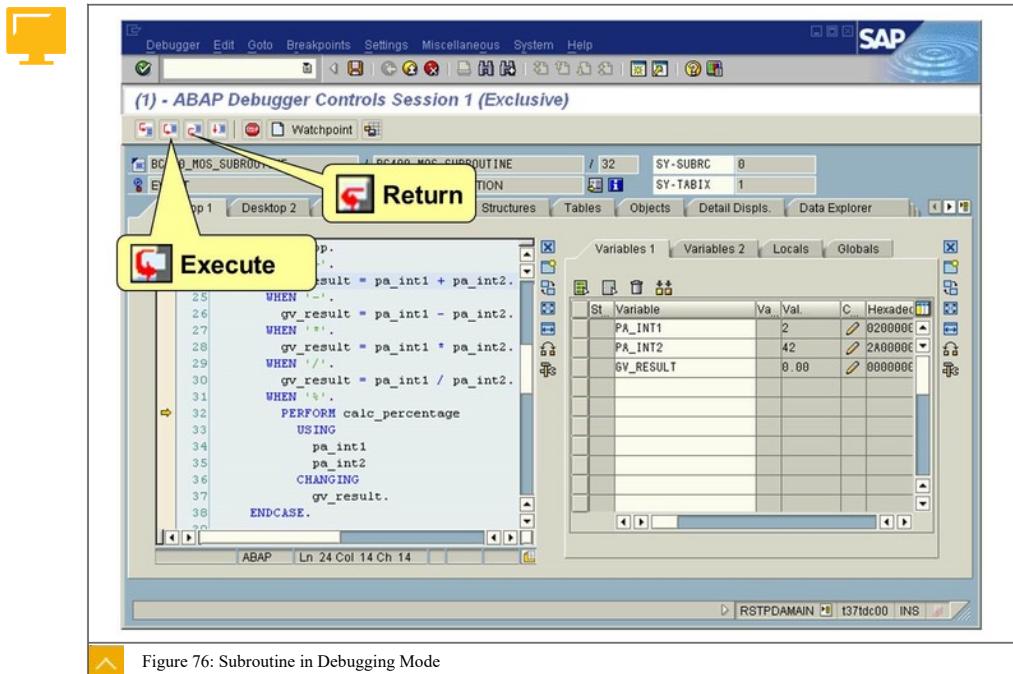


When calling a subroutine, you can make the ABAP workbench generate the PERFORM statement into your source code. Define the subroutine and then save your main program. The newly defined subroutine appears in the navigation area. Move it to the required call point in your program by using the drag-and-drop function. Then replace the formal parameters in the generated source code with corresponding actual parameters.

Alternatively, you can implement call generation by using the **Pattern** button in the ABAP Editor.

The advantage of generating the call is that it is impossible to forget or mix parameters.

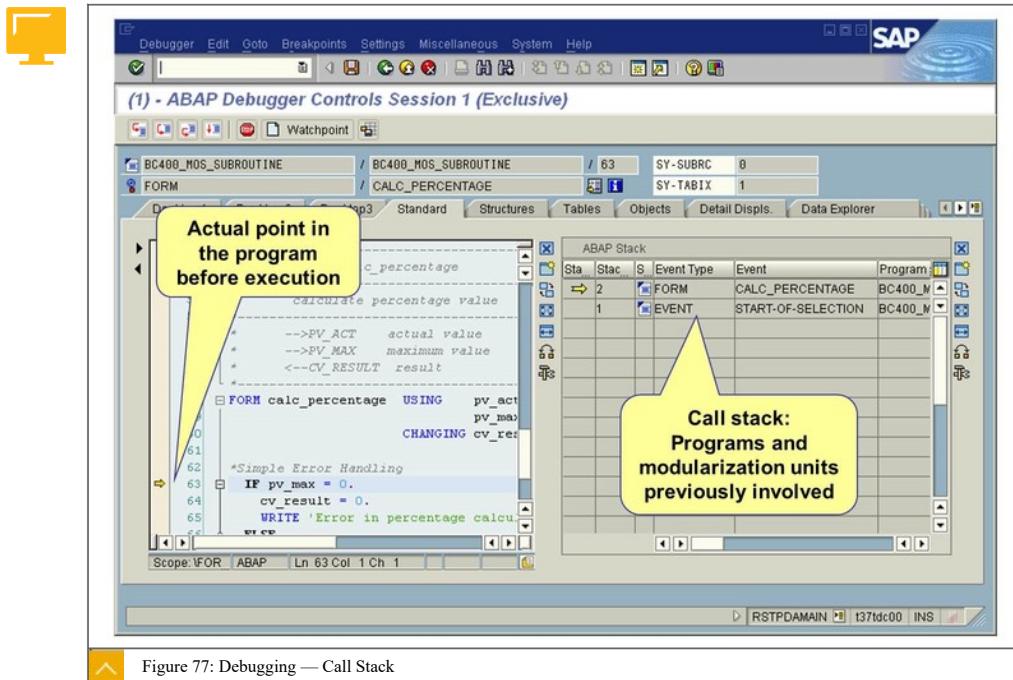
## Modularization Units in the ABAP Debugger



## Subroutines in Debugging Mode

- Consider the following when running subroutines in debugging mode.
  - If the current statement is a subroutine call, you can execute the entire subroutine without stopping by choosing **Execute**. Processing stops after the subroutine has been completed.
  - In contrast, you can use **Single Step** to stop at the first statement of the subroutine and trace its operations in more detail.
  - If the current statement is located in a subroutine, you can execute the rest of the subroutine without stopping by choosing the **Return** button. Processing stops after the subroutine has been completed.
  - Control functions of the ABAP Debugger, including single step, execute, return, and continue, are also available in the classic ABAP Debugger, and they have the same meaning.

### Call Stack



On the Standard tab in the Debugger, you can view from which programs the subroutine was called. The tool for this is the **Call Stack**.



### LESSON SUMMARY

You should now be able to:

- Modularize using subroutines

## Unit 4

### Lesson 3

# Calling Function Modules

#### LESSON OVERVIEW

This lesson explains how to search for function modules, analyze their interfaces and documentation, and test their functions. You will learn how to use existing function modules in your programs and catch and handle errors. Finally, you will create function modules and encapsulate functions that you can then reuse in other programs.

#### Business Example

You need a function for your program that is already available in the form of a function module. In addition, you want to use this function module in your program. For this reason, you require the following knowledge:

- An understanding of how to search for function modules
- An understanding of how to acquire information about the functionality and use of function modules
- An understanding of how to call function modules in your program
- An understanding of how to create a function group and a function module
- An understanding of BAPIs and their special properties

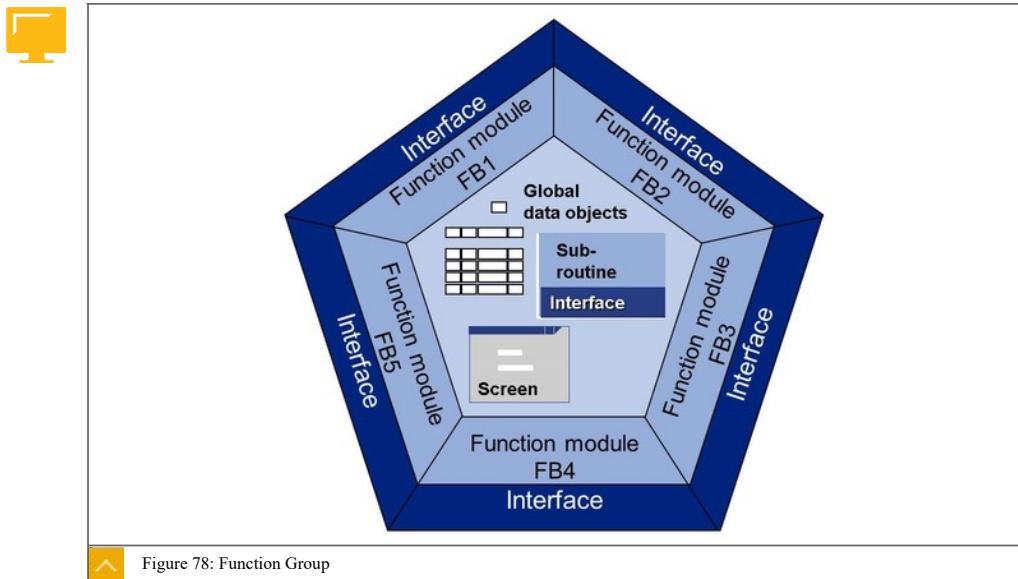


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use function modules of function groups

### Structure of Function Groups



A function module is a procedure with a corresponding function that is stored centrally in the Function Library of the SAP system. Each function module has an interface with parameters for importing or exporting data. The purpose of function modules is their reusability.

Function modules are assigned to function groups. Each function group is a collection of function modules that have related functions or process the same data.

A function group can contain the same components as an executable program.

#### Components of a Function Group

- Data objects

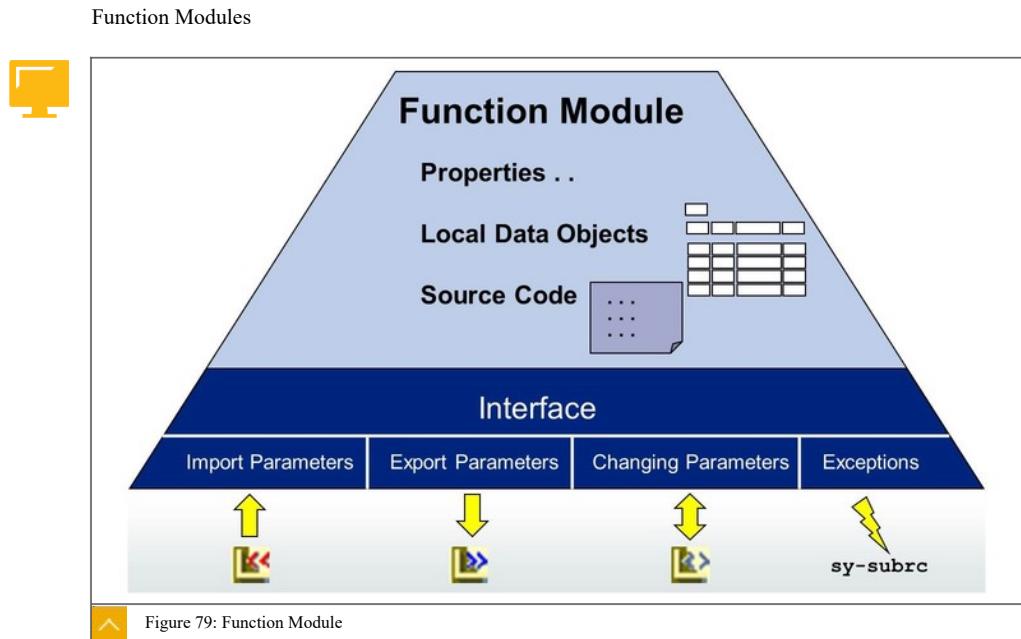
Data objects are global in relation to the function group; that is, they are visible to, and can be changed by, all function modules within the group.

- Subroutines

Subroutines can be called from all function modules in the group.

- Screens

Screens can be called from all function modules in the group.



The properties of a function module include a short description of the module and the name of the function group to which it belongs.

Similar to subroutines, a function module can contain its own local type and data object definitions. Local types and data objects can only be seen within the function module.

#### Elements of the Interface of a Function Module

The interface of a function module can contain the following elements:

##### **Import parameter**

When the function module is called, the calling program transfers values or variables to the function module. If a parameter is optional, there is no need to supply it with a value or a variable.

##### **Export parameter**

The calling program accepts the output of the function module by way of the assignment of a “receiving variable” to an export parameter. Export parameters are always optional.

##### **Changing parameter**

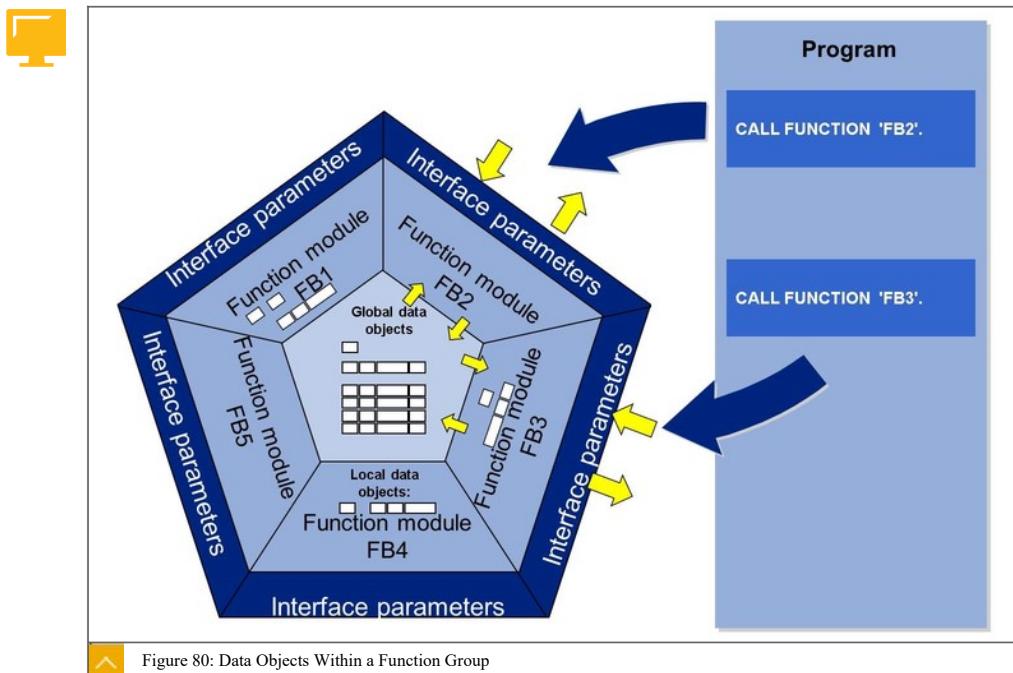
When the function module uses changing parameters, variables are transferred from the calling program to the function module and returned at the end of the call.

##### **Exceptions**

Exceptions can be raised by the function module in certain error situations and provide information about the respective processing error in the function module. Exceptions should be handled by the calling program.

In general, interface parameters are defined using types from the ABAP Dictionary.

## Data Objects Within a Function Group



When a program calls a function module, the corresponding function group is loaded and the function module is executed. The function group remains loaded in the working memory until the calling program is closed. When you call another function module of this group, it is processed without being reloaded, and with the same global data of the function group.

If a function module that writes values to the global data of the function group is called, other function modules in the same function group can access the data when they are called in the same program run.

Apart from the global data of its function group, a function module can also access its own locally defined data objects and interface parameters. Interface parameters receive data or return it to the calling program.

## Function Modules Searching

## Methods Used for Function Module Searching



- You can search for function modules as follows:
  - Application-related search  
Search within a particular application component (Application Hierarchy).
  - Application-independent search  
Perform a free search, independent of application components (Repository Information System).

- #### - Program-related search

Search for the CALL FUNCTION statement within the program source code.

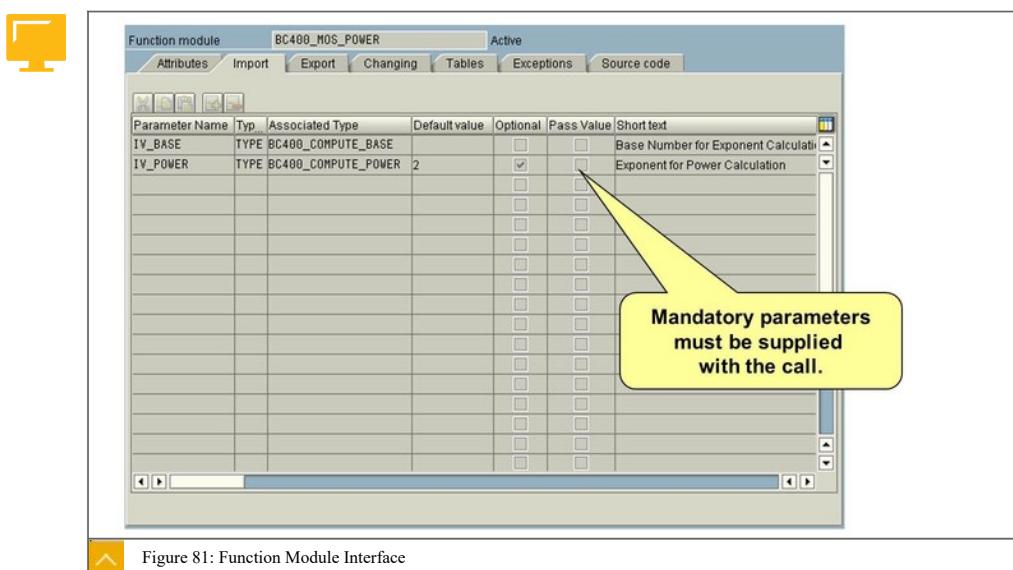
Before triggering a program function (for example, choosing a button) using “/h”, activate the ABAP Debugger and set a breakpoint at the CALL FUNCTION statement.

When the searched function module passes a screen, use F1 and “Technical Information” to determine the screen number. Navigate to it by double-clicking it, and execute the “Where-Used List in Programs”.

After you have found a function module, determine whether it has been released (attributes of the function module). You only have the right to support and upward compatibility with released function modules. It is recommended that you only use released function modules.

You can use the documentation of the function module to determine its functionality and obtain further information.

## Examination of a Function Module

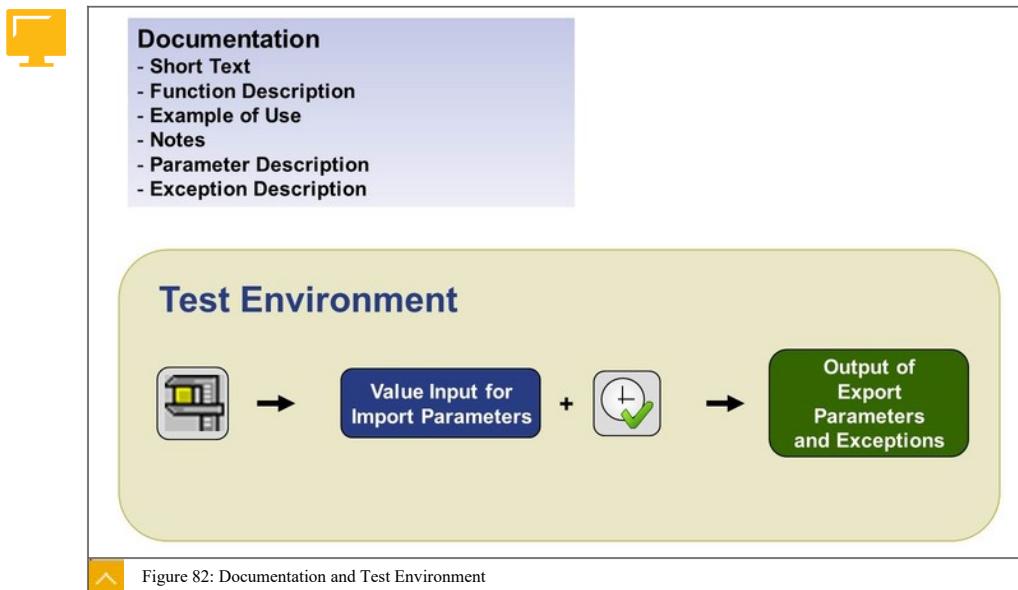


The interface of the function module consists of the import, export, changing parameters, and exceptions.

As with subroutines, you can distinguish between pass by value and pass by reference.

Non-optional parameters must be supplied when the function module is called. To find out which additional parameters are to be passed and when, refer to the function module documentation and the documentation for interface parameters.

### Documentation and Test Environment



#### Features of Documentation

- Short text
- Functional description
- Example of use
- Notes
- Parameter description
- Exception description

Function modules that are intended and released for use in other programs should be documented. A distinction is made between function module documentation, which describes the function of the module, and parameter documentation, which tells you how individual parameters or exceptions are used.

You can access the function module documentation by choosing the FModule Documentation button. Click the displayed links to go to the parameter documentation (only in display mode).

You can also access parameter documentation by choosing the button for the respective parameter in the Long Text column. The “green light” indicates that documentation has been created for the parameter.

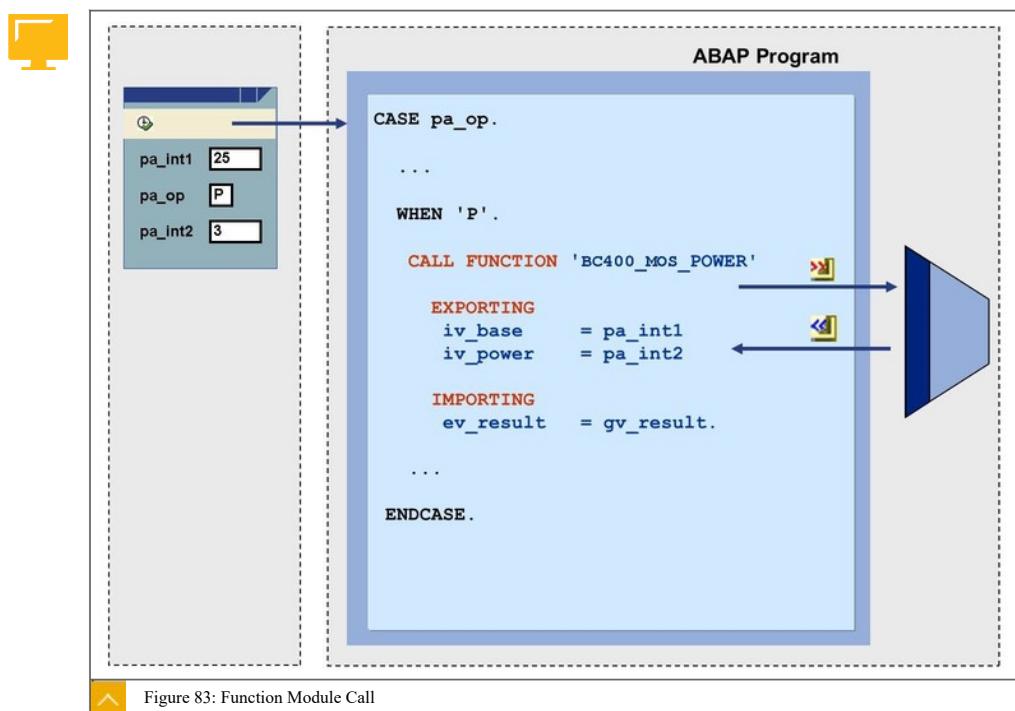
If you are unsure whether the function module does precisely what you want it to do, you can test it in the test environment prior to installing it in your program. An input template allows you to specify values for the import and changing parameters. To check values during runtime, you can also switch to debugging mode. The values in the export and changing parameters are listed after execution.

If the function module triggers an exception due to an error, the exception is displayed, where applicable, with its message text.

The test framework for function modules also displays the runtime. These values are subject to the same restrictions as the runtime analysis. Therefore, repeat the tests several times using the same data.

Store test data in a test data directory and create test sequences.

### Call of Function Modules



Function modules are called using the CALL FUNCTION statement. The name of the function module follows in capital letters enclosed in single quotation marks.

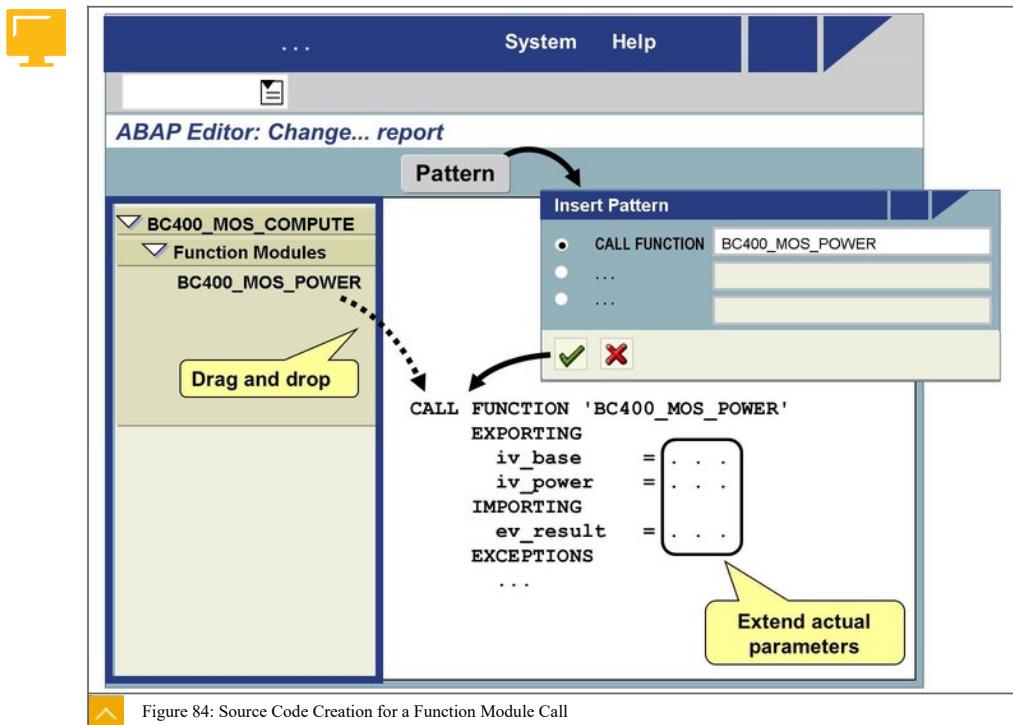
In the EXPORTING block, the system passes values to the import parameters of the function module.

In the IMPORTING block, actual parameters are assigned to the export parameters. You can use them to access the results of the call.

From the perspective of the calling program, values are exported to the import parameters of the function module and values are imported for the receiving variables assigned to the export parameters of the function module.

In the call syntax, you need to specify the name of the interface parameter (formal parameter) on the left side of the parameter assignment and the data object, or the value for the calling program (actual parameter) on the right side of the parameter assignment.

## Source Code Creation for a Function Module Call

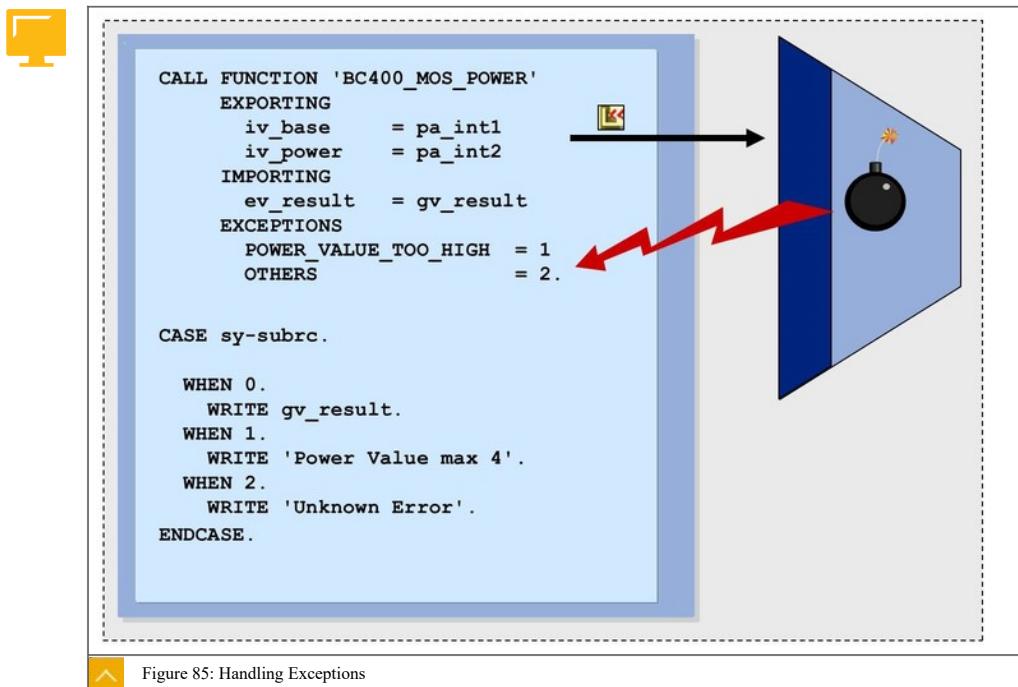


To avoid spelling errors, use the **Pattern** button to generate the call when you implement a call in the source code. Alternatively, you can display the function group in the navigation area and drag-and-drop the function module into the ABAP Editor area to generate the call in the source code.

This generates the entire call syntax, which lists all interface parameters, as well as exceptions. Optional information is commented out in the call.

Afterwards, you must fill in the actual parameters and, when necessary, implement exception handling.

## Classic Exception Handling



If application errors occur when a function module is processed (for example, a value is not suitable for the calculation), the function module raises a corresponding exception. The exception cancels the processing of the function module and returns the system to the calling program. If the exception is listed (caught) in the EXCEPTIONS block of the call, the specified return code is entered in the sy-subrc system field of the calling program. By checking this field after the call, you can determine if and, where applicable, which exception was raised so you can react accordingly. If no exception is raised by the function module, the sy-subrc of the calling program is set to zero.

You also have the option of setting a return code explicitly for some of the possible exceptions and setting a different return code for all unmentioned exceptions. To do so, list the formal exception OTHERS with the desired return code.

Catch all exceptions in your call and react to them in the program. If a raised exception is not caught, a runtime error occurs.



## LESSON SUMMARY

You should now be able to:

- Use function modules of function groups

## Unit 4

### Lesson 4

## Creating Function Modules

### LESSON OVERVIEW

This lesson explains how to create function groups and function modules.

#### Business Example

You are in charge of implementing calculation functions and want to create a global function module to calculate percentage values.

For this reason, you require the following knowledge:

- An understanding of how to create function groups and function modules

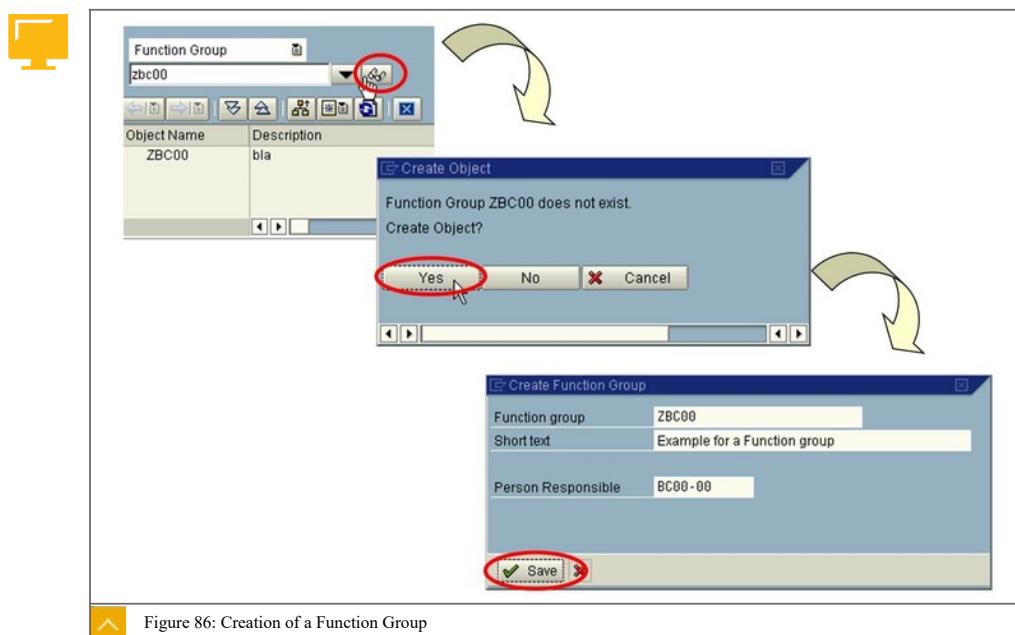


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create function groups
- Create function modules

### Creation of Function Groups



Your goal is to create function modules. Because a function module must be contained in a function group, you must first learn how to create a new function group.

### Creation of Function Modules

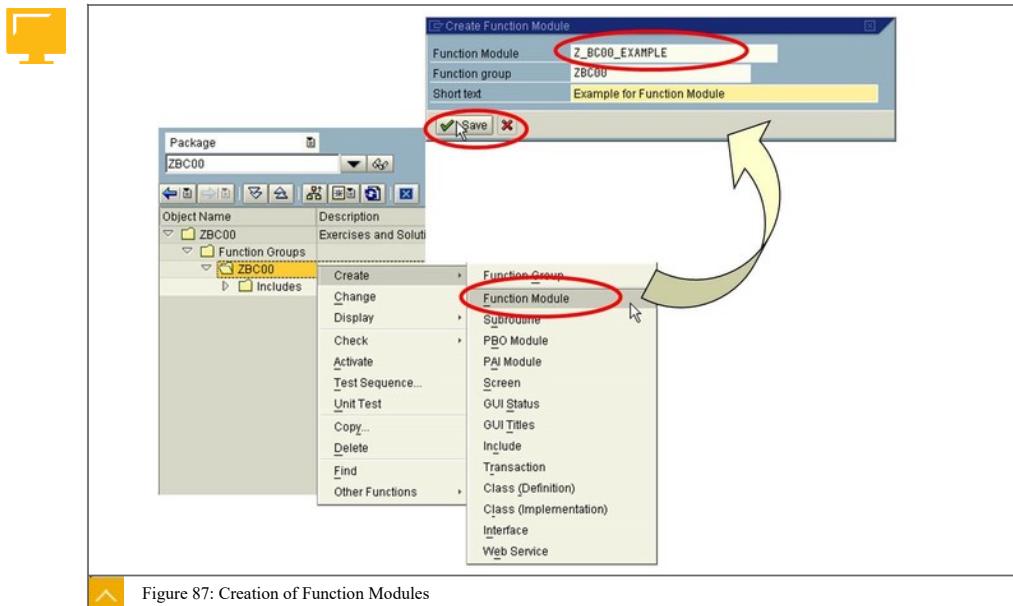


Figure 87: Creation of Function Modules



#### Note:

The recommended customer namespace for function modules is `Z_<FUNCTION_MODULE>` respectively `Y_<FUNCTION_MODULE>`.

### Source Code Editing

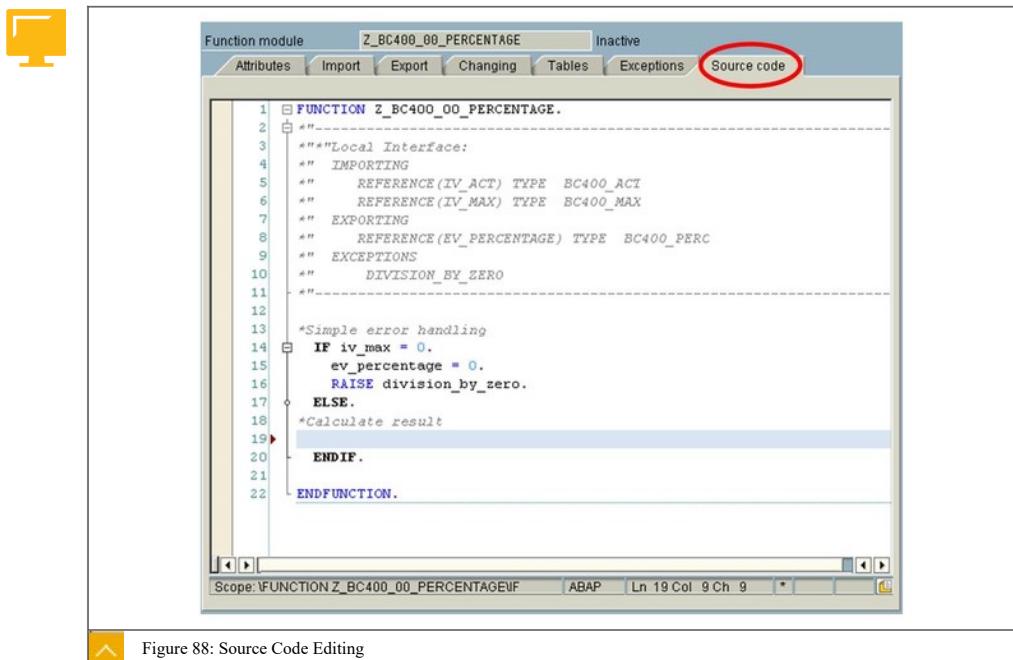


Figure 88: Source Code Editing

After defining the corresponding IMPORTING and EXPORTING parameters, you can switch to the Source code tab page to implement the functions of the function module.



#### Hint:

The comment block directly under the keyword FUNCTION is created by the Function Builder and is changed when changes are made to the parameters.



### LESSON SUMMARY

You should now be able to:

- Create function groups
- Create function modules

## Unit 4

### Lesson 5

# Describing Business Application Programming Interfaces (BAPIs)

#### LESSON OVERVIEW

This lesson introduces the Business Application Programming Interface (BAPI).

#### Business Example

You are working in the SAP system and want to access data. This can be done with the help of a BAPI. For this reason, you require the following knowledge:

- An understanding of BAPIs



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Modularize using BAPIs

#### Business Application Programming Interface (BAPI)

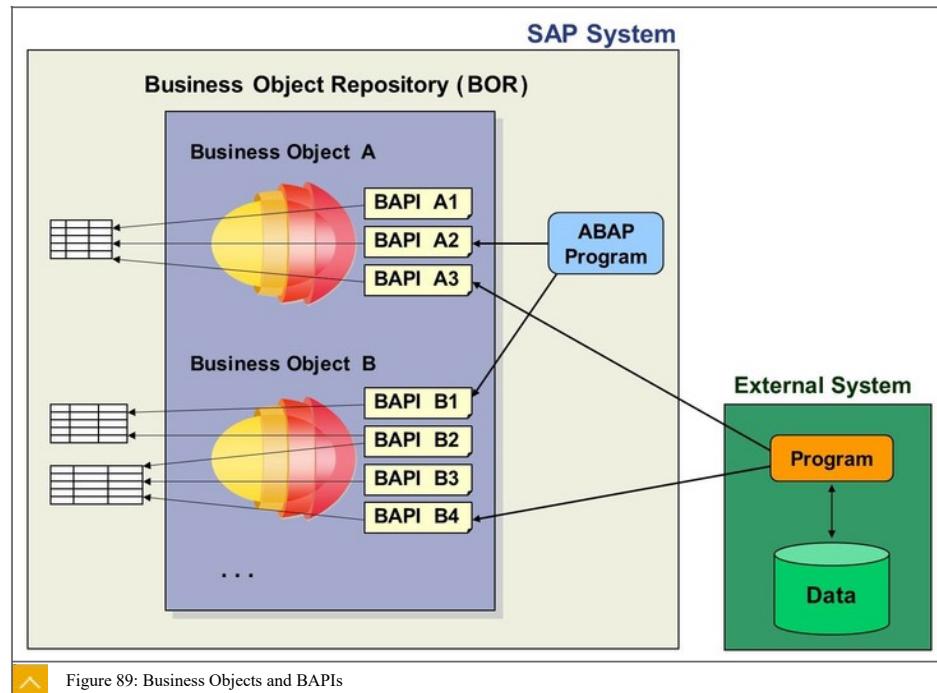


Figure 89: Business Objects and BAPIs

The Business Object Repository (BOR) in the SAP system contains business objects types. A business object type is a program that behaves like a class. It represents an SAP table or a table hierarchy. A business object has BAPIs as methods. You can call these BAPIs to access the corresponding tables. A BAPI is a means of accessing data in the SAP system.

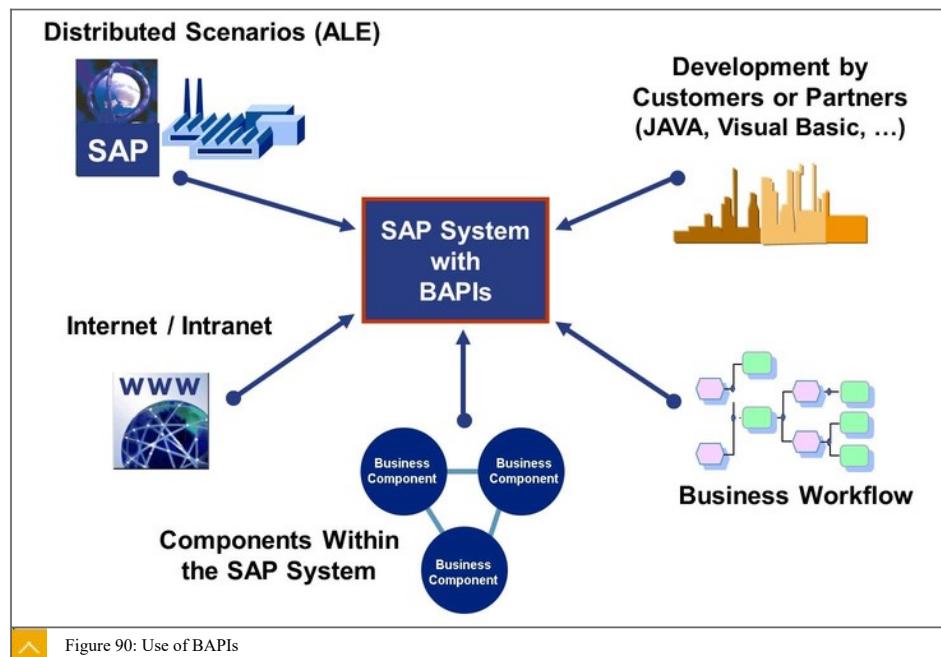
#### BAPIs Used as Functions of a Business Object Type

The functions of a BAPI are encapsulated in a function module that can be called remotely. BAPIs can, therefore, be called by ABAP programs of the same SAP system, as well as by external programs.



- BAPIs usually exist for the following functions of a business object type:
  - Creating an object
  - Retrieving object attributes
  - Changing object attributes
  - Listing objects

#### Use of BAPIs



The figure shows how BAPIs can be used.

There are standard methods in the form of BAPIs with standardized names.

#### Standard BAPIs



- GetList

All GetList BAPIs return a list of available objects that meet the specified selection criteria.

- GetDetail

All GetDetail BAPIs return detailed information (attributes) for an object (the complete key must be specified).

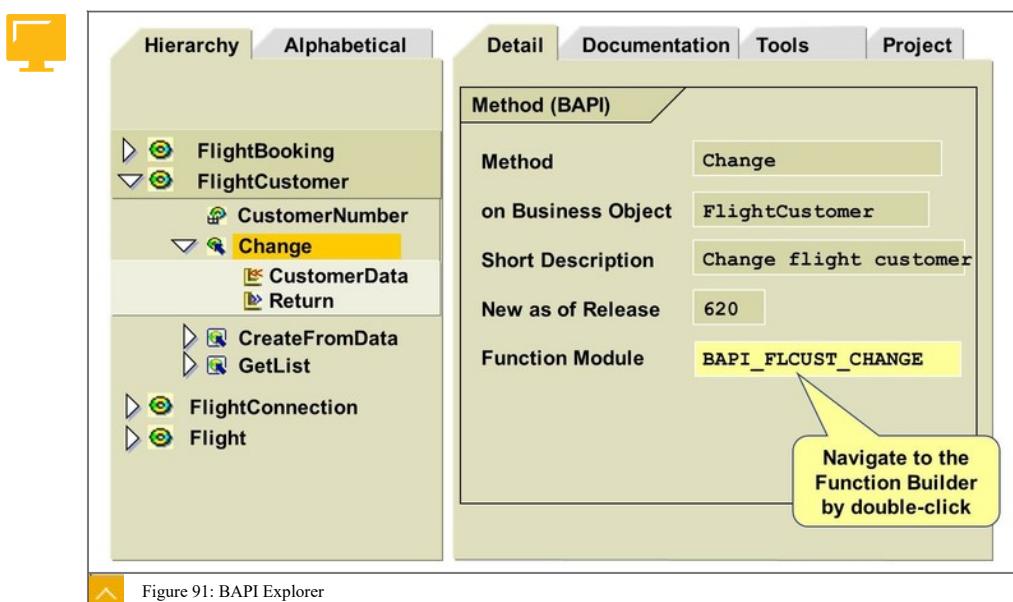
- Create, Change, Delete, Cancel

All Create, Change, Delete, Cancel BAPIs allow you to create, change, and delete objects.

- AddItem, RemoveItem

All AddItem, RemoveItem BAPIs add and remove subobjects (for example, an item for an order).

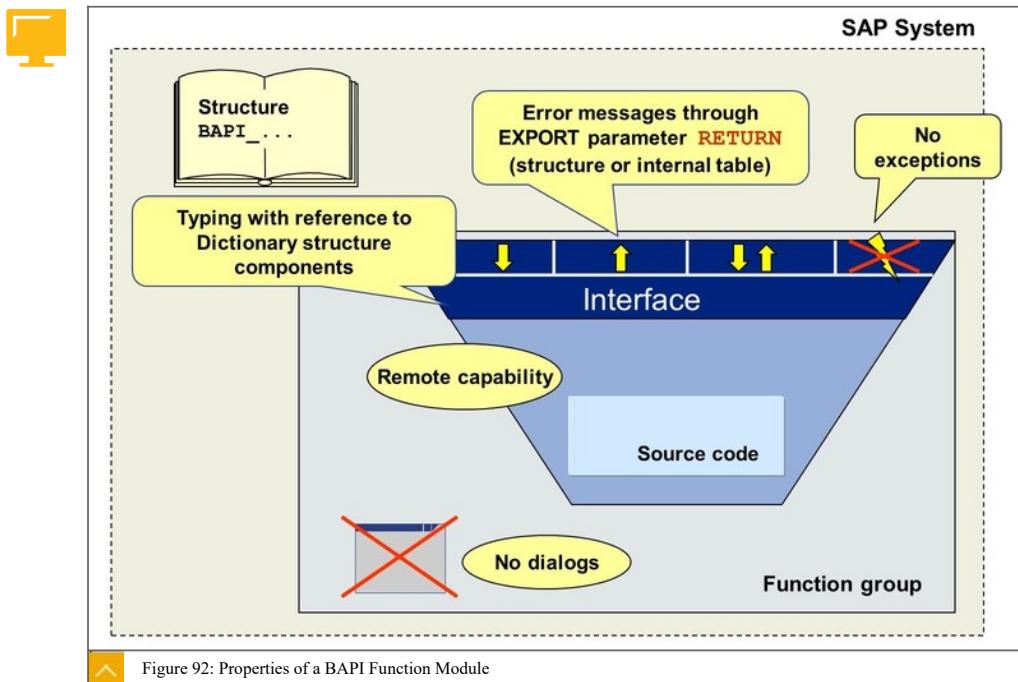
### BAPI Explorer



You can use the BAPI Explorer to list business objects as well as the corresponding BAPIs with reference to the application. To call the BAPI Explorer, use the following path in the SAP Easy Access menu: Tools → Business Framework → BAPI Explorer or transaction BAPI.

After you have located the required business object or BAPI, you can display the relevant details on the right side of the screen by selecting BAPI. You can navigate to its display using the Function Builder by double-clicking the displayed function module.

### Properties of a BAPI Function Module



### Technical Requirements for Function Modules for BAPIs

- Function modules for BAPIs must fulfill the following technical requirements:

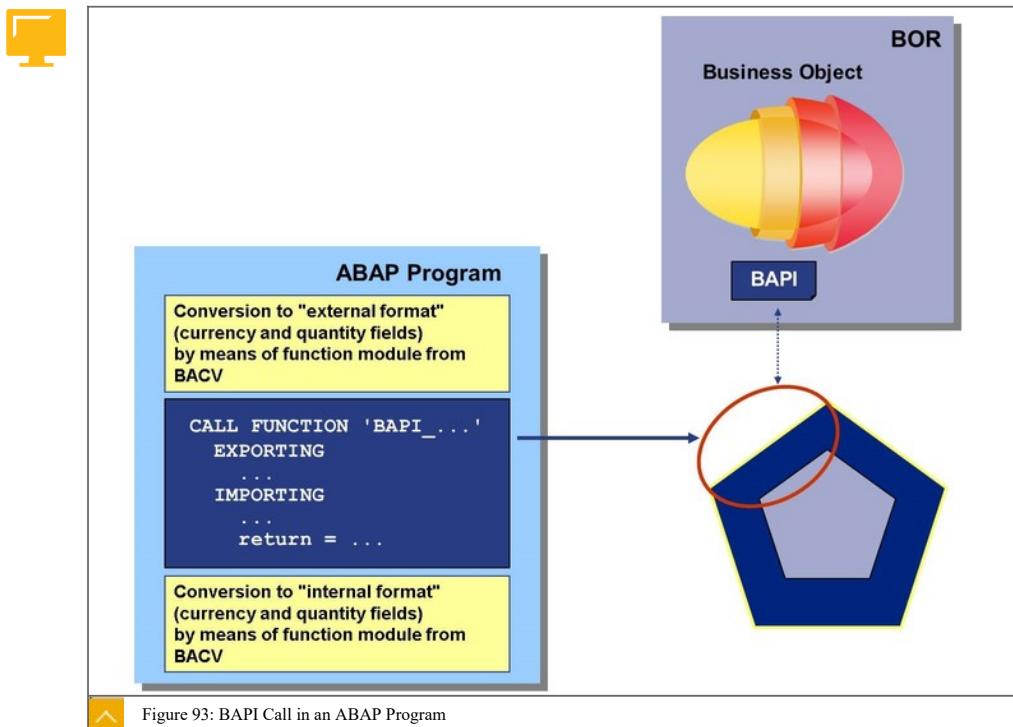
- Naming convention: BAPI <business object name> <method name>
- Remote-enabled
- No user dialogs or messages
- Name prefix for BAPI structures: BAPI

Interface parameters are typed with components of structures from the ABAP Dictionary created for this BAPI.

- No changing parameters until Release 4.6
- Raise no exceptions

Errors are reported to the user through the special export parameter RETURN.

## BAPI Call in an ABAP Program



To use a BAPI within the same SAP system, you need to call the relevant function module directly. Note the restrictions mentioned in the Properties of a BAPI Function Module section.



## Hint:

BAPI interfaces are created according to external call requirements; that is, a call from an external system. Amounts are therefore expected in an external format with 4 or 9 decimal places. During the call, the amounts must be transferred to the interface in an appropriately converted format, even if the corresponding currency has no decimal places.

For this conversion or re-conversion, you can use function modules from the BACV function group (package SBF\_BAPI).



## LESSON SUMMARY

You should now be able to:

- Modularize using BAPIs

## Unit 4

### Lesson 6

# Calling Methods of Global Classes

#### LESSON OVERVIEW

This lesson introduces you to another technique for cross-program modularization: call of methods of global classes. As with function groups and function modules, you first look for, analyze, test, and use existing classes and methods. In addition, you create a global class with a simple method.

This lesson is not intended to provide a comprehensive introduction to object-oriented programming with ABAP. Familiarize yourself with the basic terms of object-oriented programming so that you can use the functions of existing global classes in your own programs.

#### Business Example

In your program, you need a function that might already exist in the form of a method of a global class. You want to use this method in your program.

If the function does not already exist, or does not meet your requirements, you need to create your own global class and method. For this reason, you require the following knowledge:

- An understanding of the basic terms of object-oriented programming
- An understanding of how to acquire information about the function and use of global classes and their methods
- An understanding of the call of global classes in your programs
- An understanding of how to create global classes
- An understanding of how to create and implement simple methods in global classes

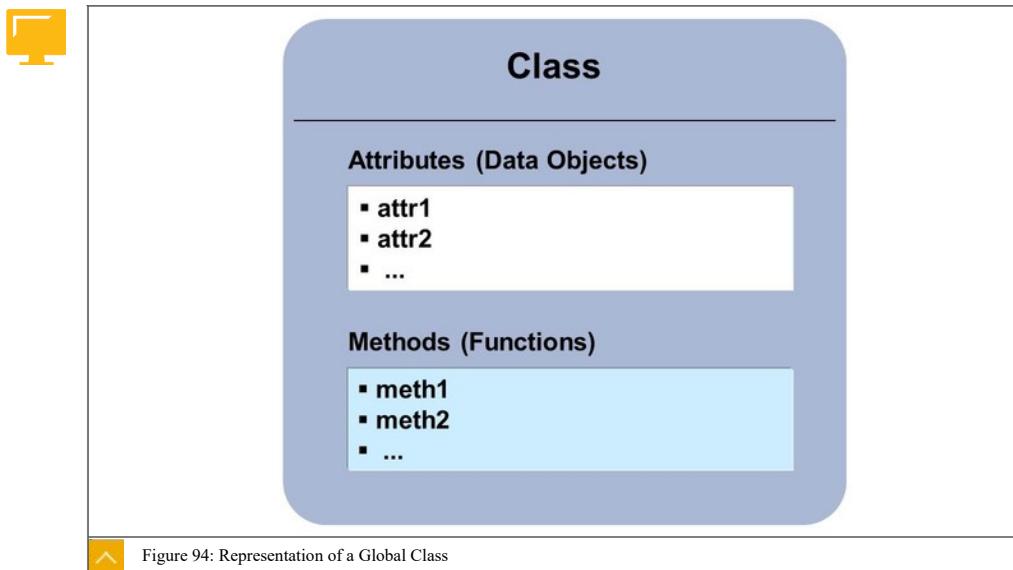


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe object-oriented programming
- Use methods of global classes
- Use instances

## Principles of Object-Oriented Programming



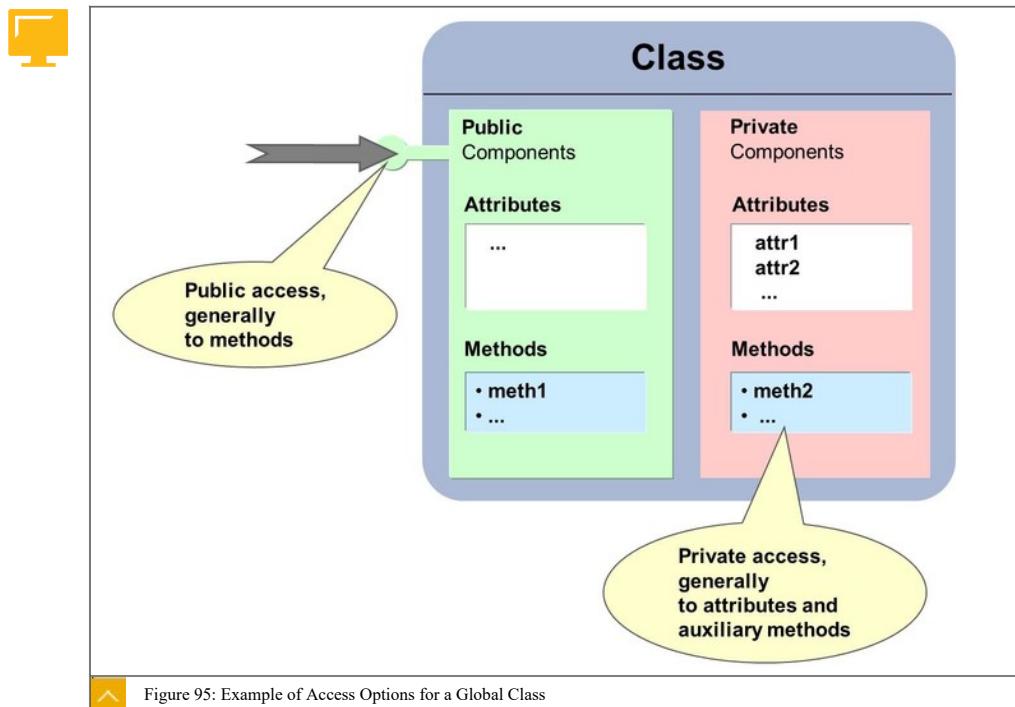
Before using global classes and methods, you need to know some basic terms used in object-oriented programming.

You have read about function groups that make reusable units available in the form of function modules.

Object-oriented enhancements to ABAP have introduced the use of global classes, which make functions available in the form of methods. Similar to function modules, methods have an interface known as a signature that comprises import, export, changing parameters, and exceptions.

Classes have other components in addition to methods. For example, classes contain global data objects known as attributes. In the same way that all function modules can access the global data objects in a function group, all the methods of a class can access the attributes of that class.

Example of Access Options for a Global Class



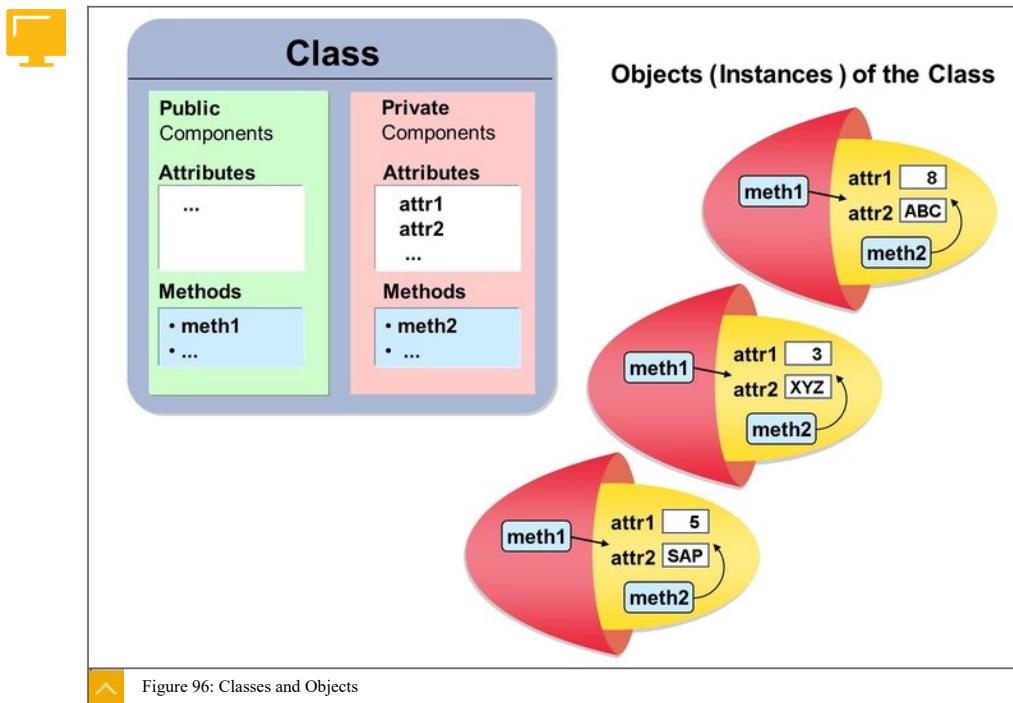
Global data objects within the function group are not visible outside the function group. This is referred to as encapsulation of data in the function group.

Attributes are normally encapsulated in the class, and can therefore only be read or changed using methods of the same class.

In contrast to function modules, classes allow you to make specific attributes visible to users of the class. A distinction is therefore made between public and private attributes.

You can apply the public and private distinction not only to attributes, but also to methods. Whereas all function modules can be called from outside the function group, only public methods are available outside the class. Private methods can only be called by methods of the same class, and are similar in this respect to subroutines within a function group.

## Multiple Instantiation of Classes



The major difference between global classes and function groups is that a function group with global data objects can only be loaded to the program context once for each main program, whereas a global class can be loaded as many times as you like. This is known as multiple instantiation of the class. Values in the global data objects of a function group are the same for all function module calls.

By contrast, a class can have several instances (also known as objects), each of which is stored separately in the program context of the main program. Each instance can, therefore, have different attribute values. Consequently, a method sees different values in the attributes, depending on the instance for which it was called.

It is necessary to generate a class instance explicitly in the ABAP source code, using the CREATE OBJECT statement.

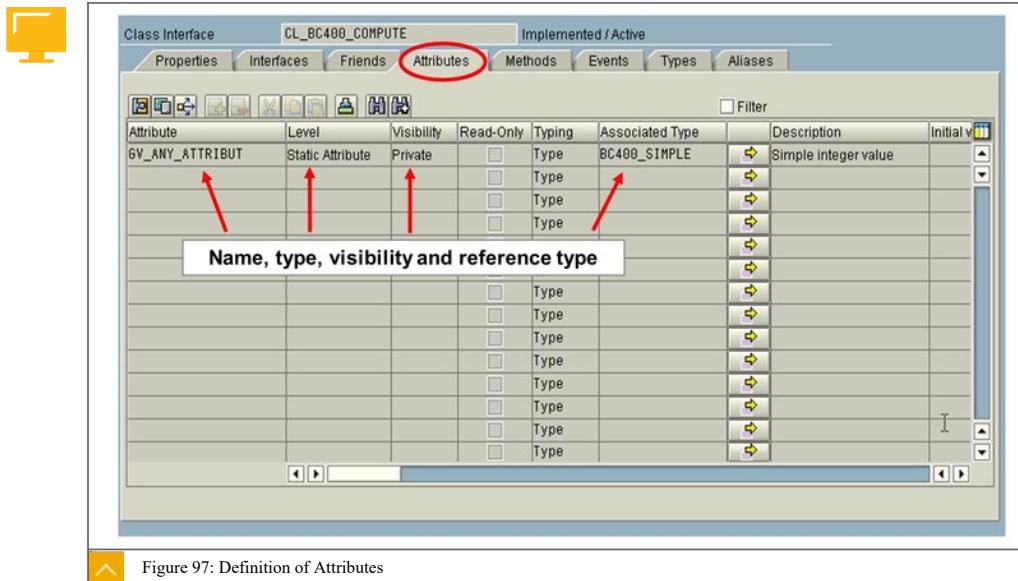
## Instance Components and Static Components

Attributes that can have a different value for each instance are known as instance attributes, to distinguish them from static attributes or class attributes. Static attributes exist only once for each program context, regardless of how many class instances are generated when a program runs. When instance methods access a static attribute, all instances read the same value. An instance method can change the value. All other instances then read the new value.

Regarding methods, a distinction is also made between static methods and instance methods. The main difference is that an instance method can only be called when a class instance is created beforehand. By comparison, static methods can be called without any previous instantiation of the class.

 Note:  
This course focuses primarily on static methods.

## Attributes of Global Classes

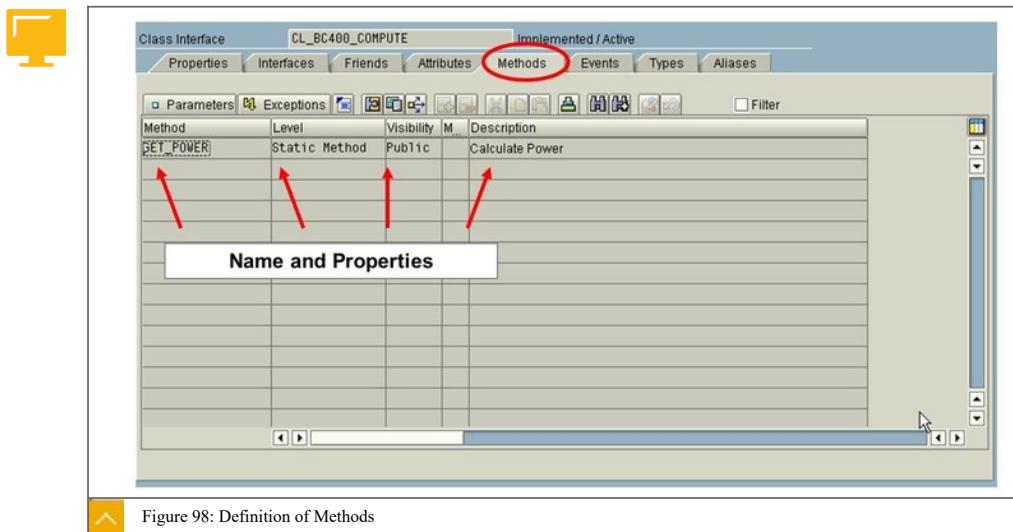


As with other Repository objects, the display and processing of global classes is incorporated in the Object Navigator. You have several options for opening a global class. In the navigation area, for example, choose **Class/Interface** and enter the name of the class in the input field directly under the name. You can also display the object list for a package first and then double-click the name of the required class in the subnode **Class Library** → **Classes**.

To open the list of attribute definitions in the class, choose the **Attributes** tab.

If you only want to use the class, only those attributes that are public are of interest. You can address these attributes directly outside the class.

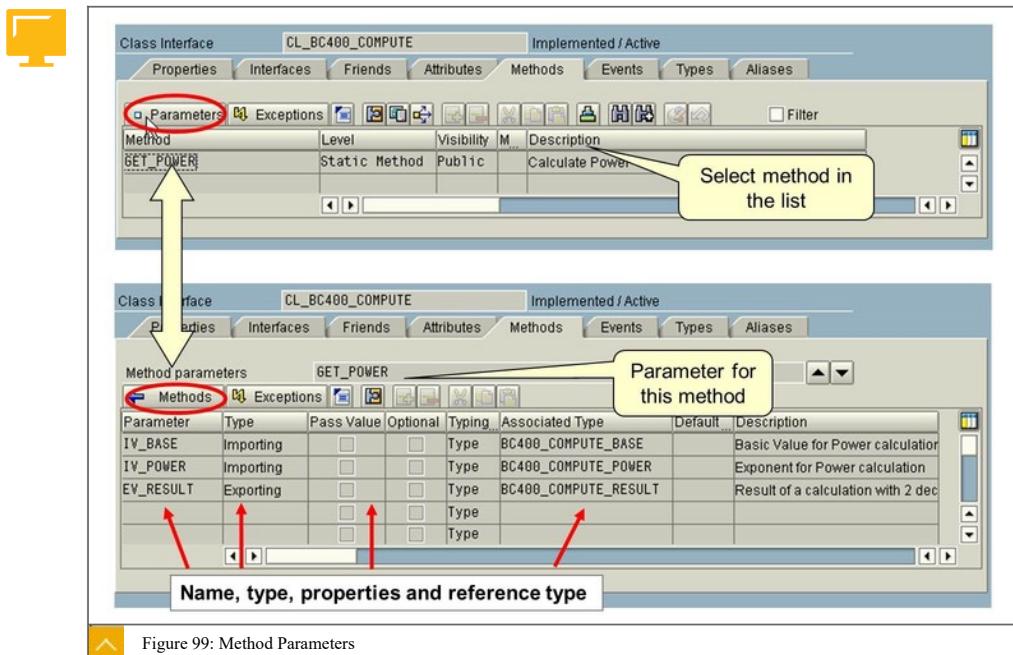
### Methods and Their Signatures



To open the list of all method definitions in the class, choose the **Methods** tab. With methods, only entries that are identified as public are of interest to the user. Only these public methods can be called externally; all other methods are used for modularization within the global class.

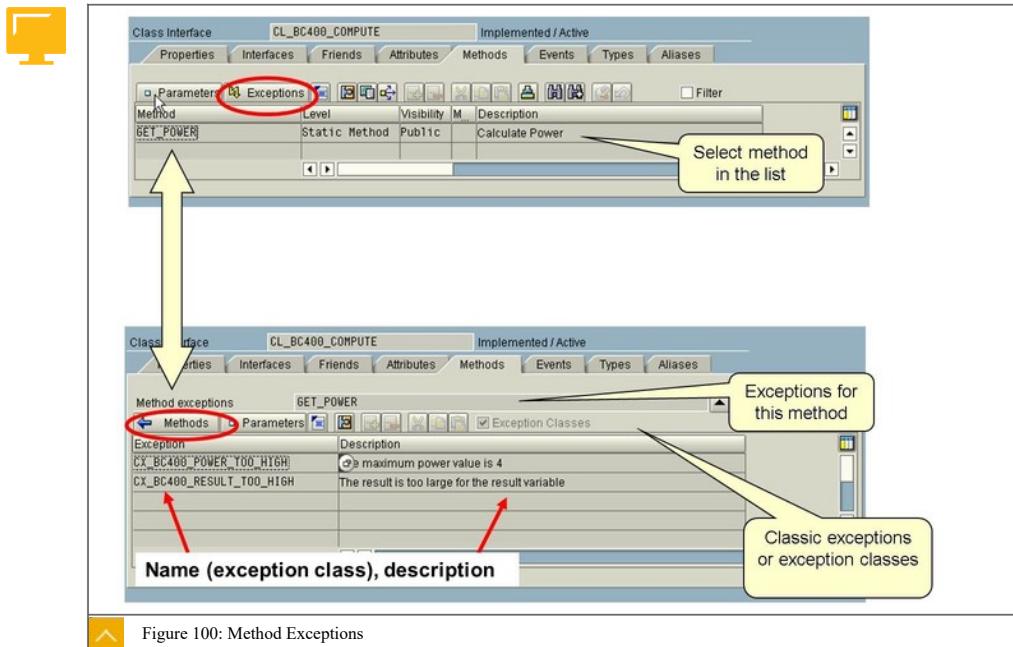
A method identified as a static method can be called directly, without the need to first generate an instance of the class. With instance methods, you need to create an instance and then call the method specifically for that instance. Under normal circumstances, the instance method then accesses instance attributes that contain different values for the various instances.

## Method Parameters



Parameters for a method are displayed in a separate window. You access the window from the method list by placing the cursor on the required method and clicking the Parameter button. In contrast to function modules, there is no separate list for export, import, and changing parameters. Instead, the parameter type is entered in the Type column.

## Method Exceptions



You need to navigate to a separate display window to display exceptions for a method. You access it from the method list by placing the cursor on the required method and clicking the Exceptions button.

Basis Release 6.10 introduced class-based exceptions in ABAP. Handling these exceptions is very different from handling the classic exceptions of function modules. A method can either raise only new class-based exceptions or only classic exceptions. It is not possible to mix both concepts. A checkbox shows you which exception concept is used by the respective method.

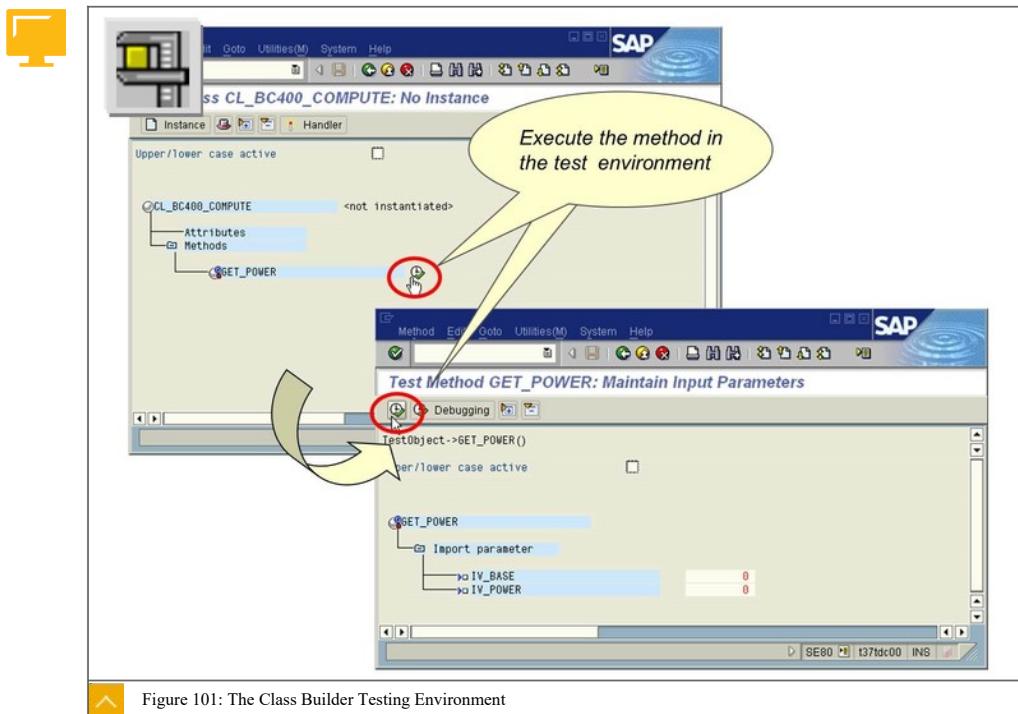
While the classic exception concept allows you to choose any exception identifier, with the new exception concept, you have to specify the names of special classes, the exception classes.



## Hint:

The class-based exception concept is also available for function modules.

### Global Class Documentation and Testing



With global class documentation, a distinction is made between the documentation of the class as a whole and the documentation of individual components.

You can access the class documentation by choosing the **Class Documentation** button. To consult the documentation for an individual method or attribute, navigate to the corresponding list, select the required component with the cursor, and choose the **Documentation** button.

You can test active global classes.

Temporary storage is allocated for the components of the class. For static components, this is allocated immediately, whereas for instance components the system only allocates storage after you have chosen the **Instance** button.

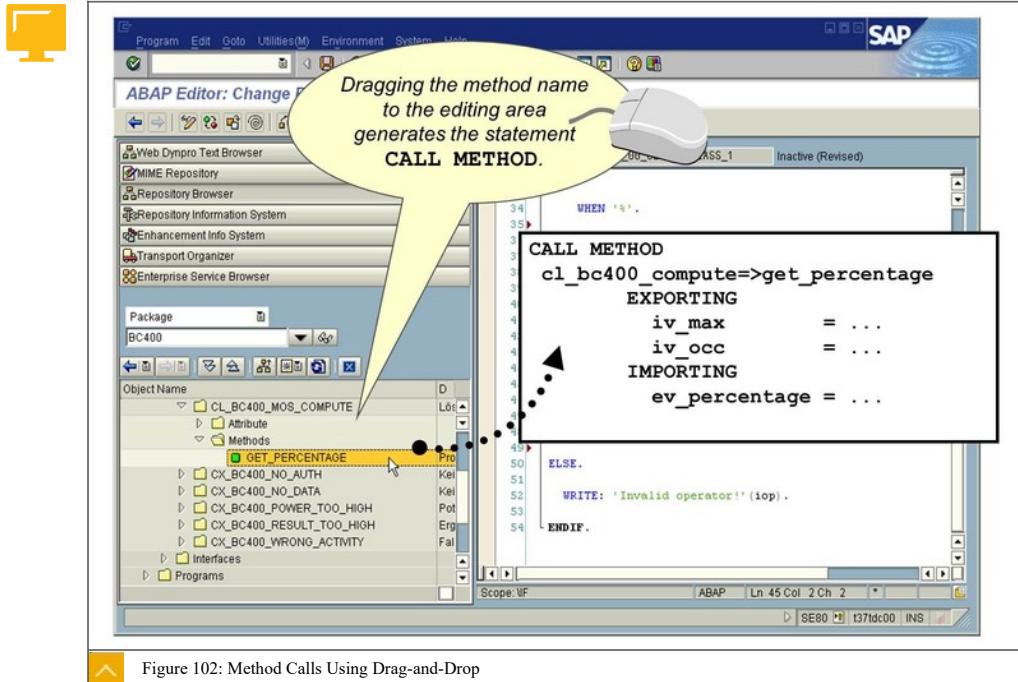
The system only lists the public components. You can test methods using the **Execute Method** button.

You do not need to generate an instance to test a static method. You can execute the static method immediately.

Any importing parameters that exist appear on the screen after you choose the **Execute** button.

After supplying the parameters with values, you can test the method. The system then displays the result of the exporting parameter.

## Static Methods



You use the CALL METHOD statement to call a method, and you then specify the method. You need to distinguish between an instance method and a static method. With static methods, this specification comprises the name of the class and the method, separated by the static component selector “=>” (double-headed arrow).

The parameters are then passed to the function module call in both an EXPORTING block and an IMPORTING block.

As with function modules, you have several options for generating the method call. SAP recommends using these options to avoid typing errors.

In the navigation area, select a method name and drag it to the editing area while holding down the left mouse button. When the statement is generated this way, you only need to add the actual parameters.

Alternatively, you can choose the Pattern button. Locate the Call Method option under ABAP Objects Pattern. Enter the class under Class/Interface and enter the name of the method under Method. There is no need to fill the Instance field with static methods.

### Exception Handling

Methods can raise either classic exceptions or class-based exceptions. Classic exceptions are handled in the same manner that you encountered with function modules.

#### Classic Exception Handling

 CALL METHOD cl\_bc400\_compute=>get\_power  
EXPORTING  
 iv\_base = pa\_int1

```
iv_power = pa_int2
IMPORTING
ev_result = gv_result
EXCEPTIONS
POWER_VALUE_TOO_HIGH = 1
RESULT_VALUE_TOO_HIGH = 2.

CASE sy-subrc.
WHEN 0.
WRITE gv_result.
WHEN 1.
WRITE 'Max Value for Power is 4'.
WHEN 2.
WRITE 'Result value was too high'.
ENDCASE.
```

In the EXCEPTIONS block of the method call, you assign a return code to the exception. When the method terminates with this classic exception, this return code is placed in the sy-subrc system field. By querying sy-subrc, the calling program can react to the exception.

In comparison, handling class-based exceptions is much more complex.

#### Class-Based Exceptions

##### Class-Based Exception Handling



```
TRY.
CALL METHOD cl_bc400_compute=>get_power
EXPORTING
iv_base = pa_int1
iv_power = pa_int2
IMPORTING
ev_result = gv_result.
WRITE gv_result.
CATCH cx_bc400_power_too_high .
WRITE 'Max Value for Power is 4'.
CATCH cx_bc400_result_too_high .
WRITE 'Result value was too high'.
ENDTRY.
```

The call must be bound between the TRY. and ENDTRY. statements. The exception is then handled in a processing block that begins with the CATCH <exception class>. statement, where <exception class> represents the exception class that is to be handled. If the corresponding exception is raised within the TRY-ENDTRY block, processing is terminated and the program flow branches directly to the relevant CATCH block.

## Exception Handling with Exception Classes

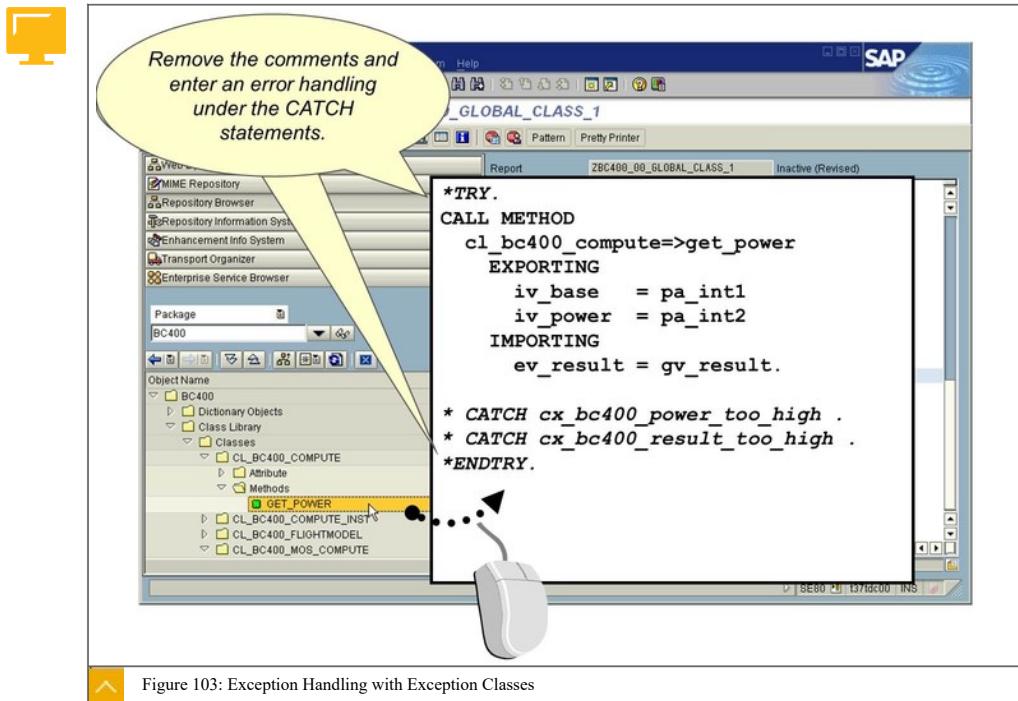
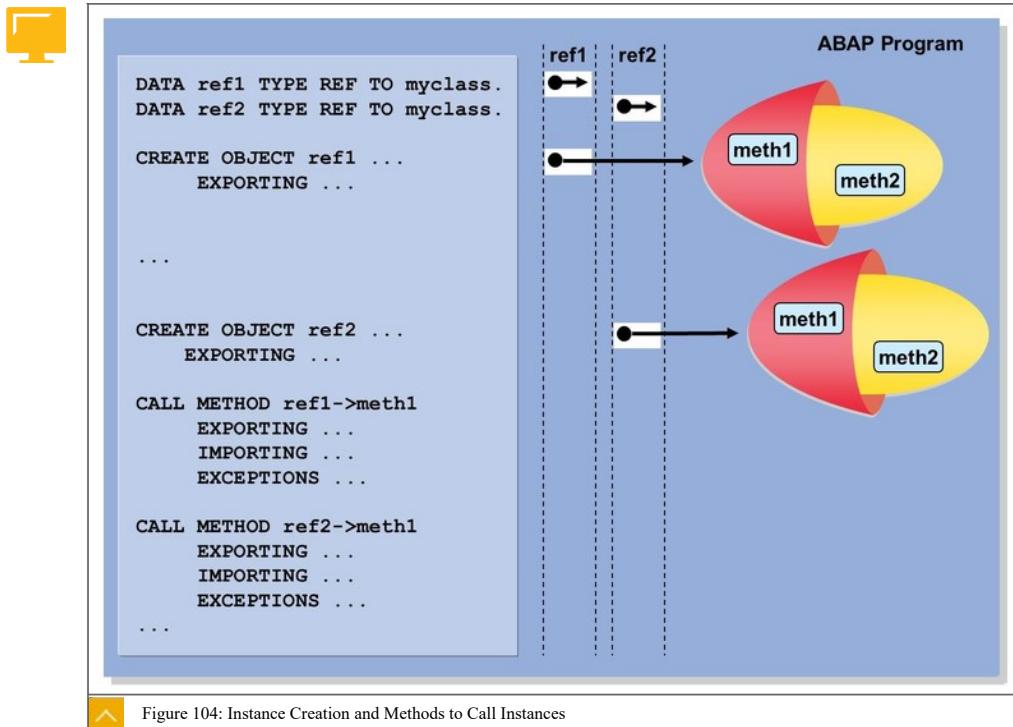


Figure 103: Exception Handling with Exception Classes

If you generate method calls by using the drag-and-drop function or by using the Pattern button, the system generates the statements for handling the class-based exceptions as well. You then need to remove the comments and implement the CATCH-blocks.

## Instance Creation and Methods to Call Instances



Because instances do not have names, you need to define reference variables to create and address instances of classes. Reference variables are pointers that you can direct to corresponding instances. Each of the reference variables has a name that can be used to address the corresponding instance.

Reference variables are defined using DATA reference\_name TYPE REF TO class\_name.

When the program starts, a reference variable still has its initial value (it does not point to an instance). After it has been used to create an instance, the reference variable no longer has the initial value and points to that instance.

You can use the CREATE OBJECT reference\_name statement to create an instance of the class that was specified in the definition of reference variables. Afterwards, the reference variable points to the newly created instance.

When you use CREATE OBJECT, you might need to supply the import parameters of the special method CONSTRUCTOR with data. This special method is automatically executed immediately after the creation of the instance. With its import parameters, this special method maintains the corresponding attributes of the new instance.

You call methods of an instance using the CALL METHOD reference\_name->method\_name statement.

In contrast to calling a function module, the method name alone is not sufficient in this case. You need to specify the relevant instance as well, because the program can have several instances of that class.



### LESSON SUMMARY

You should now be able to:

- Describe object-oriented programming
- Use methods of global classes
- Use instances

## Unit 4

### Lesson 7

# Creating Global Classes and Static Methods

#### LESSON OVERVIEW

This lesson explains how to create a global class with a method to encapsulate a function for reuse. This lesson limits itself to creating a static method. You will learn how to use global classes and static methods in your ABAP programs.

#### Business Example

You want to develop an application in which several functions are used in various programs. You are in charge of implementing calculation functions. You decide to program it as a global class with static methods. For this reason, you require the following knowledge:

- An understanding of how to create global classes
- An understanding of how to create static methods



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create simple global classes and static methods

### Creation of Global Classes

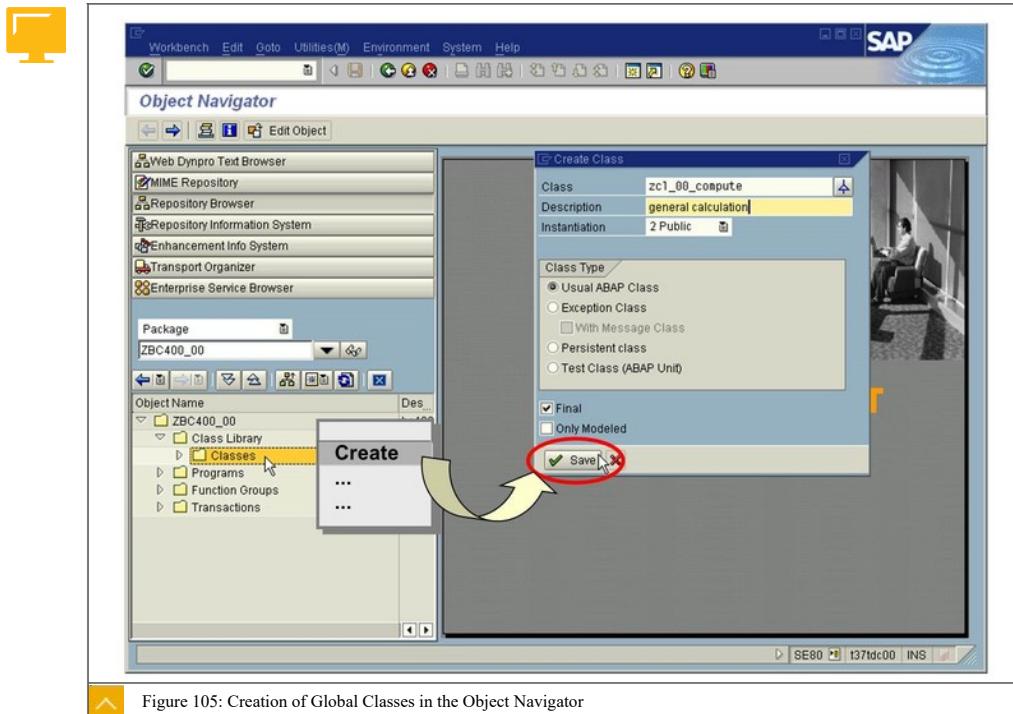
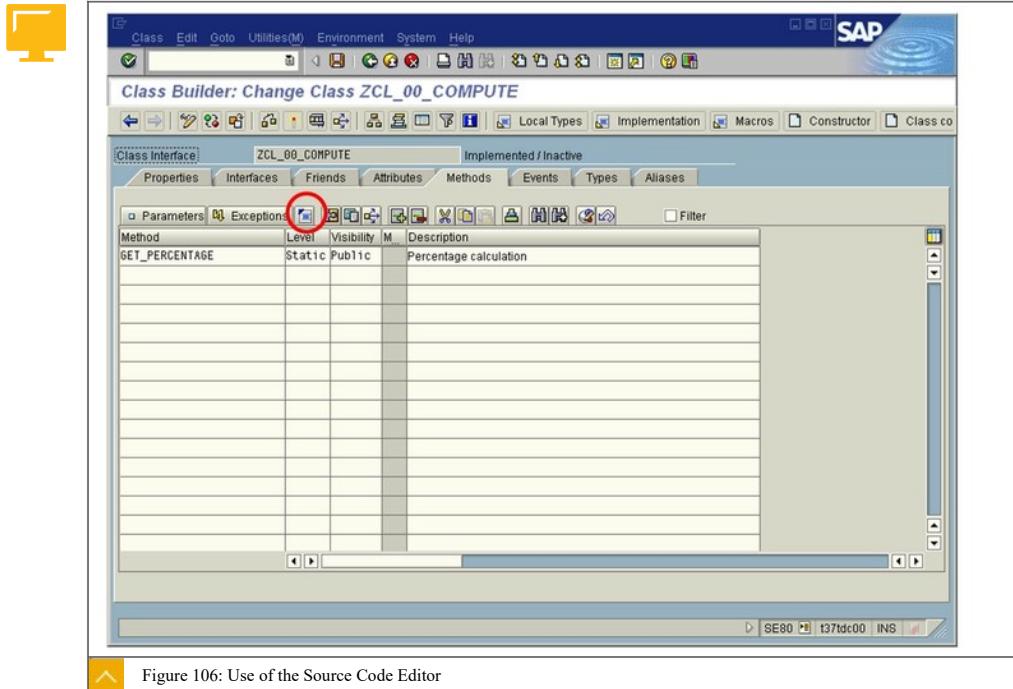


Figure 105: Creation of Global Classes in the Object Navigator

To create a global class, open the context menu for your package in the navigation area and choose Create → Class Library → Class . In the dialog box that appears, enter the name of the class, a short description of the class, and several other properties. After saving your entries, assign the class to a workbench request.

Alternatively, you can choose Class/Interface from the dropdown list in the navigation area, enter the name of the new class in the field below it, and choose the Display button. Confirm that you want to create this new class.

### Creation of Static Methods



To create a static method, simply enter the method name in the Method list on the Method tab page. You can maintain the visibility and type of method (static or instance) using the corresponding input help.

To create a parameter for a method, branch to the parameter list for the corresponding method. Switch to change mode and enter the name of the parameter in the list. Use the input help to maintain the parameter type (import, export, and so on) and specify an associated type.

To implement the source code for a method, select the method in the method list and choose Source Code . Implement the source code between the METHOD <method\_name>. and ENDMETHOD. statements.

## Display of the Signature in the Source Code Editor

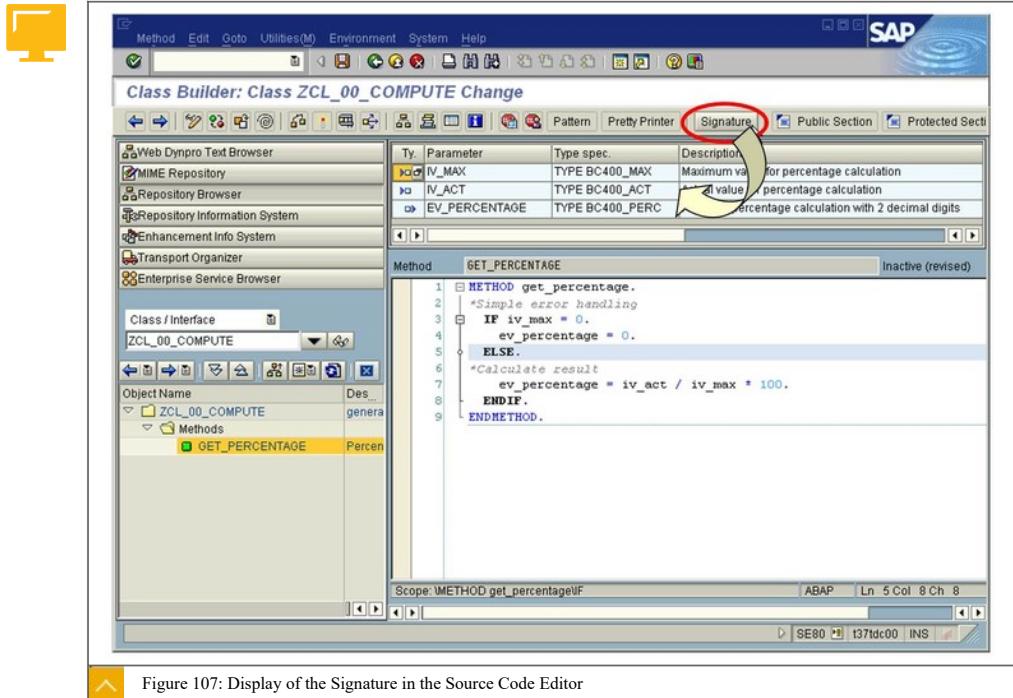


Figure 107: Display of the Signature in the Source Code Editor

While you are editing source code, you can display the method signature method. To do this, choose the **Signature**, as shown in the figure.



## LESSON SUMMARY

You should now be able to:

- Create simple global classes and static methods

## Unit 4

### Lesson 8

## Using Local Classes

### LESSON OVERVIEW

In this lesson, you are given a brief preview of how local classes enable the use of object-oriented programming techniques for internal program modularization.

#### Business Example

You want to encapsulate a function in a static method of a local class. For this reason, you require the following knowledge:

- An understanding of the definition, implementation, and use of local classes



### LESSON OBJECTIVES

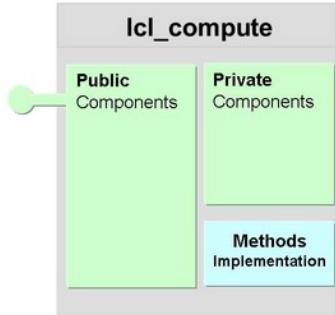
After completing this lesson, you will be able to:

- Use local classes

#### Local Classes



```
CLASS lcl_compute DEFINITION.  
  PUBLIC SECTION.  
    ...  
  
  PRIVATE SECTION.  
    ...  
  
  ENDCCLASS.  
  
CLASS lcl_compute IMPLEMENTATION.  
  ...  
  
  ENDCCLASS.
```



You have already learned how to define and use global classes. This lesson explains the definition and usage of local classes. Local classes are so named because they can only be used locally within the program in which they were defined.

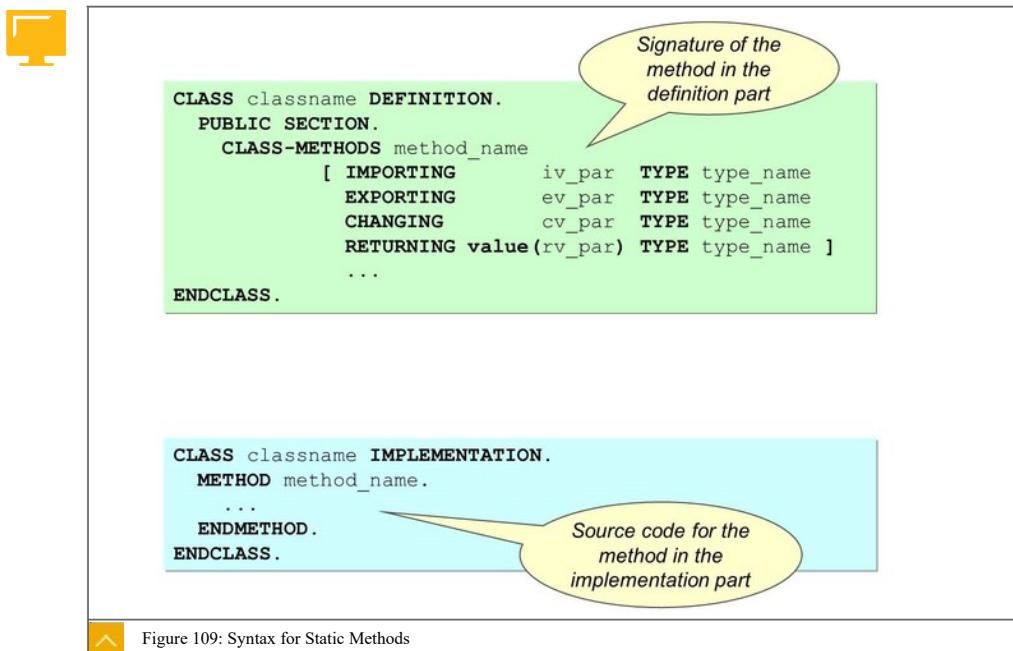
The main difference between global classes and local classes is the way in which they are defined. While global classes are maintained with a special tool called the Class Builder, local classes are created directly in the source code of the respective main program.

The figure shows how to create a local class in the source code of the program. A distinction is made between the definition part and the implementation part of the class. The description of the attributes and signatures of the methods are located in the definition part, whereas the implementation part contains the source code of the methods.

The CLASS ... ENDCLASS statement is a local definition in the program. Just as the TYPES statement defines local data types, the CLASS ... ENDCLASS statement defines local object types.

The definition part is divided into several sections in which the public and private components are defined (PUBLIC SECTION. and PRIVATE SECTION.).

#### Syntax for Static Methods



The figure shows a schematic representation of how a public static method is defined and implemented. The method is public because the definition is located in the PUBLIC SECTION. section of the class. To define an instance method, as opposed to the static method, the METHODS statement is used instead of CLASS-METHODS.

Methods have a signature (interface parameters and exceptions) that enable them to receive values when they are called and pass values back to the calling program. Methods can have any number of IMPORTING, EXPORTING, and CHANGING parameters. All parameters can be passed as values or as references.

You can define all input parameters (IMPORTING and CHANGING parameters) as optional parameters in the declaration using the OPTIONAL addition. The calling program does not have to pass these parameters when the system calls the object. The DEFAULT addition enables you to specify a start value.

## Implementation and Use of a Static Method

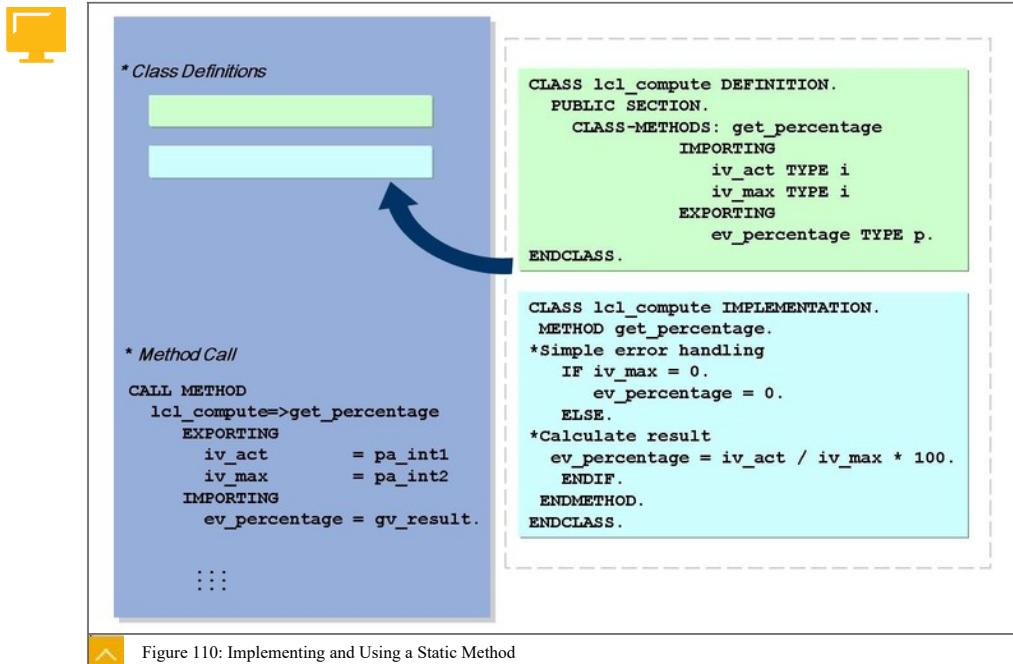


Figure 110: Implementing and Using a Static Method

The syntax example shows how to define a local class with a static method (upper-right section) and how to implement the method (lower-right section).

In the section on the left, you can see the static method call from the main program.

Notice that the call is identical to a static method call in a global class.

You can define the local class directly in the source code of the main program or in an INCLUDE program.



## Hint:

You need to define the local class before the call.

This is different from subroutines, which you normally define below the call statement.



## LESSON SUMMARY

You should now be able to:

- Use local classes

## Unit 4

### Learning Assessment

1. What are the uses of modularization?

Choose the correct answers.

- A To improve performance
- B To provide a better overview of program layout
- C To encapsulate a function that is required many times within a program for multiple use
- D To implement the central maintainability of a function within a program
- E To make a function available across the system

2. Which of the following is the name of the assignment of actual parameters to formal parameters when calling a subroutine?

Choose the correct answer.

- A Interface assignment
- B Parameter passing
- C Interface call
- D Subroutine call

3. Which of the following is the name of variables defined in the main program?

Choose the correct answer.

- A Global data objects
- B Local data types
- C Local and global data types

4. Which of the following elements does the interface of a function module contain?

Choose the correct answers.

- A Export parameter  
 B Subroutines  
 C Changing parameter  
 D Screen

5. Which of the following is the tab page that you can switch to implement the function module, after defining the corresponding interface?

Choose the correct answer.

- A Standard  
 B Exceptions  
 C Source Code  
 D Attributes

6. After defining the corresponding IMPORTING and EXPORTING parameters, you can switch to the Source code tab page to implement the functions of the function module.

Determine whether this statement is true or false.

- True  
 False

7. Data in the SAP system can be accessed by means of a Business Application Programming Interface (BAPI).

Determine whether this statement is true or false.

- True  
 False

8. Which of the following can be used as a visibility option for an attribute?

Choose the correct answers.

- A Public  
 B Private  
 C Static  
 D Instance

9. Static Methods are called using the CALL CLASS METHOD statement.

Determine whether this statement is true or false.

True

False

10. When you identify a method as a static method, it can be called directly without the need to generate an instance of the class first.

Determine whether this statement is true or false.

True

False

11. To create a static method, all you need to do is enter its name in the Constructor list.

Determine whether this statement is true or false.

True

False

12. Which of the following special tools maintains global classes?

Choose the correct answer.

A Object Builder

B Class Builder

C Method Builder

D Attribute Builder

13. To define an instance method as opposed to a static method in a local class, the METHODS statement is used instead of CLASS-METHODS.

Determine whether this statement is true or false.

True

False

## Unit 4

### Learning Assessment - Answers

1. What are the uses of modularization?

Choose the correct answers.

- A To improve performance
- B To provide a better overview of program layout
- C To encapsulate a function that is required many times within a program for multiple use
- D To implement the central maintainability of a function within a program
- E To make a function available across the system

You are correct! A modularization unit encapsulates a function. You can use most of the modularization units in more than one program. Therefore, they are often termed as reuse units. The improvement in transparency is a result of the program becoming more function-oriented. Modularization makes it easier to maintain programs, because you only need to make changes to the modularization unit, and not at various points in the main program. Read more in the lesson, Explaining Modularization, Task: Modularization Techniques, in the course BC400 (Unit 4, Lesson 1) or TAW10 Part I (Unit 10, Lesson 1).

2. Which of the following is the name of the assignment of actual parameters to formal parameters when calling a subroutine?

Choose the correct answer.

- A Interface assignment
- B Parameter passing
- C Interface call
- D Subroutine call

You are correct! The assignment of actual parameters to formal parameters when calling a subroutine is called parameter passing. Read more in the lesson, Defining and Calling Subroutines, Task: Parameter Definition for Subroutines, in the course BC400 (Unit 4, Lesson 2) or TAW10 Part I (Unit 10, Lesson 2).

3. Which of the following is the name of variables defined in the main program?

Choose the correct answer.

A Global data objects

B Local data types

C Local and global data types

You are correct! Variables defined in the main program are global data objects. They are visible and can be addressed in the entire main program, as well as in every subroutine called. Read more in the lesson, Defining and Calling Subroutines, Task: Local and Global Data Objects, in the course BC400 (Unit 4, Lesson 2) or TAW10 Part I (Unit 10, Lesson 2).

4. Which of the following elements does the interface of a function module contain?

Choose the correct answers.

A Export parameter

B Subroutines

C Changing parameter

D Screen

You are correct! Each function module has an interface with parameters for importing or exporting data. The interface of the function module consists of the import, export, changing parameters, and exceptions. Read more in the lesson, Calling Function Modules, Task: Examination of a Function Module, in the course BC400 (Unit 4, Lesson 3) or TAW10 Part I (Unit 10, Lesson 3).

5. Which of the following is the tab page that you can switch to implement the function module, after defining the corresponding interface?

Choose the correct answer.

A Standard

B Exceptions

C Source Code

D Attributes

You are correct! After defining the corresponding IMPORTING and EXPORTING parameters, you can switch to the Source code tab page to implement the functions of the function module. Read more in the lesson, Creating Function Modules, Task: Source Code Editing, in the course BC400 (Unit 4, Lesson 4) or TAW10 Part I (Unit 10, Lesson 4).

6. After defining the corresponding IMPORTING and EXPORTING parameters, you can switch to the Source code tab page to implement the functions of the function module.

Determine whether this statement is true or false.

 True False

You are correct! After defining the corresponding IMPORTING and EXPORTING parameters, you can switch to the Source code tab page to implement the functions of the function module. Read more in the lesson, Creating Function Modules, Task: Source Code Editing, in the course BC400 (Unit 4, Lesson 4) or TAW10 Part I (Unit 10, Lesson 4).

7. Data in the SAP system can be accessed by means of a Business Application Programming Interface (BAPI).

Determine whether this statement is true or false.

 True False

You are correct! The Business Object Repository (BOR) in the SAP system contains business objects types. A business object type is a program that behaves like a class. It represents an SAP table or a table hierarchy. A business object has BAPIs as methods. You can call these BAPIs to access the corresponding tables. A BAPI is a means of accessing data in the SAP system. Read more in the lesson, Describing Business Application Programming Interfaces (BAPIs), Task: Business Application Programming Interface (BAPI), in the course BC400 (Unit 4, Lesson 5) or TAW10 Part I (Unit 10, Lesson 5).

8. Which of the following can be used as a visibility option for an attribute?

Choose the correct answers.

 A Public B Private C Static D Instance

You are correct! Attributes are normally encapsulated in the class by defining them PRIVATE, and can therefore only be read or changed using methods of the same class. Classes allow you to make specific attributes visible to users of the class, if defined PUBLIC. Read more in the lesson, Calling Methods of Global Classes Task Example of Access Options for a Global Class, in the course BC400 (Unit 4, Lesson 6) or TAW10 Part I (Unit 10, Lesson 6).

9. Static Methods are called using the CALL CLASS METHOD statement.

Determine whether this statement is true or false.

True

False

You are correct! You use the CALL METHOD statement to call a method, and you then specify the method. With static methods, this specification comprises the name of the class and the method, separated by the static component selector “=>” (double-headed arrow). Read more in the lesson, Calling Methods of Global Classes, Task: Static Methods, in the course BC400 (Unit 4, Lesson 6) or TAW10 Part I (Unit 10, Lesson 6).

10. When you identify a method as a static method, it can be called directly without the need to generate an instance of the class first.

Determine whether this statement is true or false.

True

False

You are correct! You don't need to generate an instance to test a static method. You can execute the static method immediately. Read more in the lesson, Calling Methods of Global Classes, Task: Global Class Documentation and Testing, in the course BC400 (Unit 4, Lesson 6) or TAW10 Part I (Unit 10, Lesson 6).

11. To create a static method, all you need to do is enter its name in the Constructor list.

Determine whether this statement is true or false.

True

False

You are correct! To create a static method using the Class Builder, simply enter the method name in the Method list on the Method tab page. Read more in the lesson, Creating Global Classes and Static Methods, Task: Creation of Static Methods, in the course BC400 (Unit 4, Lesson 7) or TAW10 Part I (Unit 10, Lesson 7).

12. Which of the following special tools maintains global classes?

Choose the correct answer.

- A Object Builder  
 B Class Builder  
 C Method Builder  
 D Attribute Builder

You are correct! Global classes are maintained with a special tool called the Class Builder. Read more in the lesson, Using Local Classes, Task: Local Classes, in the course BC400 (Unit 4, Lesson 8) or TAW10 Part I (Unit 10, Lesson 8).

13. To define an instance method as opposed to a static method in a local class, the METHODS statement is used instead of CLASS-METHODS.

Determine whether this statement is true or false.

- True  
 False

You are correct! To define an instance method in a local class, as opposed to the static method, the METHODS statement is used instead of CLASS-METHODS. Read more in the lesson, Using Local Classes, Task: Syntax for Static Methods, in the course BC400 (Unit 4, Lesson 8) or TAW10 Part I (Unit 10, Lesson 8).

## UNIT 5

# Complex Data Objects

### Lesson 1

Using Structured Data Objects

156

### Lesson 2

Using Internal Tables

162

### UNIT OBJECTIVES

- Define structured data objects
- Implement basic ABAP statements for structured data objects
- Analyze structured data objects in debugging mode
- Define internal tables
- Implement basic ABAP statements with internal tables
- Analyze internal tables in debugging mode

## Unit 5

### Lesson 1

# Using Structured Data Objects

#### LESSON OVERVIEW

This lesson covers the definition and structured data objects (structured variables) and their analysis in the ABAP Debugger. You will also learn how to use basic ABAP statements for structured data objects.

#### Business Example

You are to process your own data structures and search your programs for semantic errors using the ABAP Debugger. For this reason, you need to know:

- How to define structured data objects (structured variables)
- How to use basic ABAP statements for structured data objects
- How to analyze structured data objects in debugging mode

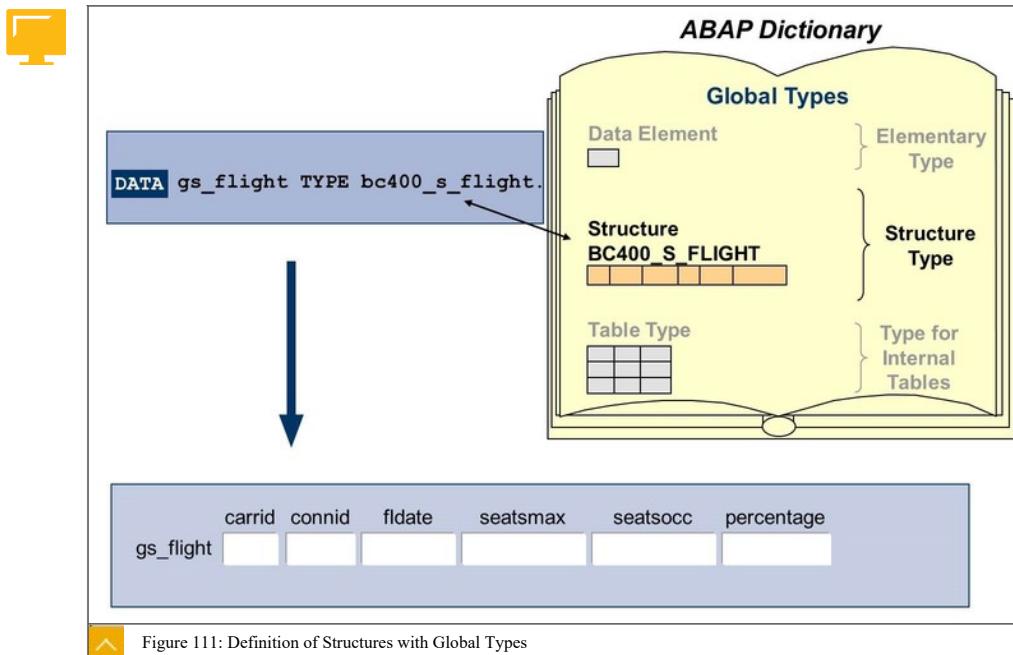


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Define structured data objects
- Implement basic ABAP statements for structured data objects
- Analyze structured data objects in debugging mode

## Definition of Structured Data Objects



## Structures Used When Setting Types

When you set the types, you can refer to the following:

- A structure in the ABAP Dictionary (global structure type)
- A structure type that you declare locally in the program

For names of structure types and structure variables, the following definitions are added to the naming conventions:

Table 4: Definitions Added to the Naming Conventions

Purpose	Prefix
Program global* or local** structured type	ts_
Program global* structured variables	gs_
Local** structured variable	ls_



## Note:

\* Program global in this case means a type or variable that is globally visible within your program. However, it is locally defined in your program. You also can define types as system global (via entries in the ABAP Data Dictionary). These type definitions are visible (and usable) within all ABAP programs of your SAP system.

\*\* Local in this case means local to a subroutine.

By defining structured data objects (known as structured variables or simply structures) in ABAP, you can combine values that belong together logically to one data object. Structures can be nested. This means that components can be made up of structures or even internal tables.

You define structure variables in the program with the DATA statement like you do with elementary data objects.

## Definition of Structures with Local Types



```
TYPES: BEGIN OF ts_flightinfo,
      carrid      TYPE s_carr_id,
      carrname    TYPE s_carrname,
      connid      TYPE s_conn_id,
      fldate      TYPE s_date,
      percentage  TYPE p LENGTH 3 DECIMALS 2,
   END OF ts_flightinfo.

DATA gs_flightinfo TYPE ts_flightinfo.
```

Declaration of a local structure type

Definition of a structure variable



Figure 112: Definition of Structures with Local Types

The figure shows the definition of a structure variable using a locally declared structure type.

Use the TYPES statement to define local structure types. Here, the components are enclosed as shown.

`TYPES: BEGIN OF structure_type, ... , END OF structure_type.`

Assign a type to each component by using the TYPE addition. For more details, refer to the keyword documentation for the TYPES statement.

Define the data object in the usual way.

If necessary, you can also define a structured data object directly. To do so, all you have to do is replace the leading keyword TYPES with DATA.

`DATA: BEGIN OF structure_name, ... , END OF structure_name.`

## Access to Structure Components



```

DATA gs_scarr TYPE scarr.

gs_scarr->carrid = 'LH'.

CALL METHOD cl_bc400_flightmodel->get_carrier
      EXPORTING iv_carrid = gs_scarr->carrid
      IMPORTING es_carrier = gs_scarr.

WRITE: / gs_scarr->carrid,
         gs_scarr->carrname,
         gs_scarr->url.

```

mandt	carrid	carrname	currcode	url	
gs_scarr	400	LH	Lufthansa	EUR	<a href="http://www.lufthansa.com">http://www.lufthansa.com</a>

Figure 113: Access to Structure Components

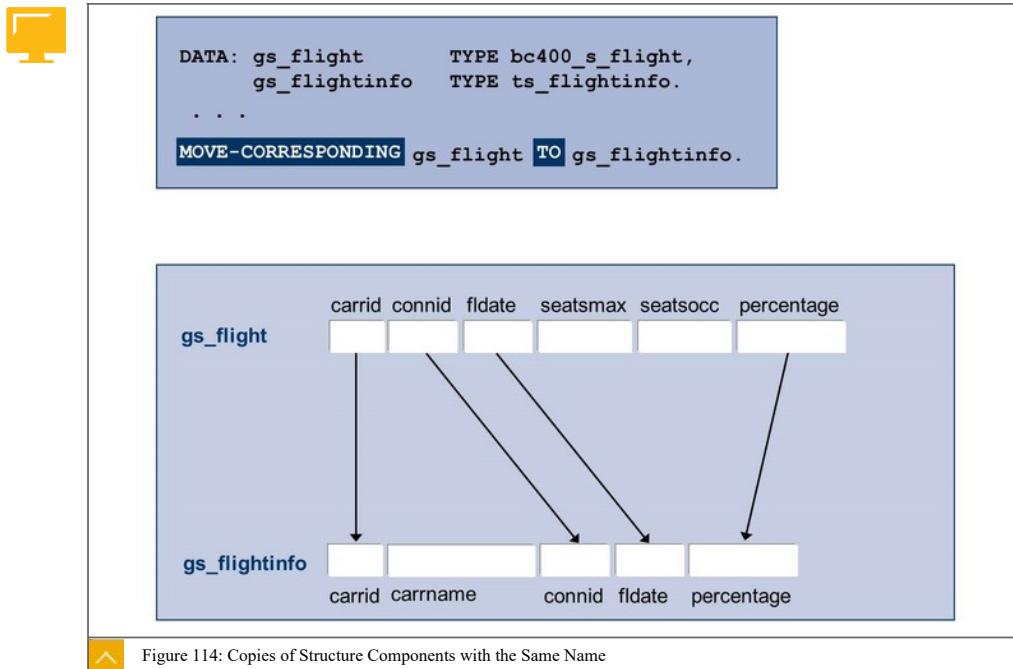
Components of a structure are always addressed using a hyphen: structure\_name - component\_name.



## Hint:

In principle, the ABAP syntax allows the names of data objects to contain hyphens, such as DATA h-var TYPE c LENGTH 5. However, to avoid confusion with the addressing structure components, you should avoid using hyphens as part of the name.

## Use of Structured Data Objects



The statement, MOVE-CORRESPONDING, copies the content of the source structure to the target structure, one component at a time. In this case, only those components that are available under the same name in both the source structure and the target structure are considered. All other components of the structures remain unchanged.

The individual assignments are executed like in the MOVE statement.

### Structured Data Objects in Debugging Mode

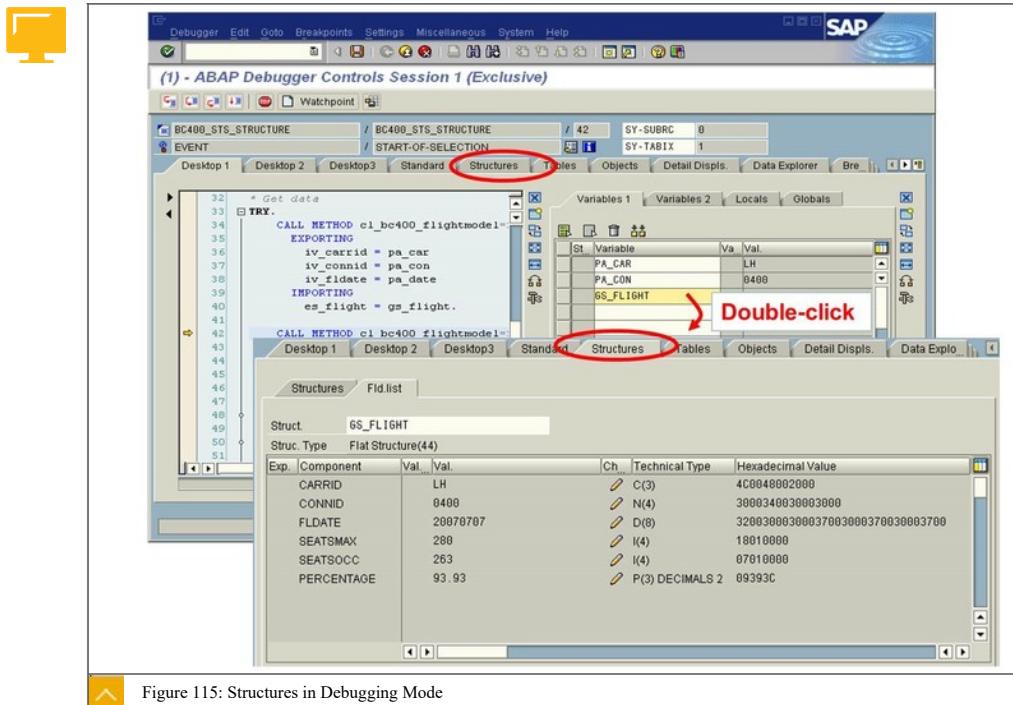


Figure 115: Structures in Debugging Mode

Trace the field content of a structure in the ABAP Debugger by entering the structure name in the Variable 1 area, or copy it from the source code by double-clicking it. Display the component of the structure by double-clicking the structure name in the Variable 1 area. In addition, you can configure a display area for structure display on one of the desktops.

To display a structure variable in the classic ABAP Debugger, use the double-click method. By double-clicking the structure in the source code, you copy the structure to the field view. Double-clicking the field view takes you to the structure display, where you can read the name, content, and type of the individual components.



### LESSON SUMMARY

You should now be able to:

- Define structured data objects
- Implement basic ABAP statements for structured data objects
- Analyze structured data objects in debugging mode

## Unit 5

### Lesson 2

# Using Internal Tables

#### LESSON OVERVIEW

This lesson explains how to define internal tables and use them in ABAP programs. You will also analyze the internal tables at runtime in the ABAP Debugger.

#### Business Example

You use table variables as data stores in your programs and then search for semantic errors in those programs using the ABAP Debugger. For this reason, you need to know:

- How to define internal tables
- How to use basic ABAP statements with internal tables
- How to analyze internal tables in debugging mode

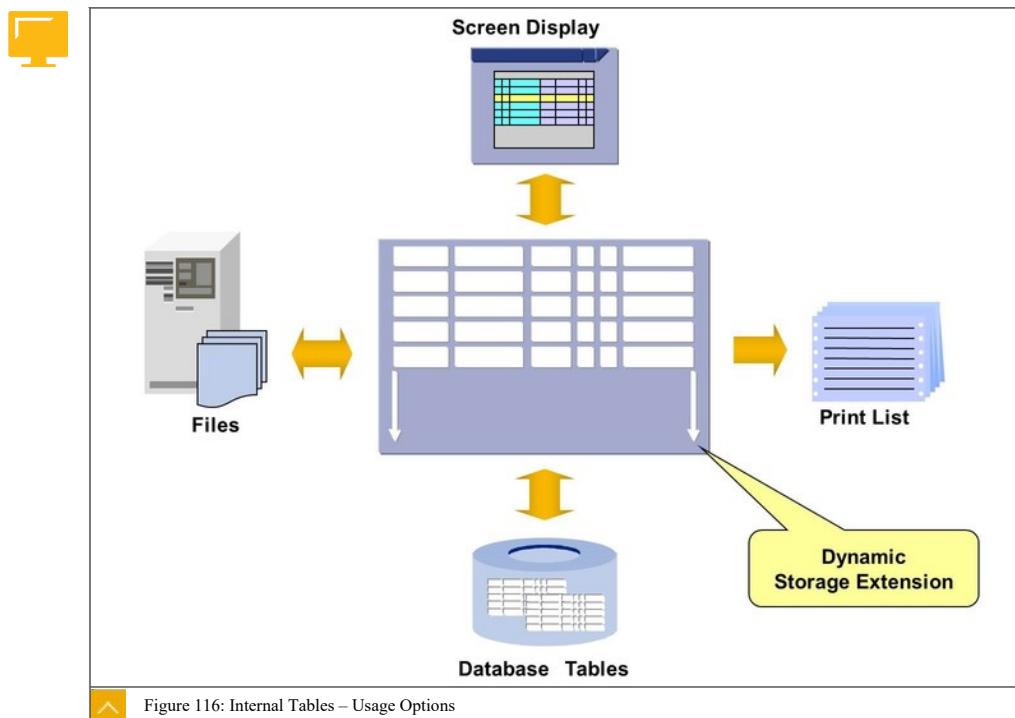


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Define internal tables
- Implement basic ABAP statements with internal tables
- Analyze internal tables in debugging mode

## Table Types



An internal table is a data object in which you can keep several identically structured data records at runtime (table variable). The number of data records is restricted only by the capacity of specific system installations.

The ABAP runtime system dynamically manages the size of the internal table. This means that memory management is not a concern for the developer.

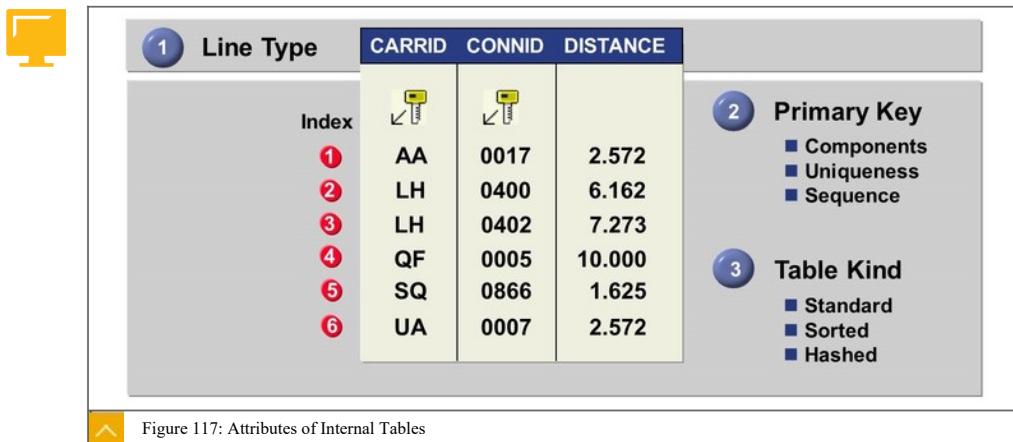
The individual data records in an internal table are known as table rows or table entries. The individual components in a row are referred to as fields or columns of the internal table.

Internal tables are a simple way of processing large data sets in a structured manner.

#### Typical Uses of an Internal Table

- Storing of data from database tables or sequential files for future processing.
- Preparation of data for screen or printer output (for example, sort).
- Preparation of data for using other services (for example, for method, function module, or subroutine calls).

## Properties of Internal Tables



## Properties of an Internal Table

- Line type

The line type describes the structure of table rows. The developer usually specifies a structure type for that, but any data types are possible.

- Primary key

The primary key of an internal table consists of key fields in a particular order. The system uses the sequence of the key fields, among other things, for sorting according to keys. Depending on the access type, the primary key can be defined as unique or non-unique. Uniqueness means that a particular combination of key field values can only appear once within the table.

- Table kind

Table kinds are of three types: standard, sorted, and hashed. Depending on the access type used, you should use the appropriate table kind for the definition to enable high performance access.

## Attributes and Uses of Tables



	Index Tables		Hashed Table
Table Kind	STANDARD TABLE	SORTED TABLE	HASHED TABLE
Index Access 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Key Access 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Key Uniqueness	NON-UNIQUE	UNIQUE/NON-UNIQUE	UNIQUE
Use in	Mainly Index Access	Mainly Key Access	Only Key Access

Figure 118: Attributes and Uses of Tables

The figure illustrates the selection of the appropriate table type.

#### Table Entry Access

You can access a table entry in the following ways:

- Index access  
Identify the table entry by its row number.
- Key access  
Identify the table entry by its values in fields.

Depending on the access type, choose the most suitable kinds of table to enable high performance access.

#### Kinds of Tables

The following kinds of internal tables exist:

##### Standard tables

The runtime environment maintains an internal row numbering (index). Both index and key accesses are possible.

Choose this kind of table if you mainly use index access for this internal table.

##### Sorted tables

The runtime environment keeps the data records sorted by the key fields in ascending order. It also maintains the internal index. Both index and key accesses are possible.

Choose this kind of table if you normally access the internal table with the key or would like the table to be automatically sorted by key.

### Hashed tables

The runtime environment manages the data records for fast key access using the hashing procedure. A unique key is required. With hashed tables, only key accesses are possible.

Choose this kind of table if the internal table is very large and you want to access it by key only.



Hint:

Only standard tables are used for this course. However, with the exception of a few special cases, the syntax is identical for all three table kinds.

### Definition of Internal Tables

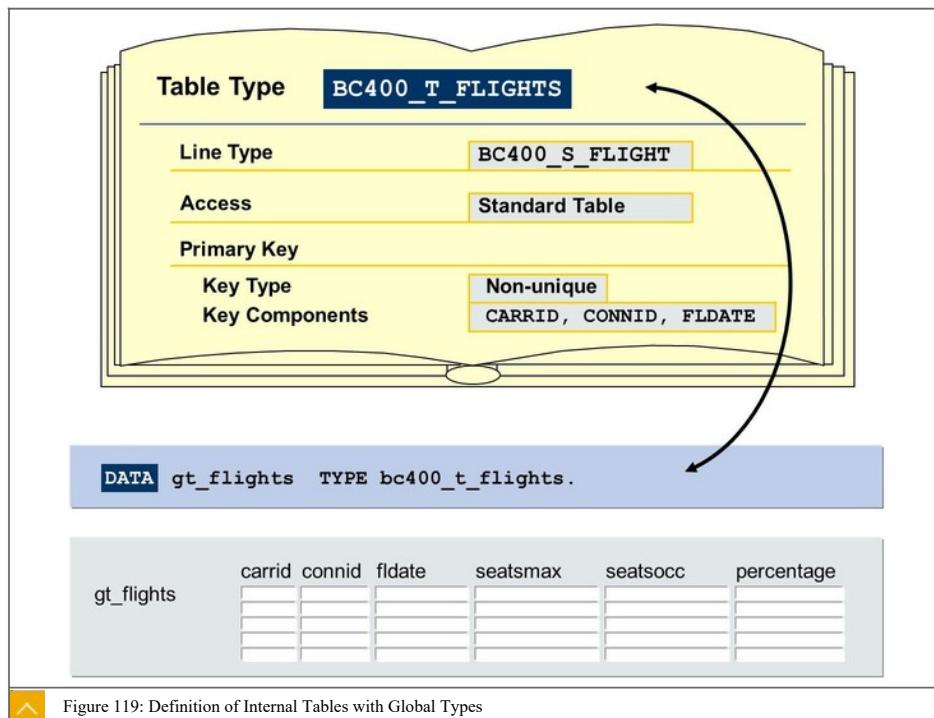


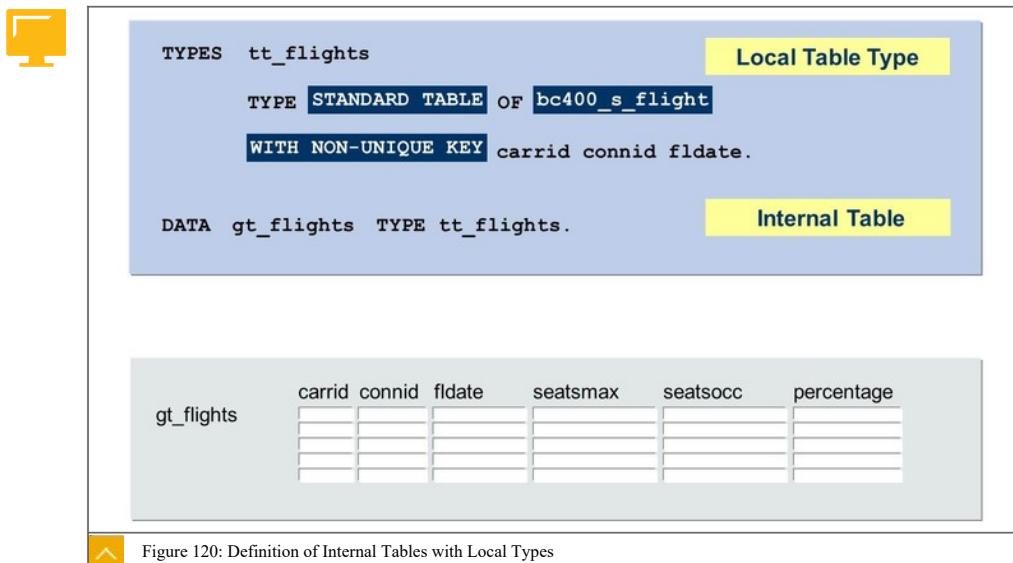
Figure 119: Definition of Internal Tables with Global Types

The type of an internal table is called a table type. Table types can be defined globally, in the ABAP Dictionary, or locally, within a program.

The figure illustrates a table type declared in the ABAP Dictionary, as well as the program-internal definition of a table variable with reference to the table type.

For detailed information about the declaration of global table types in the ABAP Dictionary, refer to the online documentation. You can access this using (Application Help) when displaying the table type.

## Definition of Internal Tables with Local Types



The figure illustrates a table type declared locally in the program as well as the internal definition of a table variable in the program with reference to the locally declared table type.

When you list key fields in the table type, note that the sequence matters for certain types of processing (such as “sort by key”). For detailed information about declaring local table types, refer to the keyword documentation for the TYPES statement.



## Hint:

Instead of defining a table type first you can also define an internal table directly. All you have to do here is use DATA instead of TYPES.



## Caution:

A common beginner’s error consists of the following syntax:

```
DATA gt_itab TYPE TABLE OF <Table type>.
```

This would result in the definition of an internal table with rows that are internal tables of the specified table type.

## Independent Definition of Internal Tables

For names of table types and internal tables (data objects) that are defined within your program, the following definitions are added to the naming conventions:

Purpose	Prefix
Program global* or local ** table type	tt_
Program global* internal table	gt_

Purpose	Prefix
Local** internal table	lt_

\* Note: Program global in this case means a type or variable that is globally visible within your program. However, it is locally defined in your program. You also can define types as system global (via the ABAP Data Dictionary). These type definitions are visible (and usable) within all ABAP programs of your SAP system.

\*\* Note: Local in this case means local to a subroutine.



```

TYPES: BEGIN OF ts_type,
        carrid TYPE s_carr_id,
        connid TYPE s_conn_id,
        ...
    END OF ts_type.

DATA gt_itab TYPE SORTED TABLE OF ts_type
        HASHED
        NON-UNIQUE KEY ...

```

Local Structure Type

Internal Table

Figure 121: Independent Definition of Internal Tables

In the previous definitions of internal tables, ABAP Dictionary objects, such as a table type (BC400\_T\_FLIGHTS) or a structure type (BC400\_S\_FLIGHT), were used. The figure illustrates an independent table definition.

The independent table enables you to implement internal tables with any kind of structure without referring to existing Dictionary types.

## Possible Definitions of Internal Tables



**Figure 122: Possible Definitions of Internal Tables**

```

1 DATA gt_itab TYPE <Table Type> .

STANDARD
2 DATA gt_itab TYPE SORTED TABLE OF <Structure Type>
      HASHED
      WITH NON-UNIQUE KEY ...
      UNIQUE

3 DATA gt_itab TYPE TABLE OF <Structure Type> .
      (Short form for definition of a standard table with
      non-unique default key)

```

The figure illustrates an overview of possible definitions of internal tables.

#### Default Value for the Short Form of a Table Definition

The short form of a table definition illustrated in the figure implicitly uses the following default values:

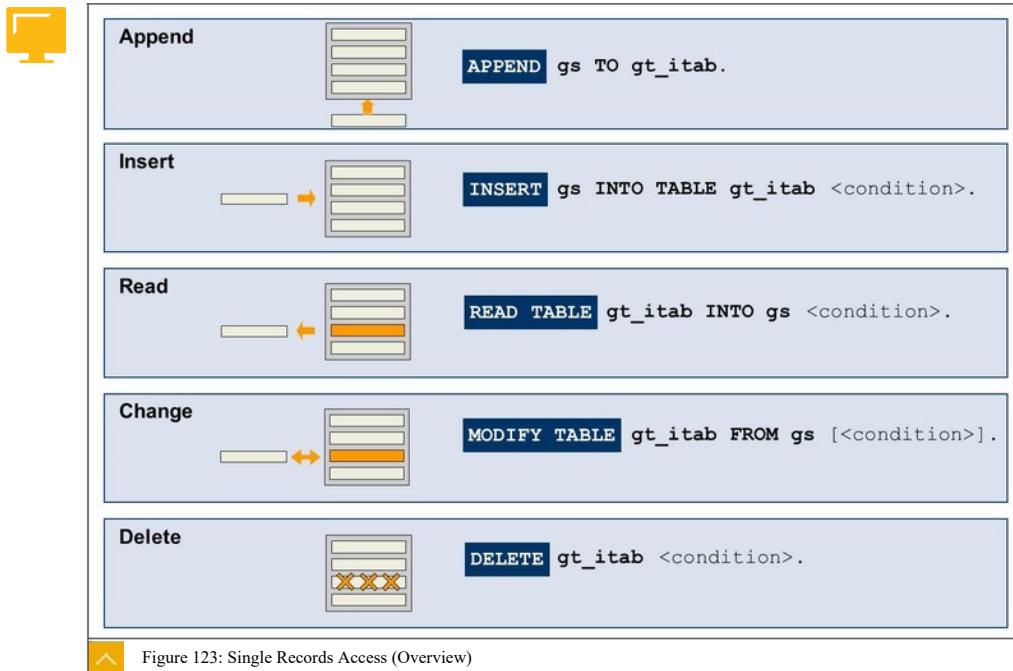
- Table kind – standard (default)
- Uniqueness of key – non-unique (only option for a standard table)
- Table key – default key (all non-numeric table fields are key fields)



#### Hint:

Because the default key usually cannot be used in a meaningful way, you should only use it to define an internal table, if you do not need the key for processing your table.

## Usage of Internal Tables



For processing single records of an internal table, a structure variable that has the same type as the line type of the internal table is required. This structure variable is known as the work area.

The figure illustrates the processing of an internal table using the corresponding work area.

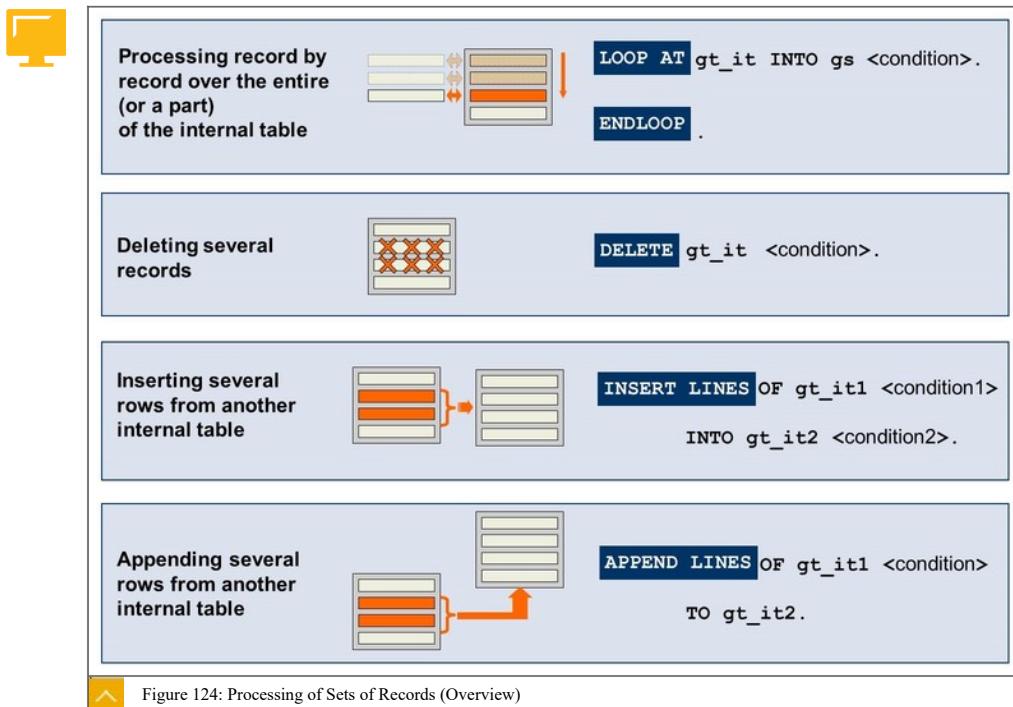
#### Statements for Accessing Single Records of Internal Tables

The following statements are used to access single records of internal tables:

- APPEND appends the content of a structure to an internal table, and this operation can only be used with standard tables.
- INSERT inserts the content of a structure into an internal table.
- READ TABLE copies the content of a table row to a structure.
- MODIFY TABLE overwrites an internal table row with the content of a structure.
- DELETE deletes a row of an internal table.
- COLLECT accumulates the content of a structure in row of an internal table that has the same key and in doing so, only non-key fields are added. Therefore, this statement can only be used for tables whose non-key fields are all numeric.

For detailed information about the ABAP statements described, refer to the relevant keyword documentation.

## Processing of Sets of Records (Overview)



## Statements for Accessing Sets of Records

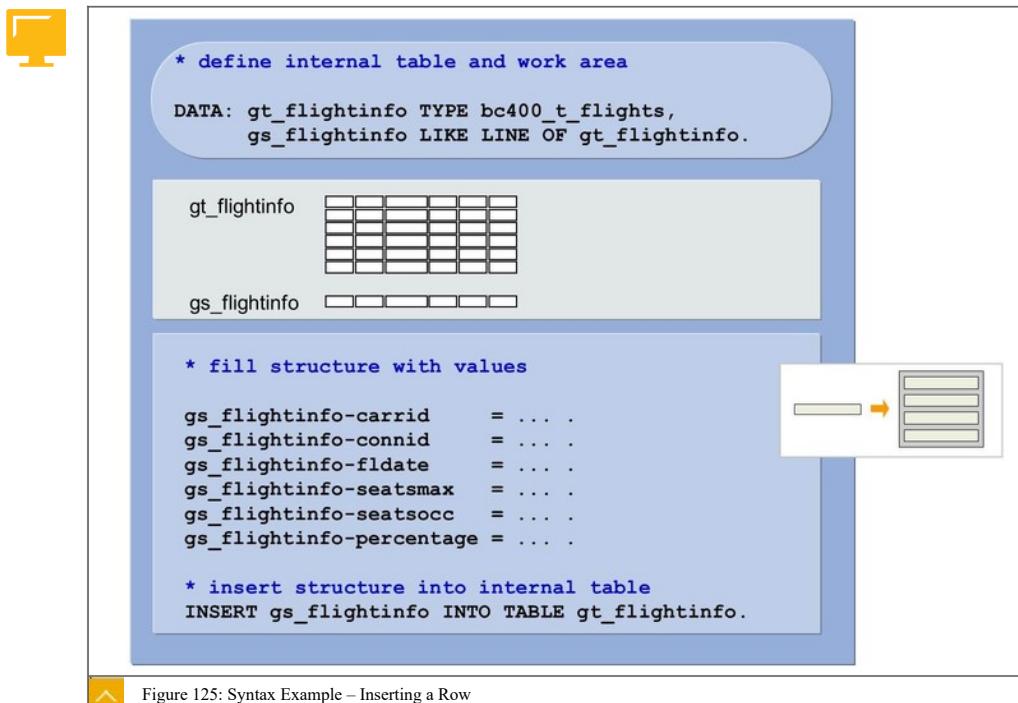
The following statements are used to access sets of records:

- `LOOP AT ... ENDLOOP` places the rows of an internal table into the structure specified in the `INTO` clause one-by-one. Within the `LOOP`, the current content of the structure can be output, or you can change and write the current content back to the table.
- `DELETE` deletes all rows of the internal table that satisfy the logical condition `<condition>`.
- `INSERT LINES OF` copies the content of several rows of an internal table to another internal table.
- `APPEND LINES OF` appends the content of several rows of an internal table to another standard table.

For detailed information about the ABAP statements described here, refer to the relevant keyword documentation.

The following figures are actual examples of syntax for the most common statements.

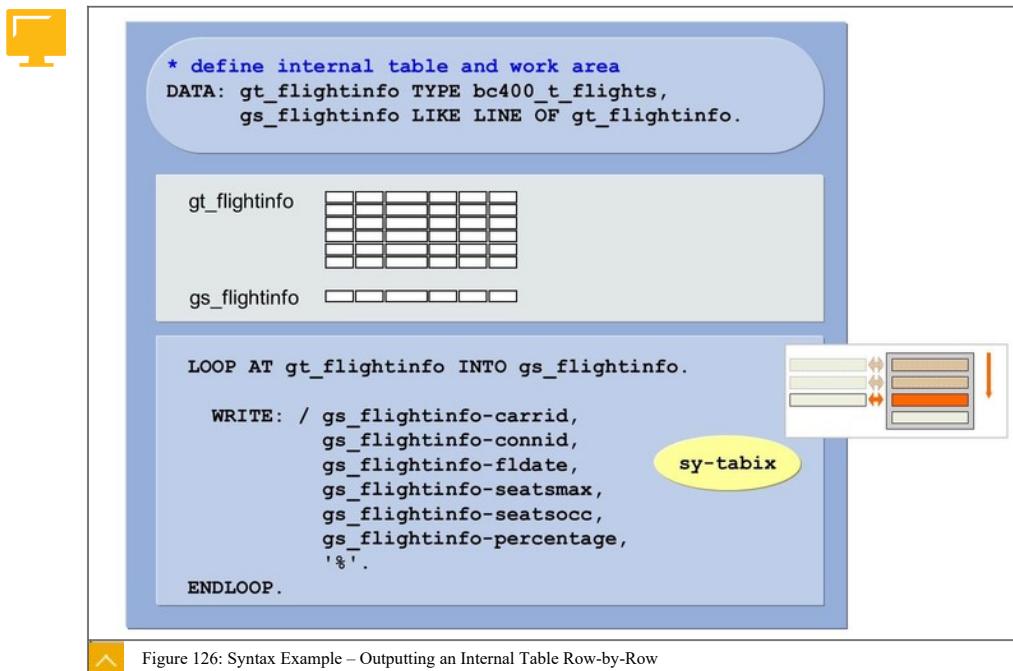
## Syntax Example – Inserting a Row



Insert a row into an internal table by writing the data for the required record into the prepared work area and then inserting it into the internal table with an INSERT statement.

With standard tables, this content is appended. With sorted tables, the row is inserted in the right place to keep the table sorted by its key fields and, in hashed tables, it is inserted according to a hash algorithm.

## Syntax Example – Outputting an Internal Table Row-by-Row



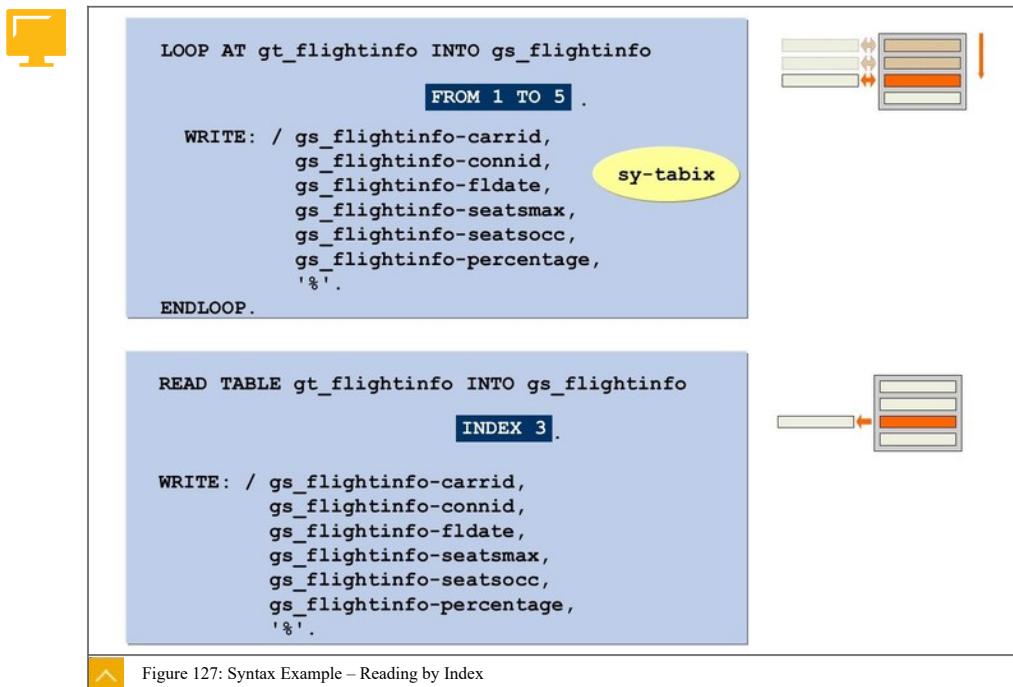
Read and edit the content of an internal table row-by-row using a LOOP. Within each cycle, the sy-tabix system field contains the row number of the current table entry.

In the example, the system processes all rows of the internal table consecutively and produces the output using the WRITE statement.

If you want to change the content of the current table row within a loop pass, change the copy of the row in the work area and then write it back to the current table row using the MODIFY statement. The syntax for this is:

```
MODIFY itab FROM wa.
```

## Syntax Example – Reading by Index

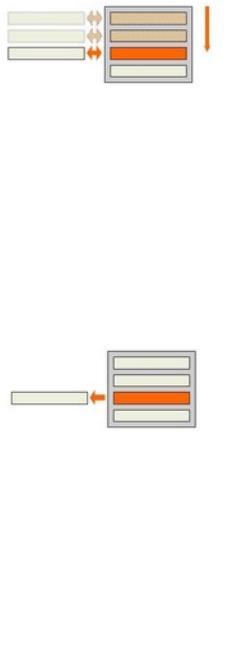


In the LOOP statement, restrict access to specific rows using the FROM-TO addition. In the example, the system only processes the first five rows of the internal table consecutively.

Use the READ TABLE statement to read a single record. Use the INDEX addition to specify the row number of the required record.

Accessing an internal table by index is only possible with index tables, such as standard and sorted.

## Syntax Example – Reading by Key



```

LOOP AT gt_flightinfo INTO gs_flightinfo
  WHERE carrid = 'LH'.
  WRITE: / gs_flightinfo-carrid,
          gs_flightinfo-connid,
          gs_flightinfo-fldate,
          gs_flightinfo-seatsmax,
          gs_flightinfo-seatsocc,
          gs_flightinfo-percentage,
          '%'.
ENDLOOP.

READ TABLE gt_flightinfo INTO gs_flightinfo
  WITH TABLE KEY  carrid = 'LH'
                  connid = '0400'
                  fldate = sy-datum.

  IF sy-subrc = 0.
    WRITE: / gs_flightinfo-seatsmax,
            gs_flightinfo-seatsocc,
            gs_flightinfo-percentage,
            '%'.
  ENDIF.

```

Figure 128: Syntax Example – Reading by Key

While in the LOOP, restrict access to specific rows using the WHERE addition. In the example, the system only processes those internal table rows in which the CARRID field has the value LH.



## Hint:

With regard to the runtime requirement, a sorted table with the CARRID field as the first key field is the most suitable type for this kind of processing.

You can use the READ TABLE syntax in the figure to read a specific row of the internal table. In this case, you have to specify the complete key value of the entry to be read after the WITH TABLE KEY addition. The return code sy-subrc is only set to zero if a corresponding row was found in the internal table.



## Hint:

With regard to the runtime requirement, a hash table is most suitable for this kind of access when you have a large dataset.

With the WITH TABLE KEY addition, you have to supply all key fields with value. If you want to supply an arbitrary set of fields with values, you must use the WITH KEY addition.

## Copies of Table Components with the Same Name

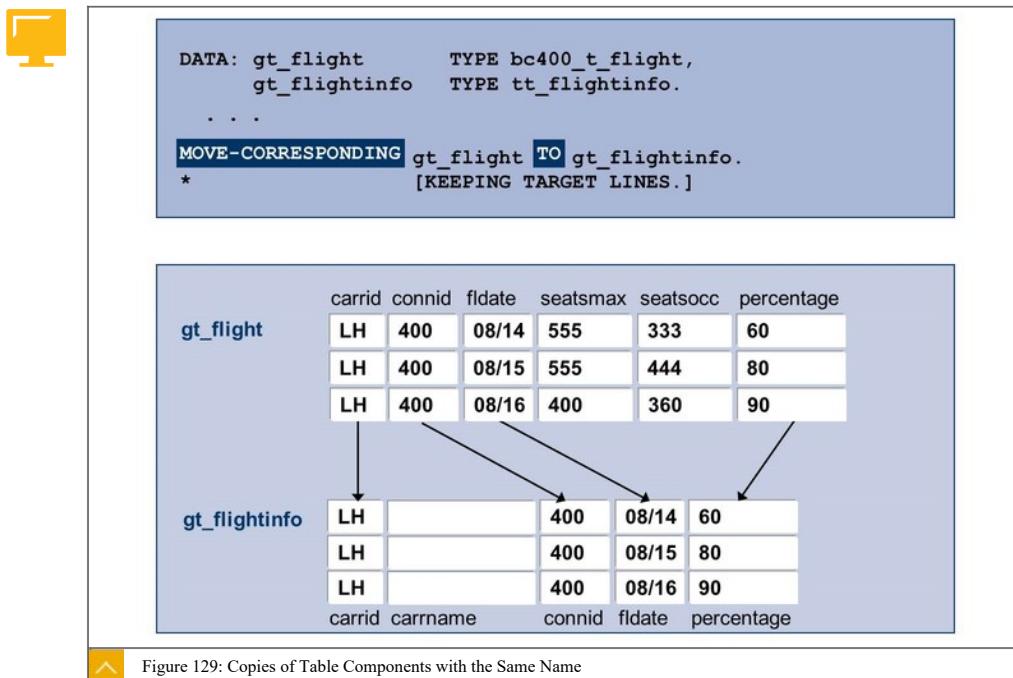


Figure 129: Copies of Table Components with the Same Name

This variant of the statement MOVE-CORRESPONDING requires internal tables to be specified for both sides. It searches for all similarly named components in the row types of both internal tables and assigns them from source table to target table. If there are components with the same name, the target table is deleted (without the addition KEEPING TARGET LINES, otherwise not) and the same number of initial rows are inserted as exist in the source table. The rows of the source table are then extracted sequentially and the content of each row is assigned to the corresponding row in the target table. The individual assignments are executed like in the MOVE statement.

If there are no components with the same name, no assignment is made and the target table is left unchanged.

## Syntax Example – Sorting and Deleting Content

The diagram consists of two main sections. The top section, titled 'Sorting', contains three lines of code: 'SORT gt\_flightinfo.', 'SORT gt\_flightinfo BY carrid.', and 'SORT gt\_flightinfo BY percentage DESCENDING carrid ASCENDING.' To the left of this section is a yellow icon of a computer monitor. To the right is a small icon of a server or database system. The bottom section, titled 'Deleting', contains three lines of code: 'REFRESH gt\_flightinfo.', 'CLEAR gt\_flightinfo.', and 'FREE gt\_flightinfo.' To the left of this section is a yellow icon of a document. To the right is a small icon of a trash can with a large red X.

```

    SORT gt_flightinfo.

    SORT gt_flightinfo BY carrid.

    SORT gt_flightinfo BY percentage DESCENDING
                      carrid          ASCENDING .

    REFRESH gt_flightinfo.
    CLEAR gt_flightinfo.
    FREE gt_flightinfo.
  
```

Figure 130: Syntax Example – Sorting and Deleting Content

Standard and hash tables can be sorted in ascending or descending order by the table key or arbitrary columns by using the SORT statement. If neither ASCENDING nor DESCENDING is specified as the sorting order for a sort field, the field is sorted in ascending order by default.

Language-specific sort rules can even be taken into account, if necessary.

Use the optional AS TEXT addition to implement lexicographical sorting. In a German-speaking environment, for example, this means that **ä** comes before **b**, and not behind **z**, as would be the case with non-lexicographical sorting.

When using the optional STABLE addition, the relative order of data records that have identical sort keys will remain intact during sorting.

For more details, refer to the keyword documentation for the SORT statement.

#### Deletion of Table Content

##### Statements for Deleting Table Content

You can use the following statements for deleting the table content:

- REFRESH

This statement deletes the entire content of the internal table. A part of the previously used memory remains available for future insertions.

- CLEAR

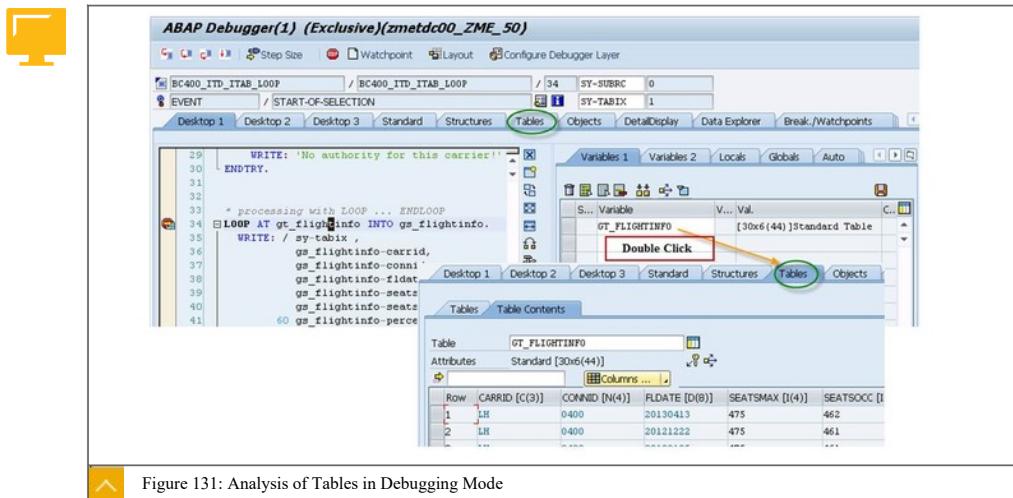
This statement has the same effect as REFRESH for most internal tables (all those that have been previously defined in the course).

For internal tables with header lines (see later), this statement only initializes the header line but does not delete the table content.

- FREE

This statement deletes the entire content of the internal table and releases the previously used memory. Use the FREE statement for internal tables that have already been evaluated and are no longer required in the further course of the program. By doing this you make the no longer required memory available again.

### Internal Tables in Debugging Mode

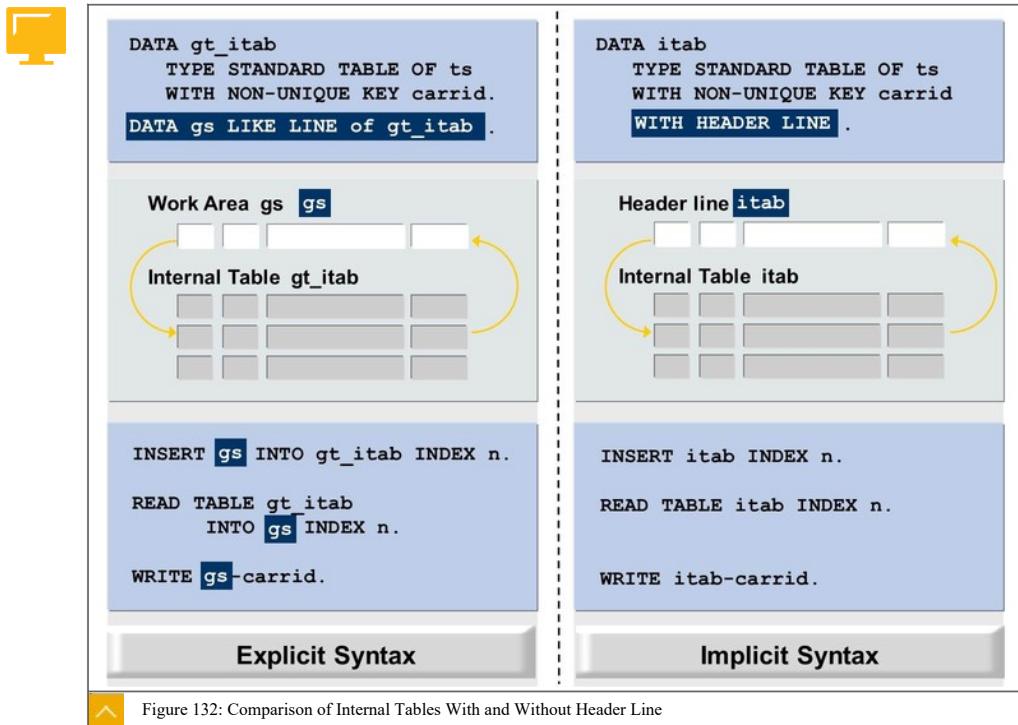


In the ABAP Debugger, trace the content of an internal table by entering the table name in the Variable 1 area and then branching to the table display by double-clicking the table name in the Variable column.

Alternatively, choose the Table tab and specify the table name in the Internal Table field. Press ENTER to display the table content. In addition, you can configure an area for displaying an internal table on one of the desktops.

To display an internal table in the classic ABAP Debugger, the tool provides a key with the label “Table”. As with the new ABAP Debugger, however, you can also navigate to this display by double-clicking the table name.

### Internal Tables with Header Lines



With the WITH HEADER LINE addition in the definition of an internal table, the table is created in an obsolete format. When this is done, a work area (header row) that suits the table is created automatically so that an additional definition of the same is not necessary. This also simplifies the syntax of the statements, because the system always uses the automatically created header line, which no longer has to be specified explicitly.

### Tables with Header Lines

#### Reasons for Not Using Tables with Header Lines

You should no longer use tables with header lines for the following reasons:

- The automatically generated work area will have the same name as the internal table, which hinders program readability.
- The table with a header line is not allowed in nested data objects (structures and internal tables that have internal tables as components) and ABAP Objects (object-oriented extension of ABAP).

Internal tables with header lines are mentioned here because some older programs still use them and you have to work with such programs from time to time. It is for this purpose that further particularities of the header line are listed.

If an internal table with header line is called "itab" you can use itab-comp to address component "comp" of the header line (work area).

Example of Addressing the Body of a Table with itab

- You can address the body of a table with itab. The following example illustrates this situation:

```
DATA itab1 TYPE TABLE OF scarr WITH HEADER LINE.  
DATA itab2 LIKE itab1.  
itab1 = itab2. "Operation only with header lines  
itab1[] = itab2[]. "Operation with the table bodies
```

Example of Defining an Internal Table with the Header Line

- The following syntax also defines an internal table with the header line, even though that is not stated specifically:

```
DATA: BEGIN OF itab OCCURS n,  
      field1 TYPE ... ,  
      field2 TYPE ... ,  
      ... ,  
      END OF itab.
```



LESSON SUMMARY

You should now be able to:

- Define internal tables
- Implement basic ABAP statements with internal tables
- Analyze internal tables in debugging mode

## Unit 5

### Learning Assessment

1. Which of the following statements copies the content of the source structure to the target structure, one component at a time?

Choose the correct answer.

- A COPY-CORRESPONDING
- B REPLACE-CORRESPONDING
- C MOVE-CORRESPONDING
- D REMOVE-CORRESPONDING

2. Which of the following statements is used for defining local structure types?

Choose the correct answer.

- A TYPES
- B BEGIN
- C END
- D START

3. Which of the following specifications are required in the definition of an internal table?

Choose the correct answers.

- A Line type
- B Primary key
- C Secondary key
- D Table kind

4. Which of the following is used for adding a row into an internal table?

Choose the correct answer.

**A** ADD LINE

**B** INSERT ROW

**C** APPEND

**D** UPDATE TABLE

## Unit 5

### Learning Assessment - Answers

1. Which of the following statements copies the content of the source structure to the target structure, one component at a time?

Choose the correct answer.

- A COPY-CORRESPONDING  
 B REPLACE-CORRESPONDING  
 C MOVE-CORRESPONDING  
 D REMOVE-CORRESPONDING

You are correct! The statement, MOVE-CORRESPONDING, copies the content of the source structure to the target structure, one component at a time. In this case, only those components that are available under the same name in both the source structure and the target structure are considered. All other components of the structures remain unchanged. Read more in the lesson, Using Structured Data Objects, Task: Use of Structured Data Objects, in the course BC400 (Unit 5, Lesson 1) or TAW10 Part I (Unit 11, Lesson 1).

2. Which of the following statements is used for defining local structure types?

Choose the correct answer.

- A TYPES  
 B BEGIN  
 C END  
 D START

You are correct! Use the TYPES statement to define local structure types. Read more in the lesson, Using Structured Data Objects Task Definition of Structures with Local Types, in the course BC400 (Unit 5, Lesson 1) or TAW10 Part I (Unit 11, Lesson 1).

3. Which of the following specifications are required in the definition of an internal table?

Choose the correct answers.

A Line type

B Primary key

C Secondary key

D Table kind

You are correct! The required specifications in the definition of an internal table are: Line type, Primary key, Table kind. Read more in the lesson, Using Internal Tables, Task: Properties of Internal Tables, in the course BC400 (Unit 5, Lesson 2) or TAW10 Part I (Unit 11, Lesson 2).

4. Which of the following is used for adding a row into an internal table?

Choose the correct answer.

A ADD LINE

B INSERT ROW

C APPEND

D UPDATE TABLE

You are correct! APPEND appends the content of a structure to an internal table, and this operation can only be used with standard tables. Read more in the lesson, Using Internal Tables, Task: Statements for Accessing Single Records of Internal Tables, in the course BC400 (Unit 5, Lesson 2) or TAW10 Part I (Unit 11, Lesson 2).

## UNIT 6

# Data Modeling and Data Retrieval

### Lesson 1

Explaining Data Models	187
------------------------	-----

### Lesson 2

Retrieving Single Database Records	196
------------------------------------	-----

### Lesson 3

Retrieving Multiple Database Records	203
--------------------------------------	-----

### Lesson 4

Describing Other Aspects of Database Access	207
---	-----

### Lesson 5

Implementing Authorization Checks	218
-----------------------------------	-----

### UNIT OBJECTIVES

- Explain the purpose and benefits of data models
- Describe the SAP flight data model
- Explain transparent tables
- Retrieve single database records
- Implement a SELECT loop
- Implement an array fetch
- Retrieve client-specific data
- Use database indexes
- Explain the SAP table buffer
- Retrieve data from several database tables
- Identify ways to change data in a database table

- Explain the authorization concept
- Implement authorization checks

## Unit 6

### Lesson 1

# Explaining Data Models

#### LESSON OVERVIEW

This lesson gives an overview of data modeling. You need to perform data modeling before application development. You will also learn to understand the description of database tables in the ABAP Dictionary.

#### Business Example

You want to develop a program that accesses database tables. To do this, you need to know the underlying data model and the structure of database tables. For this reason, you require the following knowledge:

- An understanding of the purpose and benefits of using a data model in application development
- An understanding of the SAP flight data model
- An understanding of the meaning and structure of a transparent table

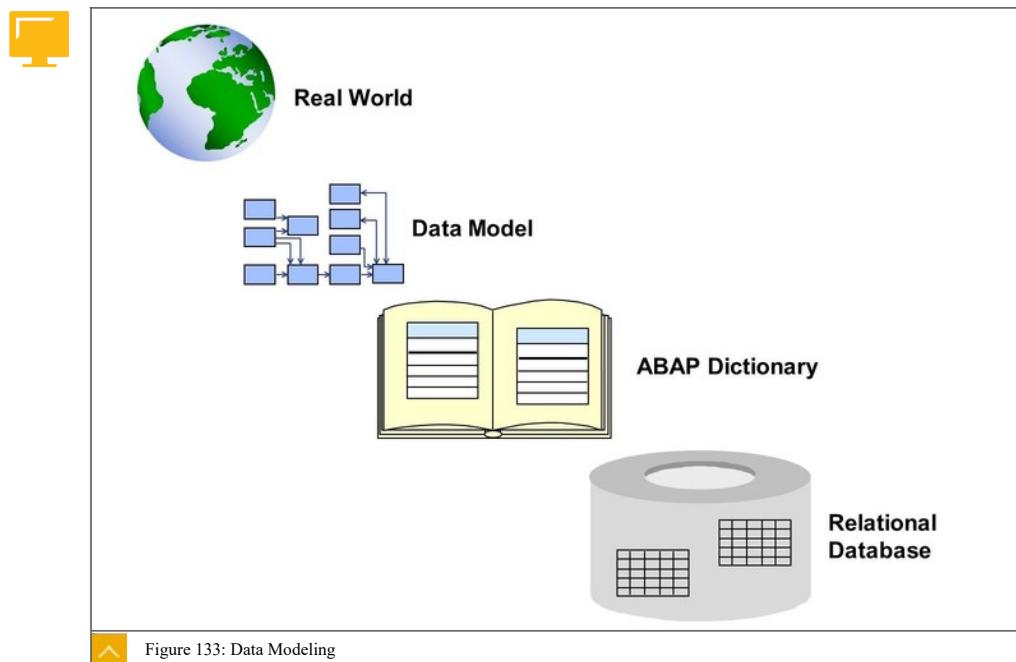


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain the purpose and benefits of data models
- Describe the SAP flight data model
- Explain transparent tables

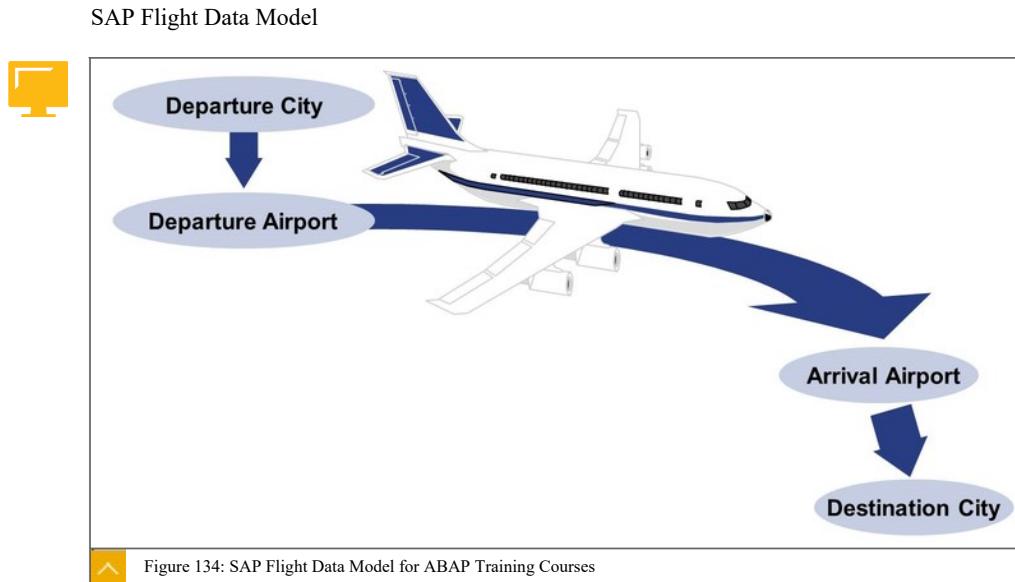
### Data Model Overview



In the development of business applications, relevant aspects of the real world must be represented in data form. A business unit represents an entity. These entities exist in relationships with each other, which are fixed in the underlying data model, also referred to as the Structured Entity Relationship Model (SERM).

The developer uses this data model as the basis for implementing appropriate table definitions (transparent tables), including their relationships with each other in the ABAP Dictionary.

By activating table definitions, corresponding database tables are automatically created in the database. The actual application data is entered into those tables later on.



#### Typical Inquiries at the Travel Agency

- Relevant airports
- Relevant flight connections
- Relevant flight times
- Relevant information about flights, for example, pricing, utilization of capacity or availability, and other details.

The ABAP training courses, online documentation, and ABAP keyword documentation use the same flight data model as an example. Repository objects for the flight data model are in the package SAPBC\_DATAMODEL.

Customers of a travel agency who want to travel from one place to another by air want to obtain the following specific information from the travel agency.

#### Required Information

- Which connection offers the best and most direct flight?
- What are the acceptable flight times on the proposed days of travel?
- Which connections offer the cheapest flights, the fastest connections, and connections with target arrival times?

On the other hand, the travel agency has different requirements. In the data model designed for managing the required data, the data is stored according to the technical criteria in tables in a central database. The amount of the stored data exceeds that needed to respond to the requests of the customer.

Therefore, you need to be able to compile the data needed to satisfy the requests of the customer using application programs.

## Relational Data Model

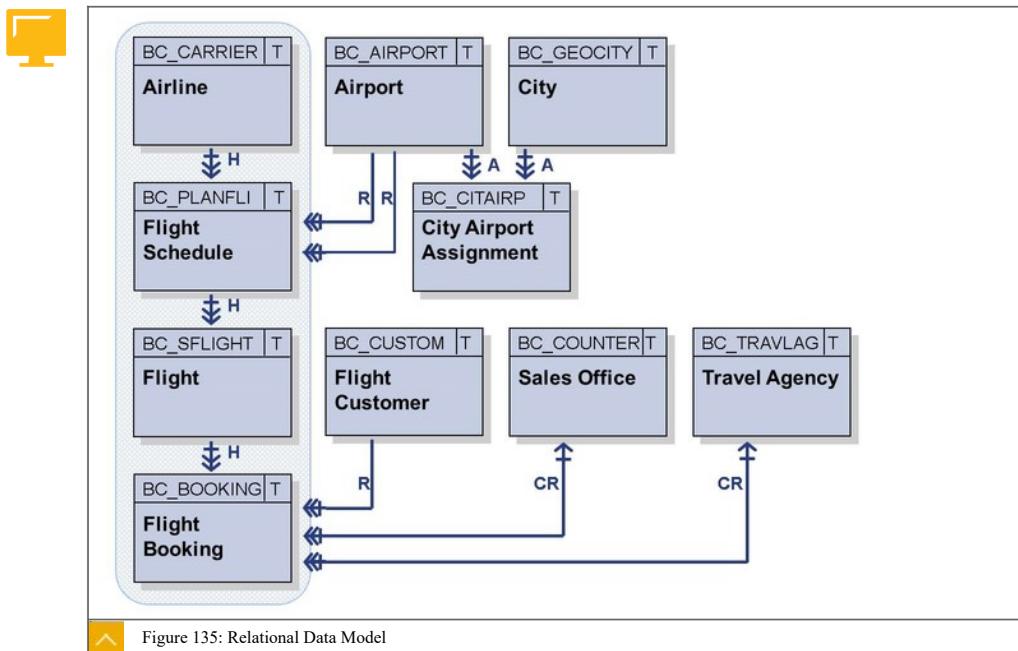


Figure 135: Relational Data Model

## Entities Within the Flight Data Model

The flight data model contains the following entities for all business information that is logically connected:

- Cities
- Airports
- Airlines
- Flight routes
- Flights

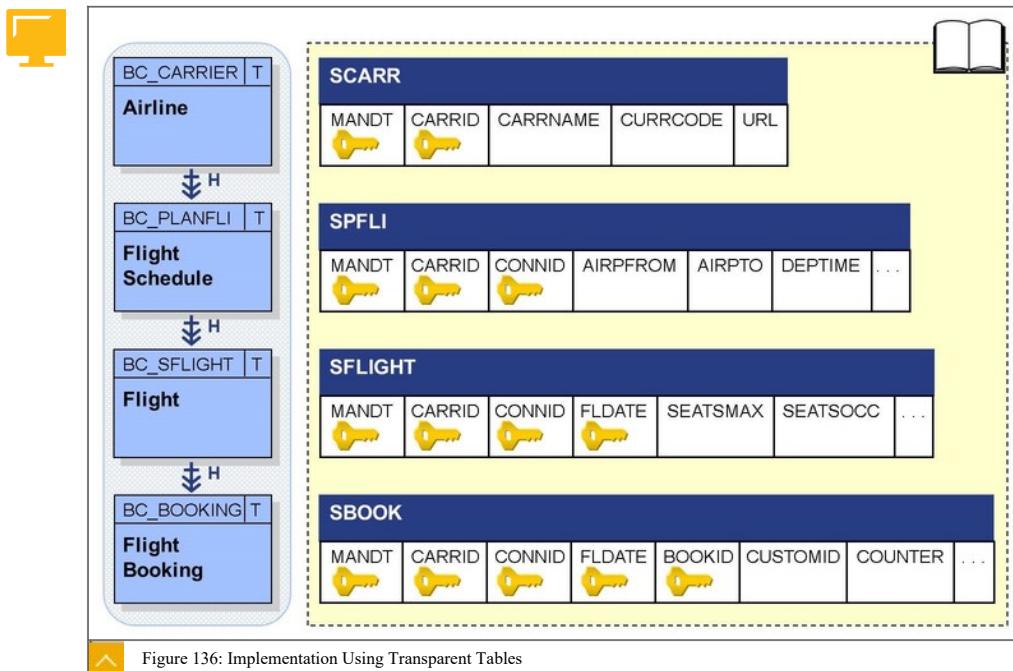
## Relationships of the Entities in the Flight Data Model

These entities relate to each other in the following ways:

- Each flight schedule contains one airline, one departure airport, and one destination airport.
- Each bookable flight belongs to one existing flight schedule.
- Each assignment can be set between the relevant cities and the nearby airports.

You can manage all of the necessary data without redundancy by using these relationships. At the same time, the travel agency is able to obtain all of the data requested by the customer.

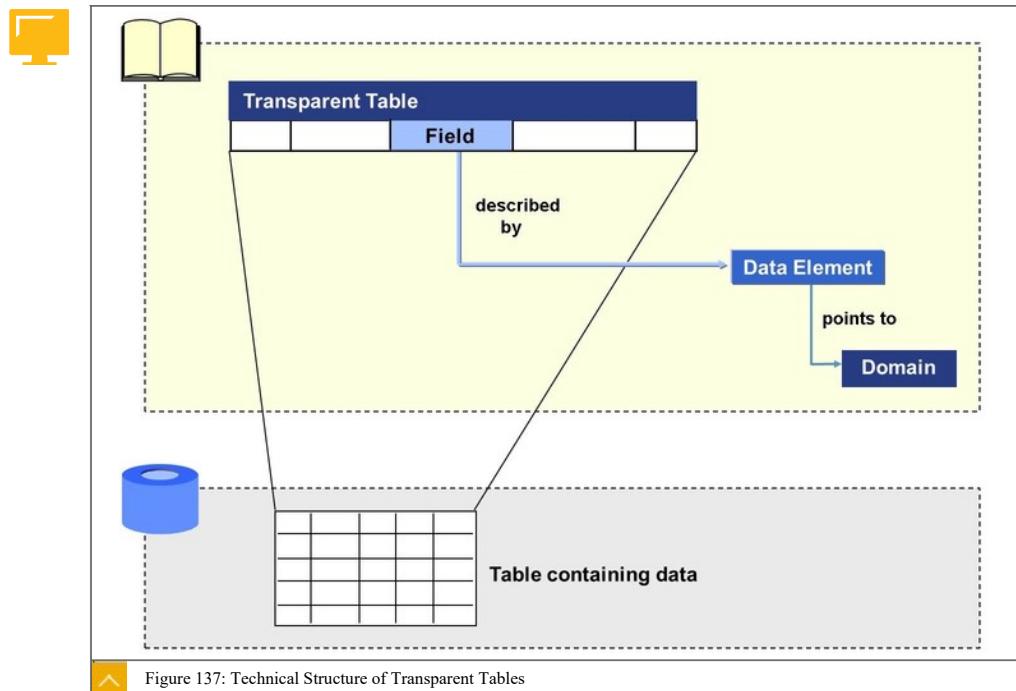
## Implementation Using Transparent Tables



For each entity fixed in the data model, the developer creates a transparent table in the ABAP Dictionary. This is a platform-independent description of the database table and not the actual database table. However, when the transparent table is activated, the system creates a table of the same name in the database.

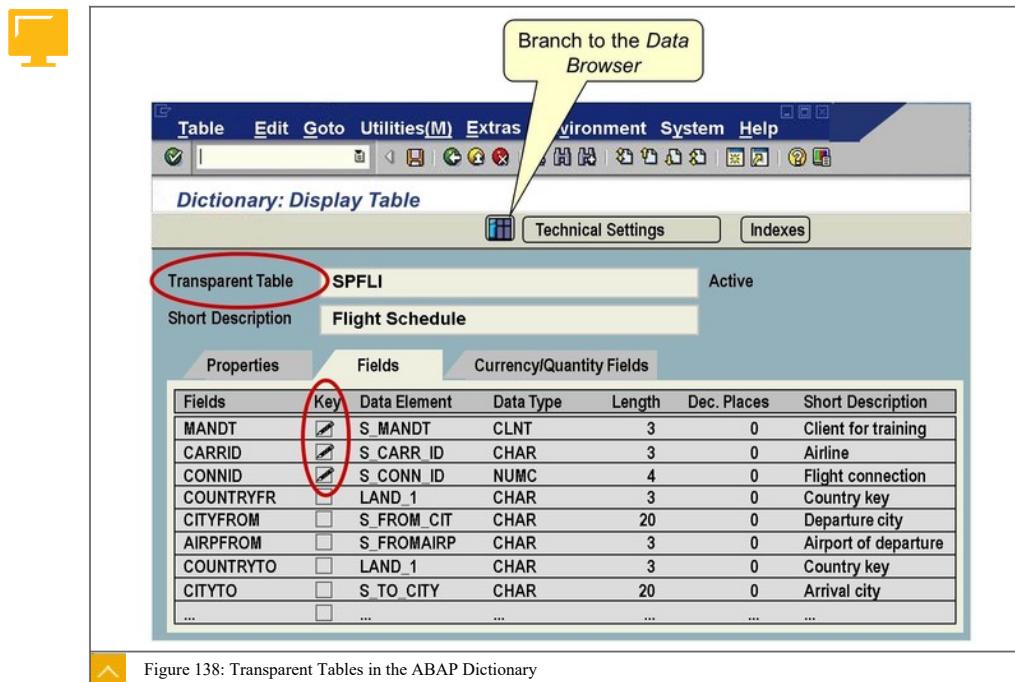
A transparent table contains various fields (columns) to store and manage data records in a structured way. Table fields are declared as key fields when the content is to be used for the unique identification of data records within the database table. The key of a table (table key) consists of key fields. This is also called a primary key. Data records in the same table must be unique with regard to primary key values. The key value of a data record is its unique ID within the table.

### Transparent Tables



A transparent table in the ABAP Dictionary is the description of the corresponding database table which contains the actual data used by the application. The fields of the transparent table form columns in the corresponding database table with identical names. Data elements, which you already know as globally defined elementary data types, are normally used for describing individual fields. Data elements refer to domains for their technical properties.

## Transparent Tables in the ABAP Dictionary



In addition to the list of fields, transparent tables contain other information that the system requires to create a table of the same name in the database and to describe the complete properties of the table.

## Additional Information in Transparent Tables

Transparent tables contain the following additional information:

- Determination of the key for the database table (key fields)
- Technical properties required by the database to create the database table (expected size and expected frequency of access)
- Settings for technologies that can speed up access to the database table (secondary indexes and buffering)

## Structures in the ABAP Dictionary

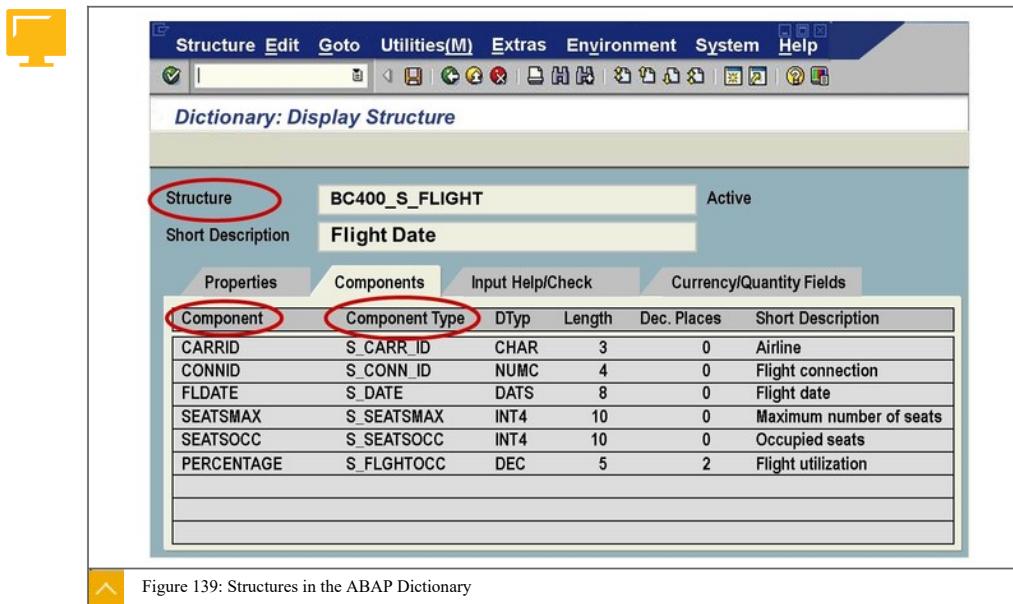
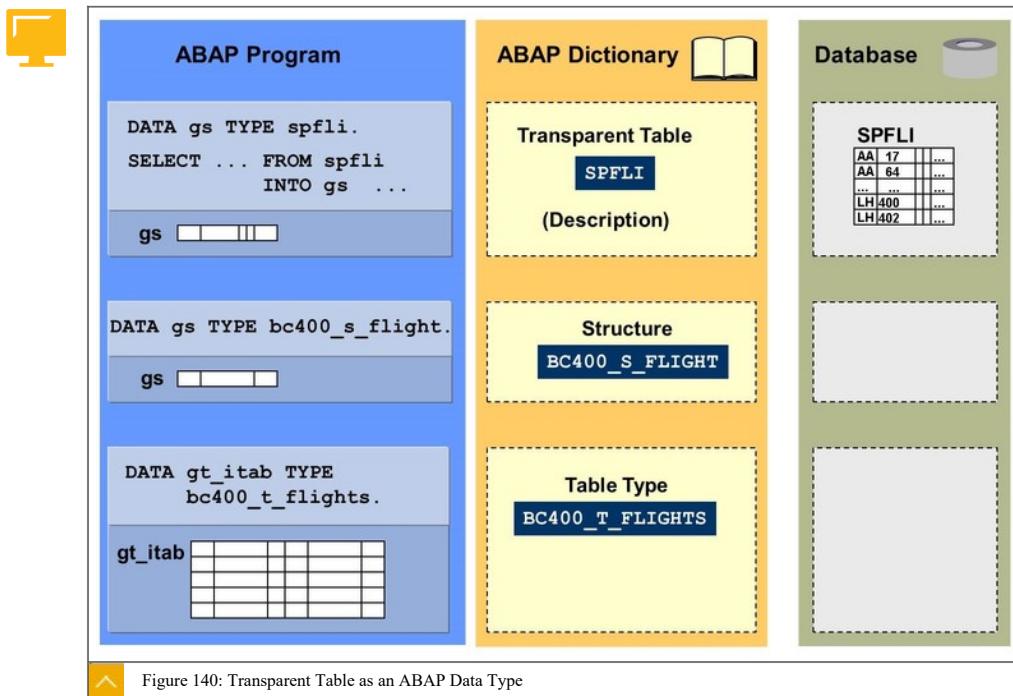


Figure 139: Structures in the ABAP Dictionary

The definition of a transparent table appears to be very similar to the definition of a global structure type. Transparent tables can be used in the same way as structure types in programming. For example, transparent tables can be used to define a structured data object (structure variable), for typing an interface parameter, or as the line type of a global or a local table type. Only the list of fields is important. Other properties of the transparent table, such as the key definition or the technical properties, are irrelevant when it is being used as a data type.

## Transparent Table as an ABAP Data Type



## Hint:

In older programs, transparent tables were very freely used as data types. Current recommendations are to use a transparent table directly in connection with access to the database. Above all, transparent tables should not be used in the process of defining user interfaces because this creates unwanted dependency between the definition of database objects or interfaces and the user interface.

In addition to the properties related to the database, there is another difference between transparent tables and structure types.

A transparent table is a list of elementary fields, whereas the components of a structure type can themselves be structured again (nested structures). The components of a structure can even be typed with a table type (deep structures).



## LESSON SUMMARY

You should now be able to:

- Explain the purpose and benefits of data models
- Describe the SAP flight data model
- Explain transparent tables

## Unit 6

### Lesson 2

## Retrieving Single Database Records

### LESSON OVERVIEW

This lesson concludes with a note about types of database accesses that initiate changes.

#### Business Example

You need to evaluate data from database tables. As there are no suitable reuse components for accessing the database tables that you want to read, you implement new ones. For this reason, you require the following knowledge:

- An understanding of the various methods for searching relevant database tables
- An understanding of how to program read access to specific columns and rows within a particular database table

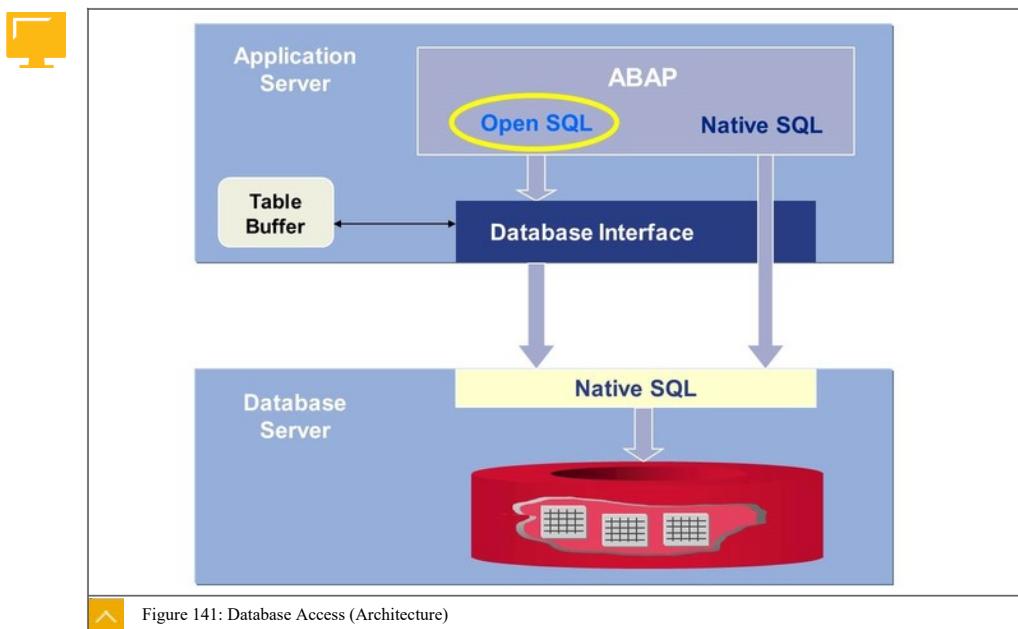


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Retrieve single database records

### Data Retrieval with Structured Query Language



Structured Query Language (SQL) is a language that enables programs to define, change, and have read access to database tables.

Every relational database system has a native SQL that is specific to the database. Therefore, an ABAP program with native SQL statements cannot be used without restrictions in all other SAP systems due to differences in database systems of various SAP systems.

In contrast, Open SQL is an SAP-defined, database-independent SQL standard used in the ABAP language. The system dynamically converts Open SQL statements to native SQL statements of the target database system and, as such, Open SQL statements remain independent of the database. Open SQL allows the ABAP programmer uniform access to data, regardless of the installed database system.

#### Database Tables Searching

##### Options for Searching Required Database Tables



- Search by application

Search within a particular application component, in the application hierarchy.

- Search by program

Search within a program that accesses the table being searched for:

- Source code search: search for SELECT statements in the ABAP Editor.

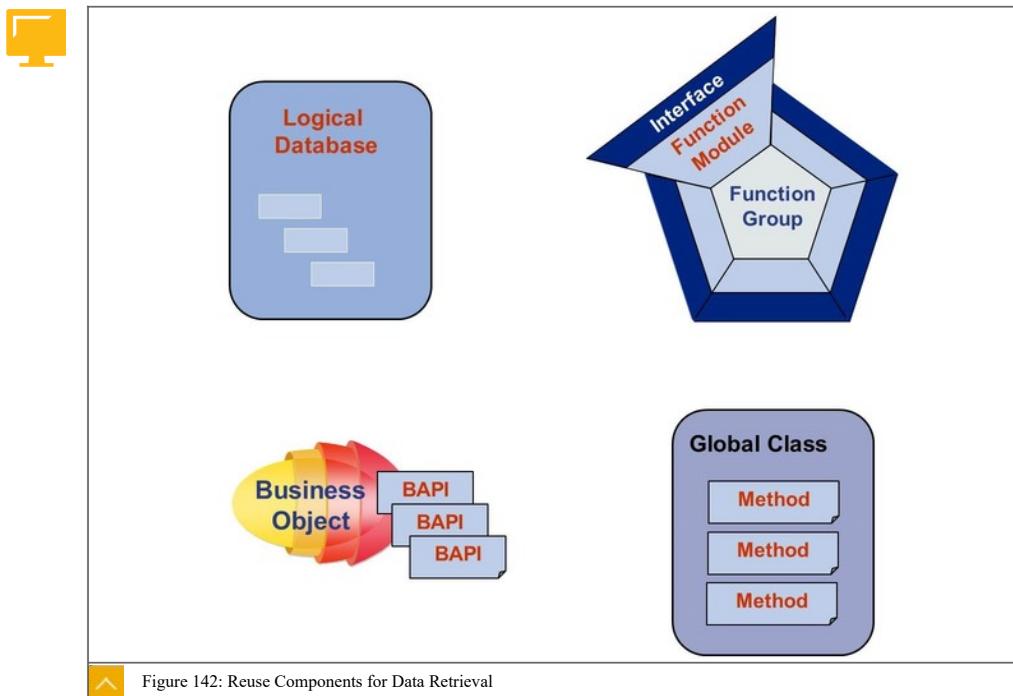
- Function debugging: switch to debugging mode (/h) prior to executing a subfunction and set a breakpoint at the SELECT statement.

- Screen field information: display a corresponding structure field using F1 + Technical Information , navigate to the relevant data element and query the Where-Used List in Table Fields.

You can also execute a free search using the Repository Information System.

Before you program direct access to database tables, look for reuse components that handle the read process. The figure gives an overview of those read routines that SAP supplies, which you can use in your program.

## Reuse Components for Data Retrieval



## Types of Reuse Components that Encapsulate Database Access

• **Logical databases**

Logical databases are data retrieval programs that read data from tables that belong together hierarchically.

• **Function modules**

Function modules are procedures stored in the function library of the SAP system with encapsulated functions, such as reading from hierarchically related tables.

• **Business Application Programming Interfaces (BAPIs)**

Examples of BAPIs include methods of business objects with read functions.

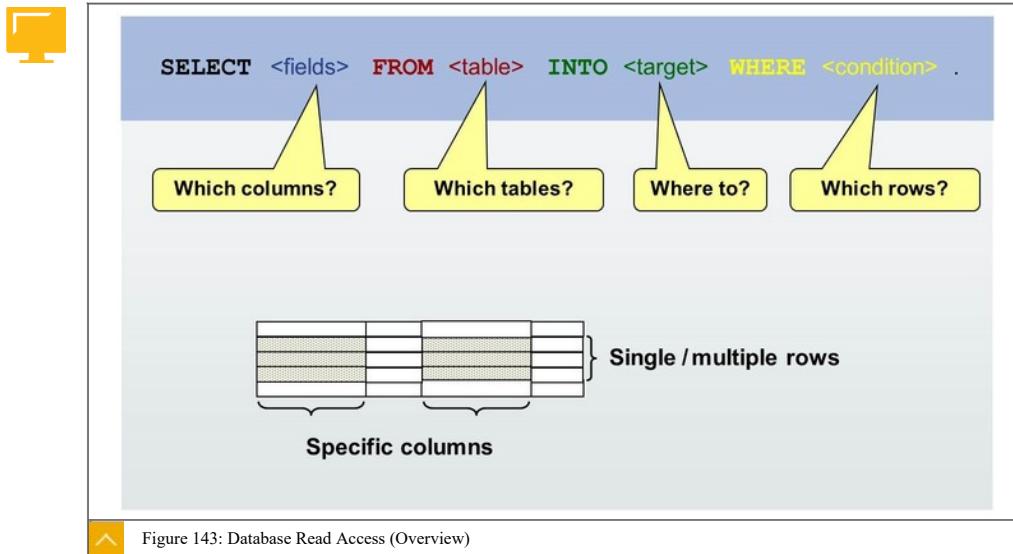
• **Methods of global classes**

Table 5: Transactions for Reuse Component Types

Reuse Component Type	Transaction
Logical databases	SE36
Function modules	SE37
BAPI	BAPI
Global class	SE24

If there are no usable reuse components available for your data selection, implement the read access using the following methods. Encapsulate the access into a reuse component; in other words, create function modules or methods of global classes.

#### Database Read Access (Overview)



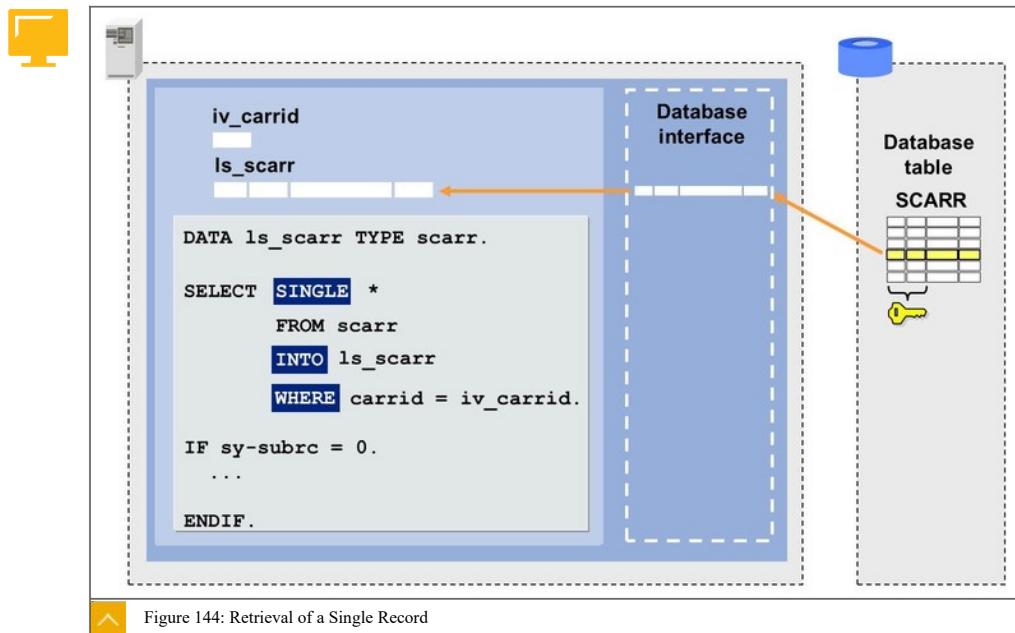
Use the Open SQL statement SELECT to program database read access.

#### SELECT Statement Clauses

- The SELECT statement contains a series of clauses, each of which has a different task as follows:
  - The SELECT clause names the fields of the table that are to be read.
  - The FROM clause names the source (database table or view) from which data is to be selected.
  - The INTO clause determines the target variable into which the selected data is to be placed.
  - The WHERE clause specifies the columns of the table that are to be selected.

For information about other clauses, refer to the keyword documentation for the SELECT statement.

## Retrieval of One Row of Data



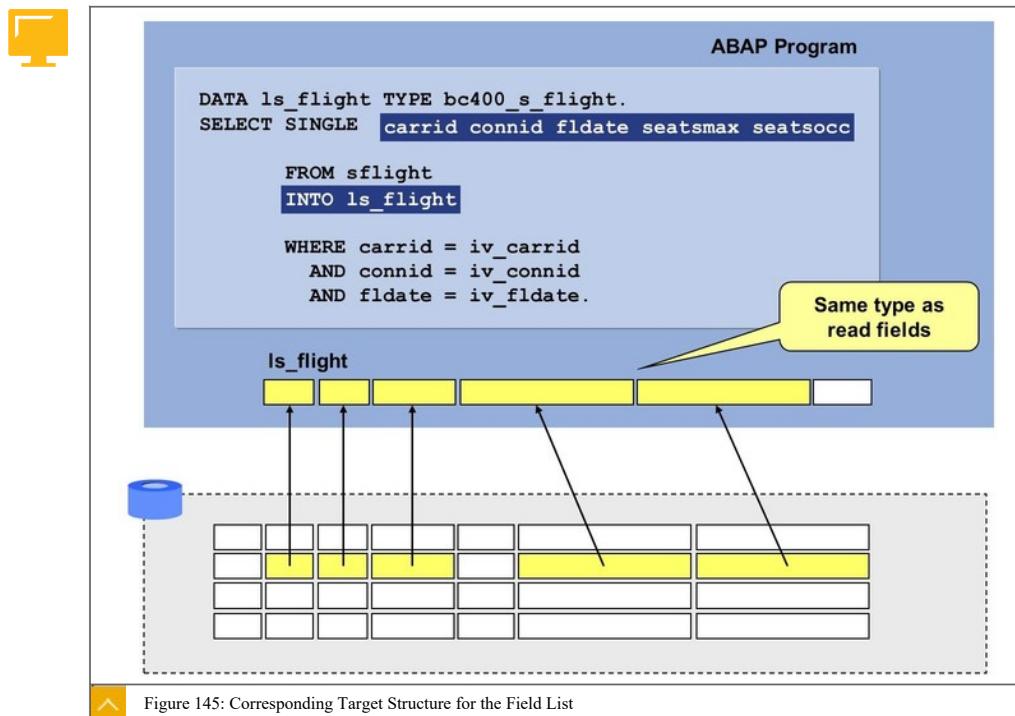
The `SELECT SINGLE` statement allows you to read a single record from the database table. To ensure a unique access, fill in all key fields of the `WHERE` clause. The client field is an exception. If nothing is specified, the current client is assumed. A client can only be specified in the `SELECT` statement in combination with the `CLIENT SPECIFIED` addition.

Use an asterisk (\*) to specify that all selected fields of the table row are to be read. If you only want a specific selection of columns, replace the \* with a list of required fields, as shown in the figure, Corresponding Target Structure for the Field List .

Use the `INTO` clause to specify the target variable to which the data record is to be copied. Target structure has to be left-justified in the same way as the field list.

If the system finds a suitable record, `SY-SUBRC` returns the value of 0.

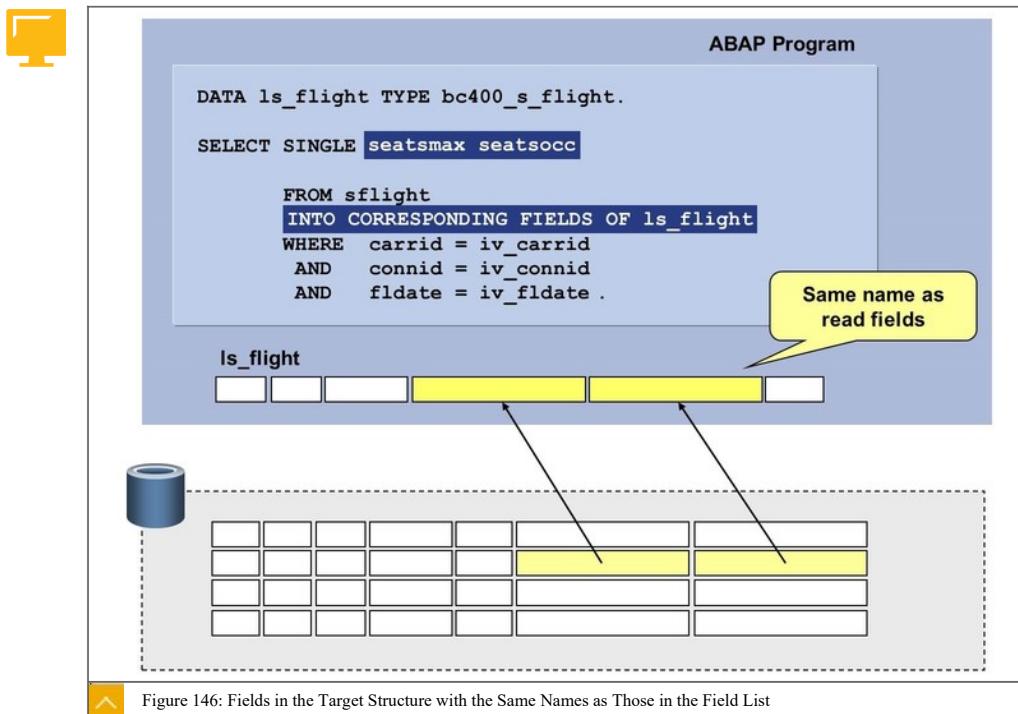
## Corresponding Target Structure for the Field List



If you only want a certain selection of fields from the table row to be read, specify these as shown in the field list within the SELECT statement (shown in the figure). Then, name a target structure variable in the INTO clause that has the same structure as the field list (at least in the beginning); in other words, it contains the names of the fields that are in the field list and in the same order. Only the corresponding field types have to match. The names of the target structure fields are not taken into account.

Alternatively, list the corresponding target fields in the INTO clause as follows: INTO (field\_1, ..., field\_n).

## Fields in the Target Structure with the Same Names as Those in the Field List



If you want to use a structure variable for receiving the read record, which has fields with the same names as the fields in the target list but has a different structure (additional fields, different order of fields), use the CORRESPONDING FIELDS OF addition. This has the effect that the system only fills the fields of the target area that have the same name as the fields of the database table. Make sure that the field types are also the same for corresponding fields; otherwise (as in the MOVE statement), a complicated conversion starts, and it is possible that only incomplete data (caused by cutoffs) will be transported to the target fields.

## Advantages of Using INTO CORRESPONDING FIELDS OF

- The target structure does not have to be left-justified in the same way as the field list.
- The construction is easy to maintain, because extending the field list or target structure does not require other changes to be made to the program, as long as there is a field in the structure that has the same name (and if possible, also the same type).



## LESSON SUMMARY

You should now be able to:

- Retrieve single database records

## Unit 6

### Lesson 3

# Retrieving Multiple Database Records

#### LESSON OVERVIEW

This lesson explains how to read multiple data records from the same database table.

#### Business Example

You need to evaluate data from database tables. But instead of single records you want to read and process a whole set of data records. For this reason, you require the following knowledge:

- An understanding of how to implement a SELECT loop
- An understanding of how to implement an array fetch

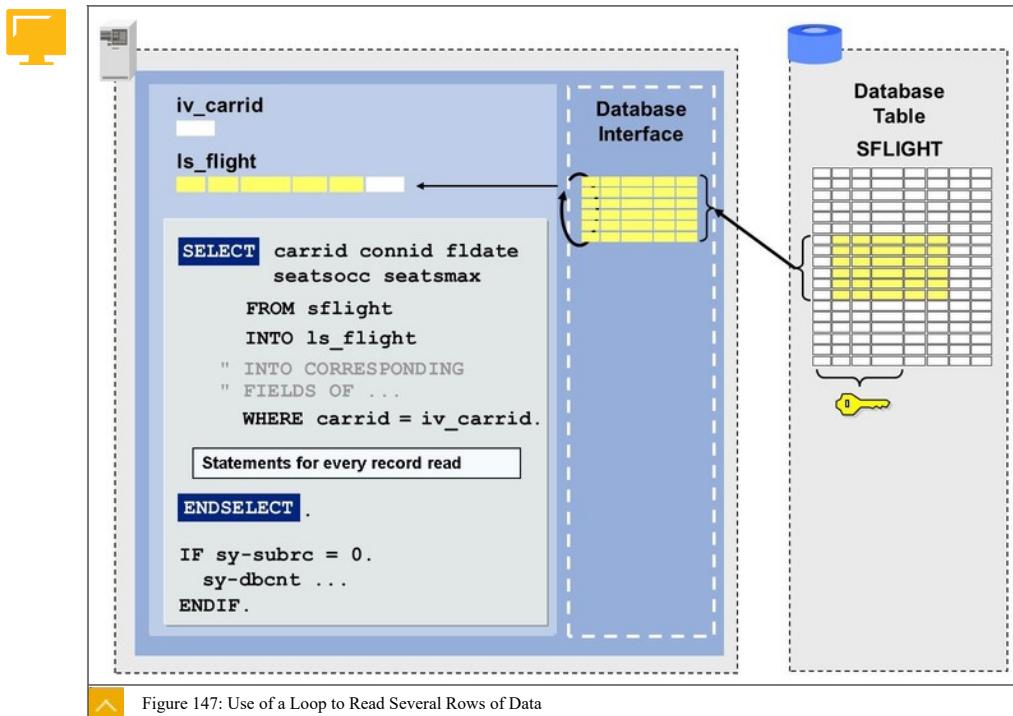


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Implement a SELECT loop
- Implement an array fetch

## Data Retrieval with Loops



You can use the SELECT loop to read several rows of a database table into the program in succession.

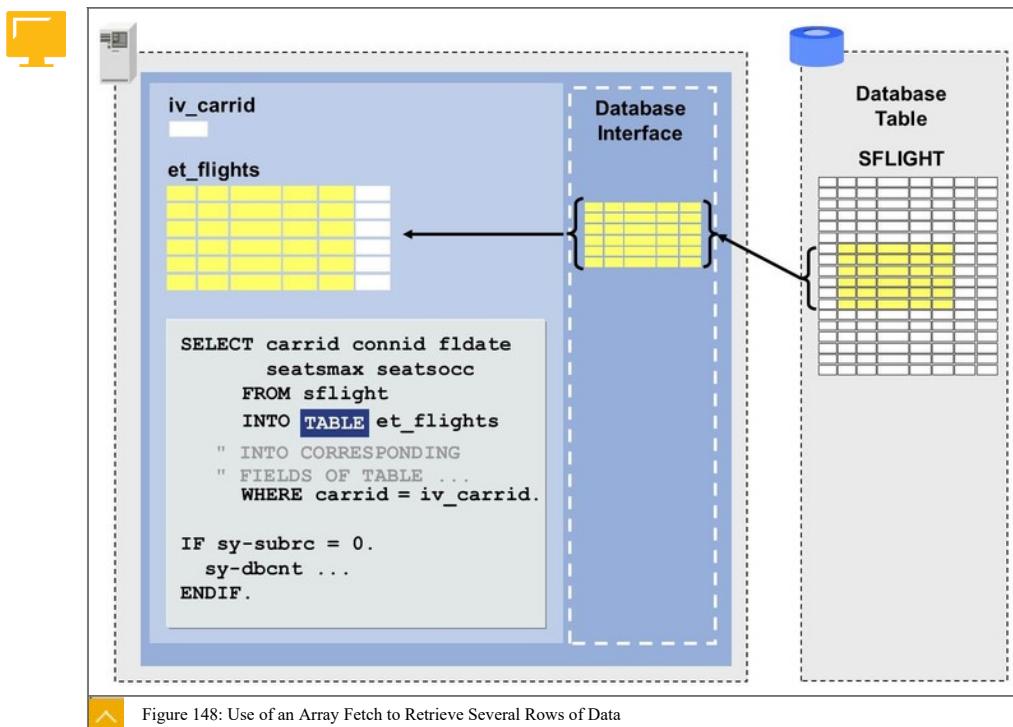
The WHERE clause determines which lines are read into the target structure and processed using the statement block specified in the loop body. You can logically connect multiple logical conditions within the WHERE clause using AND or OR.

The database delivers the data to the database interface of the application server in packages. The database interface copies the records to the target area row-by-row for processing in the processing block between SELECT and ENDSELECT.

The loop is left automatically after all of the required rows have been read and evaluated.

You should query the return value after the SELECT loop, that is, after the ENDSELECT statement. The sy-subrc returns a value of 0 if the system reads at least one row. In this case, sy-dbcnt contains the number of records read.

## Data Retrieval Using an Array Fetch



You can use the INTO TABLE addition to copy the selected part of the database directly into an internal table instead of doing so row-by-row. This technique is called an array fetch. It is a high performance technique for filling an internal table with entries from a database table, because the data transport is carried out in blocks and not in rows.

An array fetch is not a type of loop processing, so no ENDSELECT statement is required or allowed.

In the same way as in the other SELECT variants, the internal table that is specified as the target in the array fetch must be structured with left-justification just like the field list. If the internal table does not meet this prerequisite, you have to use the INTO CORRESPONDING FIELDS OF TABLE addition instead of the INTO TABLE addition. With the method, the database columns specified in the field list are copied to the columns of the internal table that have the same names. Make sure that the field types of the corresponding columns match in order to avoid complex conversions and possibly incomplete data transfer to the target table.

With array fetch, any content that might be present in the internal table is overwritten. If you want to append rows instead of overwriting, use the APPENDING TABLE addition.

If the system copies at least one record to the internal table, sy-subrc returns the value of 0. The number of rows that were copied is returned in sy-dbcnt.



### LESSON SUMMARY

You should now be able to:

- Implement a SELECT loop
- Implement an array fetch

## Unit 6

### Lesson 4

# Describing Other Aspects of Database Access

#### LESSON OVERVIEW

This lesson contains a brief outlook on some further aspects of database access.

#### Business Example

You want to know more about database access. You want to make your database access secure and efficient. Therefore, you require the following knowledge:

- An understanding of client-specific data
- An understanding of database indexes
- An understanding of the SAP table buffer
- An understanding of how to retrieve data from several database tables
- An understanding of how to identify ways to change data in a database table

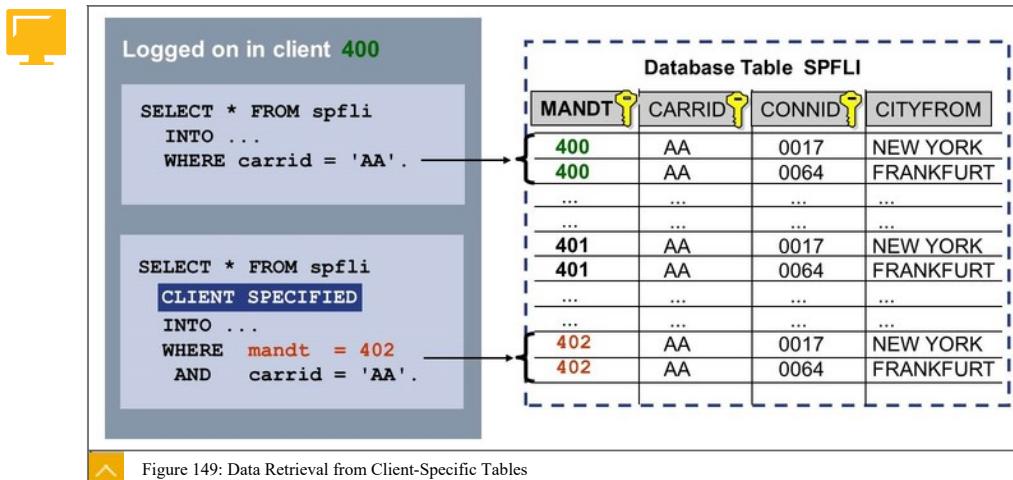


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Retrieve client-specific data
- Use database indexes
- Explain the SAP table buffer
- Retrieve data from several database tables
- Identify ways to change data in a database table

## Data Retrieval from Client-Specific Tables



A database table is classified as a client-specific table if it has a client field (data type CLNT) as the first key column and contains client-specific entries.

If you select data from a client-specific table without specifying the client, only data records from the current client are read (a restriction to the current client is automatically added to the WHERE clause of the SELECT statement).

If you want to read data from an explicitly specified client, specify the client name in the WHERE clause. However, the CLIENT SPECIFIED addition must be made after the FROM clause.

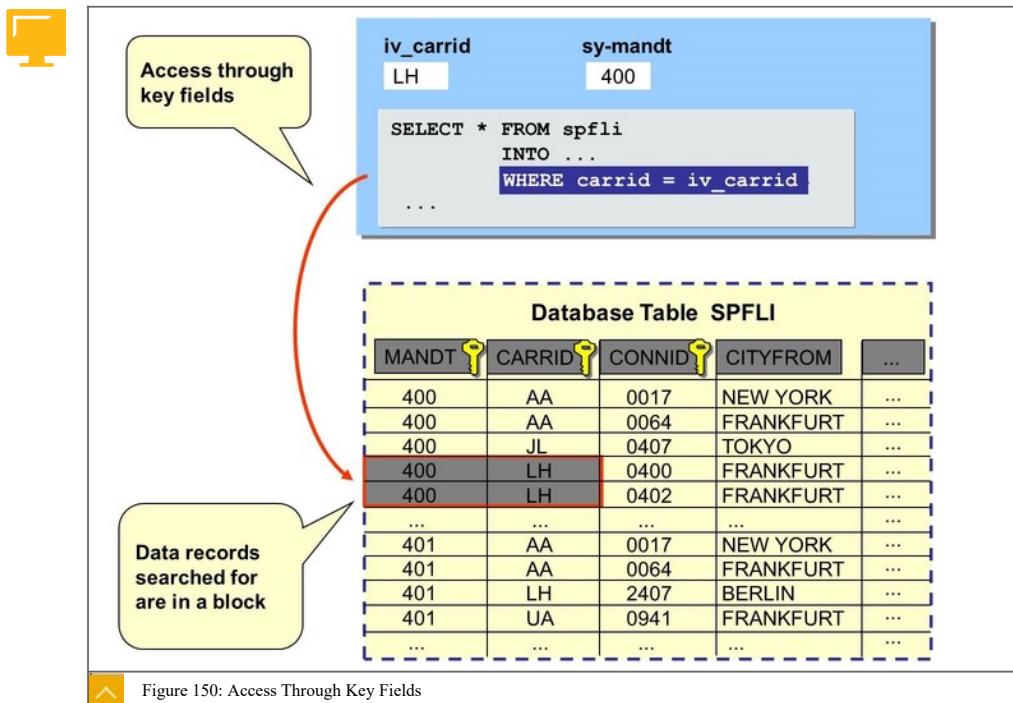
As cross-client reading is rarely requested in practice and is not relevant in the context of this course, the client field is usually omitted in presentations.

## General Database Performance Issues

In most cases, database access makes considerable demands on the runtime requirement of an ABAP application. In order to avoid unnecessarily overloading the system and to keep the wait time for the other users to a minimum, pay close attention to runtime requirements for database accesses.

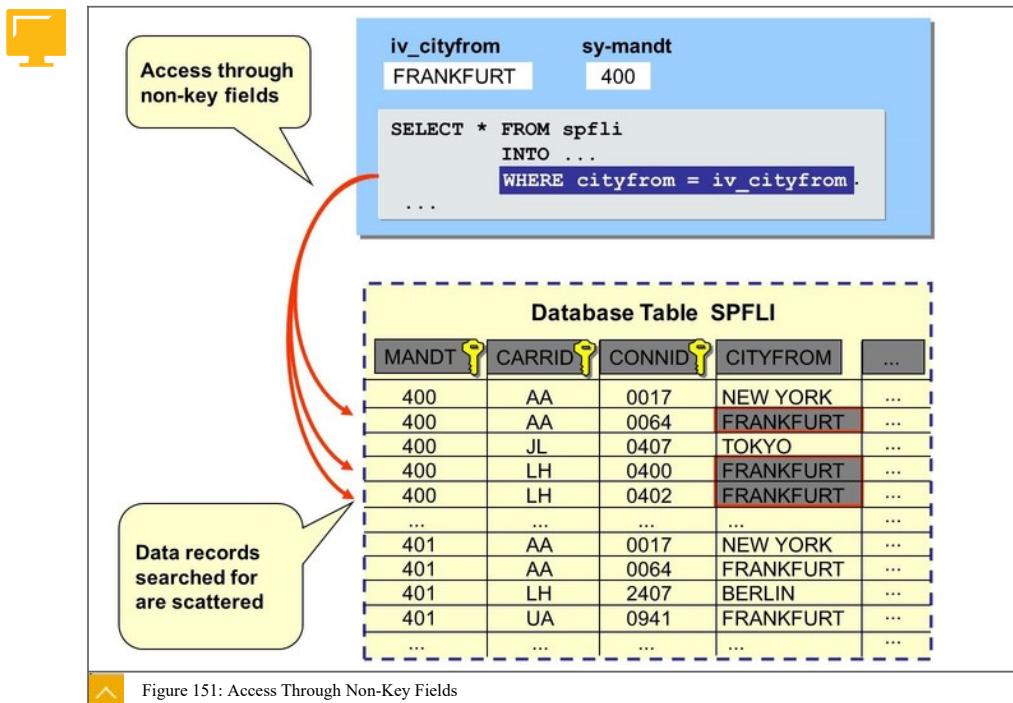
Open SQL has a range of technologies available to enable you to optimize the runtime requirement.

## Database Indexes



Every database manages the data records within the database table based on key fields. If access to the table is restricted to all, or at least the first few key fields, the database can quickly and effectively access the required data records.

## Access Through Non-Key Fields



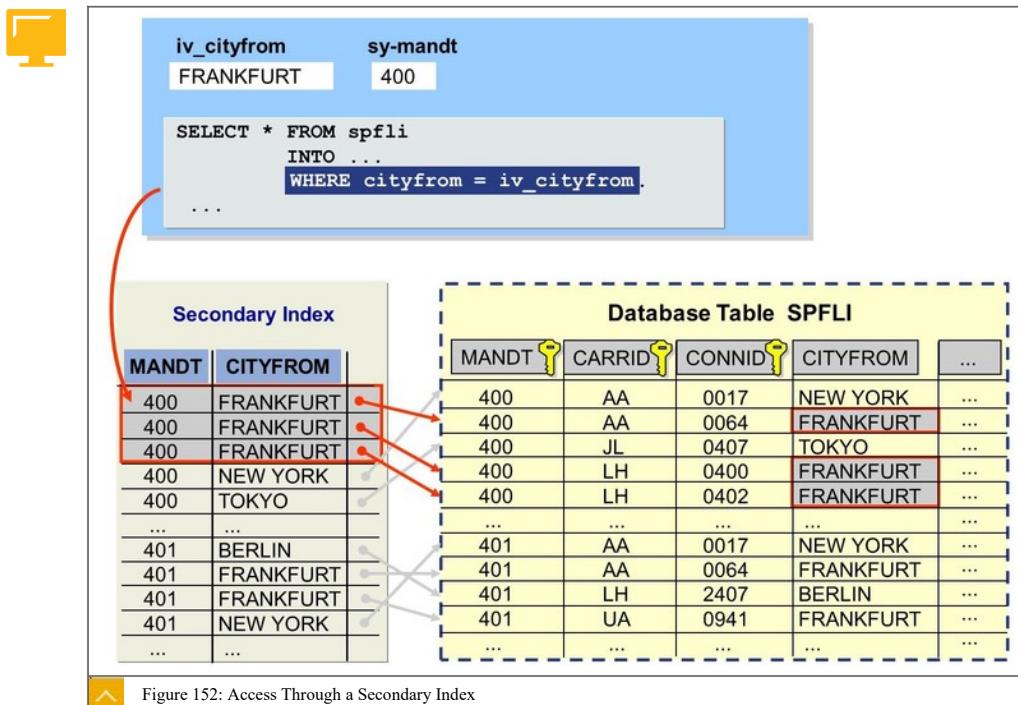
If database access is directed at fields that do not belong to the table key (non-key fields), the internal ordering principle cannot be used for rapid access. In the worst-case scenario, the entire table, or at least a very large part of the table, has to be searched for the required entries. This is referred to as a sequential search, and the wait time for the database access can be very long.



## Note:

In the figure, the data records within the database table are sorted according to the key field and stored on this basis. Some database systems actually work in this way. With others, the data records are not sorted. Instead, an index is created using the key fields. This index is referred to as the primary index. This is of no significance here.

## Access Through a Secondary Index



If you access a database table frequently using a certain selection of search fields, defining a secondary index containing the fields used in the selection will speed up corresponding accesses.

The secondary index is a separate, small table that contains the index fields and a reference to the relevant data records for each data record in the actual database table. If the fields in the selection match the fields in the secondary index (at least left-aligned and gap-free), the database searches within the secondary index and then reads the corresponding data records through the reference.

Create a secondary index in display mode for the relevant transparent table in the ABAP Dictionary using the **Indexes** button. When you activate the index, the secondary index is created in the database.

Whether and how the system uses an established secondary index in the database access is a function of the database system known as the Database Optimizer. In Open SQL, it is neither possible nor necessary to trigger the use of a secondary index by specifying one explicitly in the SELECT statement.



Hint:

For selections from client-specific tables, the client always transmits to the database (that is, it is always part of the selection). Therefore, including the client field when defining an index for such tables makes sense. At runtime, the database can use the client field to restrict the database search to the relevant client block in the secondary index.

### SAP Table Buffer

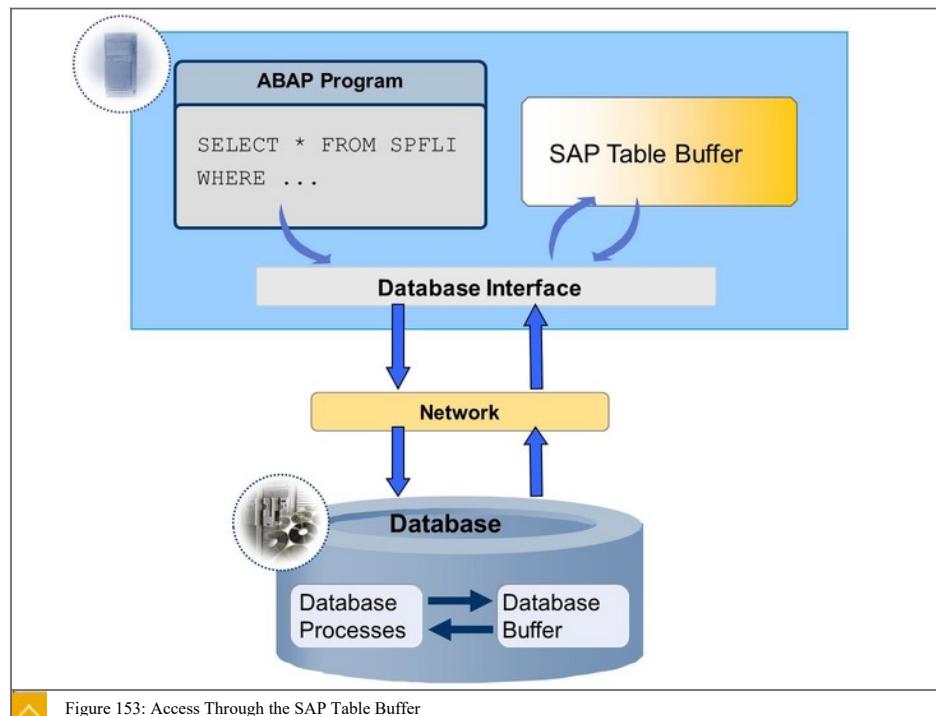


Figure 153: Access Through the SAP Table Buffer

For data retrieval, the system uses a major proportion of runtime to transfer the data from the database server to the application server. If the data is frequently read but seldom changed, you can reduce runtime by buffering the data on the application server.

Whether and how data can be buffered is a decision that needs to be made separately for each database table. Whether to buffer the data depends on how frequently the data is read or changed. The buffer settings are defined in the transparent tables in the ABAP Dictionary (Technical Settings button).



**Caution:**

The decision about buffering a database table is not simple and needs to be made by experienced ABAP developers in consultation with the system administrator. A separate SAP table buffer exists for each application server. If a system is comprised of several applications, a special synchronization mechanism is needed to ensure that the corresponding buffer contents are invalidated after changes have been made in the database. However, time lags in the synchronization process mean that for a short period of time, old data could be read from the buffer. This issue needs to be taken into account when making decisions about buffering.

If an ABAP program reads a buffered table, the database interface first tries to get the required data from the SAP table buffer. Reading data from the SAP buffer table speeds up access by a factor of between 10 and 100 compared to reading data from the database. The precise increase in speed depends on the structure of the table and on the exact system configurations.

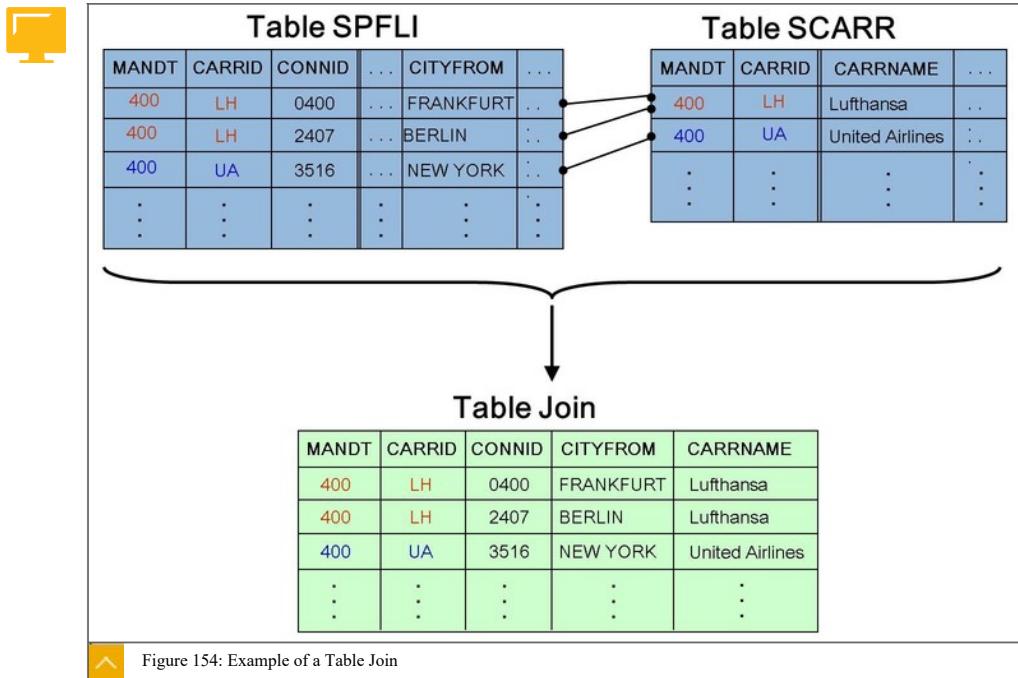
If the required data is not yet in the SAP table buffer, the database is accessed. Afterwards the database interface also stores the read data in the SAP table buffer.



**Hint:**

There are variants and additions for the SELECT statement that call for data to be read directly from the database, regardless of buffer settings. Such access is said to bypass the buffer. This kind of access can cause performance problems when accessing buffered tables and should be avoided.

## Join Mechanism

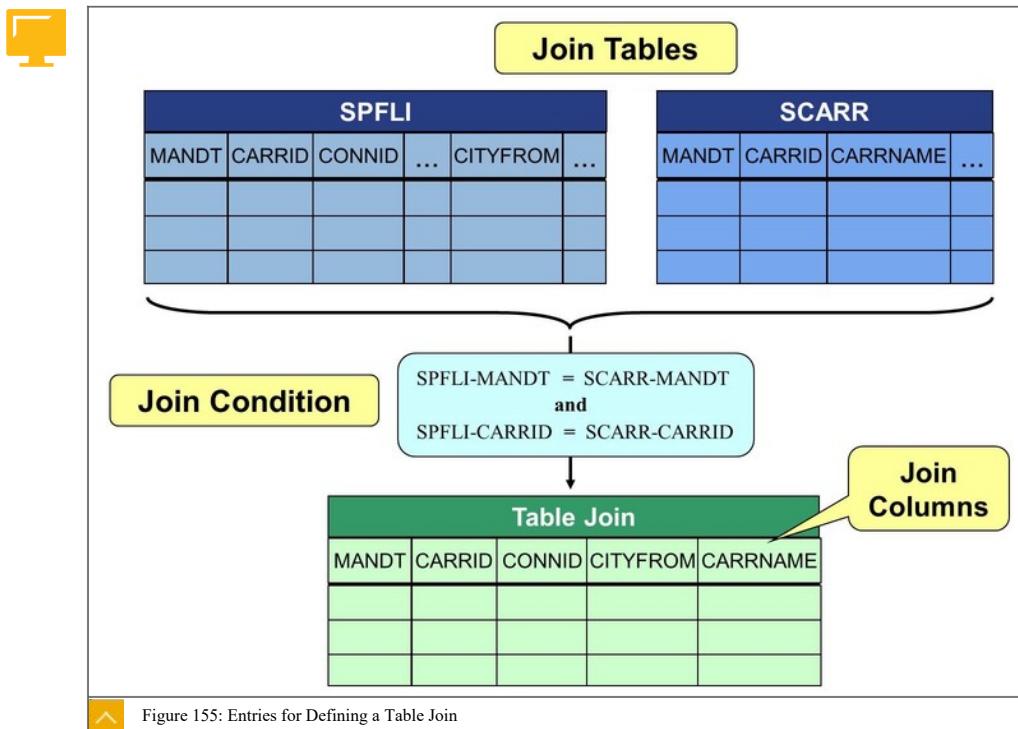


There is often a requirement to read data from various tables and display it on one table.

In general, the technique with the best performance for this task is a table join.

You will read and output records from the database table SPFLI. For each record, the full names of each respective airline, which are stored in SCARR, are to be output as well. The figure shows the logical creation of the corresponding table join from which you can select all of the required data using the SELECT statement.

## Entries for Defining a Table Join



## Table Join Definition

- Specify the following details when defining a table join:
  - Join tables  
Which database tables should be accessed and joined?
  - Join conditions  
What are the conditions for summarizing the corresponding records in the tables to be merged in a table join?
  - Join columns  
Which columns from the selected tables should be made available in the table join?

### Implementation Options for Table Joins

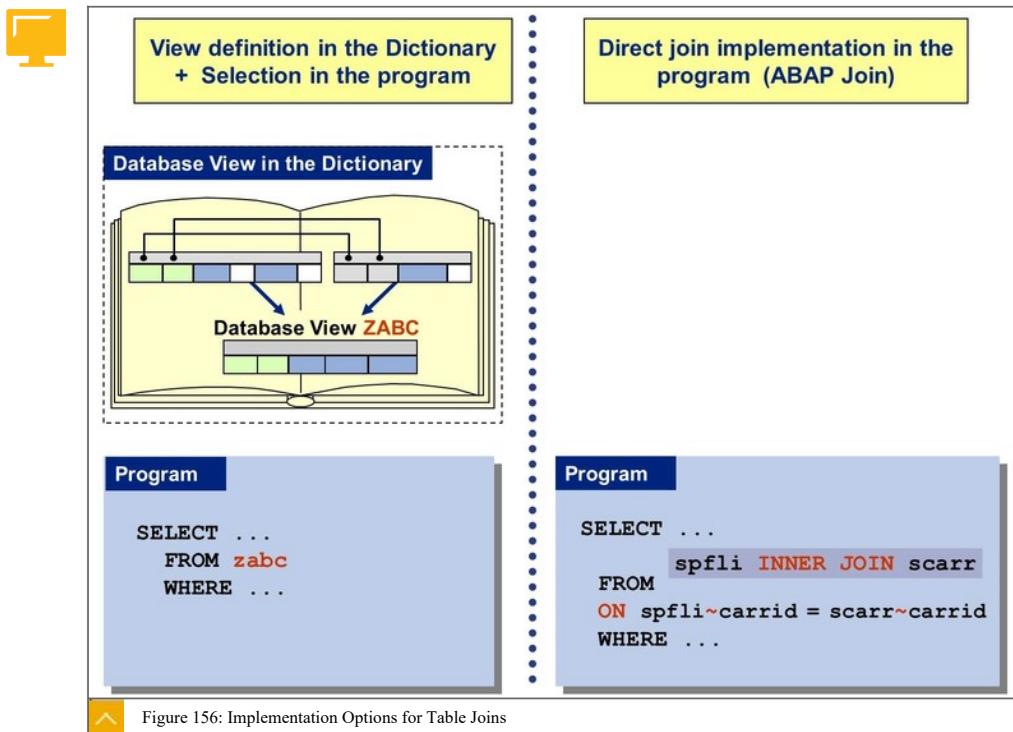


Figure 156: Implementation Options for Table Joins

### Methods for Implementing a Table Join

- The different ways for implementing a table join are as follows:
  - In the ABAP Dictionary, create a database view that corresponds to a table join and select from it in your program.  
For detailed information, refer to the online documentation for the ABAP Workbench in the ABAP Dictionary section.
  - In your program, you define the join directly in your SELECT statement (ABAP join). At runtime, the system dynamically generates an appropriate database query in the database interface.  
For more information, see the keyword documentation for the FROM clause of the SELECT statement.

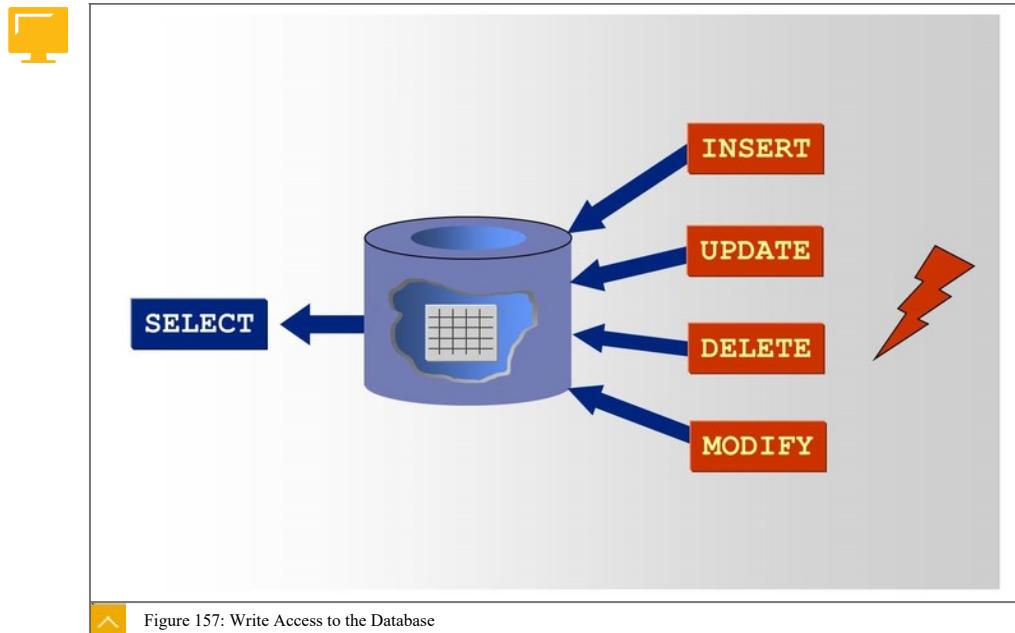


#### Hint:

Table joins and database views are only specific views on the content of database tables. They do not store the same data redundantly. When a selection is made the data is read from the underlying database tables.

You can join more than two tables in a table join.

## Write Access to the Database



In addition to the SELECT statement, Open SQL contains the UPDATE, INSERT, DELETE, and MODIFY statements. However, it is important to understand the SAP transaction concept associated with these database change accesses to avoid causing data inconsistencies.



## LESSON SUMMARY

You should now be able to:

- Retrieve client-specific data
- Use database indexes
- Explain the SAP table buffer
- Retrieve data from several database tables
- Identify ways to change data in a database table

## Unit 6

### Lesson 5

## Implementing Authorization Checks

### LESSON OVERVIEW

This lesson explains why an authorization check is useful and how to include it in your programs.

#### Business Example

Authorization checks in your programs are necessary to protect the data from unauthorized access. For this reason, you require the following knowledge:

- An understanding of the SAP authorization concept
- An understanding of how to implement authorization checks

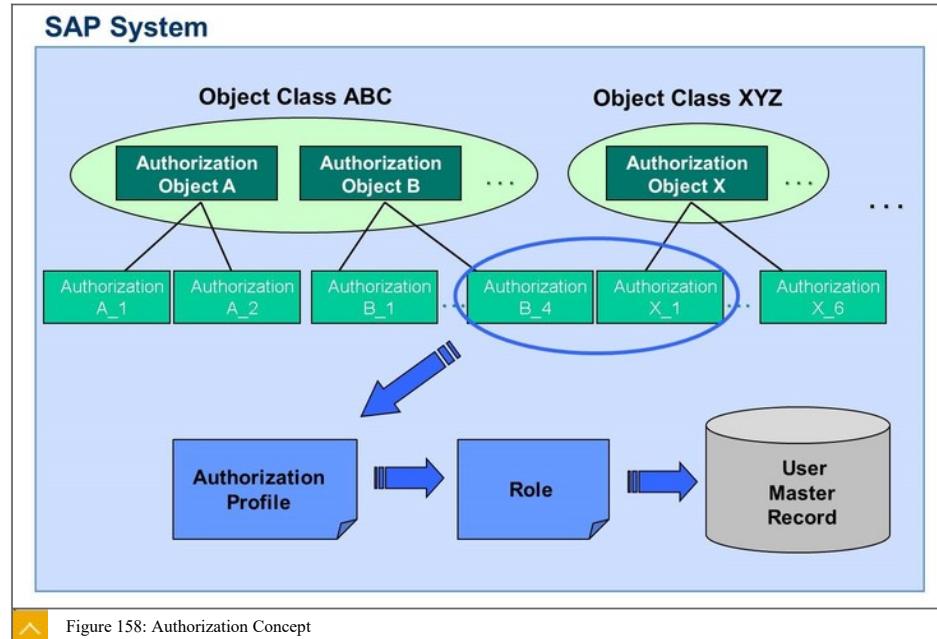


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

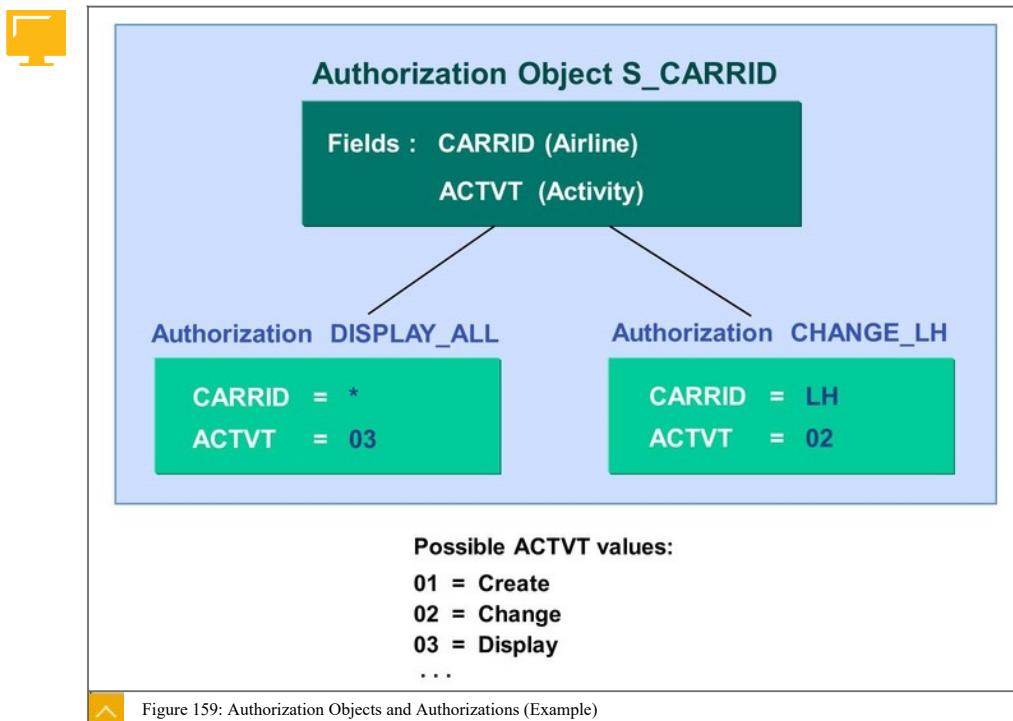
- Explain the authorization concept
- Implement authorization checks

#### Authorization Concept



You must protect critical data and parts of the functional scope of the SAP system from unauthorized access. You implement authorization checks in your program so that each user can only access areas for which he or she is authorized.

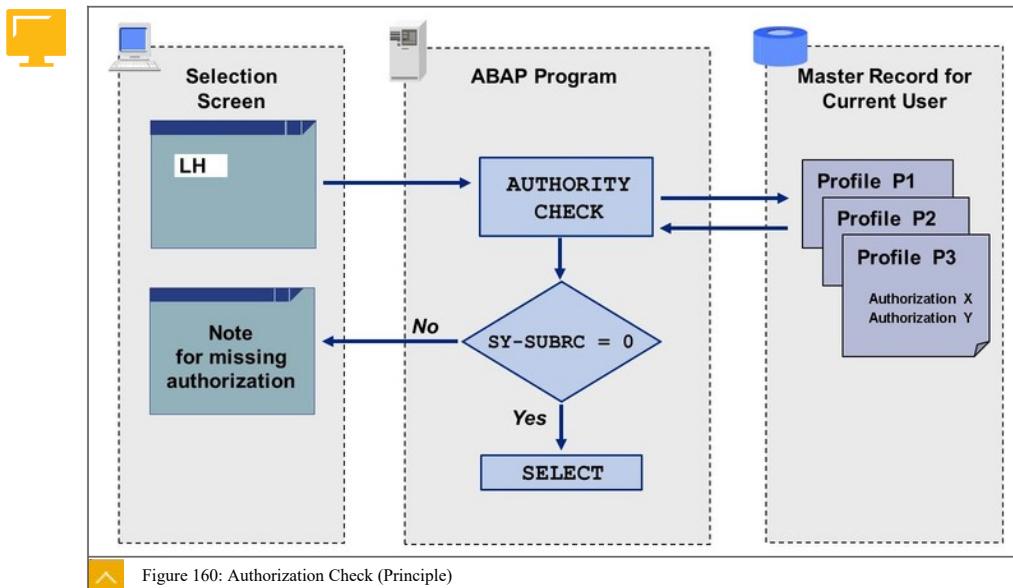
#### Authorization Objects and Authorizations (Example)



Authorization objects are defined within object classes. When defining an authorization object, the developer specifies the appropriate fields (without values). An actual authorization is derived from an authorization object by assigning values to these fields. This authorization can be integrated into the required user master record through an authorization profile.

Several different authorizations (for integration into different user master records) can be created for an authorization object.

## Authorization Check (Principle)



## Continuation of Program after Authority Check Results

- At runtime, use the AUTHORITY-CHECK statement to check whether the user has the authorization required to execute the function being called in the user master record.
- Depending on the check result (SY\_SUBRC), continue the program accordingly, as follows:

- SY\_SUBRC = 0

User has the required authorization to perform the function (for example, SELECT), therefore continue with the program.

- SY\_SUBRC <> 0

User does not have sufficient authorization, therefore, issue an error/warning message.



## Hint:

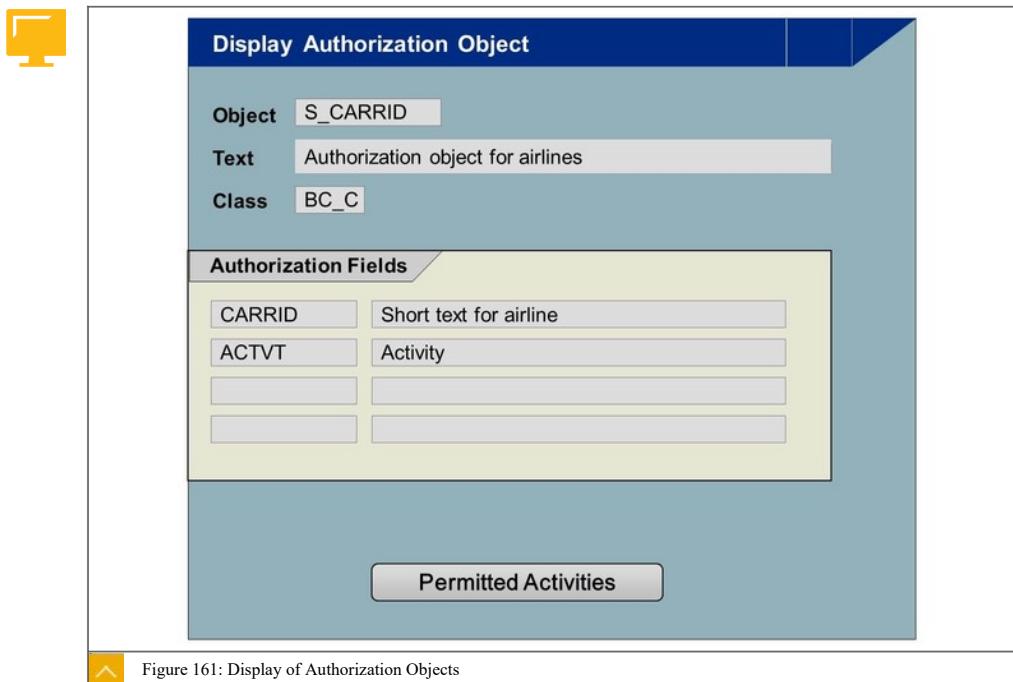
In addition to the technique described, it is also possible to control access to the entire program and transactions using authorizations. However, consider such checks as additions to, and not substitutions for the explicit authorization check by the developer.

Under normal circumstances, the definition of authorization objects is part of data modeling and the creation of database tables. In this course, you will access an existing data model so that you can use authorization object S\_CARRID in this data model.

Implementing the authorization concept is one of the tasks of the developer who programs access to database tables.

The subsequent steps, such as defining the authorizations and profiles and designing the user master records, are tasks of the administrator.

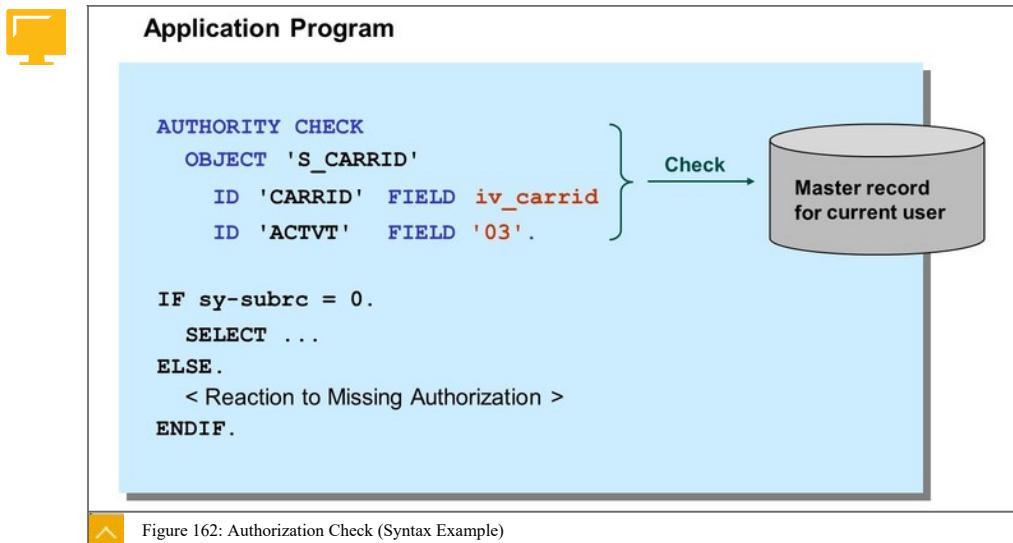
## Display of Authorization Objects



Before you can implement the required authorization check in your program, determine the structure (the fields) of the respective authorization concept. An object usually consists of the ACTVT(Activity) field and one other field, which specifies the data type to be protected (that is, material number, airline, and so on). The values of these authorization fields specify what each user is authorized to do.

You can display an authorization object directly in the ABAP Workbench (for example, by using the Other Object button or the Other tab page). You can also use transaction SU21. Here, you will get an overview of all of the authorization objects created in the system.

## Authorization Checks



The figure shows how to implement an authorization check.

For the authorization check in the program, you specify the authorization that is to be checked in the master record of the current user. You specify the authorization by specifying the authorization object, its fields, and the appropriate field values. Refer to the syntax shown in the figure.

In this example, the user authorization for access to the S\_CARRID object is checked; field CARRID (airline) contains the airline entered by the user, and field ACTVT (activity) contains the value '03' (display).

After the AUTHORITY-CHECK statement, check return code SY-SUBRC and implement the further processing of your program accordingly.



## Hint:

To exclude a check for a field, either do not enter the field in the AUTHORITY-CHECK statement or enter DUMMY as the field value. DUMMY is a predefined description entered without quotation marks.

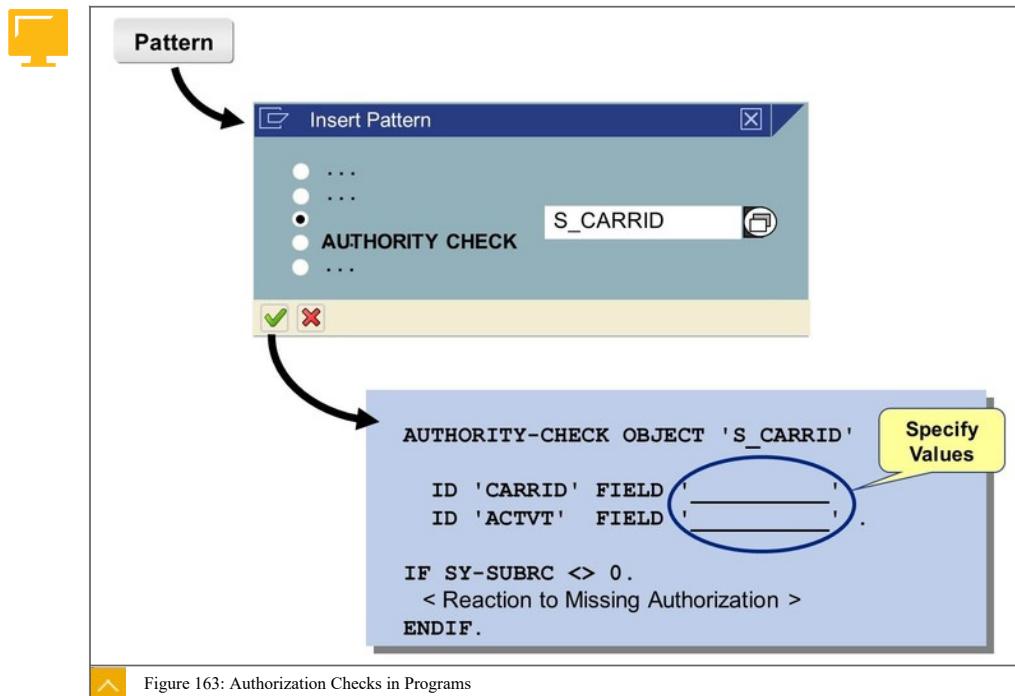
An example of a suppressed field check: When a change transaction is called, the system should immediately check whether the user has authorization to make a change for any airline. If the check fails, an appropriate message is to be output to the user immediately. Such a check can be implemented with the following syntax:

```

AUTHORITY-CHECK OBJECT 'S_CARRID'
  ID 'CARRID' DUMMY
  ID 'ACTVT'   FIELD '02'.

```

## Authorization Checks in Programs



To avoid spelling errors in object and field names, generate the AUTHORITY-CHECK statement in your source code by using the **Pattern** button. Then, maintain the field values and implement the evaluation of SY-SUBRC.



## LESSON SUMMARY

You should now be able to:

- Explain the authorization concept
- Implement authorization checks

## Unit 6

### Learning Assessment

1. Which of the following is the tool that you can use to create a transparent table?

Choose the correct answers.

- A transparent field
- B ABAP Dictionary
- C data field
- D database

2. When using the transparent table as a data type, other properties, such as the key definition or the technical properties, are relevant.

Determine whether this statement is true or false.

- True
- False

3. Which of the following are the types of reuse components that encapsulate database access?

Choose the correct answers.

- A Conceptual database
- B Function modules
- C Business Application Programming Interfaces (BAPIs)
- D Methods of local classes

4. The SELECT clause determines which lines are read into the target structure and processed using the statement block which you specify in the loop body.

Determine whether this statement is true or false.

- True
- False

5. Which of the following additions to a SELECT statement would you use to append rows to an internal table instead of overwriting the existing rows?

Choose the correct answer.

<input type="checkbox"/>	A INTO TABLE
<input type="checkbox"/>	B APPENDING TABLE
<input type="checkbox"/>	C INTO CORRESPONDING FIELD OF TABLE
<input type="checkbox"/>	D END SELECT

6. If you select data from client-specific tables without specifying the client, data records from the current and all other clients are read.

Determine whether this statement is true or false.

<input type="checkbox"/>	True
<input type="checkbox"/>	False

7. Which of the following will allow the database to search the entire table, or at least a very large part of the table for the required entries?

Choose the correct answer.

<input type="checkbox"/>	A direct search
<input type="checkbox"/>	B binary search
<input type="checkbox"/>	C sequential search
<input type="checkbox"/>	D indirect search

8. Which of the following should you specify when defining a table join?

Choose the correct answers.

<input type="checkbox"/>	A Join tables
<input type="checkbox"/>	B Join conditions
<input type="checkbox"/>	C Join rows
<input type="checkbox"/>	D Join columns

9. If an ABAP program reads a buffered table, the database interface tries to get the required data from the SAP table buffer.

Determine whether this statement is true or false.

True

False

10. Which of the following statements are contained within Open SQL?

Choose the correct answers.

A CREATE

B UPDATE

C INSERT

D SELECT

11. At runtime, which of the following statements can you use to check whether the actual user has the authorization required for executing the function in the user master record?

Choose the correct answer.

A AUTHORITY

B SELECT

C AUTHORITY-CHECK

D VALID AUTHORITY

## Unit 6

### Learning Assessment - Answers

1. Which of the following is the tool that you can use to create a transparent table?

Choose the correct answers.

A transparent field

B ABAP Dictionary

C data field

D database

You are correct! The developer can use a data model as the basis for implementing appropriate table definitions (transparent tables), including their relationships with each other in the ABAP Dictionary. Read more in the lesson, Explaining Data Models, Task: Data Model Overview, in the course BC400 (Unit 6, Lesson 1) or TAW10 Part I (Unit 12, Lesson 1).

2. When using the transparent table as a data type, other properties, such as the key definition or the technical properties, are relevant.

Determine whether this statement is true or false.

True

False

You are correct! Transparent tables can be used in the same way as structure types in programming. For example, transparent tables can be used to define a structured data object. Only the list of fields is important. Other properties of the transparent table, such as the key definition or the technical properties, are irrelevant when it is being used as a data type. Read more in the lesson, Explaining Data Models, Task: Structures in the ABAP Dictionary, in the course BC400 (Unit 6, Lesson 1) or TAW10 Part I (Unit 12, Lesson 1).

3. Which of the following are the types of reuse components that encapsulate database access?

Choose the correct answers.

A Conceptual database

B Function modules

C Business Application Programming Interfaces (BAPIs)

D Methods of local classes

You are correct! Before you program direct access to database tables, look for reuse components that handle the read process, such as Logical Databases, Function Modules, BAPIs, Methods of global classes. Read more in the lesson, Retrieving Single Database Records, Task: Types of Reuse Components that Encapsulate Database Access, in the course BC400 (Unit 6, Lesson 2) or TAW10 Part I (Unit 12, Lesson 2).

4. The SELECT clause determines which lines are read into the target structure and processed using the statement block which you specify in the loop body.

Determine whether this statement is true or false.

True

False

You are correct! The SELECT clause names the fields of the table that are to be read. Read more in the lesson, Retrieving Single Database Records, Task: SELECT Statement Clauses, in the course BC400 (Unit 6, Lesson 2) or TAW10 Part I (Unit 12, Lesson 2).

5. Which of the following additions to a SELECT statement would you use to append rows to an internal table instead of overwriting the existing rows?

Choose the correct answer.

A INTO TABLE

B APPENDING TABLE

C INTO CORRESPONDING FIELD OF TABLE

D END SELECT

You are correct! With array fetch, any content that might be present in the internal table is overwritten. If you want to append rows instead of overwriting, use the APPENDING TABLE addition. Read more in the lesson, Retrieving Multiple Database Records, Task: Data Retrieval Using an Array Fetch, in the course BC400 (Unit 6, Lesson 3) or TAW10 Part I (Unit 12, Lesson 3).

6. If you select data from client-specific tables without specifying the client, data records from the current and all other clients are read.

Determine whether this statement is true or false.

True

False

You are correct! If you select data from a client-specific table without specifying the client, only data records from the current client are read (a restriction to the current client is automatically added to the WHERE clause of the SELECT statement by the Database interface). Read more in the lesson, Describing Other Aspects of Database, Task: Data Retrieval from Client-Specific Tables, in the course BC400 (Unit 6, Lesson 4) or TAW10 Part I (Unit 12, Lesson 4).

7. Which of the following will allow the database to search the entire table, or at least a very large part of the table for the required entries?

Choose the correct answer.

A direct search

B binary search

C sequential search

D indirect search

You are correct! In the worst-case scenario, the entire table, or at least a very large part of the table, must be searched for the required entries. This is referred to as a sequential search, and the wait time for the database access can be very long. Read more in the lesson, Describing Other Aspects of Database Access, Task: Access Through Non-Key Fields, in the course BC400 (Unit 6, Lesson 4) or TAW10 Part I (Unit 12, Lesson 4).

8. Which of the following should you specify when defining a table join?

Choose the correct answers.

A Join tables

B Join conditions

C Join rows

D Join columns

You are correct! Specify the following details when defining a table join: Join tables, Join conditions, Join columns. Read more in the lesson, Describing Other Aspects of Database Access, Task: Table Join Definition, in the course BC400 (Unit 6, Lesson 4) or TAW10 Part I (Unit 12, Lesson 4).

9. If an ABAP program reads a buffered table, the database interface tries to get the required data from the SAP table buffer.

Determine whether this statement is true or false.

 True False

You are correct! If an ABAP program reads a buffered table, the database interface first tries to get the required data from the SAP table buffer. If the required data is not yet in the SAP table buffer, the database is accessed. Afterwards the database interface also stores the read data in the SAP table buffer. Read more in the lesson, Describing Other Aspects of Database Access, Task: SAP Table Buffer, in the course BC400 (Unit 6, Lesson 4) or TAW10 Part I (Unit 12, Lesson 4).

10. Which of the following statements are contained within Open SQL?

Choose the correct answers.

 A CREATE B UPDATE C INSERT D SELECT

You are correct! In addition to the SELECT statement, Open SQL contains the UPDATE, INSERT, DELETE, and MODIFY statements. Read more in the lesson, Describing Other Aspects of Database Access, Task: Write Access to the Database, in the course BC400 (Unit 6, Lesson 4) or TAW10 Part I (Unit 12, Lesson 4).

11. At runtime, which of the following statements can you use to check whether the actual user has the authorization required for executing the function in the user master record?

Choose the correct answer.

 A AUTHORITY B SELECT C AUTHORITY-CHECK D VALID AUTHORITY

You are correct! At runtime, use the AUTHORITY-CHECK statement to check whether the user has the authorization required to execute the function being called in the user master record. Read more in the lesson, Implementing Authorization Checks, Task: Authorization Check (Principle), in the course BC400 (Unit 6, Lesson 5) or TAW10 Part I (Unit 12, Lesson 5).

## UNIT 7

# Classic ABAP Reports

### Lesson 1

Implementing ABAP Lists	232
-------------------------	-----

### Lesson 2

Implementing Selection Screens	236
--------------------------------	-----

### Lesson 3

Implementing Events of ABAP Reports	244
-------------------------------------	-----

### UNIT OBJECTIVES

- Define ABAP list titles and column headers
- Describe the attributes and benefits of selection screens
- Implement options for restricting selections of selection screens
- Implement the events of ABAP reports

## Unit 7

### Lesson 1

# Implementing ABAP Lists

#### LESSON OVERVIEW

This lesson discusses classic ABAP reports. ABAP programs of this type are closely linked with the classic event control technique as well as the special selection screen and the ABAP list user dialogs.

#### Business Example

You want to develop a program that uses selections on a selection screen to read and list data from a database. For this reason, you require the following knowledge:

- An understanding of how to describe the attributes and benefits of ABAP lists



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Define ABAP list titles and column headers

## ABAP List Features

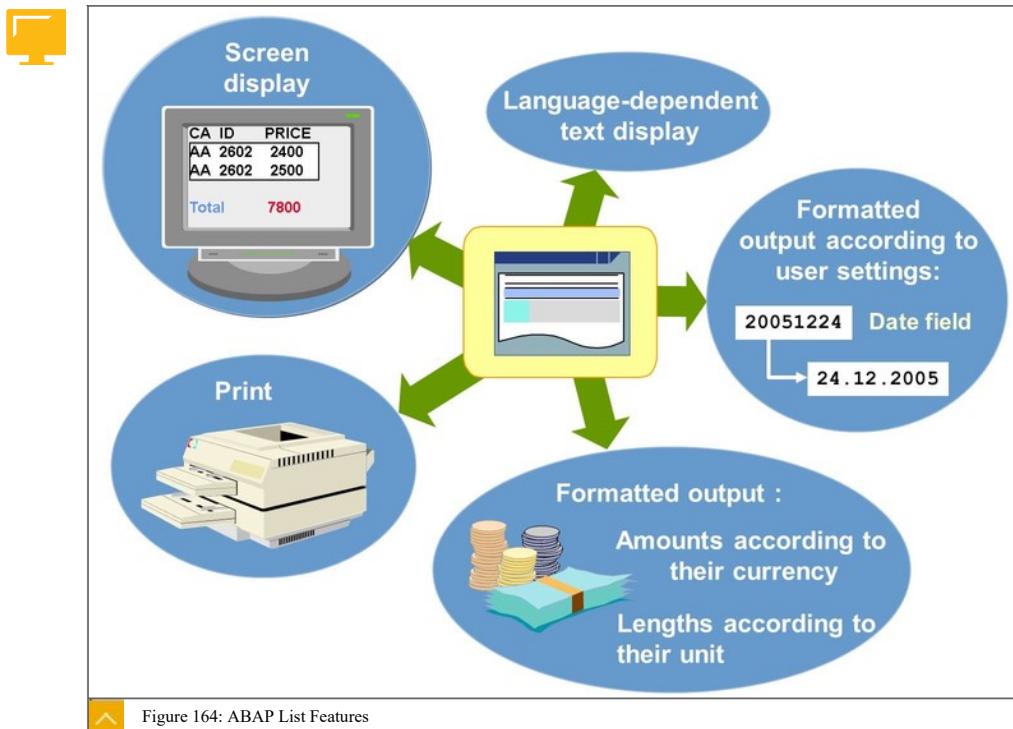


Figure 164: ABAP List Features

You use a list to minimize programming effort to output data.

#### Lists and Requirements of Business Data

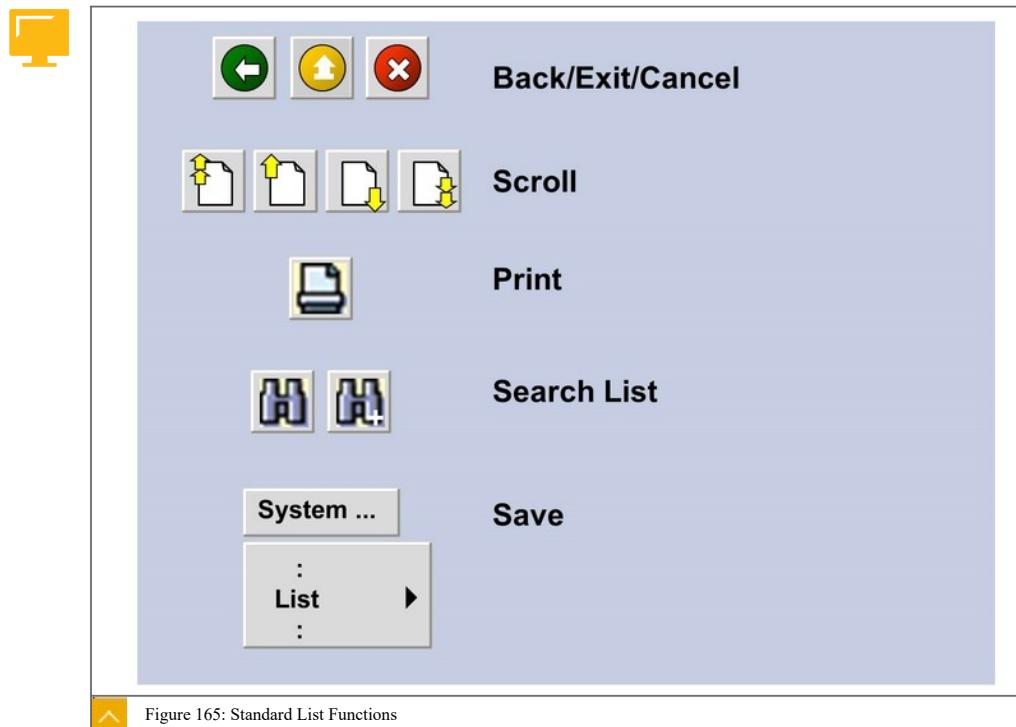
- Lists take into account the special requirements of business data in the following ways:
  - Lists can be designed for a number of languages. For example, texts and headers appear in the logon language whenever a corresponding translation is available.
  - Lists can display monetary values in the appropriate currency.

#### List Output Options

- The following options are available when outputting a list:
  - Screen  
You can add colors and icons to the screen.
  - Printer  
You can print lists.
  - Internet or intranet  
You can have the system convert the list to HTML format.
  - Save

You can save lists within the SAP system as well as outside, for example, for processing using spreadsheet programs.

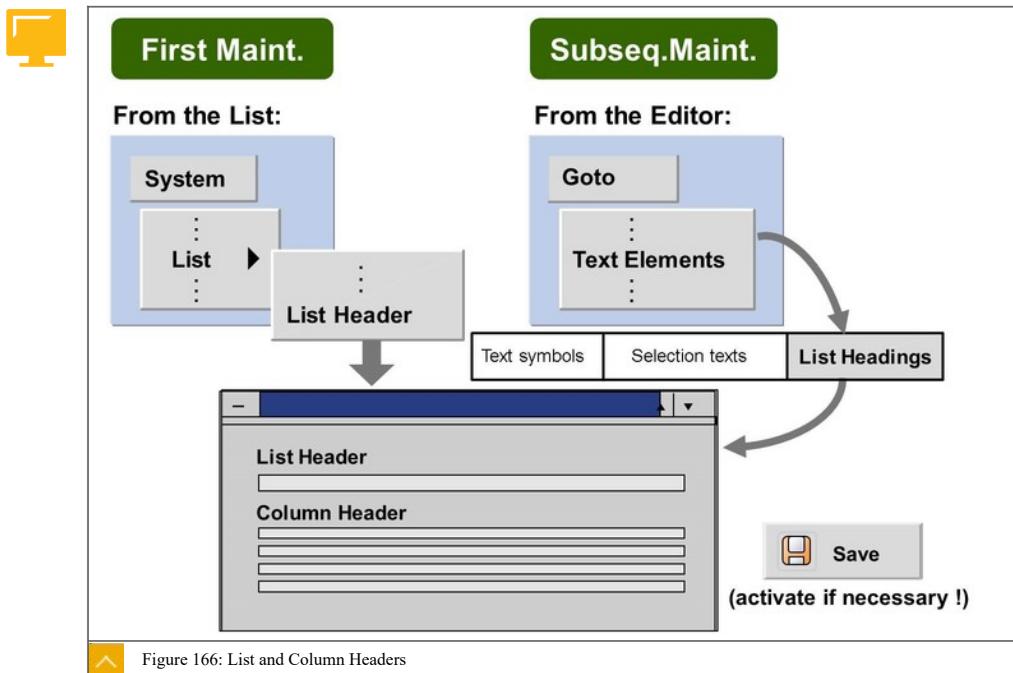
Standard List Functions



The standard user interface for a list offers a range of functions.

You can use the Menu Painter to adapt the default list interface to your own needs. More information is available in the documentation for this tool and in the corresponding training course.

## List Titles and Column Headers



You know how to implement translatable text symbols to make the interface of your program dependent on the logon language of the user. Text symbols are available in all types of programs, for example, for module pools, function groups, and global classes.

For executable programs (reports) in particular, you have the option of maintaining a single line list header and up to four rows of column headers in addition to the text symbols for the classic ABAP list.

Together with text symbols and selection texts, list and column headers belong to the text elements of a program.

For initial maintenance of the headers, you have to activate your program, then create the list by executing the program. You can maintain the headers directly above your list using System → List → List Header . The next time you start the program, they will appear in the list automatically.

To change the maintained header (follow-up maintenance), you do not have to start the program again and generate the list. Instead, starting from the editor in which you load the program, you can access the maintenance environment for changing the headers by choosing Goto → Text Elements ; tab List Headers .

To translate list and column headers from within the ABAP Editor, choose Goto → Translation .



## LESSON SUMMARY

You should now be able to:

- Define ABAP list titles and column headers

## Unit 7

### Lesson 2

# Implementing Selection Screens

#### LESSON OVERVIEW

This lesson discusses classic ABAP reports. This type of ABAP program is closely linked with the classic event control technique as well as the special selection screen.

#### Business Example

You want to develop a program that uses selections on a selection screen to be read. For this reason, you require the following knowledge:

- An understanding of the properties of the ABAP list and benefits of selection screens
- An understanding of how to implement the options for restricting selections on the selection screen

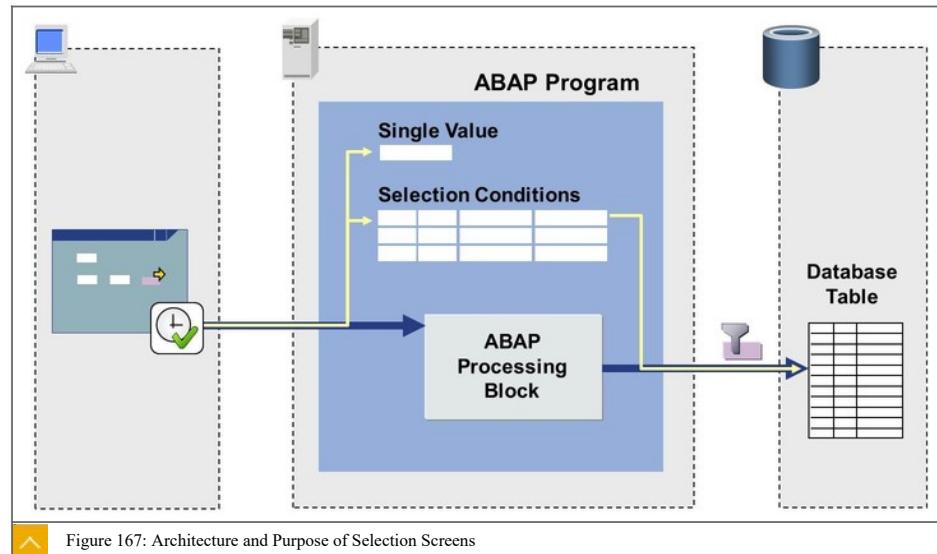


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the attributes and benefits of selection screens
- Implement options for restricting selections of selection screens

#### Selection Screen Properties



You use selection screens for entering selection criteria for data selection. For example, if the program creates a list of data from a very large database table, it makes sense for users to

select the data records they actually require and for the system to read only this data from the database. Apart from reducing the memory requirement, this also reduces the network load.

From a technical perspective, selection screens are dynpros. However, the developer does not design them directly with the Screen Painter; the system generates them in accordance with declarative statements in the source code.

#### Selection Screen Attributes

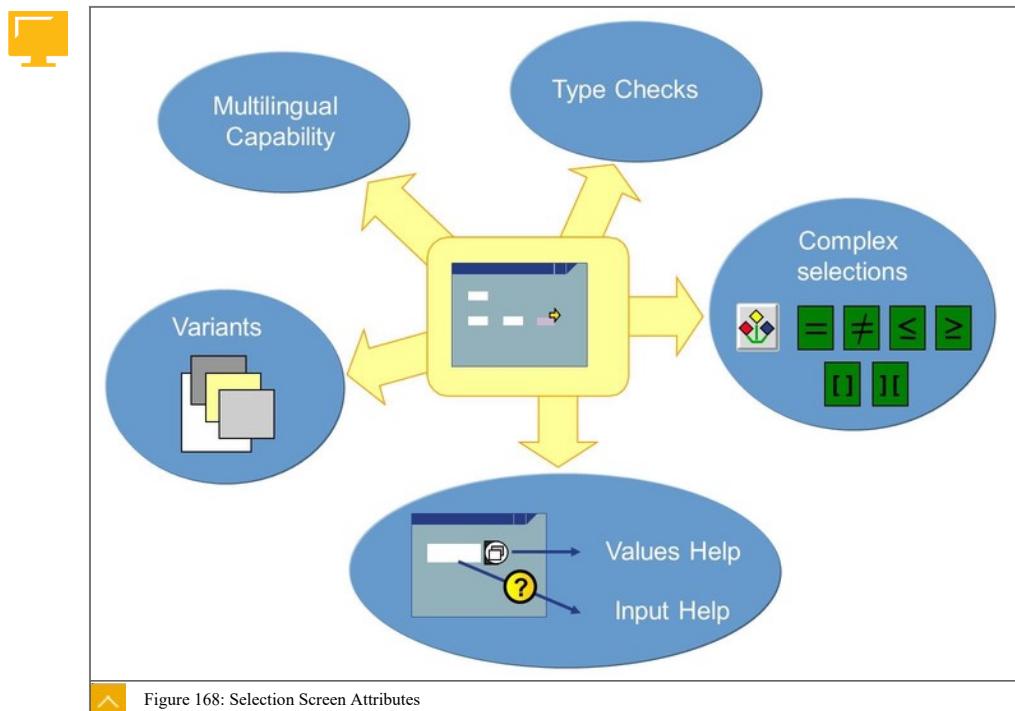


Figure 168: Selection Screen Attributes

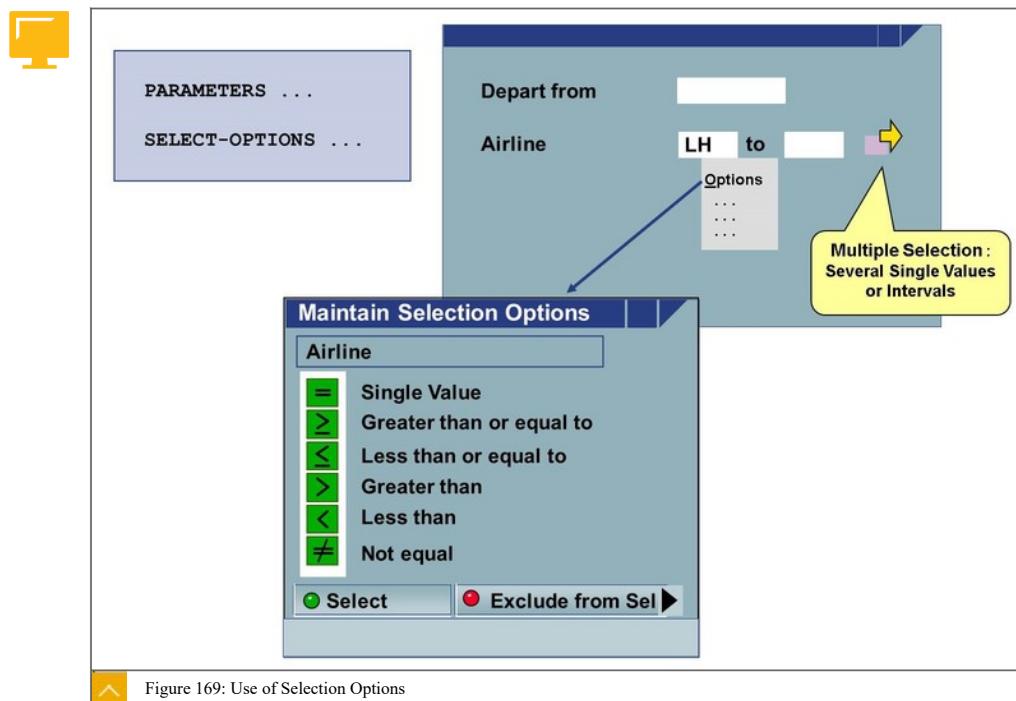
#### Standard Functions of the Selection Screen

- The selection screen has the following standard functions:
  - Texts in several languages
  - Automatic type checking
  - Complex selections for intervals, comparative conditions, and patterns
  - Field documentation for input fields
  - Search help associated with the Dictionary type
  - Saving of selection screens as variants

You can maintain texts on the selection screen in several languages. At runtime, the system displays the texts in the user's logon language. The system automatically performs a type check. If the user enters a value that does not correspond to the type of the input field, the SAP GUI will not accept it and will display an error message. For example, if the user enters characters in an input field that is defined as numeric, an error message is displayed. You can implement complex selections (SELECT-OPTIONS) on the selection screen in addition to

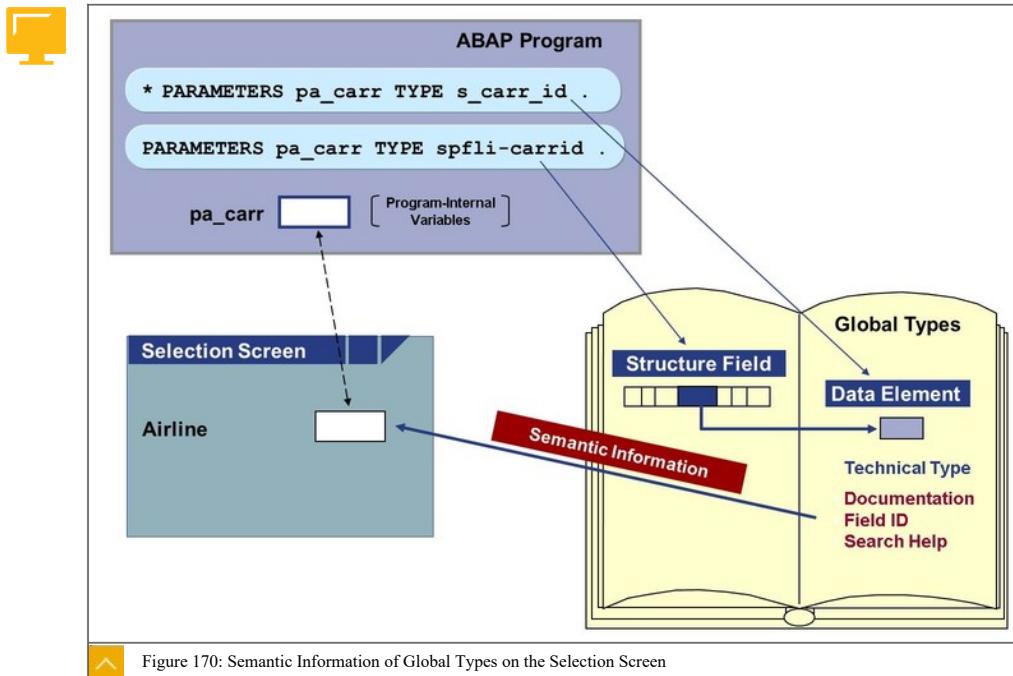
single value entries (PARAMETERS). The user can then enter intervals, comparative conditions, or even patterns as restrictions. The user can display the field documentation for the input field by pressing F1 if the developer defines the input field using a Dictionary element (for example, data element). The user can call the search help associated with the Dictionary type to display possible inputs by pressing F4. The user can save completed selection screens as variants for reuse or use in background operations.

#### Use of Selection Options



The figure illustrates the use of selection options, which enable complex entries. For more information about the operation in question, choose the Help on Screen button on the additional screen for multiple selections.

## Semantic Information of Global Types on the Selection Screen



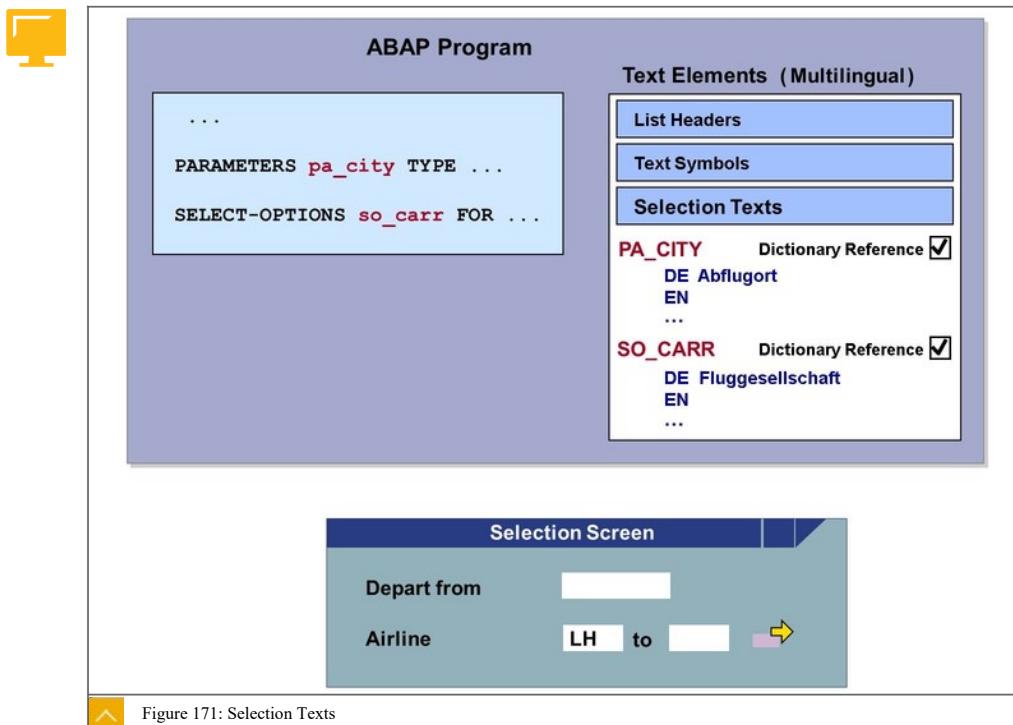
## Semantic Information on the Selection Screen

- If an input field is typed with a data element, the following additional semantic information is available on the selection screen:
  - The long field label of the data element can be adopted to describe the input field on the selection screen (selection text).
  - The documentation of the data element is automatically available as input help (F1).
  - The search help is available as input help (F4) if it is linked to the data element.

When typing an input field with a structure field instead of a data element, semantic information is also available on the selection screen. The semantic information is then taken from the data element used in the definition of the structure field

For more information, refer to the online documentation for the ABAP Dictionary.

## Selection Texts

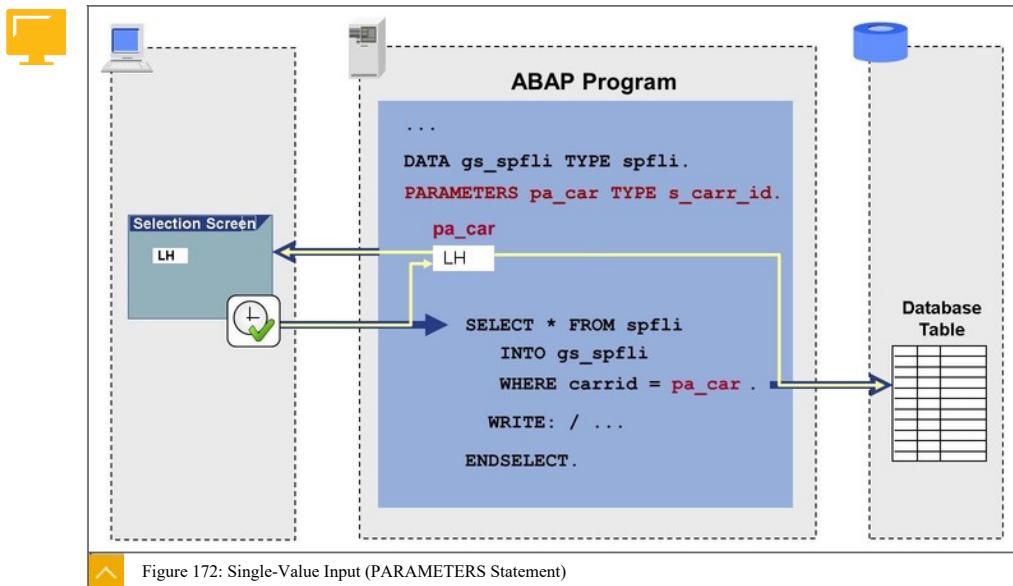


On the selection screen, the names of the input fields appear as their description by default. However, you can replace these with corresponding selection texts and then translate these texts into any other languages you require. At runtime, the system displays selection texts in the logon language of the user (automatic language).

Just like list headers and text symbols, selection texts belong to the text elements of the program. From the ABAP Editor, choose Goto → Text Elements ; tab Selection Texts to maintain them. You can implement your translation using Goto → Translation .

If you type the input field directly or indirectly with a data element, you can take over the long field name from one of the texts stored in the ABAP Dictionary (“Dictionary reference”). This provides you with an easy option for standardizing texts for all selection screens.

## Single-Value Input



An input variable is defined with a `PARAMETERS` statement instead of a `DATA` statement. The figure illustrates the usage and the runtime behavior of an input variable defined using a `PARAMETERS` statement. The definition of such an input variable creates a variable in the system and generates a selection screen with a corresponding input option.

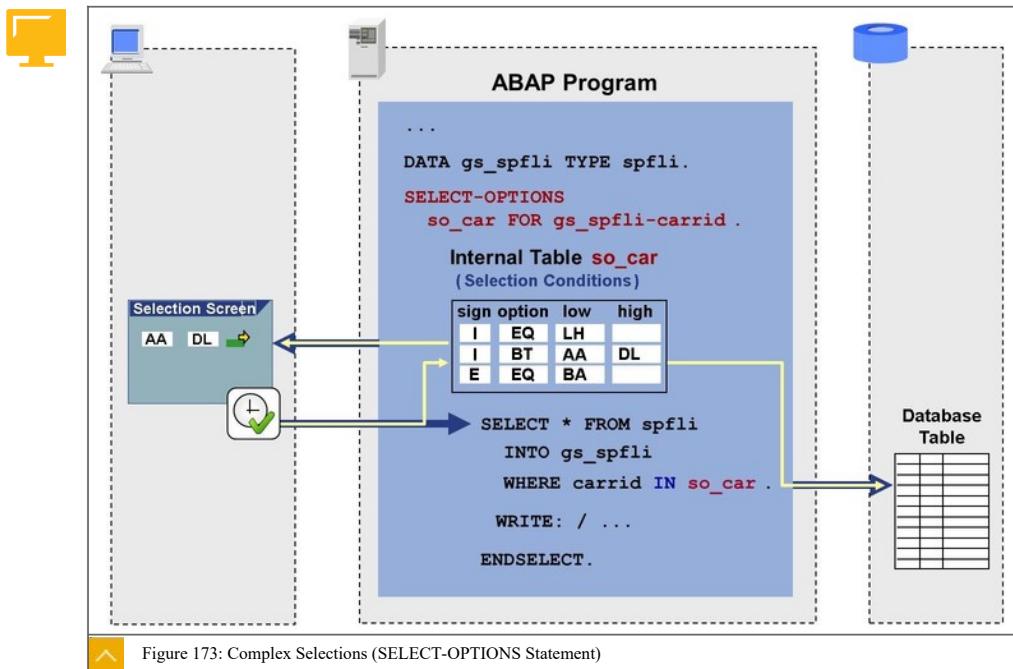
## Considerations When Defining Input Variables

- Take the following aspects into consideration:
  - The name of the input variable can be up to 8 characters long.
  - The input variable must not be typed with the standard types `F`, `STRING`, and `XSTRING`.
  - The assignment of a default value is implemented with the `DEFAULT` addition, not the `VALUE` addition.

The system displays a default value by means of the `DEFAULT` as a default value that can be overwritten.

For example, if the user enters a value and chooses the `Execute` button, the system transfers the input value to the internal variable. The system can then be used to restrict the database selection.

## Complex Selections



You can use the `SELECT-OPTIONS <name> FOR <data object>` statement to define a selection option for entering complex selections, where `<name>` is the name of the select option and `<data object>` is a predefined variable. Such a definition creates an internal table of the specified name within the program (`so_car`) and generates a selection screen with an input option for limiting the specified variable (`gs_spfli-carrid`).

The system transports user entries to the automatically generated internal table when the user chooses the `Execute` button. The internal table has four columns: sign, option, low, and high.

## Entries Created for User Input

- The figure Complex Selections (SELECT-OPTIONS Statement) illustrates the following entries created for user input:
  - If LH is entered, a row is generated with the values I (inclusive), EQ (equal), LH, and Space.
  - If the interval AA to DL is entered, a row is generated with the values I (inclusive), BT (between), AA, and DL.
  - If BA is entered as the single value to be excluded, a row is generated with the values E (exclusive), EQ (equal), BA, and Space.

After the internal table has been filled with the selection criteria, it can be used to limit the database selection, as shown in the figure.

### Table Content

#### Interpretation of Table Content

- The table content is interpreted as follows:
  - If  $I_1, \dots, I_n$  and  $E_1, \dots, E_m$  are the inclusive or exclusive conditions of the internal table, the composite condition that is used to limit the data selection is  $(I_1 \text{ OR } \dots \text{ OR } I_n) \text{ AND } (\text{NOT } E_1) \text{ AND } \dots \text{ AND } (\text{NOT } E_m)$ .
  - If the table is empty, the WHERE condition for the relevant field is met because there are no selections.



#### Hint:

The IN operator can also be used in logical expressions, for example, IF GS\_SPFLI-CARRID IN SO\_CAR.

The same three special features apply for the definition of select-options and the PARAMETERS statement.

If the program fills the internal table of the selection option using the DEFAULT addition or the APPEND statement before the display of the selection screen (INITIALIZATION), the system displays content on the selection screen as proposed conditions that can be overwritten.

For further details, refer to the keyword documentation for SELECT-OPTIONS.



#### LESSON SUMMARY

You should now be able to:

- Describe the attributes and benefits of selection screens
- Implement options for restricting selections of selection screens

## Unit 7

### Lesson 3

# Implementing Events of ABAP Reports

#### LESSON OVERVIEW

This lesson explains the implementation of ABAP reports.

#### Business Example

You want to develop a program that demonstrates implementation of ABAP reports. For this reason, you require the following knowledge:

- An understanding of how to implement the events of ABAP reports



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Implement the events of ABAP reports

#### ABAP Events

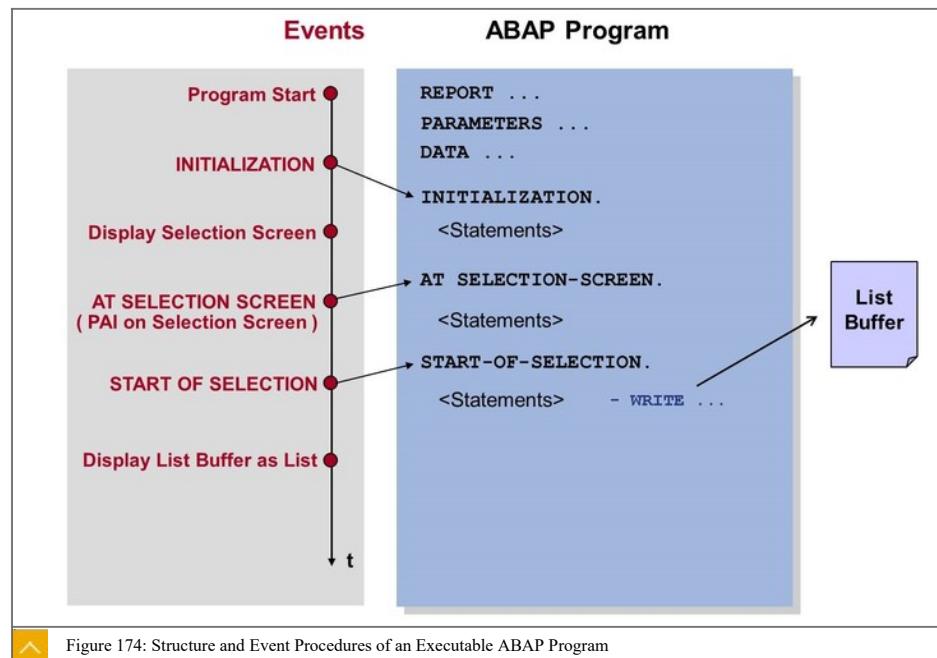


Figure 174: Structure and Event Procedures of an Executable ABAP Program



## Note:

The LOAD-OF-PROGRAM event should not be used in executable programs for the following reasons:

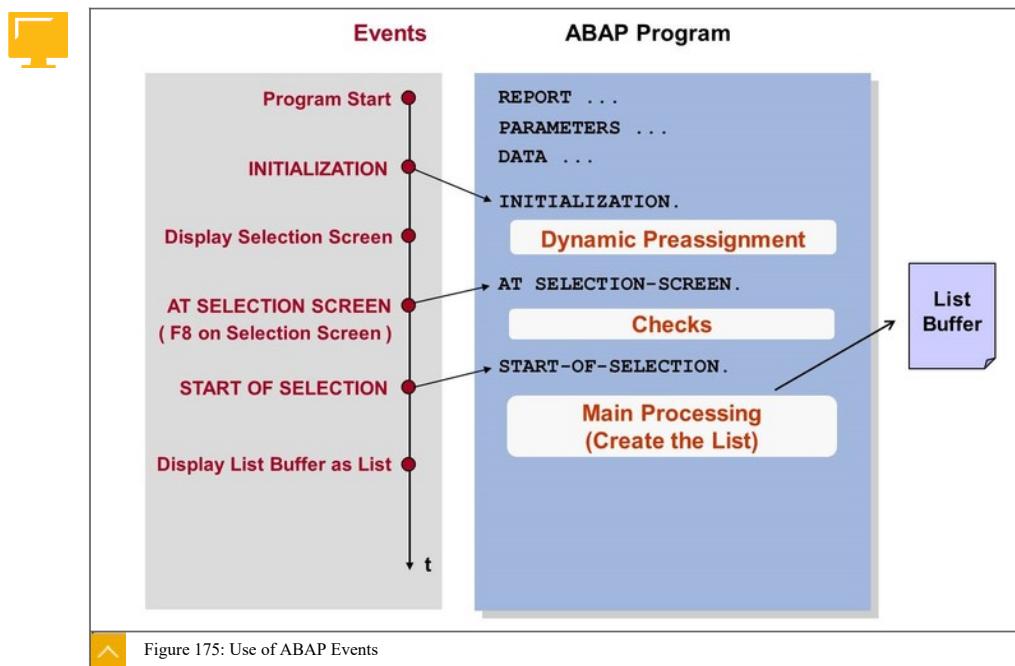
- LOAD-OF-PROGRAM, which exists since Release 4.6 parallel to INITIALIZATION, has been designed mainly for use in function groups and module pools that did not have a start action prior to Release 4.6.
- INITIALIZATION is still used for executable programs to implement the dynamic default settings for the selection screen. (See also the documentation for the corresponding events.)

When you start an ABAP program, the system creates all global data objects of the programs in the working memory. The runtime system then triggers various events in succession. If a processing block exists for a triggered event in the program, the system executes the statements in this block in sequence.

The figure, Structure and Event Procedures of an Executable ABAP Program , illustrates the basic events that are triggered, and the sequence in which they are triggered; the program implements the corresponding processing blocks. An executable ABAP program is thus a collection of processing blocks that are processed for the respective events.

Outputs created by means of WRITE statements are stored in list buffers, and are only displayed as a list after the system has processed the START-OF-SELECTION block.

## Use of ABAP Events



If you assign values to the PARAMETERS variables in the INITIALIZATION block, the system displays these as changeable default values in the input fields when the selection screen is subsequently displayed. You have the option of specifying a default value for the respective input field in the PARAMETERS definition by means of the DEFAULT addition. However, you can use the value assignment described in the INITIALIZATION block to assign another default value (dynamic pre-population of the selection screen) dynamically (that is, with reference to the situation).

If the user clicks a button on the selection screen (thus triggering Process After Input [PAI]), the system transports the entries into the corresponding internal PARAMETERS variables of the program and triggers the AT SELECTION-SCREEN event. The corresponding processing block is, therefore, suitable for an input or authorization check. If, for example, a type E message is sent in this event block because a user does not have the required authorization, the selection screen will be displayed again with the error message, allowing users to correct their entries.

Only if no error message is sent in the AT SELECTION-SCREEN block, the START-OF-SELECTION block is executed afterwards.

The main processing of the program should then take place in the corresponding processing block.

### Event Block Characteristics

#### Characteristics of Event Blocks

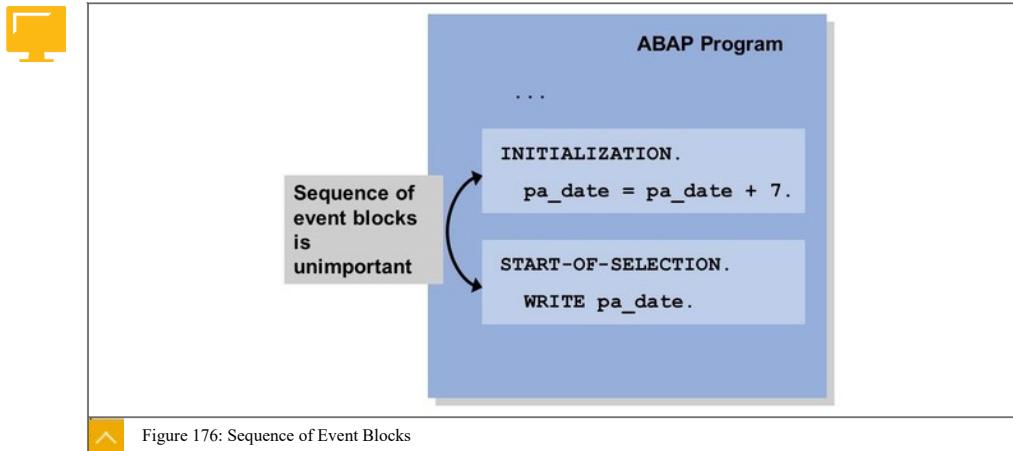


- Introduced with an event keyword
- Ends by beginning the next processing block
- Cannot be nested
- Existence is not absolutely necessary
- Sequence of event blocks is unimportant
- Implicit standard event block in executable program: START-OF-SELECTION

Processing blocks cannot be nested, because nesting would contradict the concept of ABAP events.

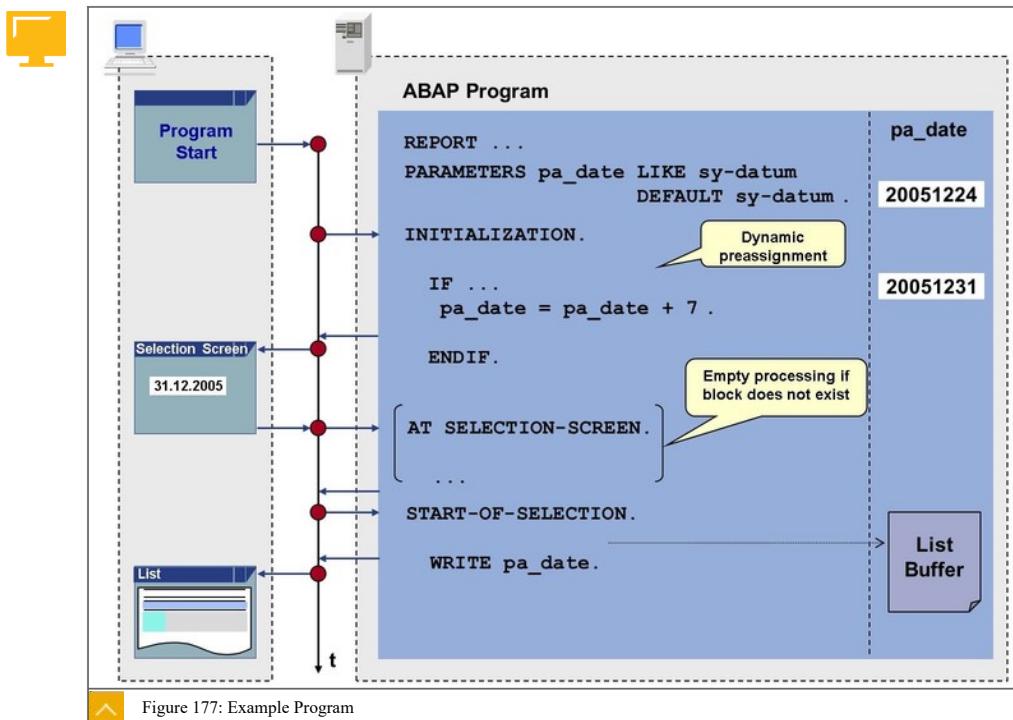
The system does not execute any statement and triggers the next event if the processing block is missing.

If no blocks are implemented in the program, the system assigns all statements to the standard processing block START-OF-SELECTION.



The ABAP runtime system controls the triggering of the events and the execution of the processing blocks. The sequence in which you place the event blocks in the program is not important.

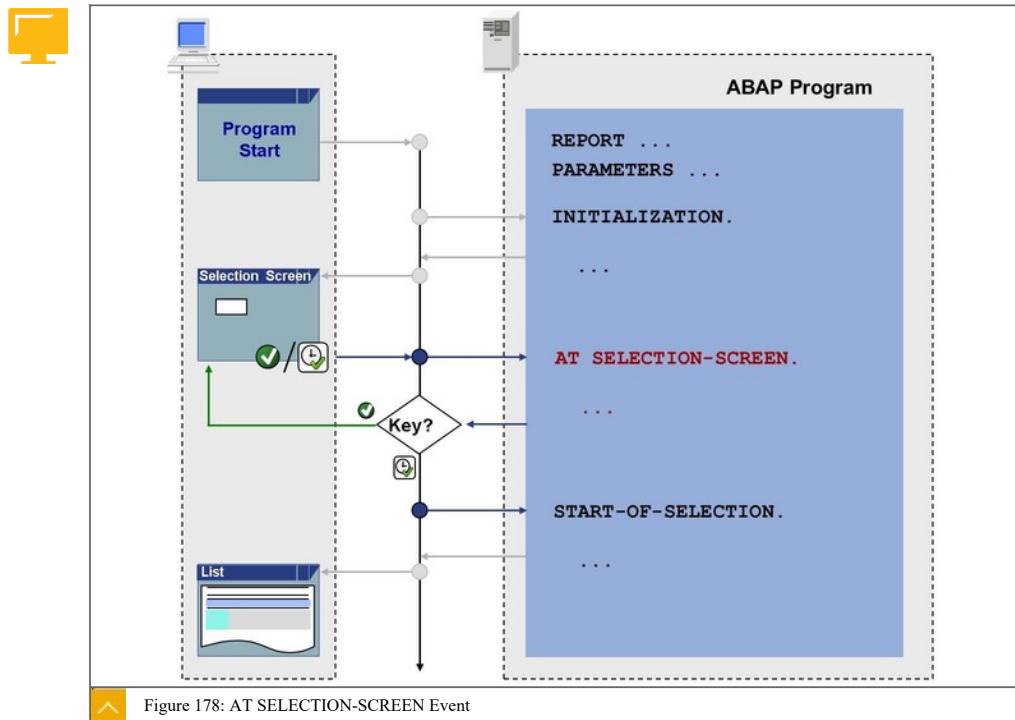
#### The INITIALIZATION Event



The example illustrated here contains a selection screen with an input field for a date. Under normal circumstances, the current date should appear as the default value (DEFAULT sy-datum). However, under certain conditions (IF), the date of the same weekday of the following week ( $pa\_date = pa\_date + 7.$ ) is to be displayed as the default value.

The figure also illustrates how the runtime system reacts when the processing block is missing. There are no statements executed for the corresponding event, and the next event is triggered.

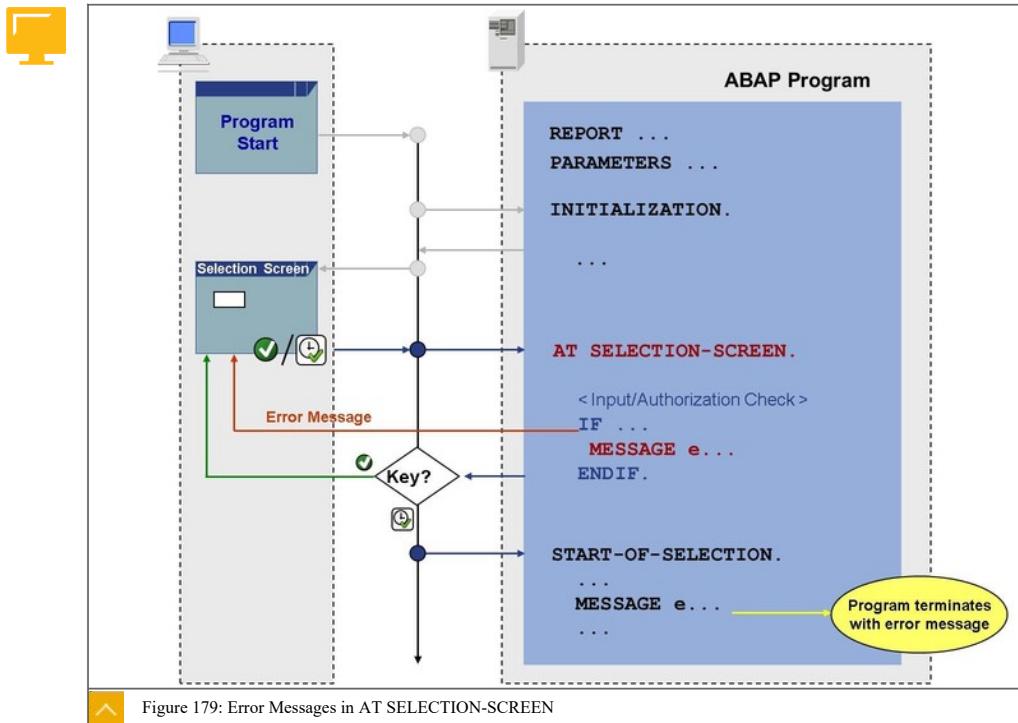
#### AT SELECTION-SCREEN Event



From the selection screen, the AT SELECTION-SCREEN event is triggered both by choosing Enter as well as F8 (Execute). After the corresponding processing block has been processed, if the user chooses the Execute button, the following START-OF-SELECTION event is triggered and the processing starts.

However, if the user chooses Enter, the system again displays the selection screen.

## Error Messages in AT SELECTION-SCREEN



Immediately before processing the AT SELECTION-SCREEN block, the system transfers user entries to the corresponding variables in the program. Hence, it makes sense to implement required input and authorization checks in this processing block. If the check result is negative, you can simply send out an error message to the user and have the selection screen displayed again. The user then has the chance to make new entries.

In contrast, if an error message is executed in the START-OF-SELECTION event, a program termination occurs.

Syntax Example – Authorization Check with Error Dialog



```
...
PARAMETERS pa_car TYPE s_carr_id.

CONSTANTS gc_actvt_display TYPE activ_auth VALUE '03'.

* Event processed after leaving the selection screen

AT SELECTION-SCREEN.

TRY.
    CALL METHOD cl_bc400_flightmodel=>check_authority
        EXPORTING
            iv_carrid = pa_car
            iv_activity = gc_actvt_display.
    CATCH cx_bc400_no_auth .
        * Show selection screen again with error message
        MESSAGE e046(bc400) WITH pa_car.
    ENDTRY.

START-OF-SELECTION.
...
```

Figure 180: Syntax Example – Authorization Check with Error Dialog

The figure illustrates a simple example program with an authorization check and an error dialog on the selection screen.

Additional information can be found in the keyword documentation for AT SELECTION-SCREEN.



LESSON SUMMARY

You should now be able to:

- Implement the events of ABAP reports

## Unit 7

### Learning Assessment

1. Which of the following can you use to adapt the default list interface to your own needs?

Choose the correct answer.

- A WRITE statement
- B Menu Painter
- C Pattern function

2. Which of the following are standard functionality on a selection screen?

Choose the correct answers.

- A Singular capability
- B Type check
- C Variants
- D Value entry

3. Which of the following statements is used for defining an input variable?

Choose the correct answer.

- A DATA
- B VALUE
- C PARAMETERS
- D DEFAULT

4. Which of the following are the characteristics of an event block?

Choose the correct answers.

- A Ends by beginning the next processing block
- B Can be nested
- C Existence not absolutely necessary
- D Sequence of event blocks important

## Unit 7

### Learning Assessment - Answers

1. Which of the following can you use to adapt the default list interface to your own needs?

Choose the correct answer.

- A WRITE statement  
 B Menu Painter  
 C Pattern function

You are correct! You can use the Menu Painter to adapt the default list interface to your own needs. Read more in the lesson, Implementing ABAP Lists, Task: Standard List Functions, in the course BC400 (Unit 7, Lesson 1) or TAW10 Part I (Unit 13, Lesson 1).

2. Which of the following are standard functionality on a selection screen?

Choose the correct answers.

- A Singular capability  
 B Type check  
 C Variants  
 D Value entry

You are correct! The selection screen has the following standard functions: Texts in several languages, Automatic type checking, Complex selections for intervals, comparative conditions, and patterns, Field documentation for input fields, Search help associated with the Dictionary type, Saving of selection screens as variants. Read more in the lesson, Implementing Selection Screens Task Standard Functions of the Selection Screen, in the course BC400 (Unit 7, Lesson 2) or TAW10 Part I (Unit 13, Lesson 2).

3. Which of the following statements is used for defining an input variable?

Choose the correct answer.

A DATA

B VALUE

C PARAMETERS

D DEFAULT

You are correct! An input variable is defined with a PARAMETERS statement instead of a DATA statement. The definition of such an input variable creates a variable in the system and generates a selection screen with a corresponding input option. Read more in the lesson, Implementing Selection Screens, Task: Single-Value Input, in the course BC400 (Unit 7, Lesson 2) or TAW10 Part I (Unit 13, Lesson 2).

4. Which of the following are the characteristics of an event block?

Choose the correct answers.

A Ends by beginning the next processing block

B Can be nested

C Existence not absolutely necessary

D Sequence of event blocks important

You are correct! Characteristics of an Event Block are: Introduced with an event keyword. It ends by the beginning of the next processing block, Cannot be nested, Existence is not absolutely necessary, Sequence of event blocks is unimportant. Read more in the lesson, Implementing Events of ABAP Reports, Task: Event Block Characteristics, in the course BC400 (Unit 7, Lesson 3) or TAW10 Part I (Unit 13, Lesson 3).

## UNIT 8

# Screens

### Lesson 1

Creating Screens

255

### Lesson 2

Creating Input and Output Fields

274

### Lesson 3

Implementing Data Transport

277

### UNIT OBJECTIVES

- Create screens and understand screen processing
- Create input and output fields on screens
- Implement data transport on screens

## Unit 8

### Lesson 1

# Creating Screens

#### LESSON OVERVIEW

This lesson explains how to design and program simple screens with input and output fields and buttons.

#### Business Example

You want to develop a program that allows data to be entered and output to be displayed on screens. Furthermore, you need to create buttons on screens to address corresponding functions in the program. For this reason, you require the following knowledge:

- An understanding of the attributes and benefits of screens
- An understanding of how to implement simple screens with input and output fields and link them to a dialog application
- An understanding of the concept and implementation of program-internal processing for screen calls

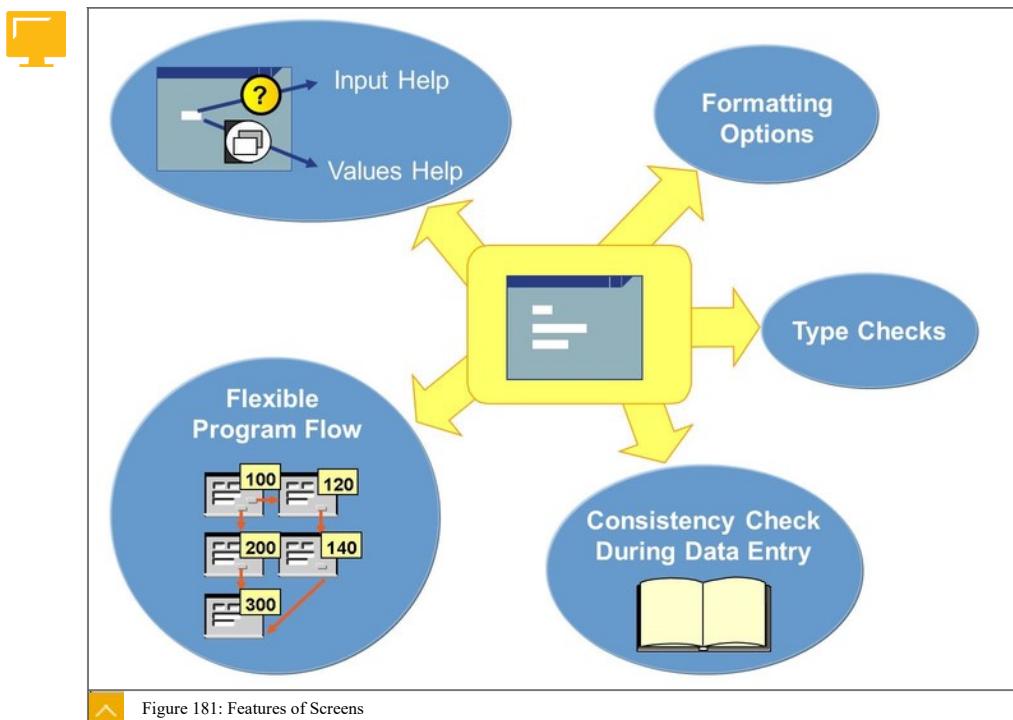


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create screens and understand screen processing

## ABAP Screens



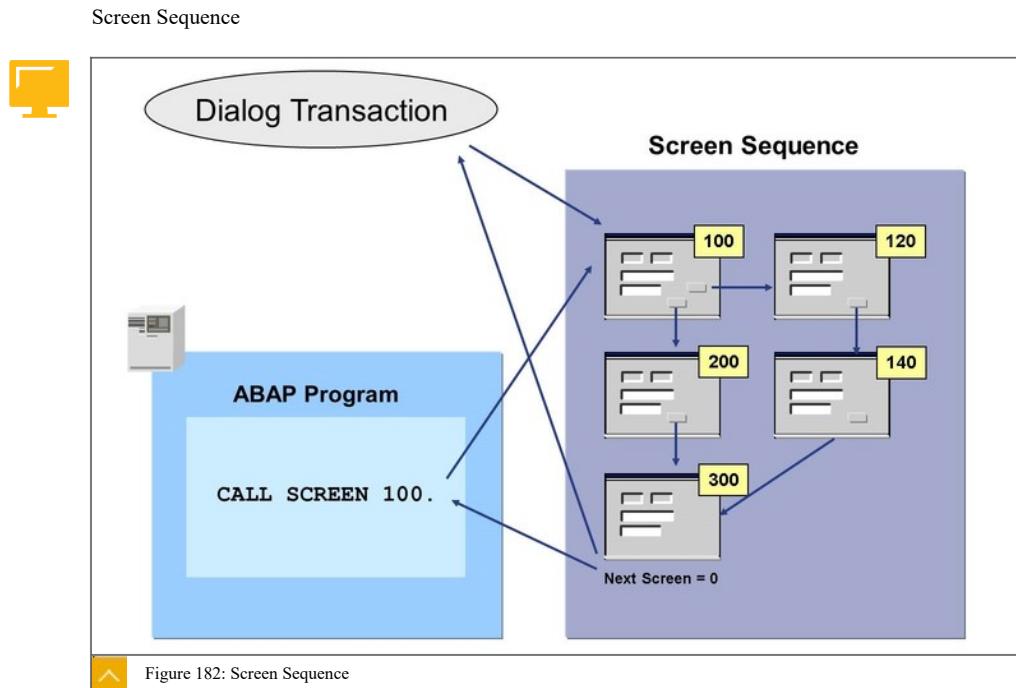
A screen not only consists of its layout with input/output fields, buttons, and other screen elements, but also processing logic (source code excerpts that are processed as the preprocessing or postprocessing of the screen display).

The fact that the ABAP Dictionary is integrated into the SAP system means that automatic consistency checks for screen input fields are provided. These checks include type checks, foreign key checks, and fixed value checks. All of these checks are automatically supplied with information from the ABAP Dictionary.

Other program-specific checks complement the checks described earlier. There are techniques available for screens that allow you to control the order in which checks are performed and, if errors occur, to make the fields input-ready again.

You can design screen layouts very flexibly. A screen layout can contain input fields, output fields, radio buttons, check fields, and buttons.

Screens have the same formatting options as lists and selection screens. The system formats the fixed point numbers and dates according to the settings specified in the user master record. The time is set to HH:MM:SS. Sums of money are formatted according to the content of the currency field, and physical measures (lengths, weights, and so on) are formatted according to the content of a unit field.

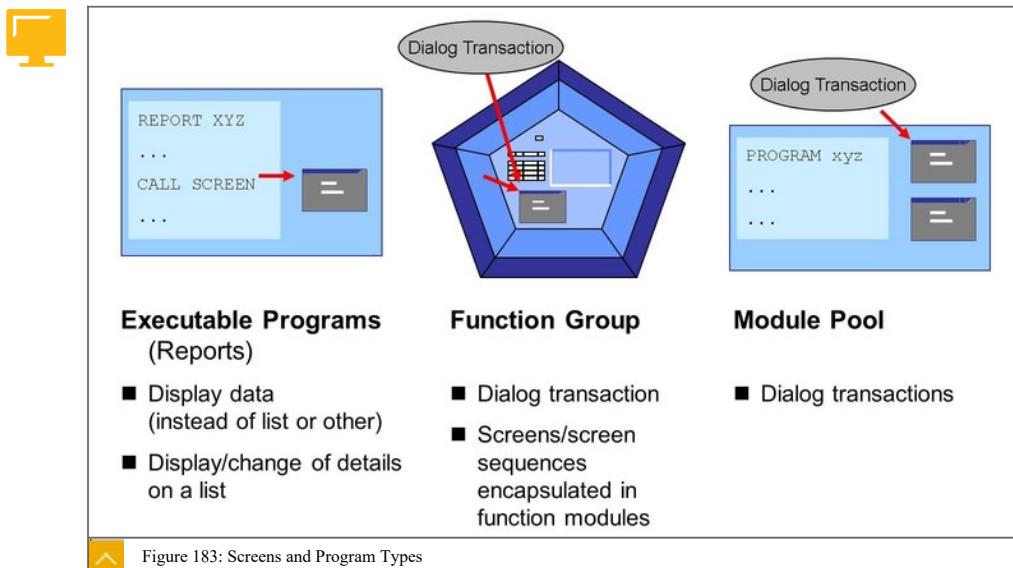


#### Screen Sequence Start-Up

- Start a screen sequence by performing one of the following actions:
  - Call the first screen (CALL SCREEN statement) from a processing block in your program.
  - Create a transaction that references the program and the first screen (dialog transaction).

After a screen is processed, the screen sequence, whether defined statically or dynamically, is processed. The symbolic next screen 0 either returns processing to the point where the screen was called or ends the dialog transaction.

## Screens and Program Types



## Screens and Program Types

The following program types may contain screens:

- **Executable program (report)**

Screens are used in executable programs to display data in addition to the list output, or to replace the list output completely. In addition, you can also use screens to enter and change data in the list. For the purpose of reusability and data encapsulation, however, SAP recommends that you use screens from function groups rather than create screens directly in reports.

- **Function group**

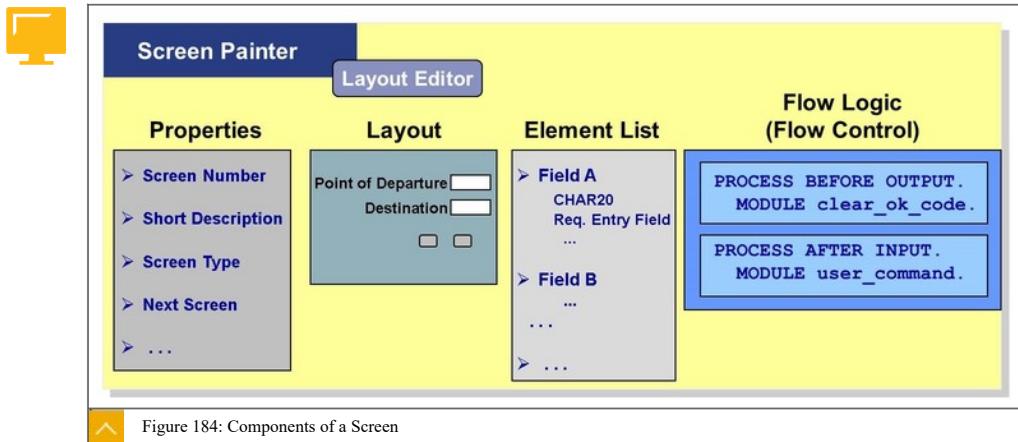
Screens in function groups can be addressed using dialog transactions. You can also start such screens from the source code of a function module using the CALL SCREEN statement. SAP recommends that you use function groups to create screens and screen sequences easily available for reuse.

- **Module pool**

Screens in module pools can only be addressed using dialog transactions. In contrast to screens in function groups, you cannot encapsulate screens in module pools, nor can you provide them with a well-defined external interface.

For this reason, SAP recommends that you use function groups rather than create new module pools, even if you have not (yet) planned to address the screen using function modules.

## Components of a Screen



## Components of a Screen

These are the components of a screen:

- **Properties (attributes)**

Properties contain, for example, a four-digit number as the screen name, a short text, and information about the screen type (for example, **Normal** for full screen size) and the default next screen.

- **Layout**

Layouts contain input or output fields, texts, buttons, and other elements that you can place on your screen. Such elements are called screen elements.

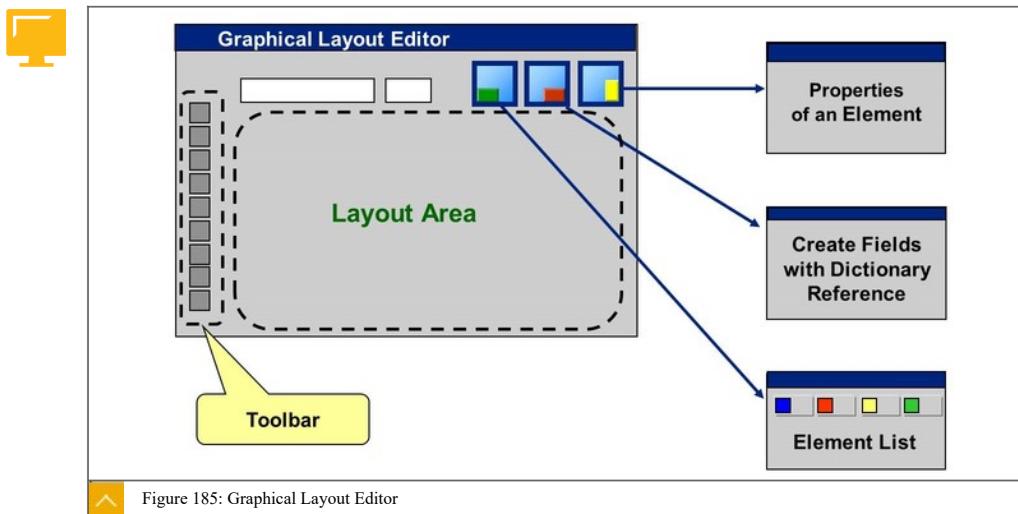
- **Element list**

The element list of a screen contains all screen elements and their attributes, such as position, size, data type, and so on.

- **Flow logic**

The flow logic of a screen consists of Process Before Output (PBO) and Process After Input (PAI). PBO contains references to processing blocks (PBO modules) that are processed in preparation for the screen display (for example, data selection) before the screen is sent. PAI contains references to processing blocks (PAI modules) that are processed as a reaction to user input and actions (for example, save data).

Graphical Layout Editor



Using the graphical **Screen Painter**, you can design the layout of the screen (toolbar).

#### Graphical Layout Editor

You can also perform the following maintenance functions using the buttons depicted in the figure:

- **Maintain element attribute s**

With this function, you view and modify all attributes for the selected screen element on a separate dialog box.

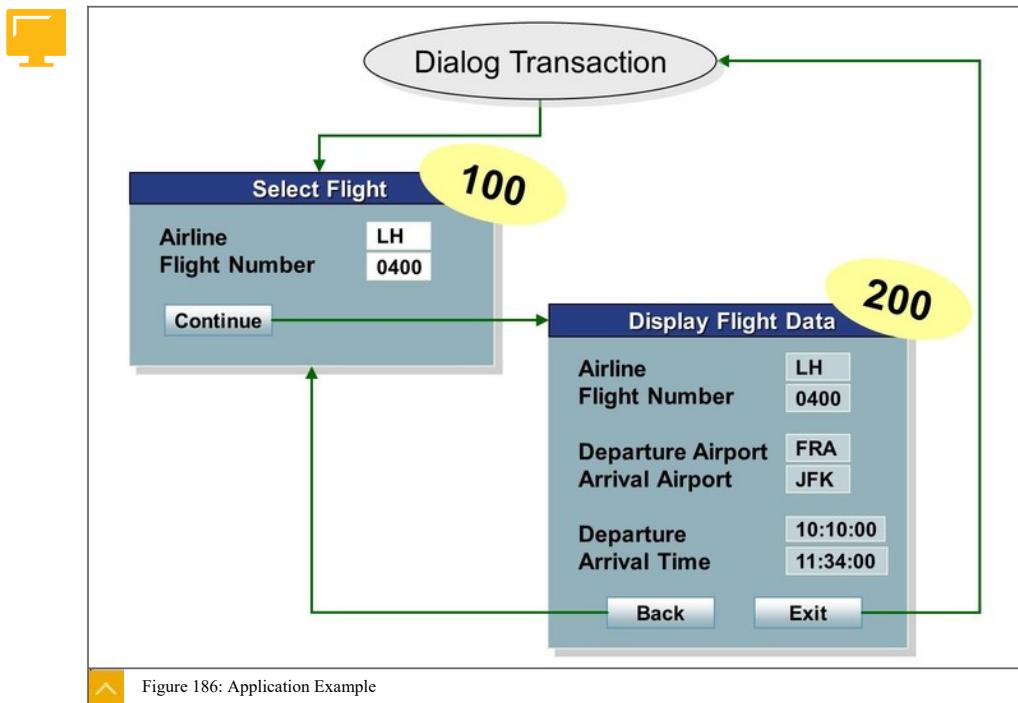
- **Get Dictionary or program fields**

With this function, you create screen fields with reference to fields from Dictionary structures or fields defined within the program.

- **Show element list**

With this function, you show and maintain all available screen elements with their attributes on a list.

## Application Example



You need to develop a program in several steps that will enable the system to display data for individual flight connections.

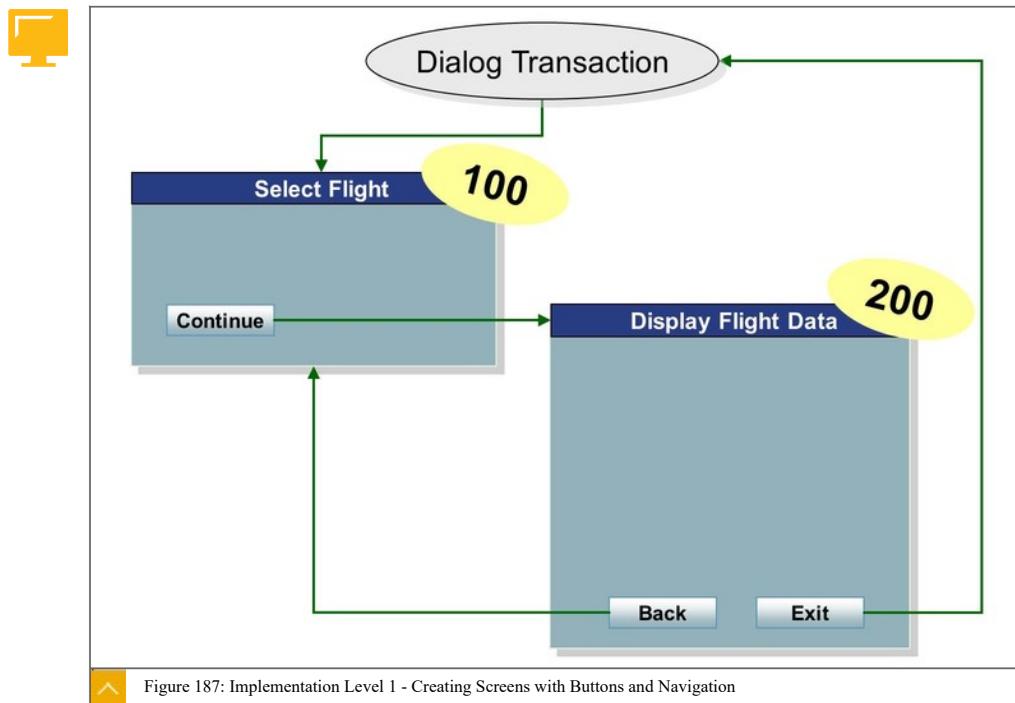
The program comprises a sequence of two screens. On the first screen, the user chooses a flight connection. On the second screen, the details for this connection are displayed.

A dialog transaction starts the screen sequence. If the user chooses **Continue** on the first screen, the entries are processed, the data is read from the database, and the system navigates to the second screen.

The second screen displays the flight connection data. If the user chooses **Back**, the program returns the user to the first screen, where the user can select another flight connection.

If the user chooses **Exit**, the program exits the screen sequence. In this case, the program cancels the dialog transaction.

## ABAP Screen Creation and Navigation



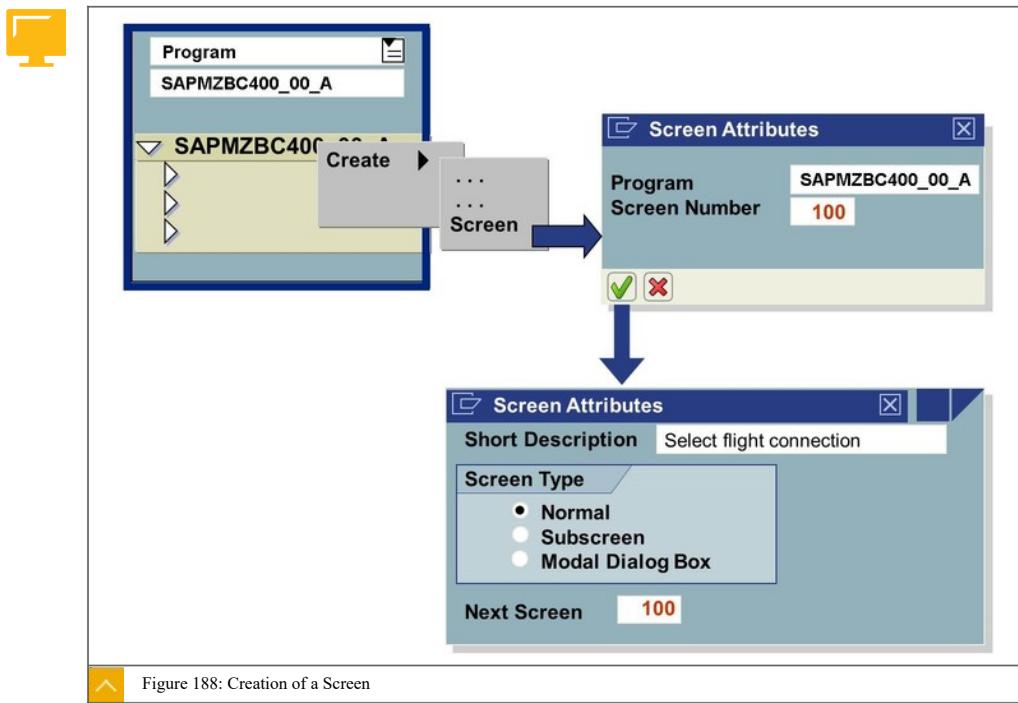
To begin developing your program, you need to create two screens on which buttons are located. You need to implement the program so that the user selects a button to either navigate to the other screen or exit the screen sequence. You create a dialog transaction to start the screen sequence.

## Implementation Level 1 – Creating Screens with Buttons and Navigation



- To accomplish these tasks, you need to understand the following concepts:
  - Creation of screens
  - Flow logic of a screen in PBO and PAI event blocks
  - PBO and PAI modules as processing blocks for the corresponding events
  - Implementation of buttons and evaluation of user actions
  - Creation of dialog transactions

## Creation of a Screen



## Screen Creation

- Use one of the following functions to create a screen:

- The Object Navigator

Create a new program object ( **Screen** ) for your program on the object list in the navigation area using the context menu for your program, as illustrated in the figure.

- Forward navigation from the ABAP Editor

Double-click the screen number in the ABAP Editor to create a screen (in the CALL SCREEN nnnn statement). The system opens the **Screen Painter** for maintaining the new screen.

When you create a screen, the system asks you to enter screen attributes. Enter a short description of the screen, select screen type **Normal**, and enter the number of the subsequent screen.

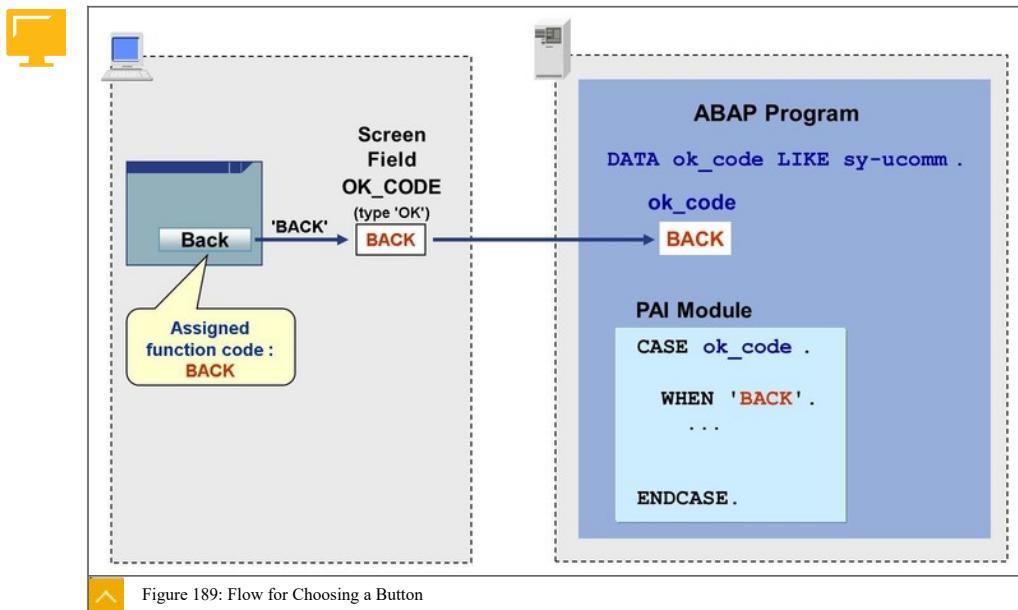
If you enter **0** for the subsequent screen, the system processes the screen completely, and then returns to processing at the point where the screen was called.



**Caution:**

Because zero ( **0** ) is the initial value of the field, it is hidden when the system displays the screen attributes.

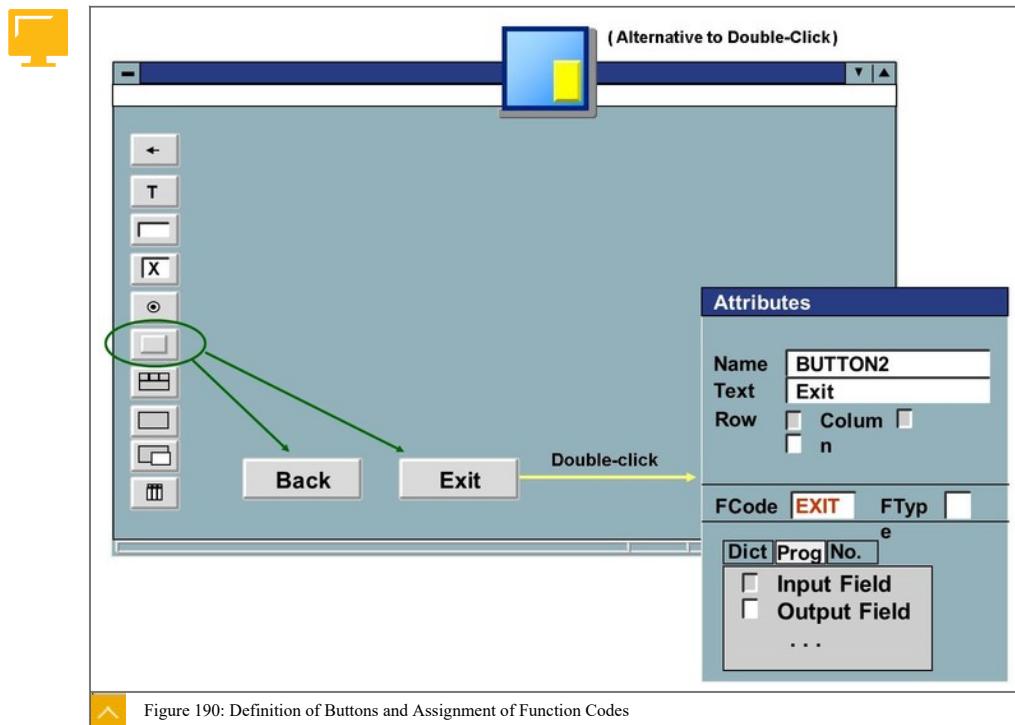
Flow for Choosing a Button



If the user chooses a button, the runtime system copies the assigned function code to a special screen field of the OK type. This screen field is typically called OK\_CODE.

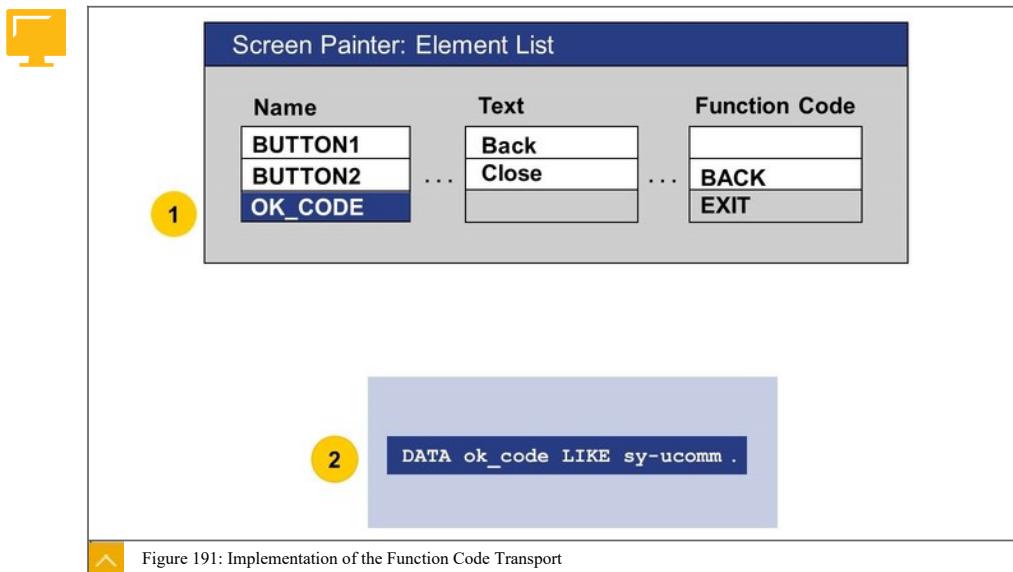
The system transports the content of this special screen field if there is a data object of the same name within the program. Following that, the system triggers the PAI processing, which processes the user action by transferring the function code to the program and performing appropriate processing.

## Definition of Buttons and Assignment of Function Codes



The figure illustrates how you define buttons in the graphical Screen Painter. To assign a name and a function code to each button, you need to maintain the field attributes.

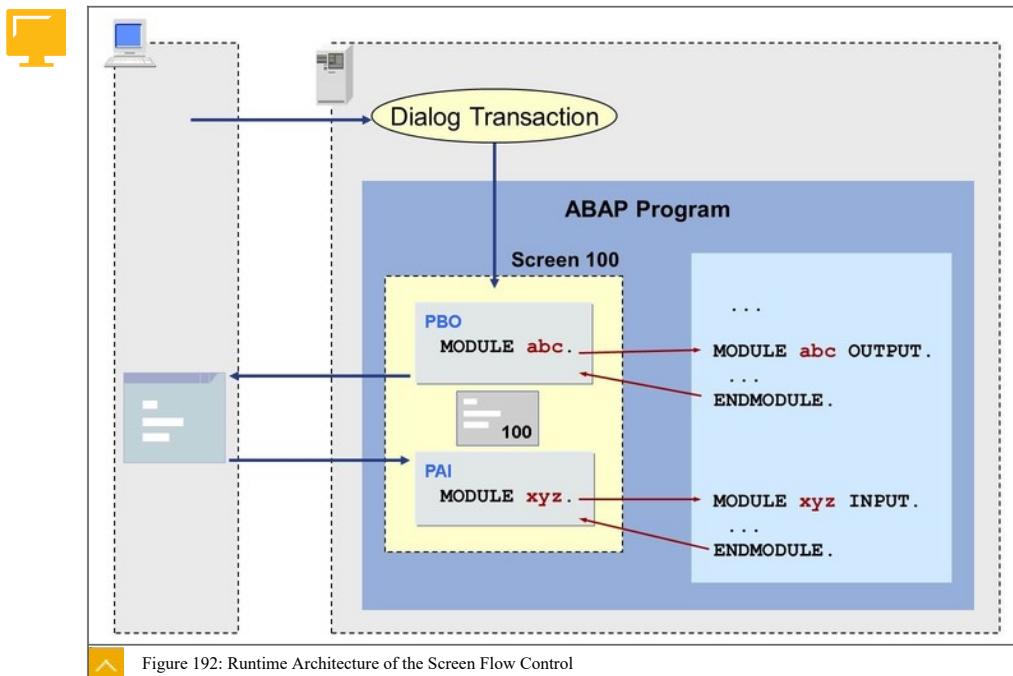
## Implementation of the Function Code Transport



The command field is the special screen field with which the system transports the respective function code to the program. The command field is available on the screen by default. To use the command field, you must enter a name for it. Normally, the name OK\_CODE is used.

Then declare a program-internal data object with the same name as the command field. Use system field SY-UCOMM to set the data type, as illustrated in the figure.

## Runtime Architecture of the Screen Flow Control



Use a dialog transaction to trigger screen processing.

#### Screen Processing



- Screen processing consists of the following steps:
  1. PBO processing
  2. Field transport from the program to the screen
  3. Screen display
  4. Field transport from the screen to the program
  5. PAI processing

##### **1. PBO processing**

In preparation for the screen display, the PBO modules listed in the PBO block are processed successively.

##### **2. Field transport from the program to the screen**

After PBO has been completed, the data to be displayed is transported to the screen fields.

##### **3. Screen display**

The screen to which values have been assigned is sent to the presentation server (SAP GUI) and shown to the user.

#### 4. Field transport from the screen to the program

Each user action on the screen triggers the transport of the screen field contents back to the program.

#### 5. PAI processing

As a reaction to the user action, the modules listed in PAI are processed successively.

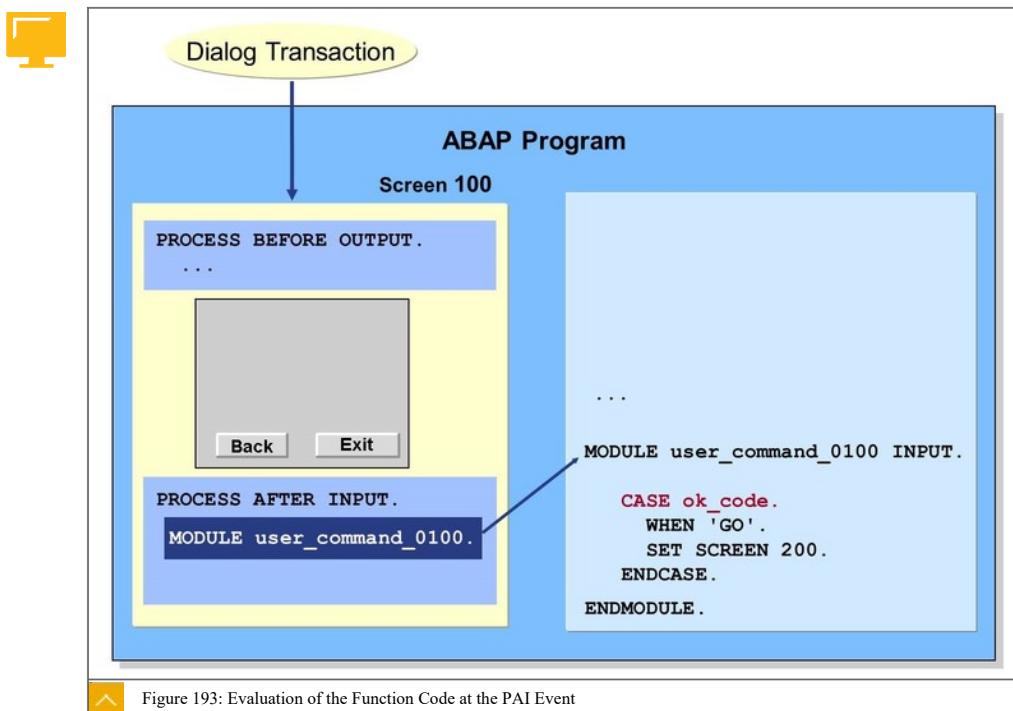
Modules are source code blocks without interface. They are enclosed by the ABAP statements MODULE and ENDMODULE. For each subfunction in PBO or PAI, implement a corresponding module.



**Caution:**

The flow logic of the screen (PBO or PAI) contains references to modules, which are implemented in the program and consist of ABAP statements. You must not enter the ABAP source code directly in the flow logic.

Evaluation of the Function Code at the PAI Event



The figure illustrates the program's reaction to the user action. The system evaluates the corresponding function code that is transferred to the OK\_CODE field immediately before PAI processing.

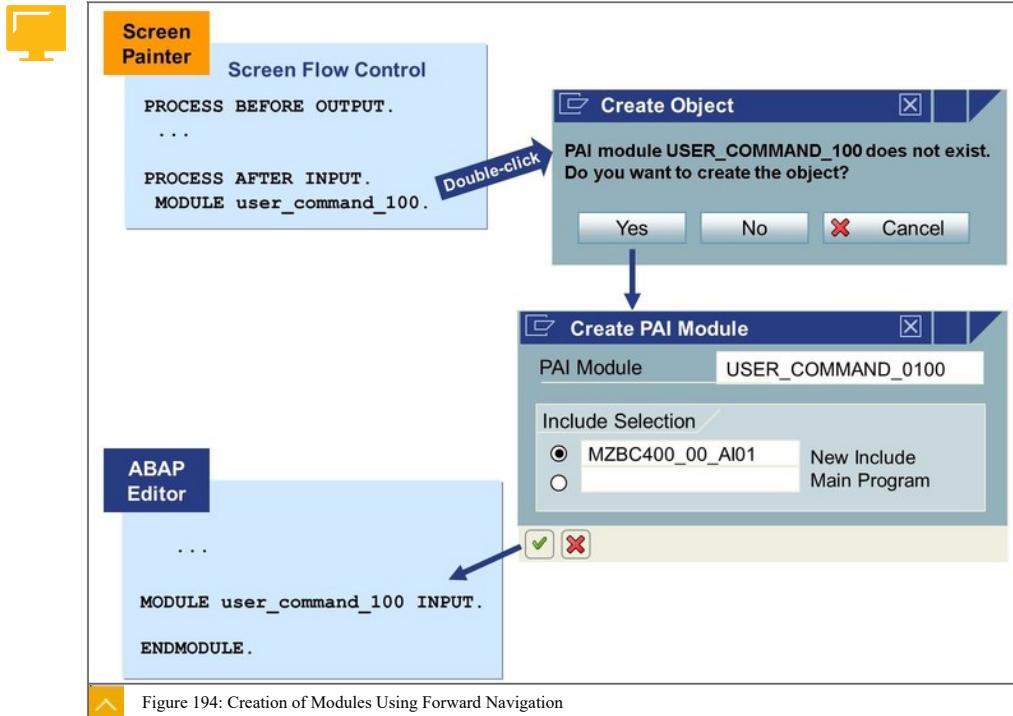
Usually, the module with the main processing in PAI is called USER\_COMMAND\_nnnn, where nnnn represents the screen number.



## Hint:

In a module, you can access all global data objects of the program. Variables that you define within the module are always global.

## Creation of Modules Using Forward Navigation



## Module Creation

- You can create modules in the following ways:
  - Use forward navigation from the flow logic  
To create a module, double-click the corresponding module reference, as illustrated in the figure.
  - Use the navigation area of the Object Navigator  
To create a module from the object list in the navigation area, use the context menu of the program. Include a corresponding reference to the newly created module in the flow logic of your screen.

A module can be called during runtime from the flow logic of several screens (reusability).



Note:

Modules starting with MODULE ... OUTPUT are PBO modules and can only be called by the PBO of a screen. Accordingly, PAI modules, which start with MODULE ... INPUT, can only be called by PAI.

Next Screen Same Screen (Effect)

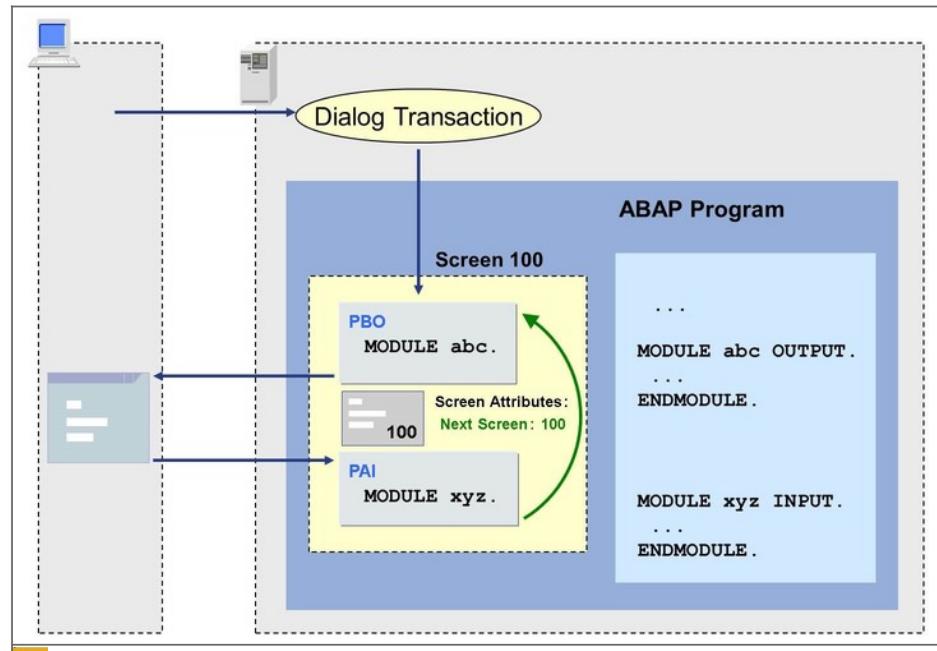
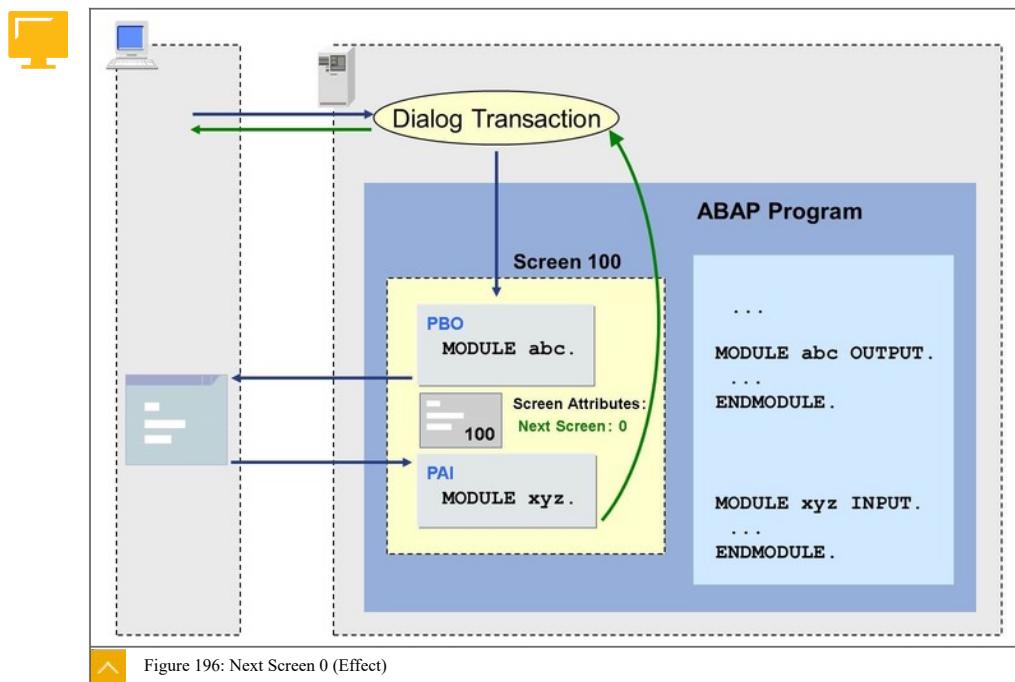


Figure 195: Next Screen Same Screen (Effect)

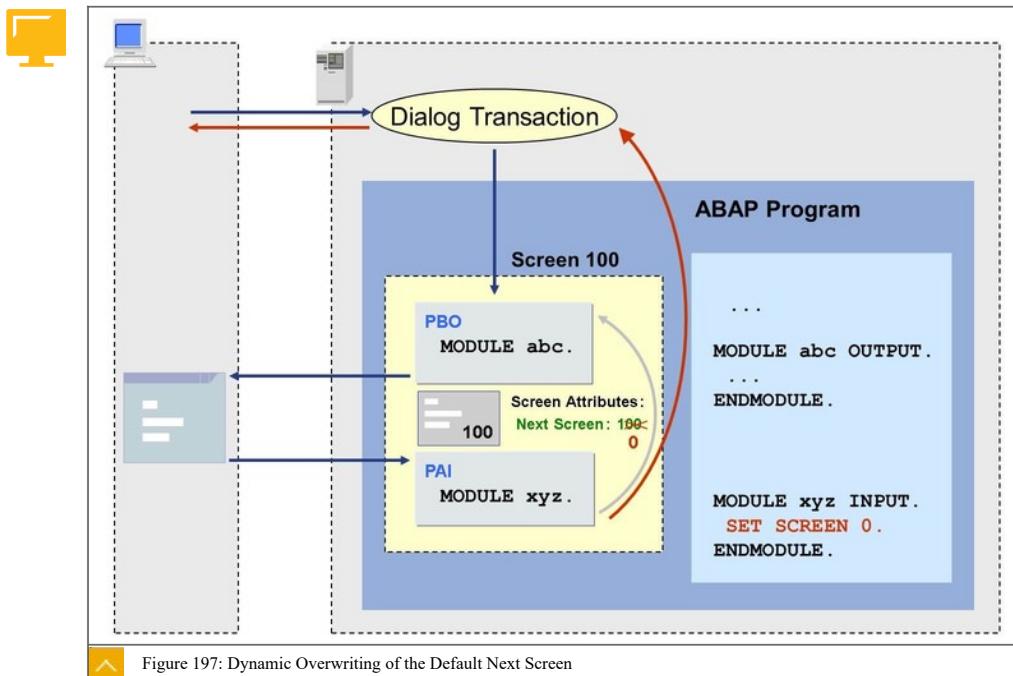
If you assign a screen its own screen number as the next screen, the system processes it again after it is closed.

## Next Screen 0 (Effect)



If you enter screen attribute next screen = 0, the system processes the entries on your screen and then returns to processing at the point where the screen call is set.

## Dynamic Overwriting of the Default Next Screen



To dynamically overwrite the default next screen specified in the screen attributes, use the ABAP statement `SET SCREEN` from a module (typically, a PAI module), as illustrated in the figure. You can use this option to implement the following SAP standard: when you press `ENTER`, you return to the same screen; only certain buttons take you to other screens. To implement this standard for your screen, enter its own screen number as the default next screen and branch to other screens (using the `SET SCREEN` command) from PAI only if you press certain buttons.

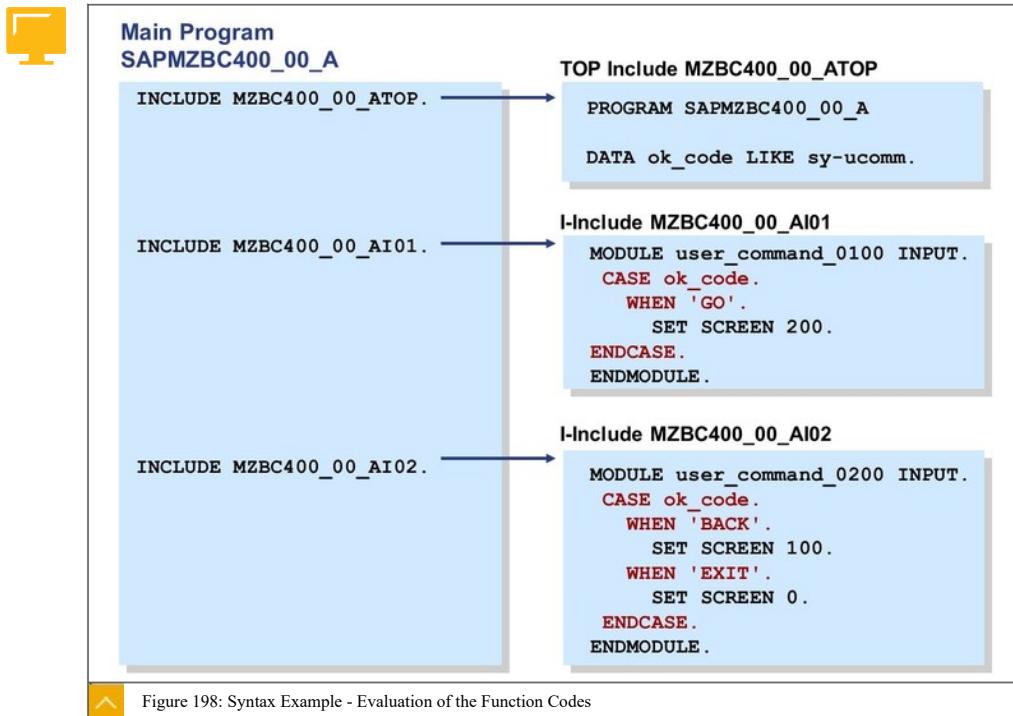


## Note:

By default, choosing `Enter` does not insert a function code into the command field of the screen. Therefore, the system transports the initial value of the field (space) to the program as the function code.

If the system processes the same screen again, it runs through all PBO modules again. If you decide to fill the TABLES structure in a PBO module, ensure that data changes made by the user are not overwritten on the screen if the module is called twice.

## Syntax Example – Evaluation of the Function Codes



## Button Behavior

- For this scenario, choose one button on the first screen and two on the second, and set them to behave as follows:
  - If the user chooses Continue (function code GO) on the first screen, the next screen is set dynamically to 200 to get to the second screen.
  - If the user chooses Back (function code BACK) on the second screen, the next screen is set dynamically to 100 to return to the first screen.
  - If the user chooses Exit (function code EXIT), the next screen, which is automatically set to zero, returns the user to the call point of the screen and terminates the dialog transaction.



## LESSON SUMMARY

You should now be able to:

- Create screens and understand screen processing

## Unit 8

### Lesson 2

## Creating Input and Output Fields

### LESSON OVERVIEW

This lesson explains how to design and program simple screens with input and output fields.

#### Business Example

You want to develop a program that allows data to be entered and output to be displayed on screen layouts. For this reason, you require the following knowledge:

- An understanding of how to create input and output fields on screens

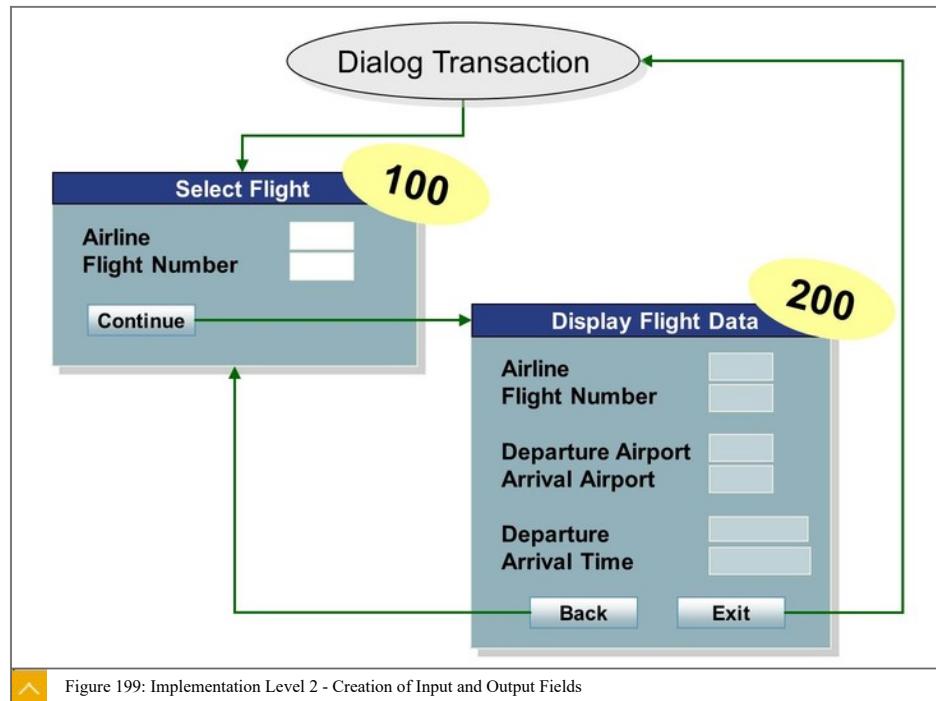


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create input and output fields on screens

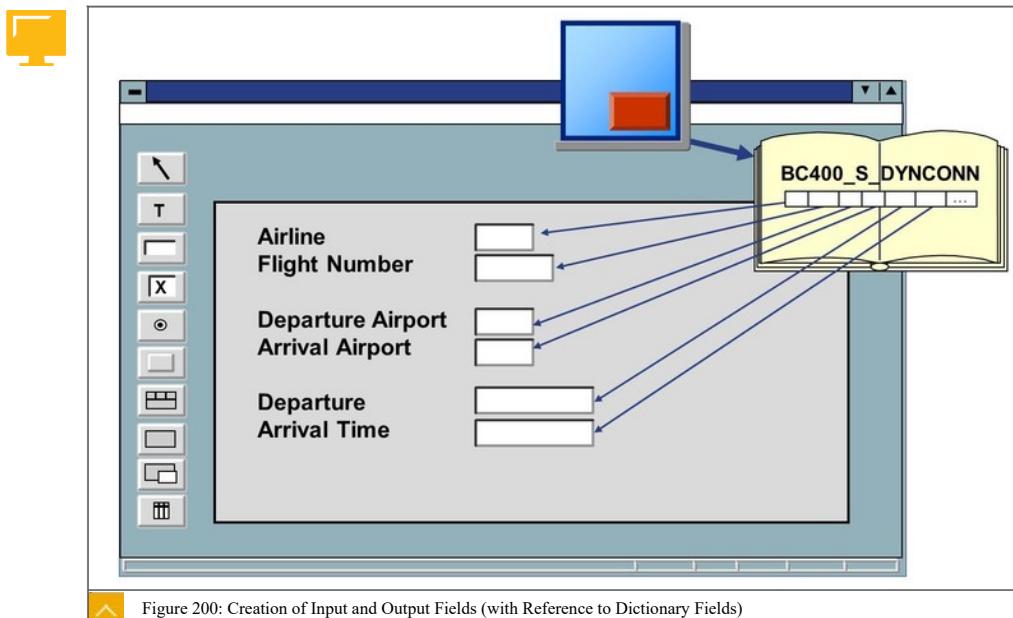
### Input and Output Fields on a Screen



At this point in your program's development, you need to create input and output fields on both screens. The first screen needs to contain **Airline and Flight Number** as ready-for-input

fields. The second screen needs to contain only display fields. In addition to the Airline and Flight Number fields, the second screen also needs to include the Departure Airport , Destination Airport , Departure Time , and Arrival Time fields.

#### Creation of Input and Output Fields (with Reference to Dictionary Fields)



#### Assignment Options of Field Attributes

Use one of the following options to implement the assignment of field attributes when you create screen fields:

##### **Get from Dictionary**

Use this option to copy the field type, as well as the field attributes, of a Dictionary field when creating a screen field. All Dictionary information, then, is available for the screen field, including semantic information about the corresponding data element and foreign key dependencies. The name of the Dictionary field is used as the field name.

##### **Get from program**

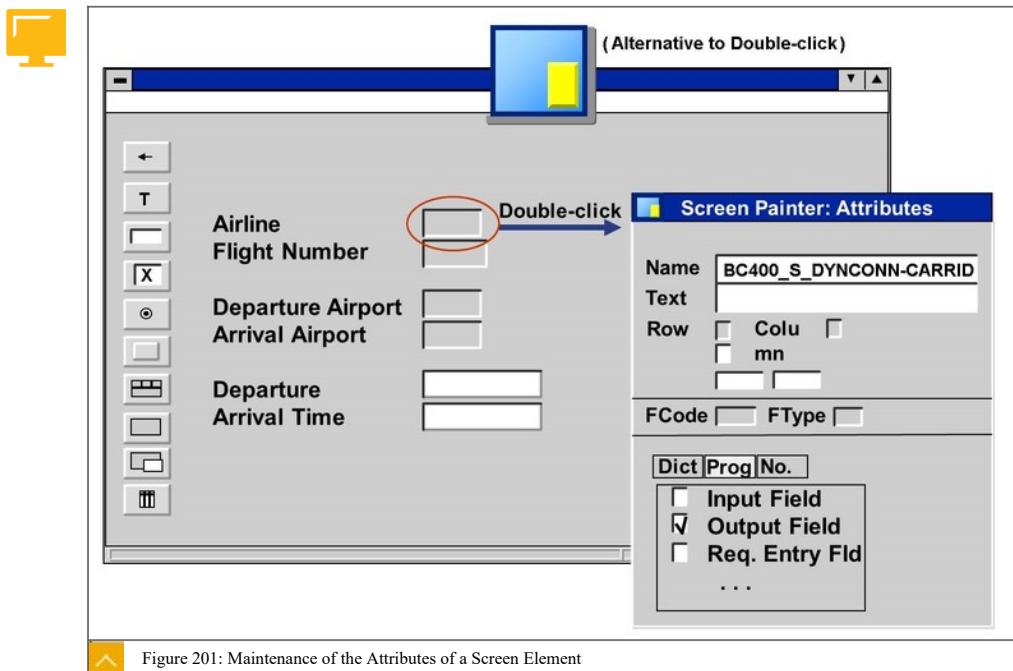
Use this option to copy the field attributes of a program-internal data object for a screen field. To do so, a generated version of the program must be available (the generated version is created automatically at the time of activation). The name of the data object is used as the field name.

Use the graphical layout editor to define further screen elements, such as texts, boxes, radio buttons, and checkboxes. Click the required screen element in the toolbar to select it and then position the element in the screen maintenance area.

To delete a screen element, select it with the mouse and then choose **Delete**.

To move a screen element to a new position, drag it and drop it in the desired position.

## Maintenance of the Attributes of a Screen Element



To maintain the attributes of a screen element, double-click it. The system displays its attributes in an additional window for maintenance. (Instead of double-clicking, you can also select the element, and then choose the **Attributes** button.)

To make input mandatory, assign the **Required** attribute to a screen field. At runtime, the system marks the field accordingly if it is blank. If the user does not fill out the field, an error dialog follows any user action. After this dialog, the input fields become ready for input again.



## LESSON SUMMARY

You should now be able to:

- Create input and output fields on screens

## Unit 8

### Lesson 3

## Implementing Data Transport

### LESSON OVERVIEW

This lesson discusses the concept and implementation of program-internal processing for screen calls.

### Business Example

You want to develop a program that shows how to transfer data between the program and the screens.

For this reason, you require the following knowledge:

- An understanding of how to implement data transport on screens

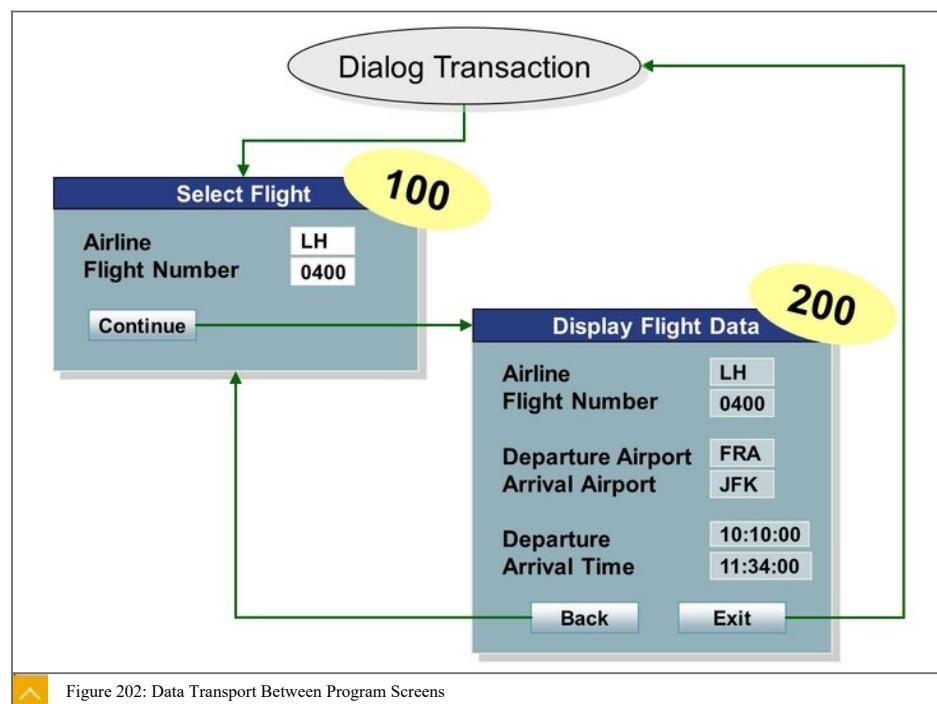


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

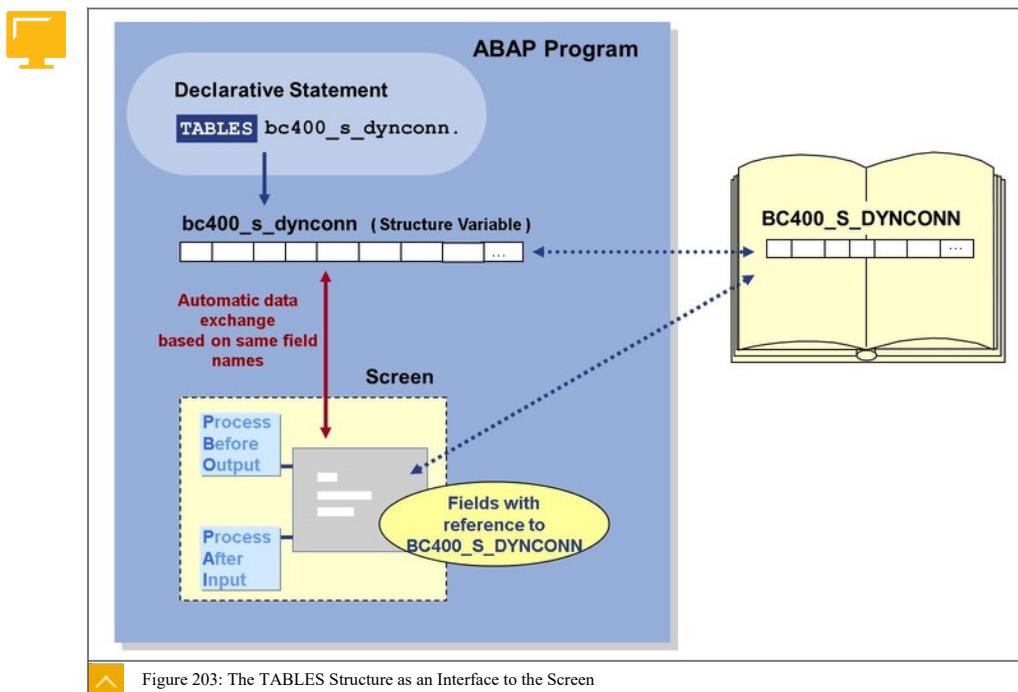
- Implement data transport on screens

### Data Transport Between Program Screens



At this point in your program's development, the user can select a flight connection on the first screen and navigate to the second screen. However, only the Airline and Flight Number fields on the second screen contain a value, because they have exactly the same name as the input fields on the first screen. You must now program the data transport, both from the first screen to the program and from the program to the second screen, to enable the program to process user entries and display relevant information on the second screen.

The TABLES Structure as an Interface to the Screen

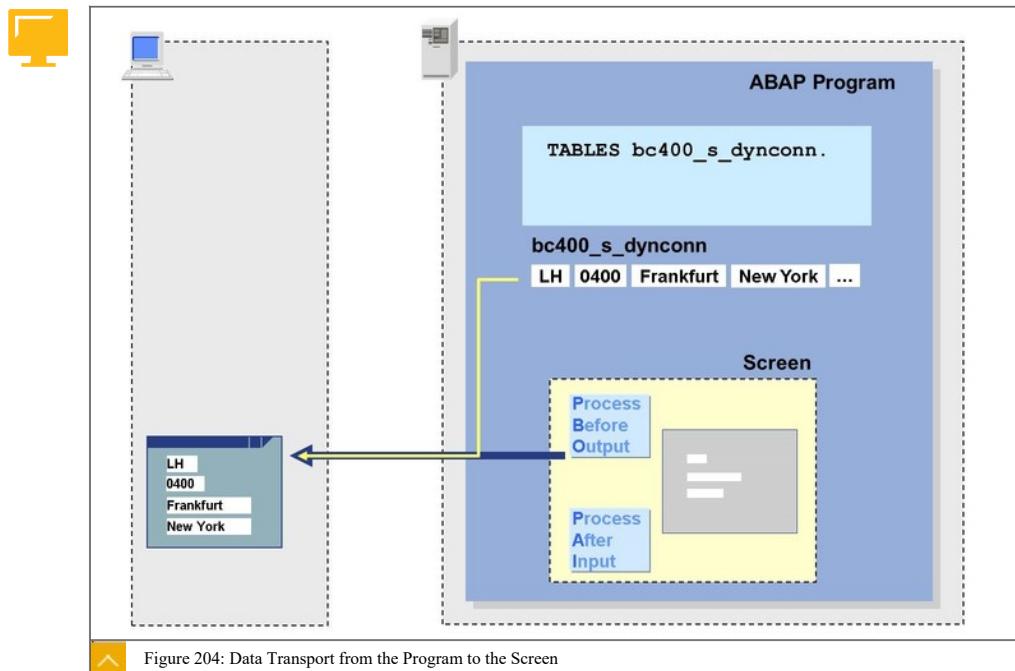


You use the TABLES statement to define a structure variable of the same type and name with reference to the specified Dictionary structure (for example, a transparent table) within the program. This structure variable then serves as the interface between the program and the screen.

If the screen fields and the TABLES statement refer to the same structure type, then the system exchanges data in their fields on the basis of the fields having the same names: after PBO, from the TABLES structure to the screen fields; and before PAI, from the screen fields to the TABLES structure.

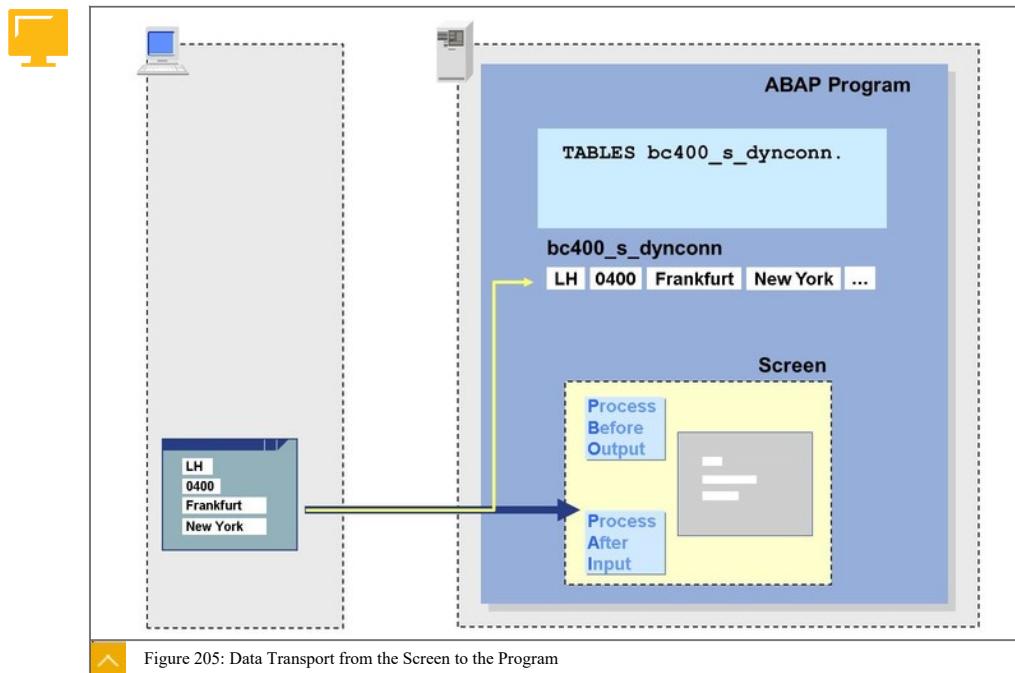
A special structure that contains the fields that are to be displayed on the screens of an application is normally created in the ABAP Dictionary. This structure can also contain fields from several database tables. Define fields on the screen with reference to this structure, and implement an interface structure within the program using the corresponding TABLES statement. This way, you still have access to a clear interface structure for data exchange between the program and the screen, even though the structure contains fields from various tables.

## Data Transport from the Program to the Screen



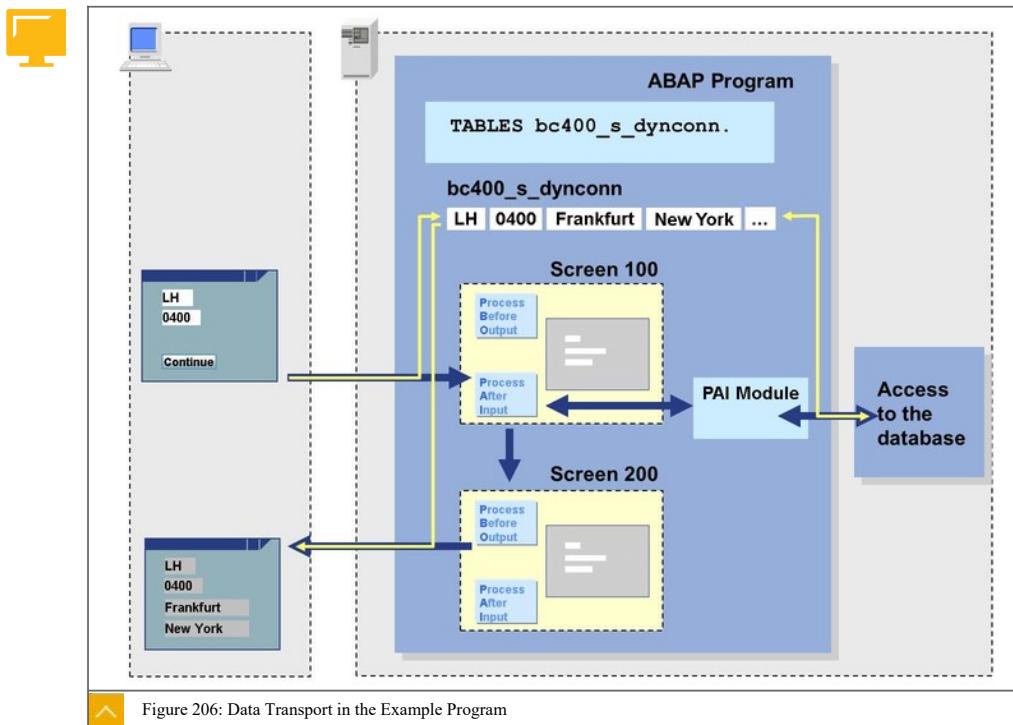
After processing the PBO event, the system copies the field content of the TABLES structure within the program into the screen fields with the same names. The system then sends the screen to the presentation server. Data transport occurs in this sequence because the data for the screen display in the TABLES structure is prepared in PBO.

## Data Transport from the Screen to the Program



After the user completes actions on the screen, the system transports content from the screen fields into the TABLES structure fields with the same names before the processing of the PAI event. Data transport occurs in this sequence because user entries are supposed to be processed further in PAI. Therefore, they must be available in the program at that time.

## Data Transport in the Example Program



## Data Transport Process of the Example Program

The data transport process of the example program involves the following steps:

1. When the user completes an action on screen 100, the system transports the user entries to TABLES structure BC400\_S\_DYNCONN within the program.
2. The system processes PAI module USER\_COMMAND\_0100.
3. Depending on the function code, the program calls a function module for data retrieval in the PAI module. It transfers the user entries from the TABLES structure to the function module in the process. The program then places the data to be displayed on screen 200 in the same TABLES structure.
4. After PAI for screen 100, the program navigates to screen 200.
5. After PBO for screen 200, the system transports the data from the TABLES structure to the screen fields.

## Syntax of the Example Program



```

MODULE user_command_0100 INPUT.

CASE ok_code.
  WHEN 'GO'.
    MOVE-CORRESPONDING bc400_s_dynconn TO gs_connection.

    CALL FUNCTION 'BC400_DDS_CONNECTION_GET'
      EXPORTING
        iv_carrid      = gs_connection-carrid
        iv_connid      = gs_connection-connid
      IMPORTING
        es_connection = gs_connection
      EXCEPTIONS
        no_data        = 1
        OTHERS         = 2.

    IF sy-subrc <> 0.
      MESSAGE e042(bc400) WITH gs_connection-carrid
                                gs_connection-connid.
    ELSE.
      MOVE-CORRESPONDING gs_connection TO bc400_s_dynconn.
      SET SCREEN 200.
    ENDIF.
  ENDCASE.

ENDMODULE.          " USER_COMMAND_0100 INPUT

```



Figure 207: Syntax of the Example Program

The program evaluates the function code in the PAI module USER\_COMMAND\_0100. If the user chooses Continue (function code GO), the program calls a function module for data retrieval.

The program transfers the user entries from the TABLES structure to the function module. To separate the interfaces to the screen and to the function module clearly, the program does not transfer the data directly from the TABLES structure to the function module. Instead, the data is copied from the TABLES structure to the data object that serves as the interface to the function module (global, structured data object gs\_connection, type BC400\_S\_CONNECTION).

After retrieving data successfully, the program copies the result back to the TABLES structure and triggers navigation to the display screen. If the function module terminates with an exception, a corresponding error message is sent. The system displays this error message directly on screen 100.



## Hint:

If error messages (message type “E”) are output during the processing of a PAI module, the system displays the relevant screen again immediately without processing PBO again. With the use of the corresponding programming techniques, screen fields can be ready for input again.



Hint:

In the example, the data is read at PAI for the input screen. Alternatively, data can also be retrieved in a PBO module for the display screen, or in other words, after navigation. Use the second option to make the reuse of the display screen easier.



#### LESSON SUMMARY

You should now be able to:

- Implement data transport on screens

## Unit 8

### Learning Assessment

1. Which of the following can you use to start screen processing?

Choose the correct answer.

- A Screen flow control setting
- B Dialog transaction
- C Create processing statement
- D Start screen processing setting

2. Which of the following should be assigned to a screen field to make input mandatory?

Choose the correct answer.

- A Insert
- B Required
- C Graphical layout
- D Delete

3. Which of the following statements is true regarding the implementation of data transport?

Choose the correct answer.

- A System field sy-ucomm serves as the interface for data transport between the program and the screen.
- B Data transport is performed when you enter values on the screen.
- C The TABLES statement serves as the interface for data transport between the program and the screen.

## Unit 8

### Learning Assessment - Answers

1. Which of the following can you use to start screen processing?

Choose the correct answer.

- A Screen flow control setting
- B Dialog transaction
- C Create processing statement
- D Start screen processing setting

You are correct! Use a dialog transaction to trigger screen processing. Read more in the lesson, Creating Screens, Task: Runtime Architecture of the Screen Flow Control, in the course BC400.

2. Which of the following should be assigned to a screen field to make input mandatory?

Choose the correct answer.

- A Insert
- B Required
- C Graphical layout
- D Delete

You are correct! To make input mandatory, assign the Required attribute to a screen field. Read more in the lesson, Creating Input and Output Fields, Task Maintenance of the Attributes of a Screen Element, in the course BC400.

3. Which of the following statements is true regarding the implementation of data transport?

Choose the correct answer.

- A System field sy-ucomm serves as the interface for data transport between the program and the screen.
- B Data transport is performed when you enter values on the screen.
- C The TABLES statement serves as the interface for data transport between the program and the screen.

You are correct! You use the TABLES statement to define a structure variable of the same type and name with reference to the specified Dictionary structure within the program. This structure variable then serves as the interface between the program and the screen. Read more in the lesson, Implementing Data Transport, Task: The TABLES Structure as an Interface to the Screen, in the course BC400.

## UNIT 9

# SAP List Viewer

Lesson 1

Using the SAP List Viewer

288

### UNIT OBJECTIVES

- Describe EnjoySAP controls
- Implement an SAP List Viewer

## Unit 9

### Lesson 1

## Using the SAP List Viewer

### LESSON OVERVIEW

You use the SAP List Viewer to display an internal table on a screen. In addition to the function of the graphic formatting of the data, the SAP List Viewer offers a range of standard functions, such as sorting and filtering. An SAP standard class is used to control the SAP List Viewer.

### Business Example

You want to present data that exists in an internal table in the user dialog in an interesting way and provide the user with additional functions with minimum effort. For this reason, you require the following knowledge:

- An understanding of how to use the ALV grid control (SAP List Viewer) to display an internal table on a screen

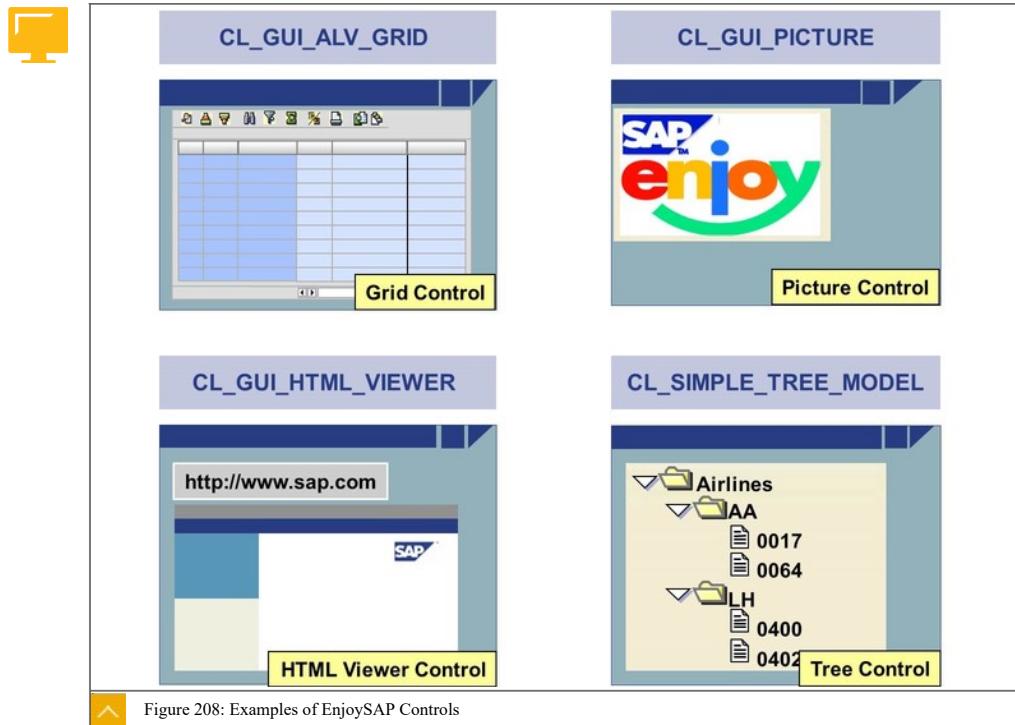


### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe EnjoySAP controls
- Implement an SAP List Viewer

## EnjoySAP Controls



## Selection of EnjoySAP Controls

- With Release 4.6, SAP delivered a large number of EnjoySAP controls, which you can use to design screens more ergonomically and interestingly.
  - Grid Control**: For displaying an internal table on a screen with various functions, such as sort, filter, and totals
  - Picture Control**: For displaying a picture on the screen
  - HTML Viewer Control**: For displaying an HTML file or Web page on the screen
  - Tree Control**: For depicting a hierarchical list in the form of a tree structure on the screen

Classes and methods that are delivered with the SAP standard system are used to communicate between these controls and an ABAP program. You will use class `CL_GUI_ALV_GRID` to address the most interesting and popular EnjoySAP controls.

## SAP List Viewer Usage

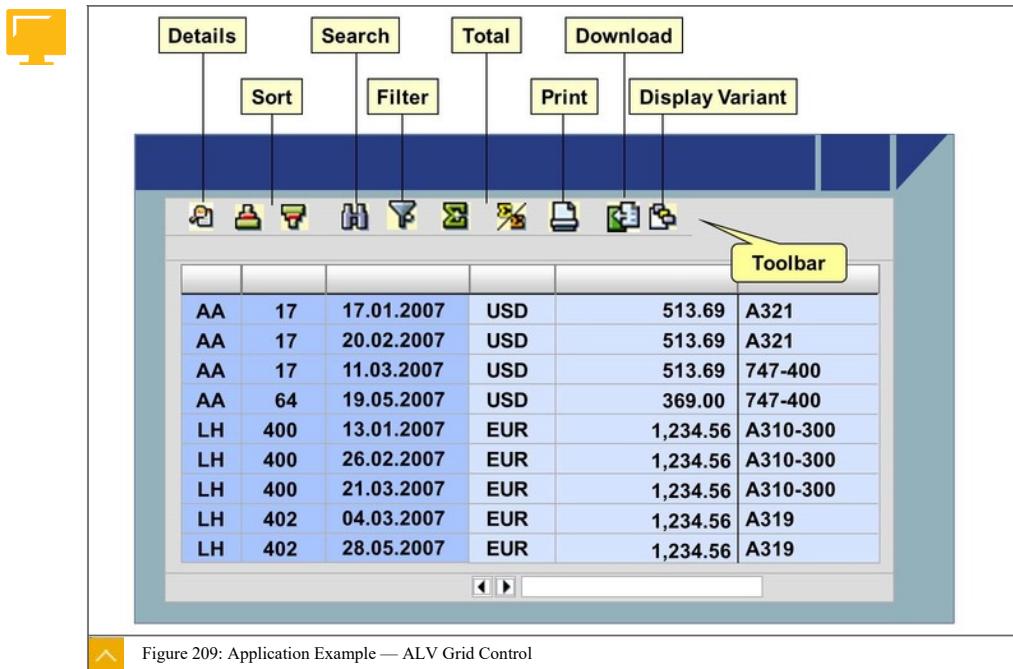


Figure 209: Application Example — ALV Grid Control

The ALV grid control, also known as the SAP List Viewer or ABAP List Viewer (ALV), is used to display an internal table on a screen. It has numerous user functions.

On the screen, the user can vary the width of the columns, or the width can be automatically adjusted to the current data. The user can also change the display position of the columns using drag and drop.

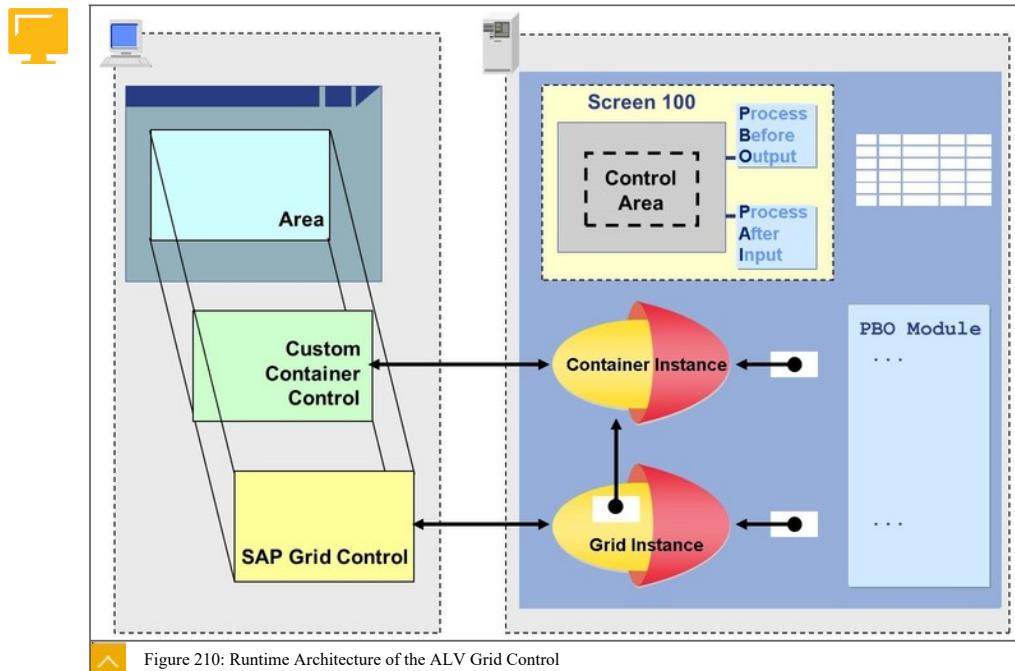
## Application Example – ALV Grid Control

- The standard buttons of the control can be used to execute the following functions:
  - Showing the fields that were previously selected with the cursor in a model dialog window.
  - Specifying complex sorting criteria for the columns.
  - Searching for a character string in rows or columns within a selected area.
  - Forming totals for one or several numeric columns together.
  - Printing and downloading.
  - Saving settings as a display variant and reusing them at a later time.

The detailed display shows the fields that were previously selected with the cursor in a model dialog window. The sort function provides the user with the option of specifying complex sorting criteria for the columns. The search function provides the user with the option of searching for a character string in rows or columns within a selected area. The totals function provides the user with the option of forming totals for 1 or several numeric columns together. The user can then use the Subtotals function to set up control level lists. Select the non-

numeric columns that you want to include before choosing this function. The system then displays the corresponding control level totals. The corresponding buttons enable you to print and download. The grid control enables users to save their settings as a display variant and reuse them at a later time.

#### Runtime Architecture of the ALV Grid Control

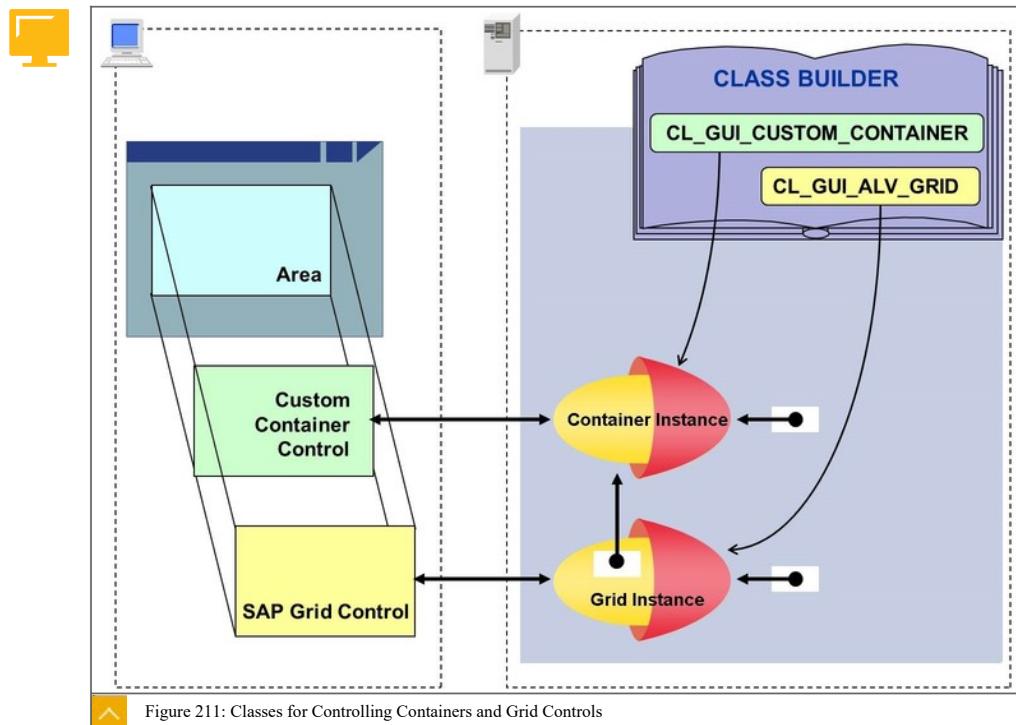


An EnjoySAP control is always displayed together with a screen. There are various options for the attachment of the EnjoySAP control to the screen. The simplest is to use the Graphical Layout Editor to create a special control area on the layout of the screen (see upper left part of the figure).

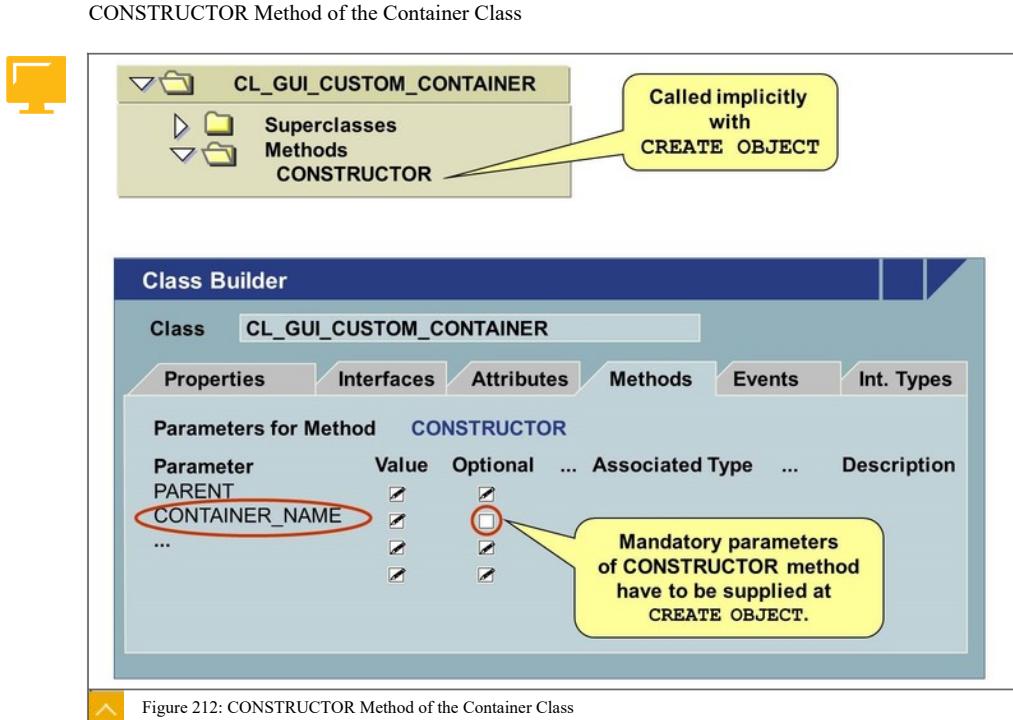
An SAP Container Control (container) is used to link the EnjoySAP control and the control area on the screen. The EnjoySAP control is embedded in the container, which in turn is integrated in the control area.

To implement the grid control and the container on the GUI, corresponding instances have to be created as substitutes in the ABAP program. You can use these instances to address the elements in the GUI. To do so, your SAP system has standard classes from which you can generate the container and the grid control instance.

Classes for Controlling Containers and Grid Controls



When you create an instance, the class-specific constructor (special method CONSTRUCTOR of the class) is called implicitly. The task of this method is to use its own input parameters to fill the attributes of the instance to be created. You must, therefore, supply the required import parameters of the constructor with values when you create an instance (CREATE OBJECT statement).



#### Access to Information about a Global Class or Method

- To obtain detailed information about a global class or method, you can navigate to the Class Builder using the following steps:
  - Display the object list of the class in the navigation area of the Object Navigator .
  - Double-click the class for a detailed display in the Class Builder . (Alternatively, you can also go to the Class Builder by double-clicking the class names within an ABAP program.)
  - Select the required method with the cursor and choose the Parameter button to display the interface parameters of the method.

The CONSTRUCTOR method of global class CL\_GUI\_CUSTOM\_CONTAINER (class for the container) has the required parameter CONTAINER\_NAME. When you create the container, you have to supply at least this parameter with data, namely with the name of the control area on the screen.

## Important Methods for the Grid Control Class

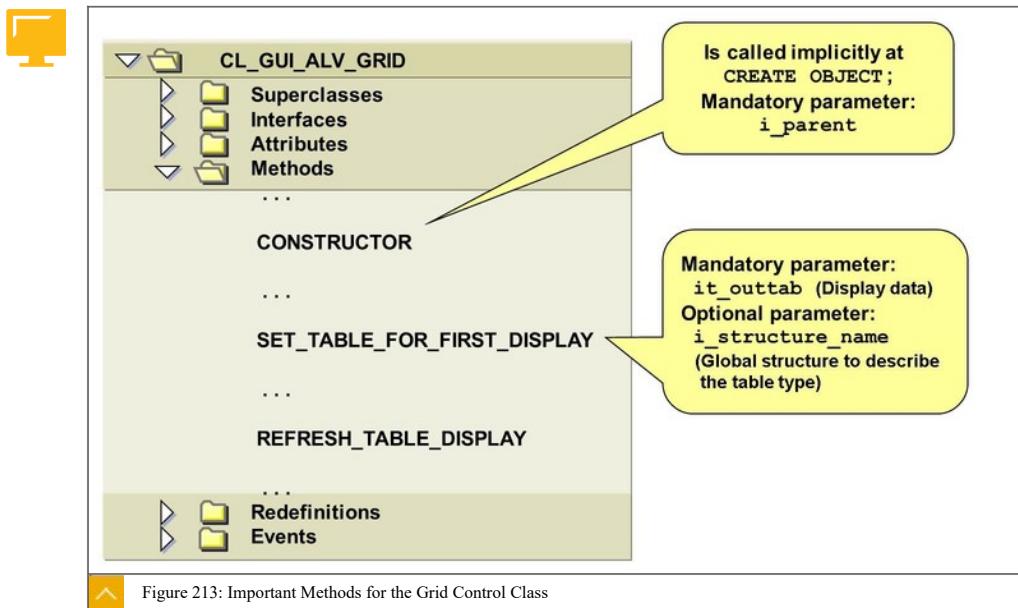


Figure 213: Important Methods for the Grid Control Class

The CL\_GUI\_ALV\_GRID global class has numerous methods that can be called for the corresponding grid control functions.

#### Display of the Content of an Internal Table with an ALV Grid Control

To display the content of an internal table with an ALV grid control, you need to know the following methods in detail:

- **CONSTRUCTOR**

The grid class has a constructor. The only required parameter is **I\_PARENT** to which the previously created container instance, in the form of a pointer, has to be transferred (when the grid control instance is created).

- **SET\_TABLE\_FOR\_FIRST\_DISPLAY**

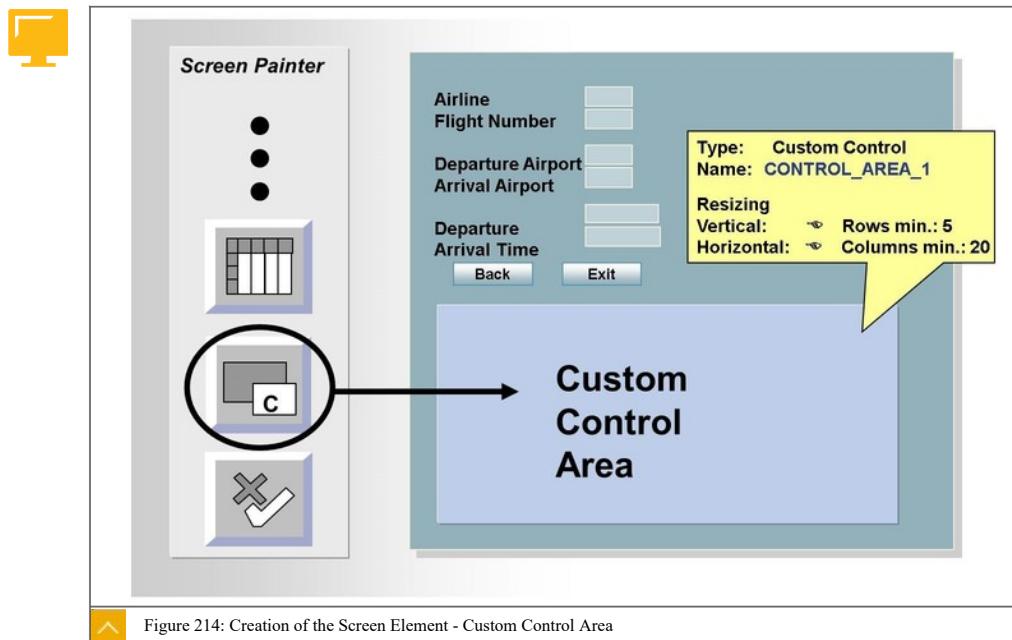
This method of the created grid control instance is used for transferring data and settings to the grid control. The internal table to be displayed must be transferred to parameter **IT\_OUTTAB**. This must be a standard table (see typing of the parameter).

Furthermore, technical information is required for the formatting of the grid columns. The easiest thing to do is to use a Dictionary structure or a transparent table as the row description of the internal table. In this case, all you have to do is transfer the name of the Dictionary object to parameter **I\_STRUCTURE\_NAME**. (Alternatively, you can also set up a field catalog and transfer it to parameter **IT\_FIELDCATALOG**.)

- **REFRESH\_TABLE\_DISPLAY**

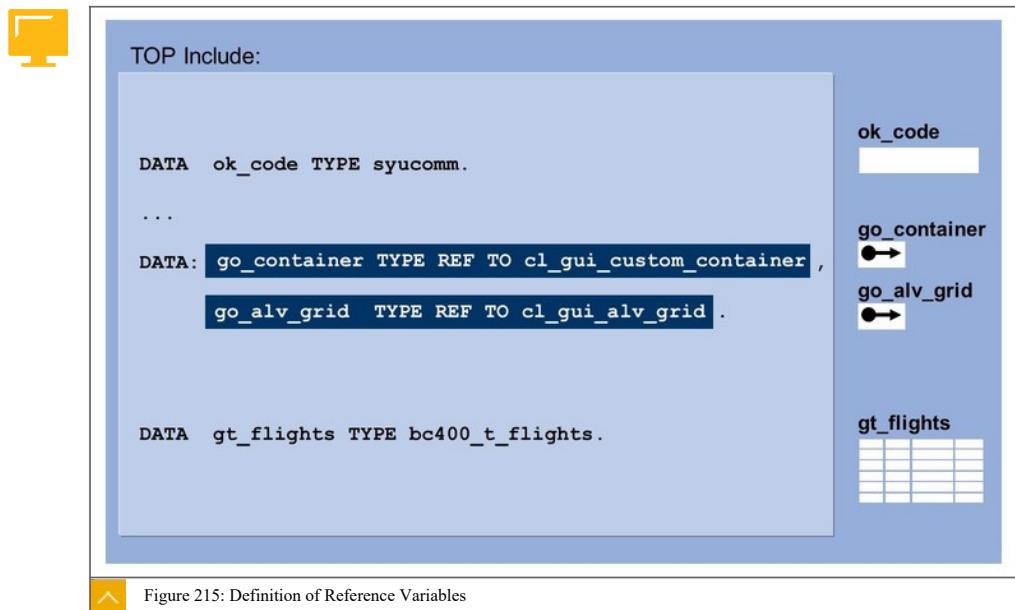
This method is to refresh the output table in the grid control and is necessary if data has been changed.

## Display of Table Content Using the SAP List Viewer



In the graphical layout editor , you can define a control area on your screen by choosing the Custom Control button from the toolbar. Select the button and specify the size and position of the area on the screen. Enter a name for the new screen element (for example, **CONTAINER\_AREA\_1**).

### Definition of Reference Variables



### Reference Variables Required in the ABAP Program

- Certain reference variables are required to be defined in the ABAP program:
  - A reference variable that should point to a container instance, which is yet to be created (for example: **GO\_CONTAINER**).
  - A reference variable that should point to a grid control instance, which is yet to be created (for example: **GO\_ALV\_GRID**).

## Creation of Instances



O INCLUDE:

```

MODULE init_control_processing_0200 OUTPUT.

IF go_container IS INITIAL.

    CREATE OBJECT go_container
        EXPORTING container_name = 'CONTROL_AREA_1'.

    CREATE OBJECT go_alv_grid
        EXPORTING i_parent      = go_container.

    ...

ENDIF.

ENDMODULE.
```

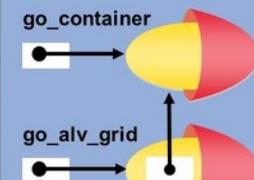


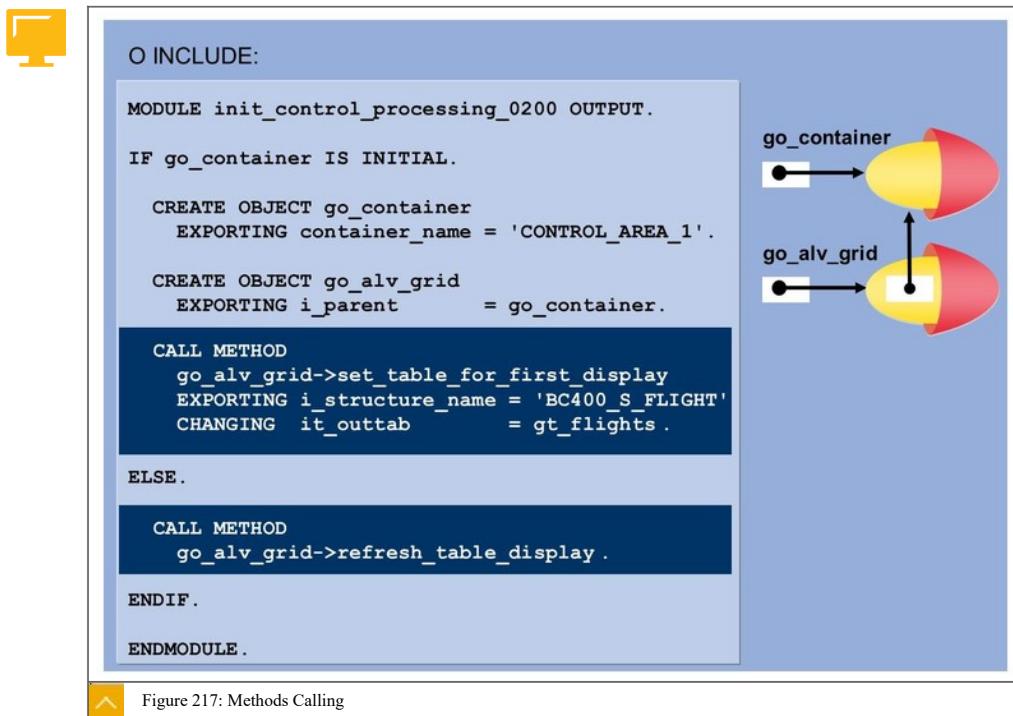
 Figure 216: Creation of Instances

You create the instances with the CREATE OBJECT statement. Use "Pattern" to generate the statement, insert the name of the reference variable and supply the parameters with values in the call.

You only have to instantiate the controls on a screen once. This means that this step is omitted when a screen is processed again. This is easily done by querying one of the two reference variables: IF go\_container IS INITIAL. The call syntax is very similar to that of the function module.

It makes sense to implement the creation of the control instances before displaying the screen, in other words, in a PBO module.

## Methods Calling



To transfer the content of an internal table and its row description to the ALV grid control, call the SET\_TABLE\_FOR\_FIRST\_DISPLAY method of the grid control instance. Here, you should also generate the call by using drag and drop.

## Parameter Values

- You also have to supply the parameters with the following values:
  - Transfer the filled internal table to parameter IT\_OUTTAB. The internal table has the global structure type BC400\_S\_FLIGHT as a line type here, so you can pass this name to the I\_STRUCTURE\_NAME parameter. The corresponding Dictionary information is then loaded automatically and transferred to the control.
  - If the content of the internal table changes during the program, you must call the REFRESH\_TABLE\_DISPLAY method to update the grid display before the next screen is displayed.



## To Display Tables with the SAP List Viewer

- In the graphical layout editor , you can define a control area on your screen by choosing the Custom Control button from the toolbar. Select the control area and specify the size and position of the area on the screen. To do so, choose the editing area where you want to place the top-left corner of the custom control and hold down the mouse key. Drag the cursor diagonally down to the right to where you want the bottom-right corner. After you release the mouse button, the bottom-right corner is fixed in position.

Enter a name for the new screen element (for example, **CONTAINER\_AREA\_1**).

Use the **Resizing Vertical** and **Resizing Horizontal** attributes to specify whether the area of the custom control should be resized when the main screen is resized. When you set these attributes, you can also set the minimum values for the area using the additional attributes **Min. Lines** and **Min. Columns**.

**2.** Define the reference variables required by the grid control classes in the ABAP program.

- A reference variable that should point to a container instance, which is yet to be created (as in the figure: **GO\_CONTAINER**).

```
DATA: go_container type ref to cl_gui_custom_container,
      go_alv_grid type ref to Cl_gui_alv_grid.
```

- A reference variable that should point to a grid control instance, which is yet to be created (as in the figure: **GO\_ALV\_GRID**).

**3.** You create the instances with the CREATE OBJECT statement. By generating your statement in your source code, for example, using drag and drop, you can avoid typing errors and omissions.

To generate your statements in the source code, perform the following steps:

- a) Display the object list of the respective class in the navigation area of the Object Navigator .
- b) Drag and drop the class name in your source code.
- c) Insert **go\_alv\_grid** for xxxxxxxx and supply the parameters with values in the generated call.

You only have to instantiate the controls on a screen once. This means that this step is omitted when a screen is processed again. This is easily done by querying one of the two reference variables: IF go\_container IS INITIAL.

The call syntax is very similar to that of the function module. However, the parameters to be supplied with values with CREATE OBJECT are the interface parameters of the respective constructor.

It makes sense to implement the creation of the control instances before displaying the screen, in other words, in a PBO module.

**4.** To transfer the content of an internal table and its row description to the ALV grid control, call the **SET\_TABLE\_FOR\_FIRST\_DISPLAY** method of the grid control instance. Here, you should also generate the call by using drag and drop. In the generated call, use **go\_alv\_grid** instead of xxxxxxxx.

You also have to supply the parameters with the following values:

- a) Transfer the filled internal table to parameter **IT\_OUTTAB** (as in the figure, Methods Calling).
- b) The internal table has the global structure type **BC400\_S\_FLIGHT** as a line type here as in the figure, Methods Calling , so you can pass this name to the **I\_STRUCTURE\_NAME** parameter. The corresponding Dictionary information is then loaded automatically and transferred to the control.

If the content of the internal table changes during the program, you must call the REFRESH\_TABLE\_DISPLAY method to update the grid display before the next screen is displayed.



#### LESSON SUMMARY

You should now be able to:

- Describe EnjoySAP controls
- Implement an SAP List Viewer

## Unit 9

### Learning Assessment

1. Which of the following is the Container Class that is used to link the EnjoySAP control with the control area on the screen?

Choose the correct answer.

- A** CL\_GUI\_CUSTOM\_CONTAINER
- B** CL\_ALV\_CONTAINER
- C** CL\_CONTAINER
- D** CL\_ENJOY\_CONTAINER

2. Which of the following are methods of global class CL\_GUI\_ALV\_GRID?

Choose the correct answers.

- A** CONSTRUCTOR
- B** SET\_TABLE\_FOR\_FIRST\_DISPLAY
- C** CREATE\_TABLE\_FOR\_DISPLAY
- D** REFRESH\_TABLE\_DISPLAY

## Unit 9

### Learning Assessment - Answers

1. Which of the following is the Container Class that is used to link the EnjoySAP control with the control area on the screen?

Choose the correct answer.

- A CL\_GUI\_CUSTOM\_CONTAINER**
- B CL\_ALV\_CONTAINER**
- C CL\_CONTAINER**
- D CL\_ENJOY\_CONTAINER**

You are correct! An SAP Container Control (container), such as CL\_GUI\_CUSTOM\_CONTAINER, is used to link the EnjoySAP control and the control area on the screen. The EnjoySAP control is embedded in the container, which in turn is integrated in the control area. Read more in the lesson, Using the SAP List Viewer, Task: Runtime Architecture of the ALV Grid Control, in the course BC400.

2. Which of the following are methods of global class CL\_GUI\_ALV\_GRID?

Choose the correct answers.

- A CONSTRUCTOR**
- B SET\_TABLE\_FOR\_FIRST\_DISPLAY**
- C CREATE\_TABLE\_FOR\_DISPLAY**
- D REFRESH\_TABLE\_DISPLAY**

You are correct! The CL\_GUI\_ALV\_GRID global class has numerous methods that can be called for the corresponding grid control functions. The methods CONSTRUCTOR, SET\_TABLE\_FOR\_FIRST\_DISPLAY and REFRESH\_TABLE\_DISPLAY are some of them. Read more in the lesson, Using the SAP List Viewer, Task: Important Methods for the Grid Control Class , in the course BC400.

## UNIT 10

# Web Dynpro ABAP

### Lesson 1

Describing Web Dynpro ABAP	304
----------------------------	-----

### Lesson 2

Implementing Navigation in Web Dynpro	312
---------------------------------------	-----

### Lesson 3

Implementing Data Transport in Web Dynpro	322
---	-----

### UNIT OBJECTIVES

- Describe Web Dynpro ABAP
- Implement navigation in Web Dynpro
- Implement data transport in Web Dynpro

## Unit 10

### Lesson 1

# Describing Web Dynpro ABAP

#### LESSON OVERVIEW

This lesson gives an overview of the properties, usage scenarios, and programming and runtime architecture of Web Dynpro for ABAP. You will also learn how to structure and program a simple Web Dynpro for ABAP application with input and output fields and buttons.

#### Business Example

You need to explain the properties and usage scenarios as well as the programming concept and runtime architecture of Web Dynpro for ABAP. For this reason, you require the following knowledge:

- An understanding of the properties and usage scenarios of Web Dynpro for ABAP
- An understanding of the programming and runtime architecture of Web Dynpro for ABAP
- An understanding of how to implement simple Web Dynpro applications with input and output fields and buttons



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe Web Dynpro ABAP

#### Web Dynpro Essentials

#### Comparison of Web Dynpro Terms

Table 6: Web Dynpro Terms

Classic Dynpro	Web Dynpro
Program (for example, module pool)	Web Dynpro component
Dialog transaction	Web Dynpro application
Screen	Web Dynpro view
Screen sequence	Web Dynpro window
TABLES structure	Web Dynpro context
Function code	System response
PAI module user_command_nnnn	onactionxyz methods
SET SCREEN nnnn	FIRE_PLUGxyz

### Properties of Web Dynpro



- Programming model independent of client and protocol
- Separation of layout, program flow, and business logic
- Graphical support in layout design and program flow logic design
- Development tools implemented in the Object Navigator
- Reusable components
- Back-end systems through the use of web services
- Support of Accessibility Standard Section 508

With SAP NetWeaver 7.0, SAP introduced Web Dynpro for ABAP, which is a new programming model that is independent of the clients used (for operating Web Dynpro applications) and the respective protocol. This independence is achieved through the use of metadata.

Web Dynpro for ABAP will become the standard UI technology used by SAP for the development of Web applications in the ABAP environment.

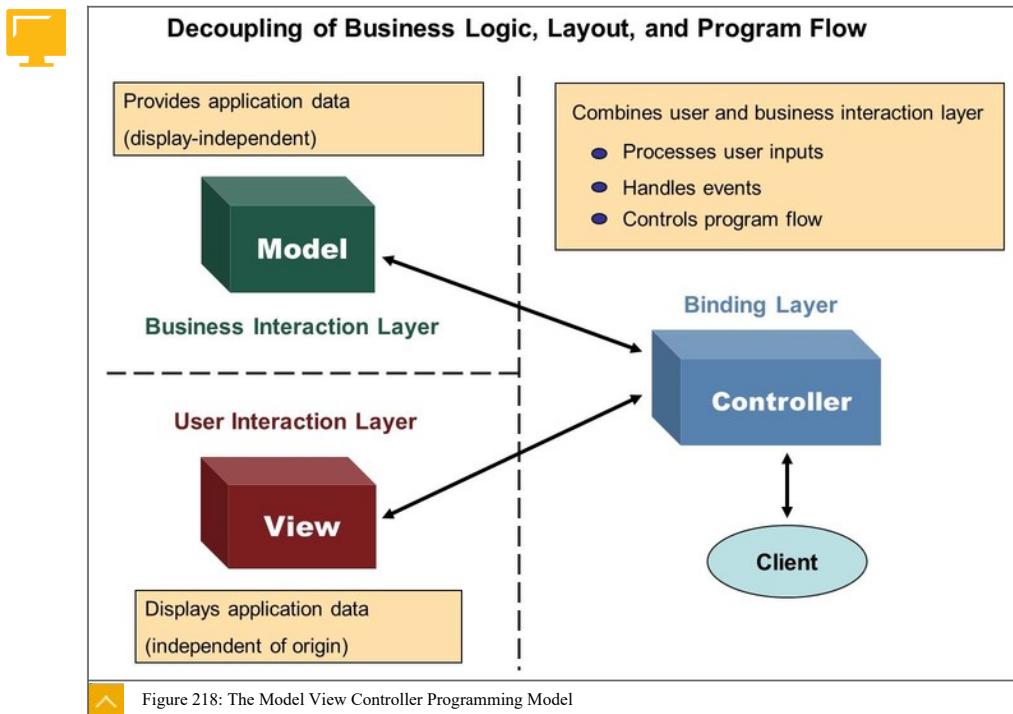
In addition to having its own runtime environment, Web Dynpro for ABAP, in particular, has a graphic development environment ( Web Dynpro Explorer ), whose tools are fully integrated into the ABAP development environment ( Object Navigator ). You use SAP GUI to develop a Web Dynpro application in the SAP system. It is called through the respective URL or the Internet browser, or from the SAP Portal.

Maintenance and further development of Web Dynpro for ABAP applications is simplified through the reusability of components and the strict separation of layout, program flow, and business logic.

In addition, Web Dynpro applications can be made usable for visually impaired users, as Web Dynpro supports Accessibility Standard Section 508 through keyboard input and screen-reading functions.

Web Dynpro supports the use of web services, which enable you to access business functions of various systems from any application in a standardized, easy-to-use way.

## The Model View Controller Programming Model



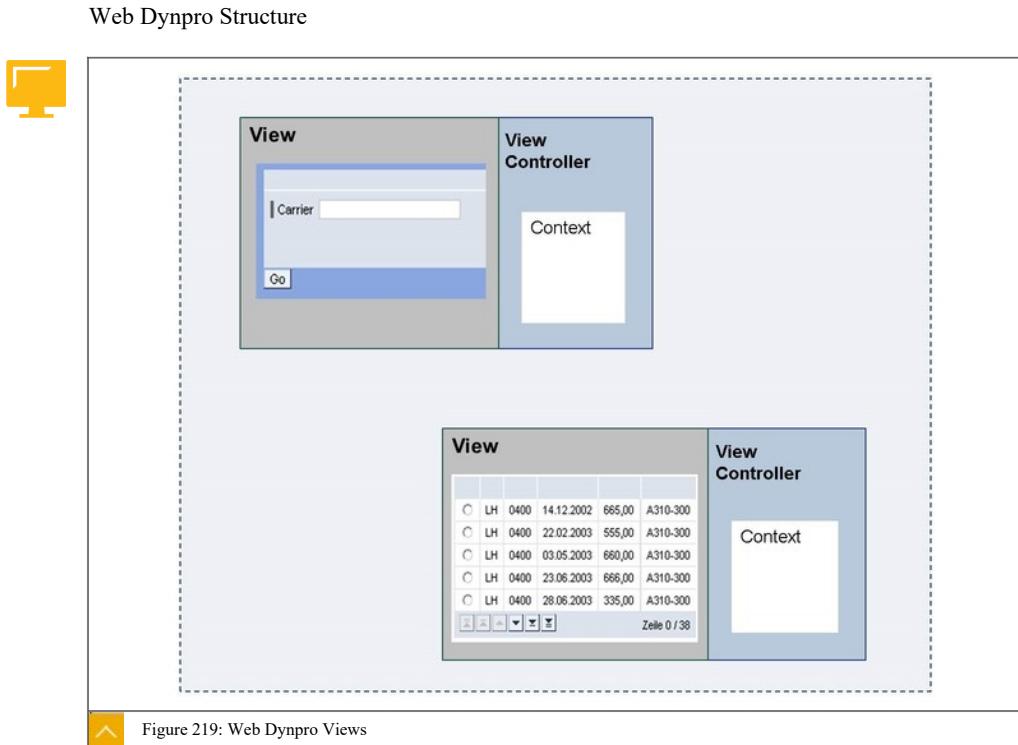
Web Dynpro applications are structured according to the Model View Controller (MVC) programming model.

While the model in the business interaction layer is responsible for display-independent data access to the back-end system, the view takes care of the display of data (layout), irrespective of the source, in the user interaction layer.

The controller in the binding layer is between the view and the model. It formats the model data that is to be displayed in the view, processes the user entries, and returns them to the model. In addition, it controls the flow of the program.

In Web Dynpro, the model can consist of application classes that encapsulate data access. However, existing Business Application Programming Interfaces (BAPIs), function modules, or Web services can be used as well.

The view and the controller are implemented using graphic tools.



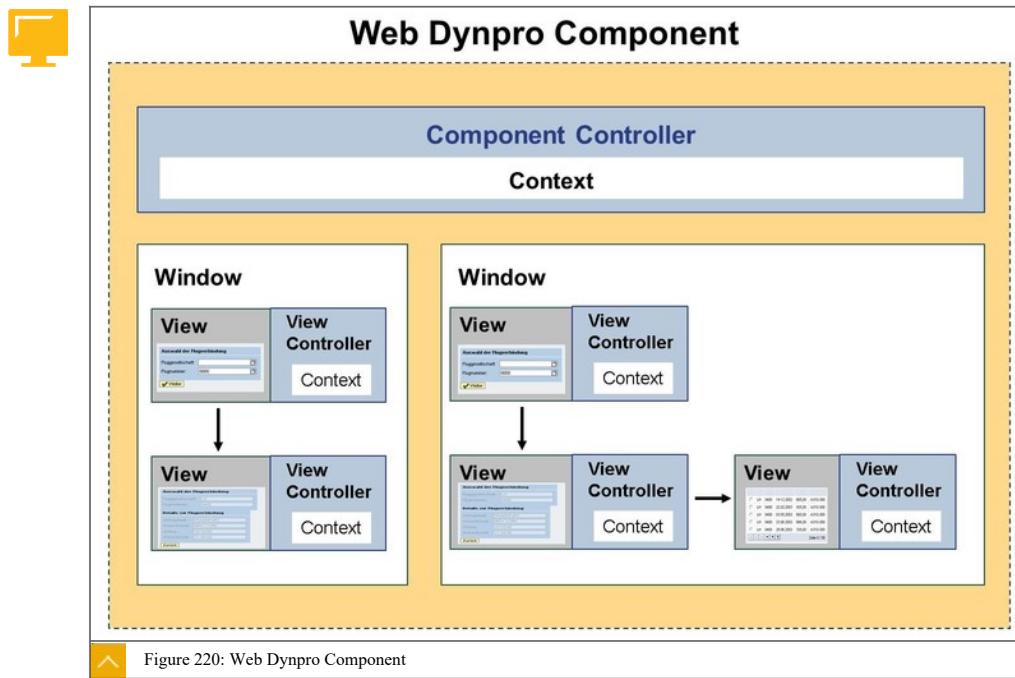
The layout of a view consists of a set of UI elements (screen elements such as input fields, tables, and buttons) that are grouped together using a graphic tool. At runtime, the system displays the views one after the other, not simultaneously, on the screen. Therefore, they correspond to classic screens without flow logic.

As with classic Dynpro, input checks and input helps (F4) do not have to be implemented manually, but can be used as UI services by referring to the ABAP Dictionary.

You can integrate Web Dynpro views into other views using the Container technology. This enables modularization.

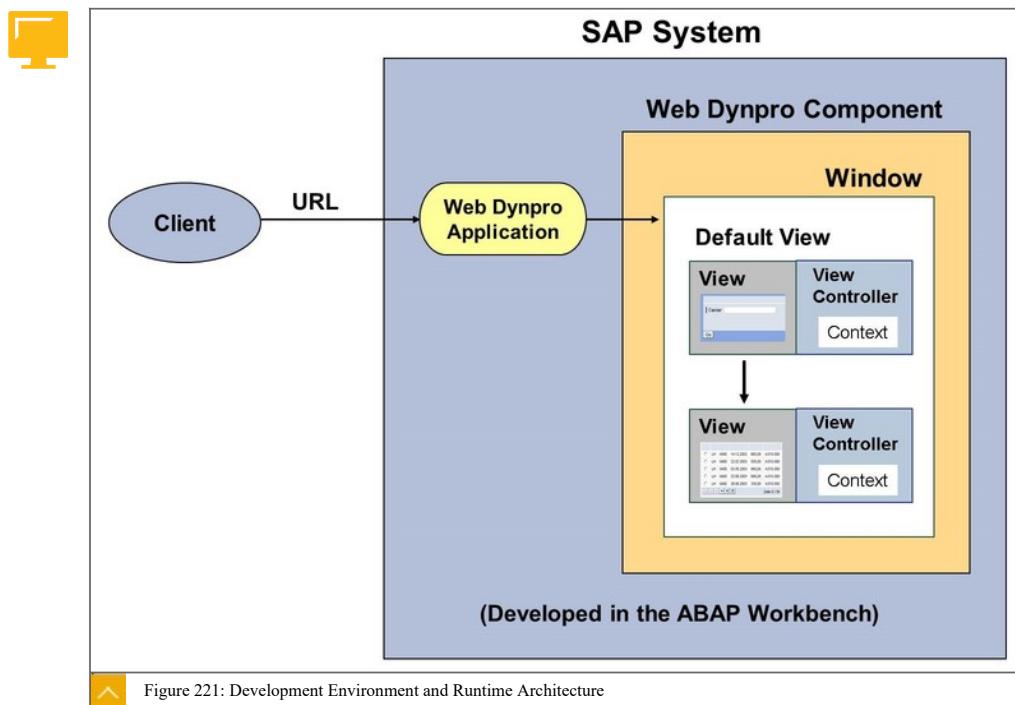
Each view has its own assigned view controller. Technically speaking, the view controller is an ABAP class. The developer only partially implements the source code for the view controller. A large part of the source code is automatically generated in accordance with the developer's design specifications. The context of the view controller serves as a data container and contains the data to be displayed in the view (corresponds to the TABLES structure in classic Dynpro).

Web Dynpro Component



Windows bundle several views and define navigation options between them (view sequences). A component contains one or several windows and has its own controller with a context, where data to be displayed in several component views is stored.

## Development Environment and Runtime Architecture



A Web Dynpro application points to a window with a default view, which is displayed when the application is called.

A component with appropriate windows is usable as a self-contained unit in different Web Dynpro applications. This reduces implementation and maintenance work to a minimum.

## Web Dynpro Context

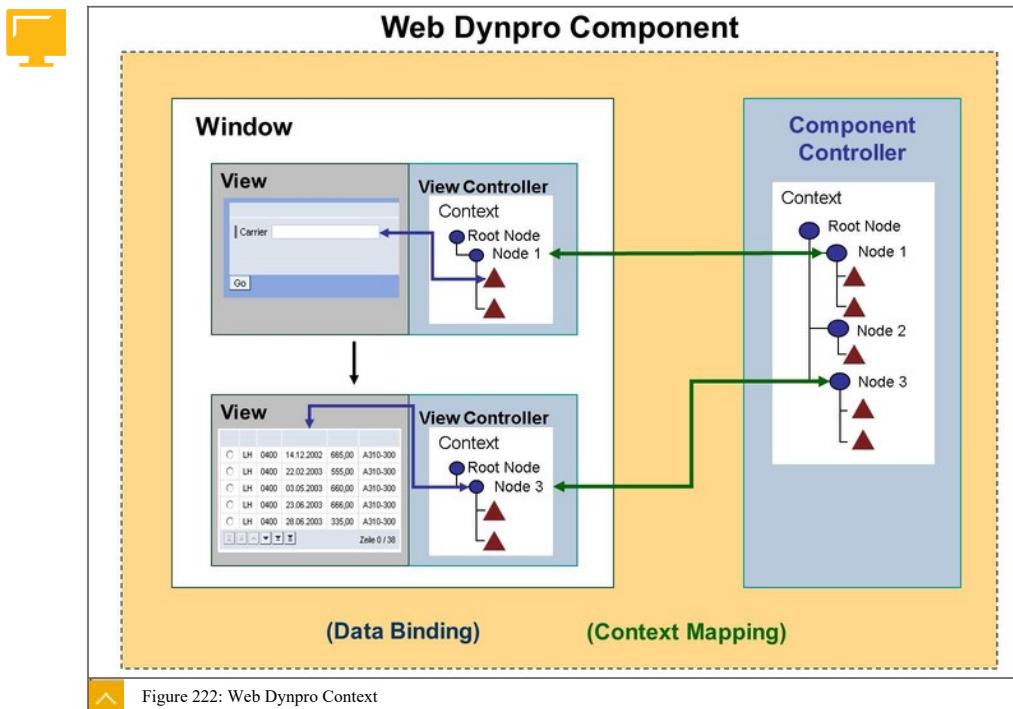
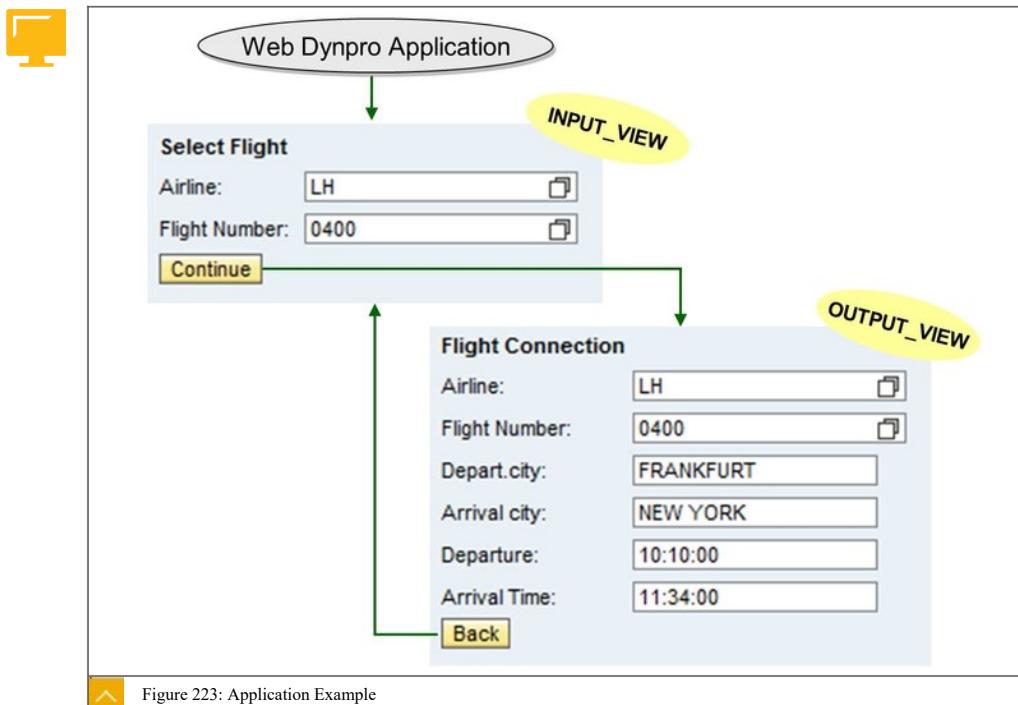


Figure 222: Web Dynpro Context

The context of each view controller contains data that is to be displayed in the view (data container). The data transport between the view context and the UI elements of the view takes place when the appropriate assignment is made (data binding).

A component also has a context. Here, data that is to be displayed in the different views of the component is stored. In such cases, references to component data are implemented mostly in the corresponding view contexts (context mapping).

## Application Example



You can develop a Web Dynpro application in several steps that will enable you to display the data for individual flight connections.

The program comprises a Web Dynpro component with two views. In the first view, the user chooses a flight connection. In the second view, the details for this connection are displayed.

Both views are embedded in a Web Dynpro window for the component and connected by navigation links in such a way that the user can navigate between views using buttons.

The screen sequence defined in the window is started by way of the Web Dynpro application.



## LESSON SUMMARY

You should now be able to:

- Describe Web Dynpro ABAP

## Unit 10

### Lesson 2

# Implementing Navigation in Web Dynpro

#### LESSON OVERVIEW

This lesson explains how to structure and program a simple Web Dynpro for ABAP application.

#### Business Example

You want to create a Web Dynpro component with several views and navigation between these views. For this reason, you require the following knowledge:

- Create a Web Dynpro component
- Create Web Dynpro views
- Embed views in Web Dynpro windows
- Set up the navigation
- Create buttons and evaluating user actions
- Create Web Dynpro applications

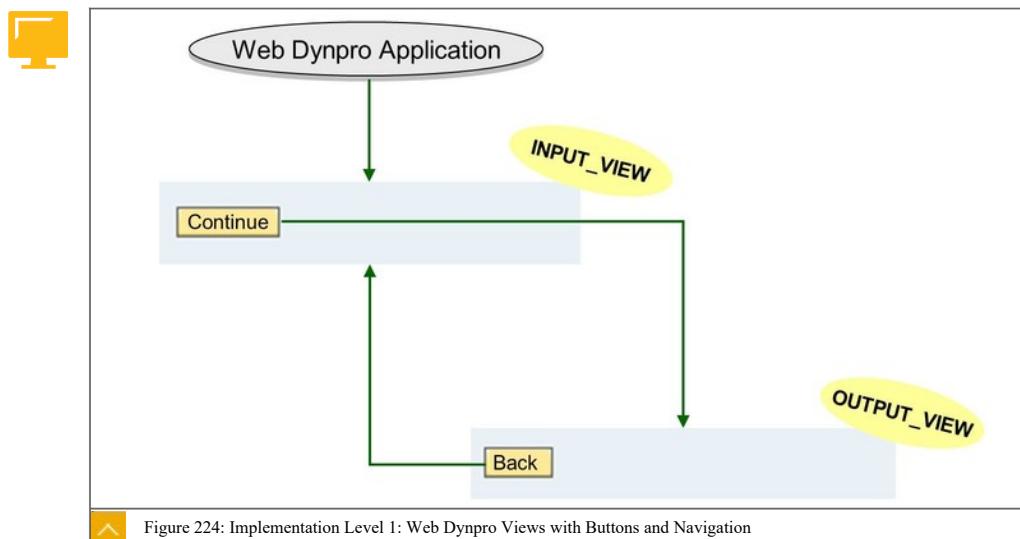


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Implement navigation in Web Dynpro

#### Web Dynpro Navigation



At the first level of our example, we want to create a Web Dynpro component with two views. Each of the views should have one button. The program should be implemented in such a way that when you click a button, you navigate to the other view. To start the view sequence, you need to create a Web Dynpro application.

### Web Dynpro Component

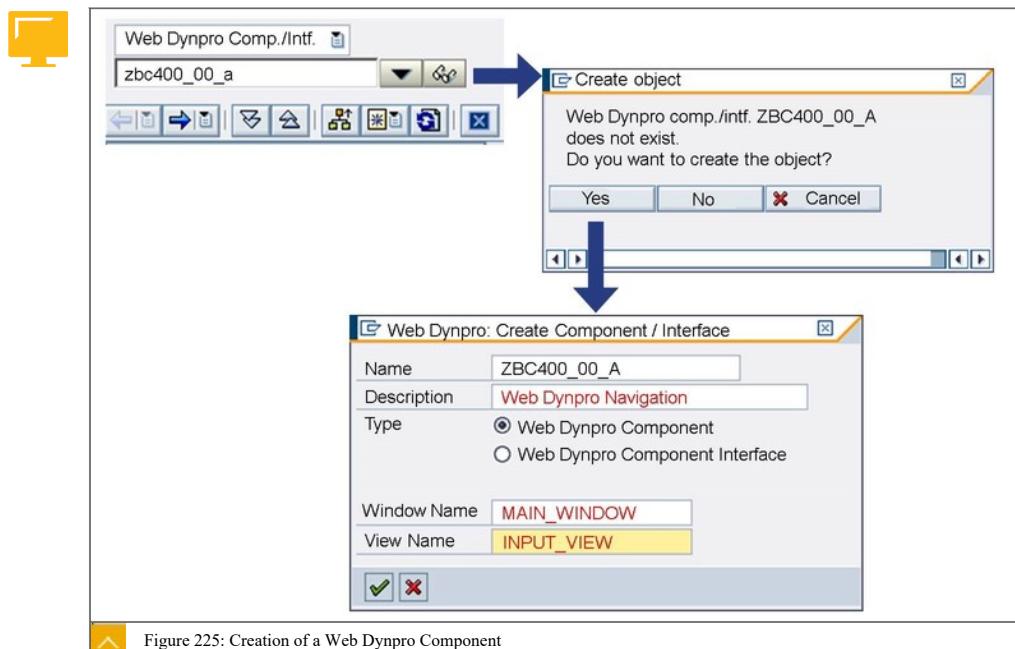
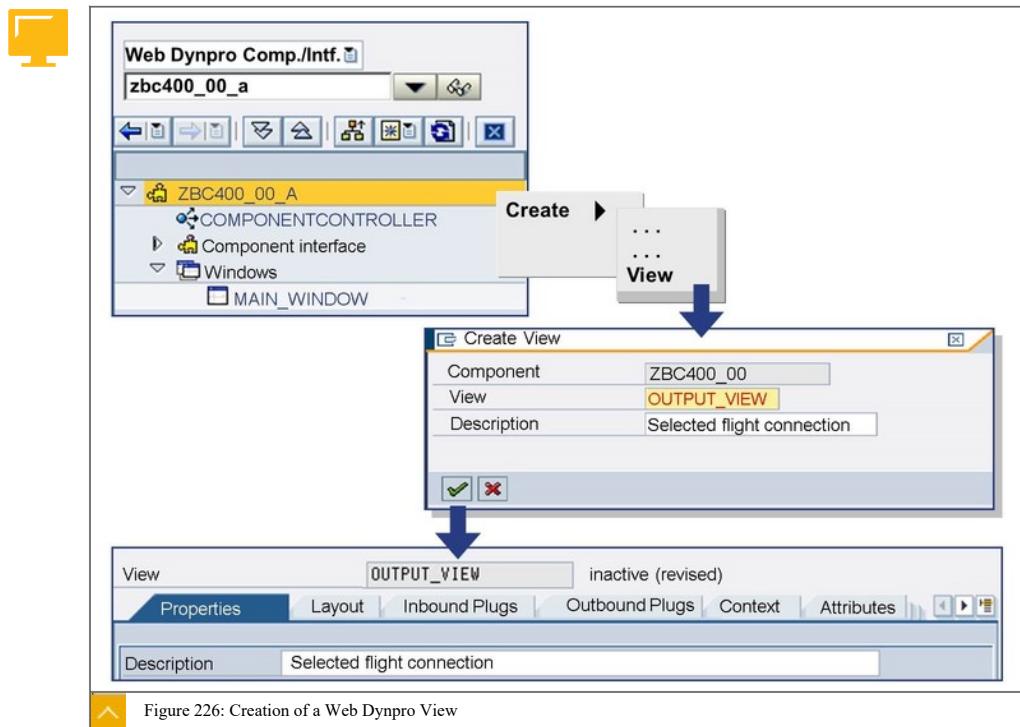


Figure 225: Creation of a Web Dynpro Component

You create a Web Dynpro component in a similar manner to the way you create a new ABAP program, from the context menu in the navigation area of the Object Navigator .

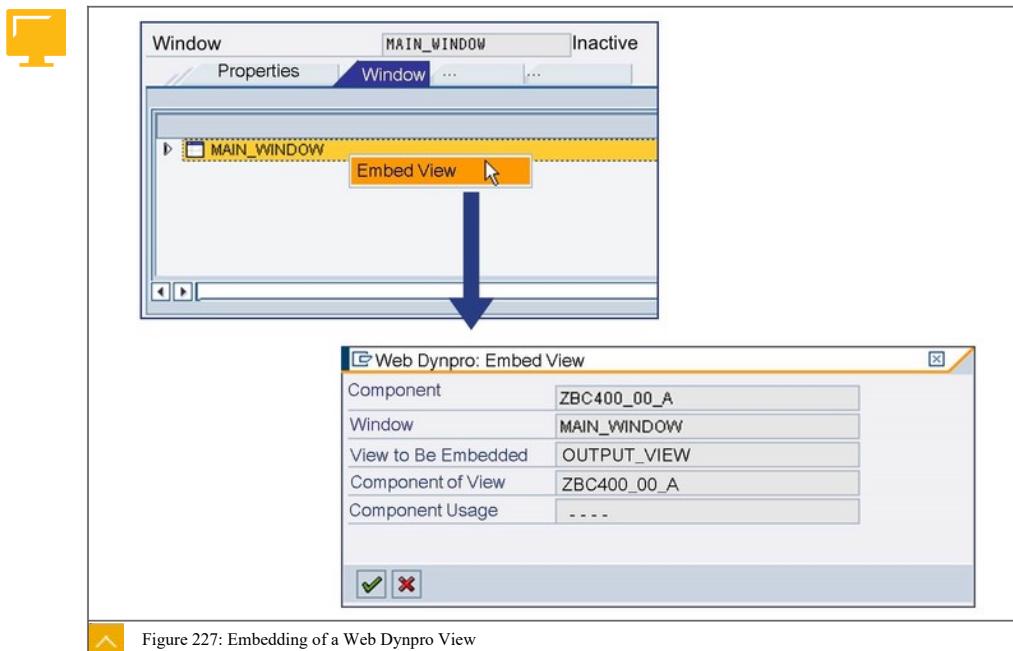
When you create a new Web Dynpro component, the system creates a Web Dynpro window and a Web Dynpro view. At a later stage, you can create additional windows and views in the component.

### Views and Windows



To create a Web Dynpro view, use the context menu in the navigation area of the Object Navigator. Assign a name and description to the view.

### Embedding of a Web Dynpro View



To embed a Web Dynpro view in a window, open the window in the Object Navigator and display the window structure.

#### Methods for Embedding Views

You can use the following options to embed the views:

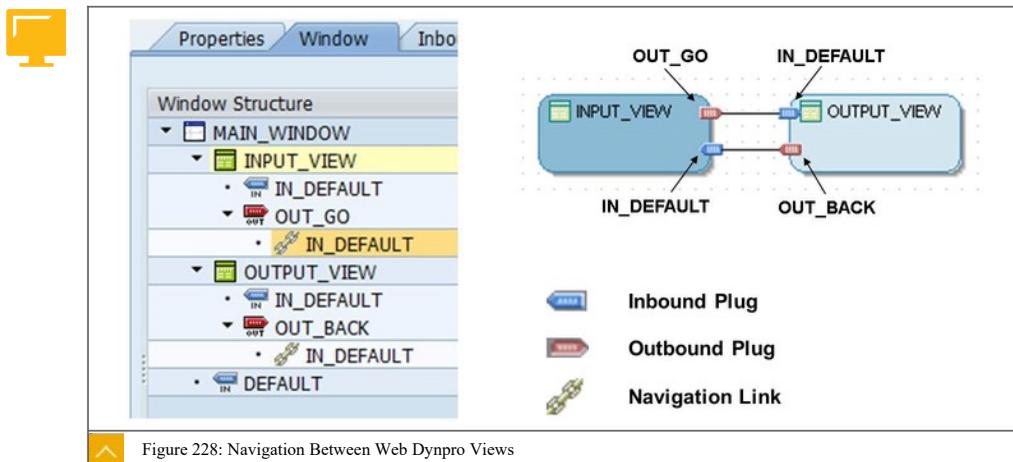
- Context menu

In the window structure, the **Embed View** entry appears in the context menu for the name of the window (see figure). On the entry screen that follows, you can select the view to be embedded using the input help (F4).

- Drag-and-drop from the navigation area

If you open the **Views** node in the navigation area, you can drag the view to be embedded directly into the window structure and onto the window by drag-and-drop.

### Navigation Between Views



To navigate within a window from one view to another, you have to create a navigation link between the views. A navigation link leads from an outbound plug in one view to an inbound plug in the other.

The outbound plugs and inbound plugs of a view determine the possible entrances and exits. A view can have more than one inbound and one outbound plug. The navigation link in the figure enables you to navigate to OUTPUT\_VIEW, when you exit INPUT\_VIEW using its outbound plug OUT\_GO. OUTPUT\_VIEW is then started by way of its inbound plug IN\_DEFAULT.

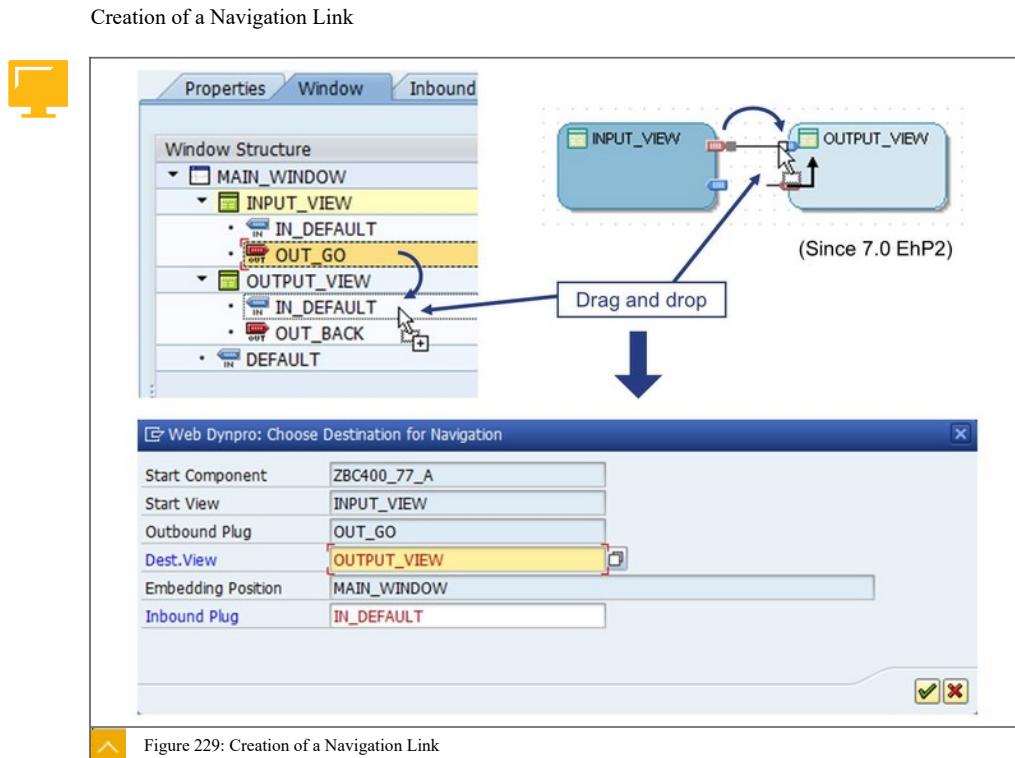
When you create the navigation links, this determines the view sequence within the respective window. This occurs on the basis of the existing plugs and independently of the implementation of the views.

You create a plug very simply by processing the corresponding view and entering a name and a description for the plug on the Inbound Plugs or Outbound Plugs tab pages. You can also apply parameters to the outbound plugs for advanced applications.



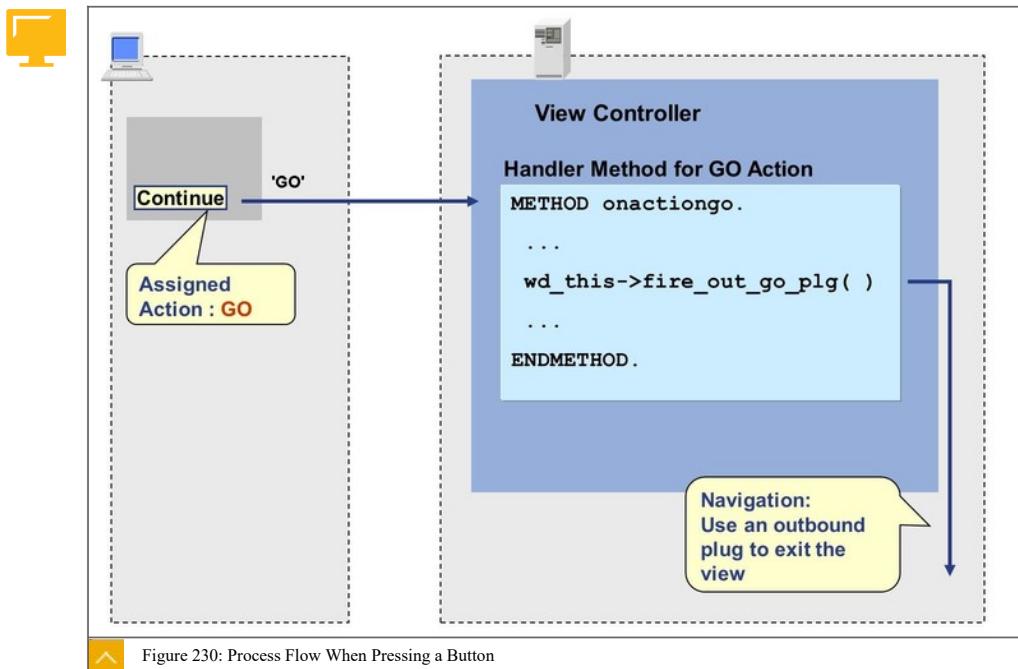
**Hint:**

After SAP NetWeaver 7.0 EhP2 has come into use, you can switch the Window Editor view between a hierarchical view and a graphical modeler view. You can decide which one is more suited for your work.



After you have created the plugs, you can see them in the structure of a window next to the embedded views. You can create navigation links between these plugs by using drag-and-drop.

### Buttons and Navigation



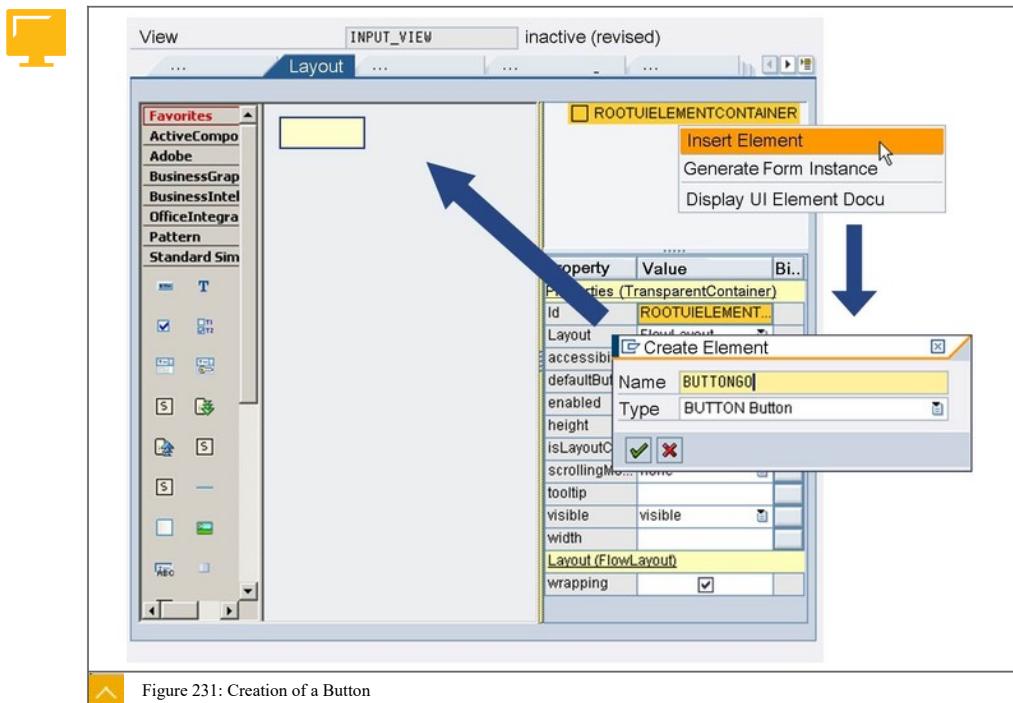
In the example program, navigation should be triggered in each instance by pressing a button.

#### Process Flow When Pressing a Button

The process flow when pressing a button is as follows:

- The user presses a button, and an action code is transferred from the browser to the application server. This action code was defined by the developer for the respective view and assigned to the button.
- There is a special method (Action Handler Method in the view controller) for each action code that now executes in reaction to the action.
- To navigate to the next view, you leave the current view using the corresponding outbound plug. The expression used is that the outbound plug is “fired”.

## Creation of a Button



## Options for Creating Buttons in Layout Views

To create a button in the layout of the view, you can use the following options:

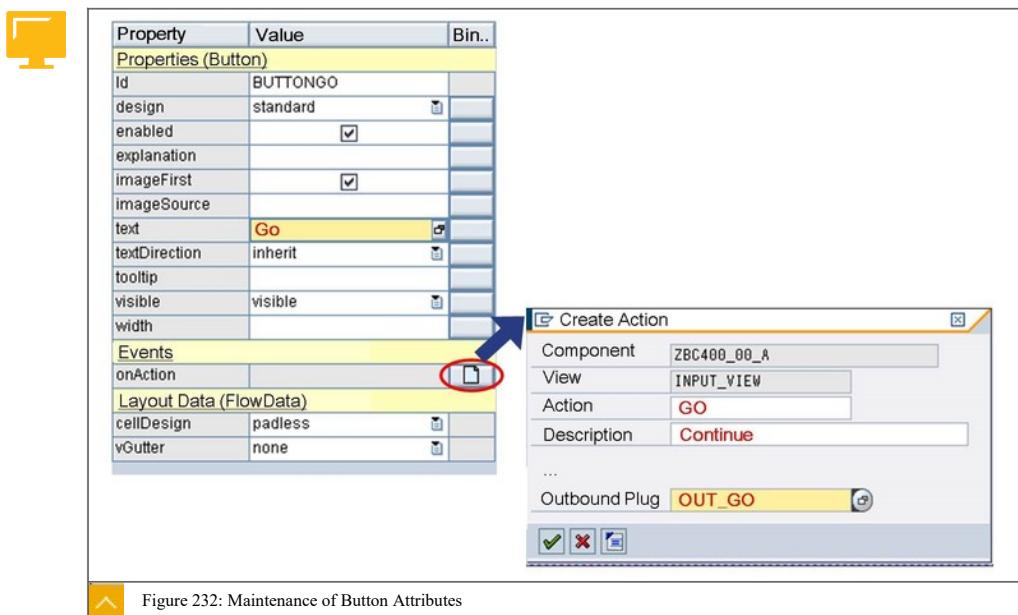
- Via the context menu in the element hierarchy

In the element hierarchy, open the context menu for the **ROOTUIELEMENTCONTAINER** node in the top-right corner and choose **Insert Element** from the drop-down list. Enter the name and type of the UI element (as illustrated in the figure).

- Via the toolbar

From the toolbar, choose the **Button** tool on the left and position the button in the middle of the layout preview.

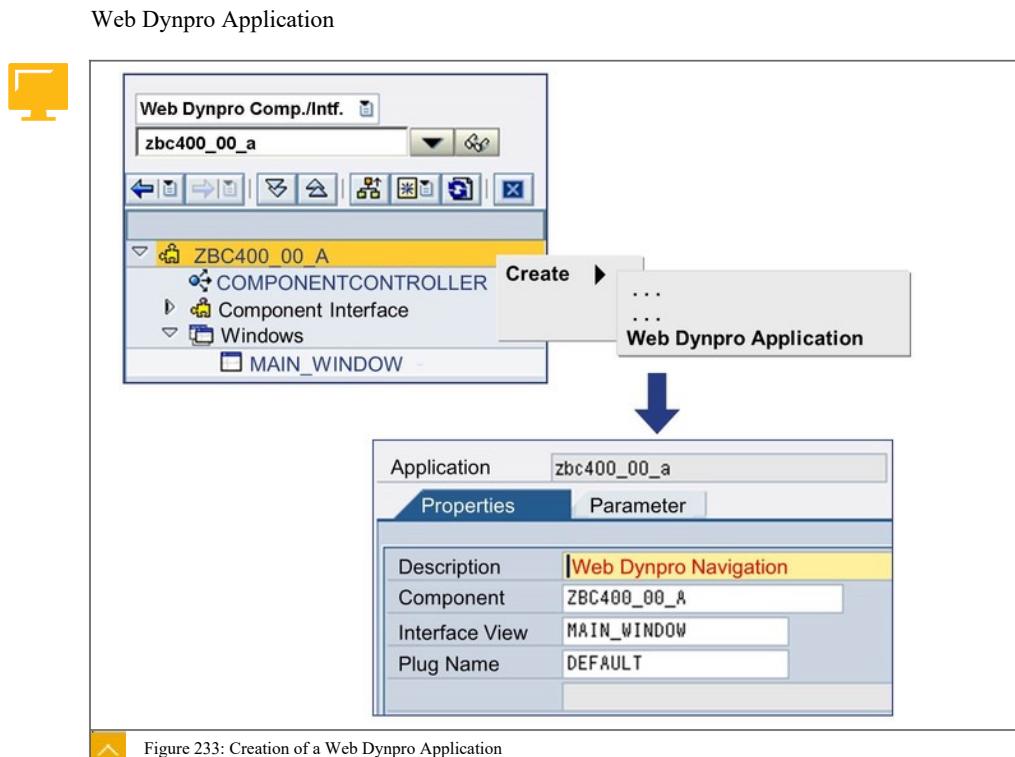
## Maintenance of Button Attributes



In the attribute window in the lower-right corner, the attributes are displayed for a specific element of the layout. You can enter text for a button or assign an icon to a button here.

To assign an action to the button, choose the `onAction` attribute and click the Create icon in the third column (as illustrated in the figure). Assign an action to the button in the Create Action dialog box that follows. If the action has not already been defined for the view (Actions tab page), you can create it here again directly.

In the `Outbound Plug` field, you have to specify the outbound plug that is fired as a reaction to this action. The Action Handler method is then not created empty, but automatically contains the source code for firing the outbound plug.



The entry point in a Web Dynpro view sequence, in other words, a Web Dynpro window, is always a Web Dynpro application. It points to the window or the interface view of the same name and the inbound plug **DEFAULT** that was automatically created for it.

You create a Web Dynpro application from the context menu of the **Object Navigator** (as illustrated in the figure). It is connected to the component, the interface view, and the standard plug in the dialog box that follows.

You can test a Web Dynpro application directly from the development environment using a corresponding button. Alternatively, you can enter the assigned URL directly in the address bar of a Web browser.



#### LESSON SUMMARY

You should now be able to:

- Implement navigation in Web Dynpro

## Unit 10

### Lesson 3

# Implementing Data Transport in Web Dynpro

#### LESSON OVERVIEW

This lesson explains how to program a simple Web Dynpro for ABAP application with input/output fields and data processing.

#### Business Example

You want to create a Web Dynpro component with several views and data transport between these views. For this reason, you require the following knowledge:

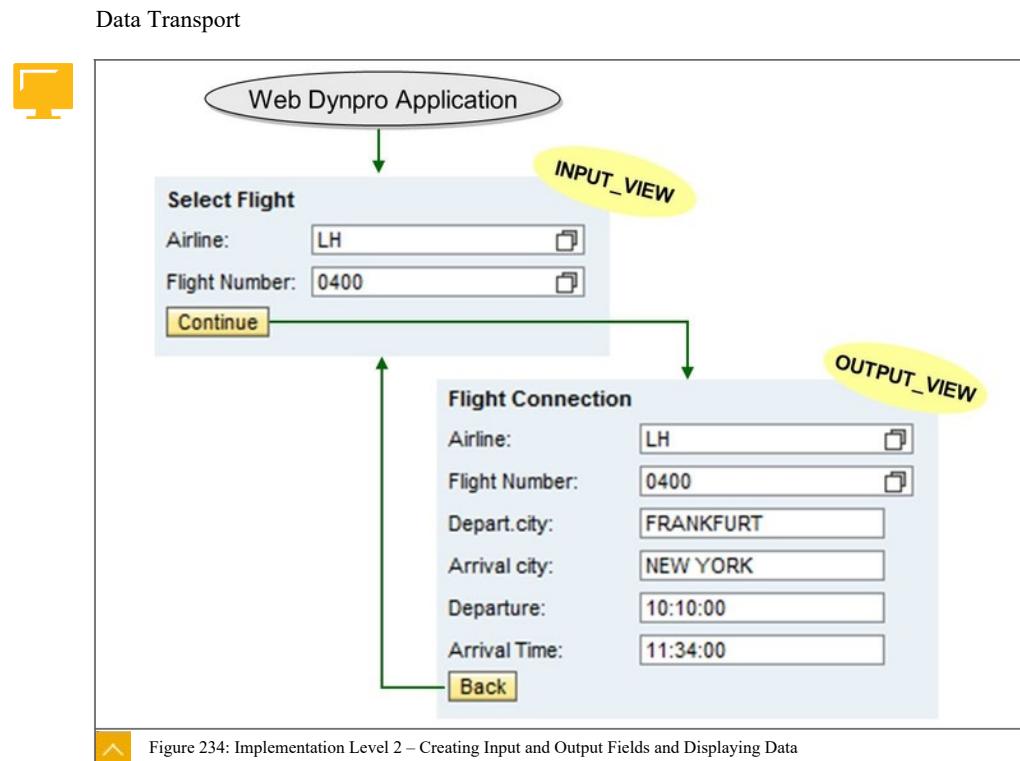
- An understanding of how to generate a service call
- An understanding of how to define context mapping
- An understanding of how to define data binding
- An understanding of how to create and arrange elements on the view layout



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Implement data transport in Web Dynpro

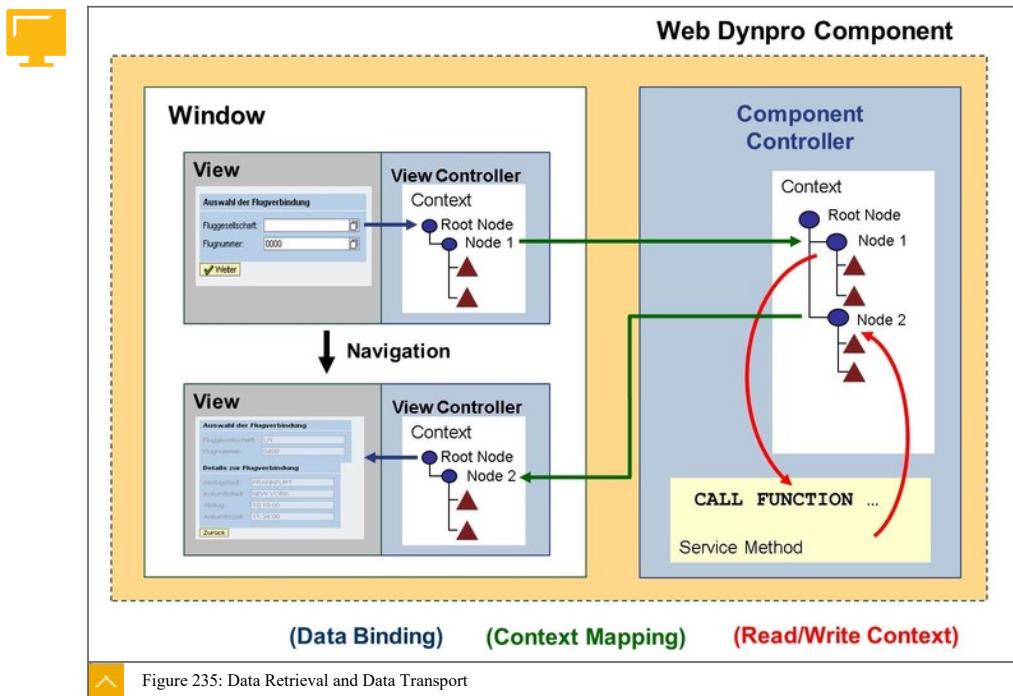


At the second level of the example, the Web Dynpro views should be extended with the input and output fields, and data acquisition and transport should be implemented.

In the first view, the user should select a flight connection using the airline ID and the flight number. When the button is pressed, this information should be further processed in the program, and detailed information about the airline should be read from the database.

After navigation, the second view should display the detailed information.

## Data Retrieval and Data Transport



To correspond to the MVC programming model, each data retrieval of a Web Dynpro component should be encapsulated centrally in a Component Controller method. In this case, this service call should contain a function module call. The data, which the function module requires, is taken from the context of the component controller. Accordingly, after the function module is called, the result is stored again in the context.

Data is transported between the central component controller and the individual view controllers using context mapping. This reference occurs from one context to another, so the data transport functions automatically in both directions.

Data is transported between the input and output fields of a view and the context for this view by data binding.

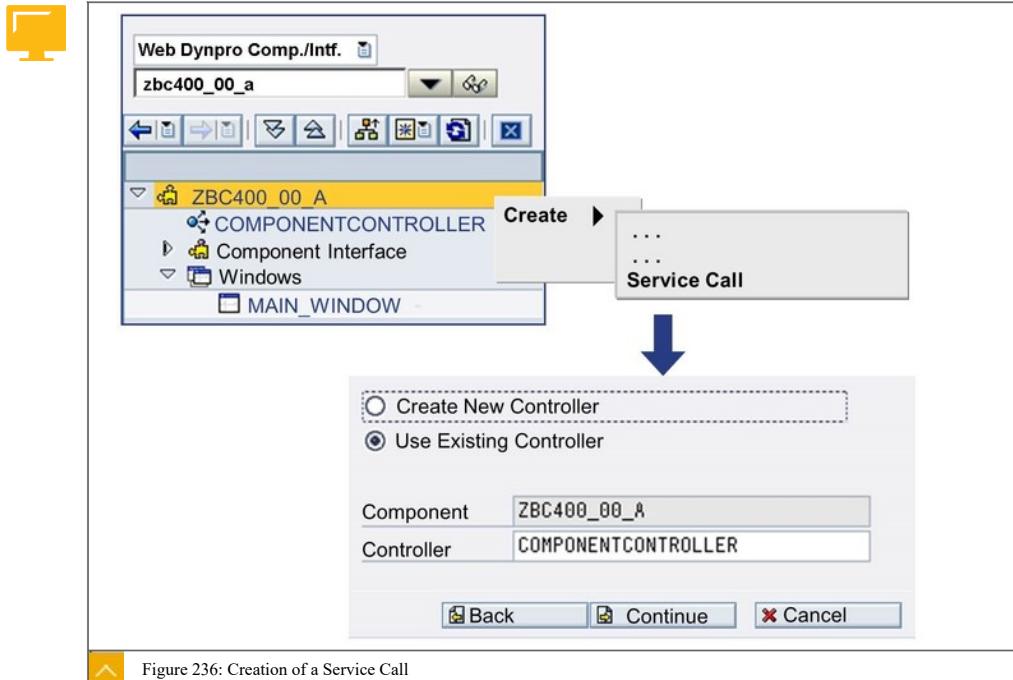
## Tasks for Data Retrieval and Data Transport



- You need to perform the following tasks:
  - Create a Web Dynpro service call.
  - Set up context mapping.
  - Create input or output fields.
  - Set up data binding.
  - Call the service method for data retrieval.

**Note:**  
You do not need to create the controller context. The service call wizard creates the controller context for you, based on the interface of the function module.

### Service Call Generation



The ABAP Workbench provides a Web Dynpro wizard for creating service calls, which you can address in the navigation area of the Object Navigator using the context menu. After you have started the wizard, you need to select the controller in which you want to create the service call. In this case, this is the existing component controller. You can create an additional controller in the wizard as well.

However, the wizard still needs to know which function module is to be called, the name of the service method, and which function module parameters to supply with actual parameters when the function module is called.

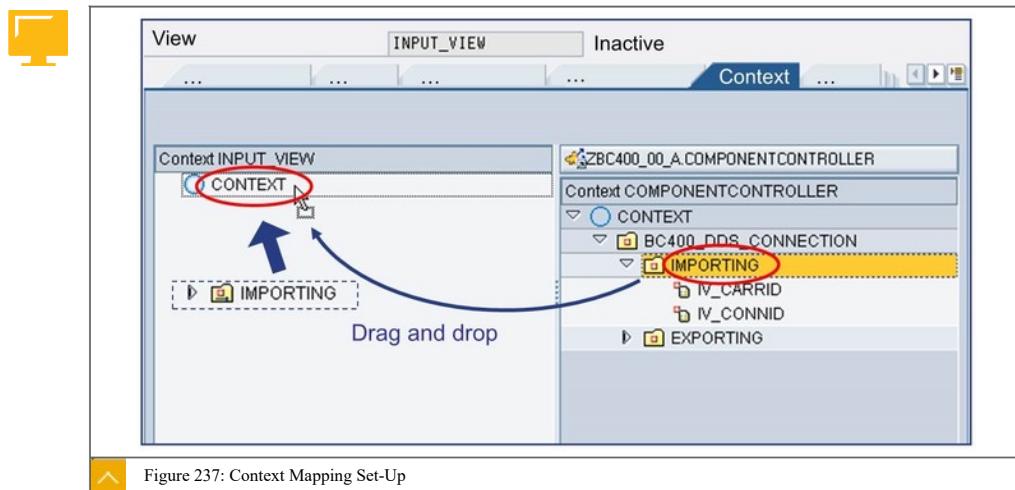
#### Generation of Components by the Wizard



- The wizard generates the following components in the selected controller:
  - A service method (name begins with "EXECUTE\_")
  - Context nodes for all function module parameters that have to be supplied during the call
  - The function module call in the service method

- The source code for reading the data from the context
- The source code for filling the context with the result of the function module call

### Context Mapping

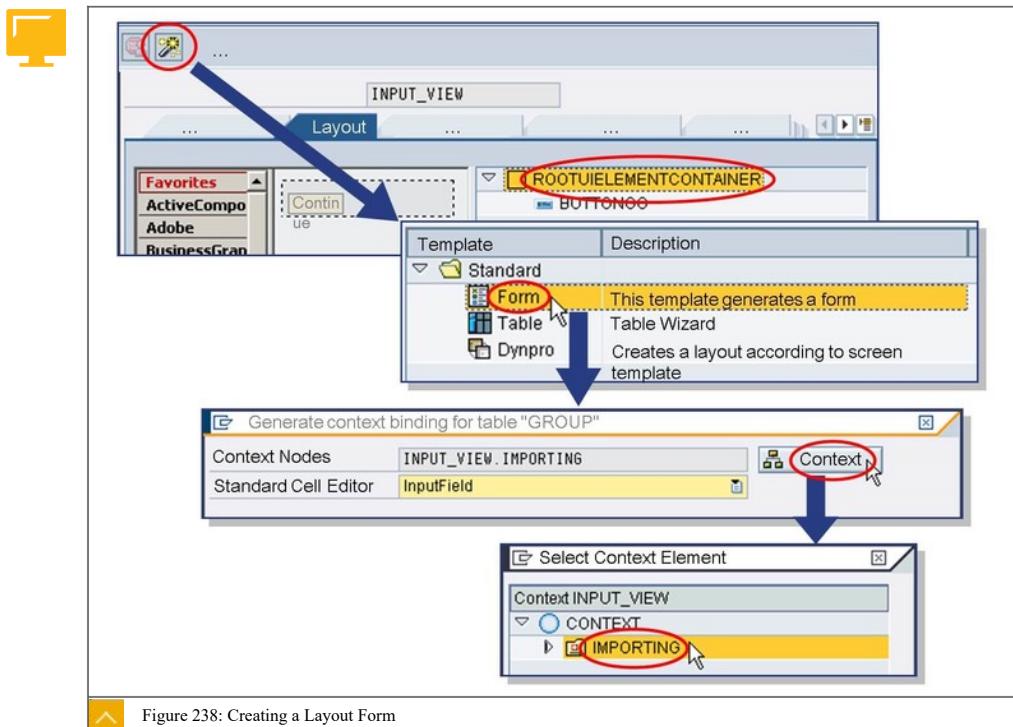


To make the data from the component controller context available in the views, copy the corresponding nodes in the view controller context and map them to the component controller nodes. Context mapping is also necessary to make user entries available on a view in the component controller.

To copy the nodes at the same time as you set up context mapping, open the corresponding view in change mode and choose the **Context** tab page.

On the left, you will see the view controller context (still empty, apart from a root node named CONTEXT) and on the right, the generated component controller context. Copy a node by dragging it from the right and dropping it on the root node for the view controller context on the left.

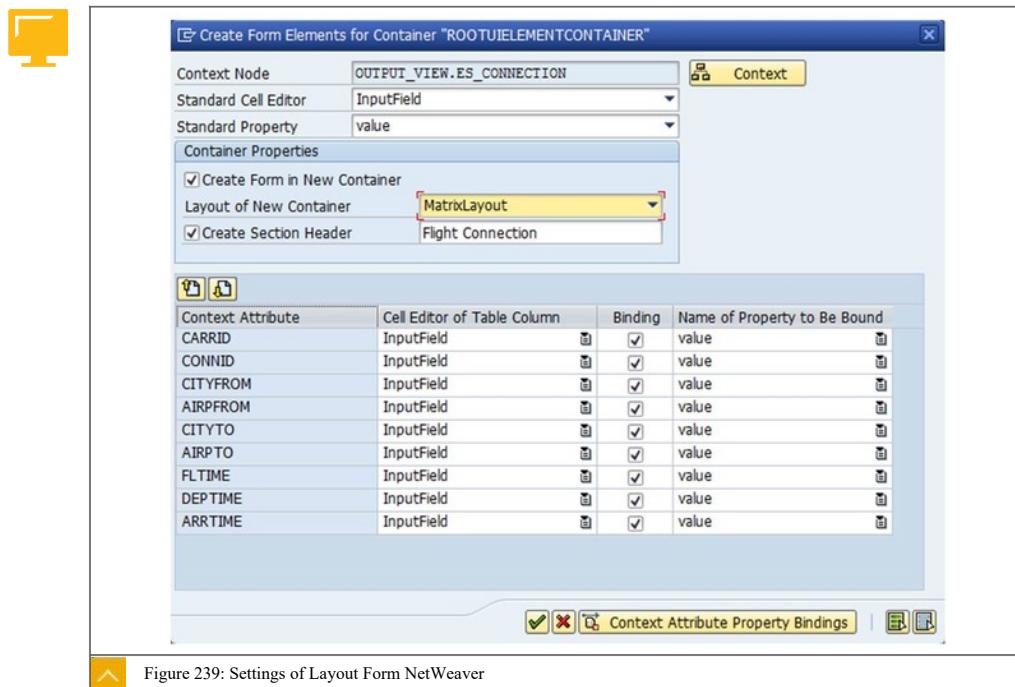
### Data Binding



The ABAP Workbench also provides a wizard for creating input or output fields and setting up data binding. The prerequisite to use this wizard is that the node that is to be bound already exists in the view controller context. If this is the case, it is easy to create a layout form for this node.

Open the view in change mode and choose the **Layout** tab page. Choose entry **ROOTUIELEMENTCONTAINER** in the element hierarchy (top-right). Start the Web Dynpro wizard using the corresponding button (as illustrated in the figure) and select the **Form** template. Choosing **Form** takes you to a dialog in which you can select the node for which you want to create the form directly.

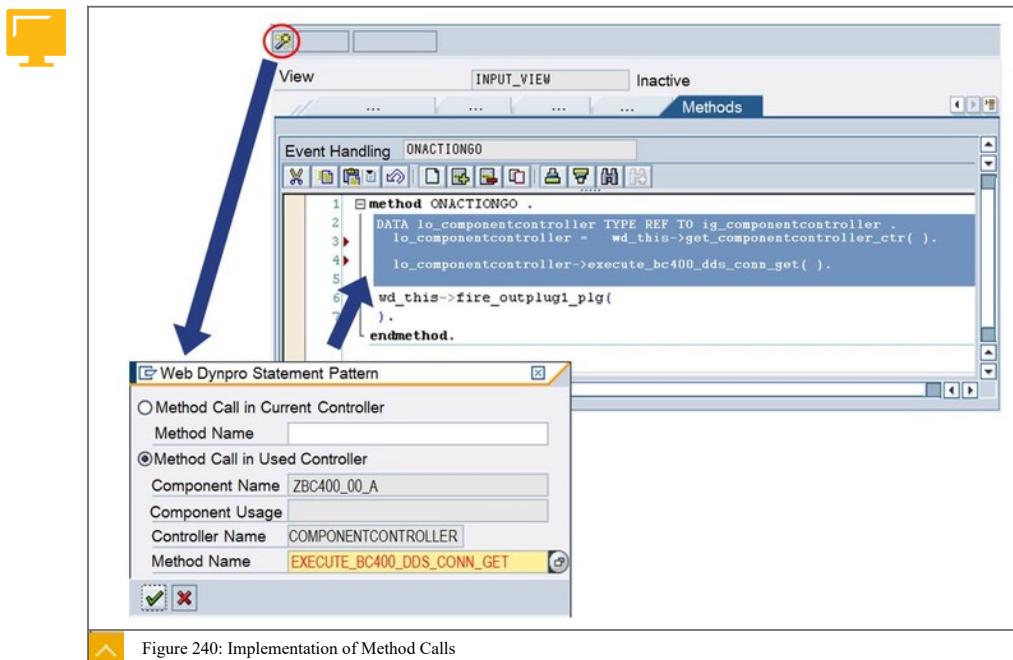
## Settings of Layout Form NetWeaver



Back in the dialog in which you have chosen the **Context** button, you can adapt the form in some detail. If you decide to have an own UI container for the form, you can also decide which layout should be used for it.

In addition, you can enter a text for the header of the form. If the node has several fields, you can select or exclude these fields individually. You can choose the elements to be generated from a list of simple UI elements.

### Implementation of Method Calls



To enable the flight connection details to be displayed on the second view, the service method has to be called before you navigate to the view. In the example, this call is made in the Action Handler method (ONACTION... method) of the input view.

You can also generate method calls with the Web Dynpro wizard. To do this, open the source code for the Action Handler method, position the cursor in front of the source code to fire the outbound plug, and click the appropriate button to launch the wizard (as illustrated in the figure).

On the following screen, select the  Method Call in Used Controller radio button and use input help to enter the component controller as the used controller. Use input help again to choose the method you want to call.

To conclude the activity, the wizard generates the source code to call the method.



### LESSON SUMMARY

You should now be able to:

- Implement data transport in Web Dynpro

## Unit 10

### Learning Assessment

1. What starts the screen sequence defined in the window of a Web Dynpro Component?

Choose the correct answer.

- A Web Dynpro structure
- B Web Dynpro application
- C Web Dynpro component
- D Web Dynpro content

2. What can be defined inside of a Web Dynpro component?

Choose the correct answers.

- A View
- B Component controller
- C Application
- D Window
- E Dynpro

3. What is used to transport data between the central component controller and the individual view controllers?

Choose the correct answer.

- A Service call
- B Context mapping
- C Data binding
- D Input/output fields

## Unit 10

### Learning Assessment - Answers

1. What starts the screen sequence defined in the window of a Web Dynpro Component?

Choose the correct answer.

- A Web Dynpro structure  
 B Web Dynpro application  
 C Web Dynpro component  
 D Web Dynpro content

You are correct! The screen sequence defined in the window is started by way of the Web Dynpro application. Read more in the lesson, Describing Web Dynpro ABAP, Task: Application Example, in the course BC400.

2. What can be defined inside of a Web Dynpro component?

Choose the correct answers.

- A View  
 B Component controller  
 C Application  
 D Window  
 E Dynpro

You are correct! A component contains one or several windows and has its own controller, the COMPONENTCONTROLLER. Windows embed one or more views and define navigation options between them (view sequences). Read more in the lesson, Describing Web Dynpro ABAP, Task: Web Dynpro Component, in the course BC400.

3. What is used to transport data between the central component controller and the individual view controllers?

Choose the correct answer.

A Service call

B Context mapping

C Data binding

D Input/output fields

You are correct! Data is transported between the central component controller and the individual view controllers using context mapping. Read more in the lesson, Implementing Data Transport in Web Dynpro, Task: Data Transport, in the course BC400.

## UNIT 11

# Program Analysis Tools

Lesson 1

Improving the Quality of ABAP Code with the Code Inspector

334

### UNIT OBJECTIVES

- Describe the Code Inspector
- Use the Code Inspector

## Unit 11

### Lesson 1

# Improving the Quality of ABAP Code with the Code Inspector

#### LESSON OVERVIEW

This lesson explains the basic function and purpose of the Code Inspector. This is only a brief introduction to the tool. You will find more information in the relevant documentation or by attending the advanced courses.

#### Business Example

You need to check your programs for performance, typical semantic programming errors, and security gaps. For this reason, you require the following knowledge:

- An understanding of the purpose and use of the Code Inspector
- An understanding of the most important properties of the Code Inspector
- An understanding of how to use the Code Inspector for simple analysis of your programs



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe the Code Inspector
- Use the Code Inspector

#### The Code Inspector

The Code Inspector offers you the option of analyzing your programs with regard to performance, security, and typical semantic errors. Here are a few examples of what you can check:

#### Typical Quality Issues with ABAP Code



- Typical Performance Issues
  - Are indexes used for database access?
  - Are SELECT statements embedded in loops?
- Typical Security Issues
  - Is data read from a client other than the login client?
  - Is the database table or the WHERE clause dynamically specified in the SELECT statement?
- Typical Semantic Errors

- Is the sy-subrc field checked after each AUTHORITY-CHECK statement?
- Is a client actually specified for CLIENT SPECIFIED?

### Calling of the Code Inspector

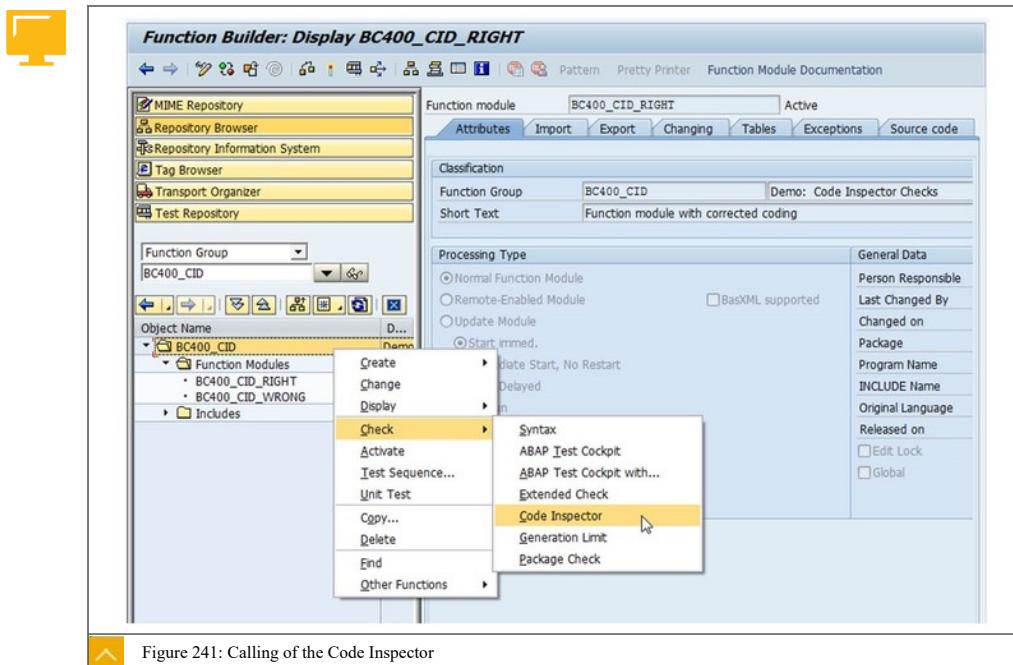


Figure 241: Calling of the Code Inspector

There are several different ways to call the Code Inspector.

The figure shows one possibility. Call the Code Inspector from the context menu of the object list by choosing **Check → Code Inspector** and perform a standard inspection. You will get the same call if you choose **Function Module → Check → Code Inspector**.

### Use of the Code Inspector

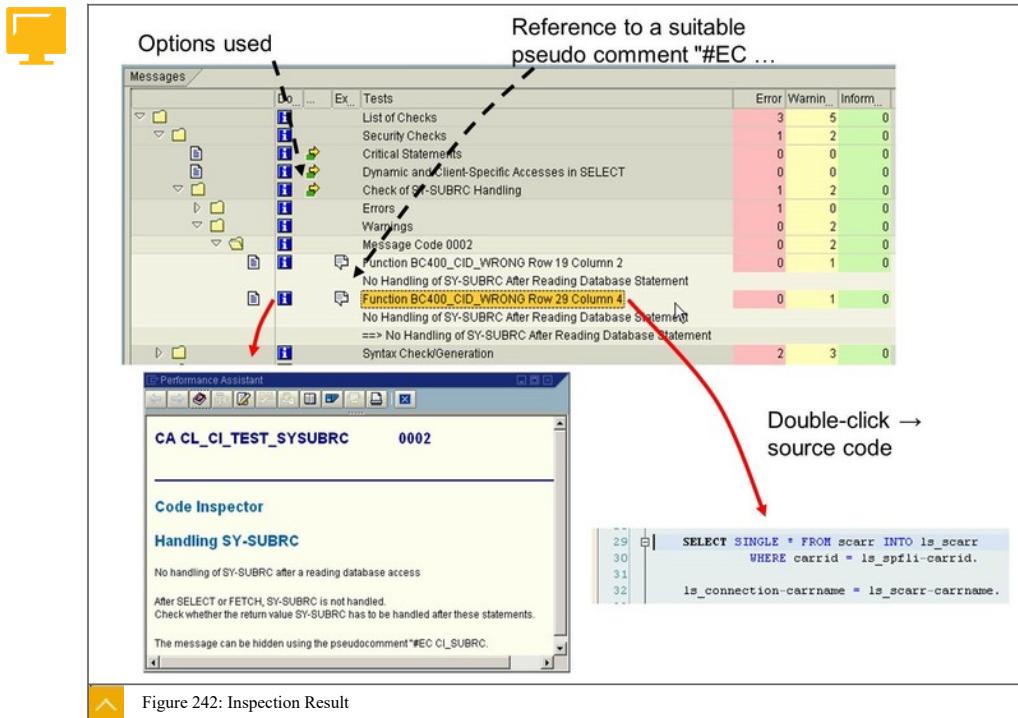


Figure 242: Inspection Result

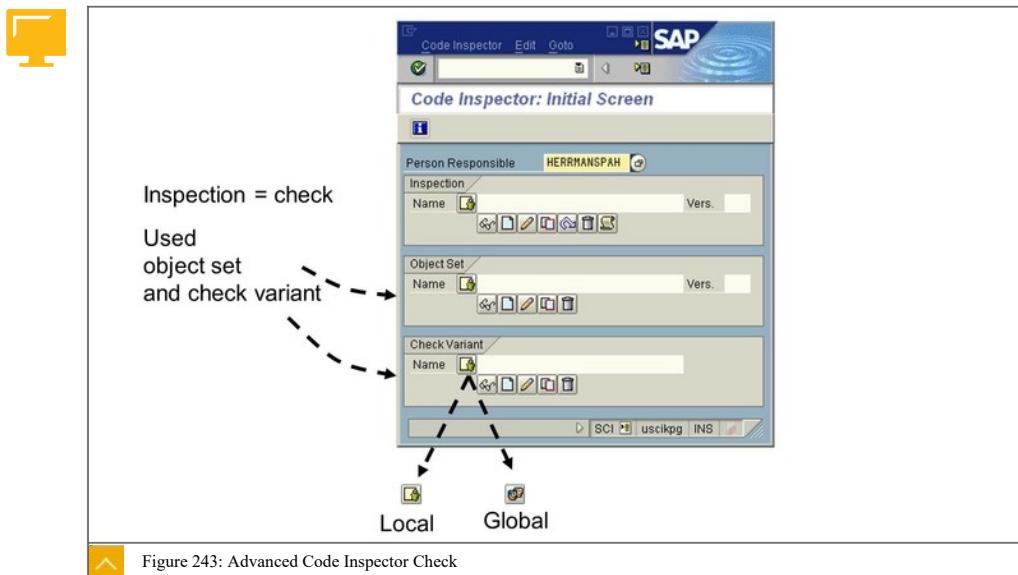
As the result of an inspection, you receive a list of error and warning messages. The information button that belongs to the message displays a detailed error description as well as suggestions for improvement. However, if you double-click the error text, you branch to the corresponding program statement.

In certain instances, it is practical, or even essential, to write ABAP source code that the Code Inspector would query, such as a transaction call. You can provide the comment "#EC \* for such commands, which is the same action to be taken with an extended syntax check.

When you initiate a check from inside the ABAP Editor, the Code Inspector uses a default check variant with a predefined set of checks to be performed. You can overwrite this default check variant by creating a new check variant with the name DEFAULT.

However, this new check variant will override the standard variant for your user in all clients. If you delete your default check variant, the standard variant is automatically used for future checks. For this reason, we recommend that you define your own check variants instead of overwriting the default variant.

## Definition of Check Variants, Object Sets, and Inspections



To define individual checks, start the Code Inspector with transaction code **SCI**, or from the SAP Easy Access menu by choosing Tools → ABAP Workbench → Test → Code Inspector .

## Areas of the Code Inspector

- The initial screen contains the following areas:

- Check Variant

The check variant determines which checks to perform in the system, such as the programming conventions or a performance check.

- Object Set

The object set determines which Repository objects to check.

- Inspection

Inspection refers to the name of the actual check. To save the inspection and its results or execute it in the background, you must give it a name. If you do not enter a name for the inspection, the system does not save the result.

Inspection uses a previously defined check variant and object set. However, it is also possible to create temporary object sets and check variants that are only valid for one inspection and cannot be reused in other inspections.

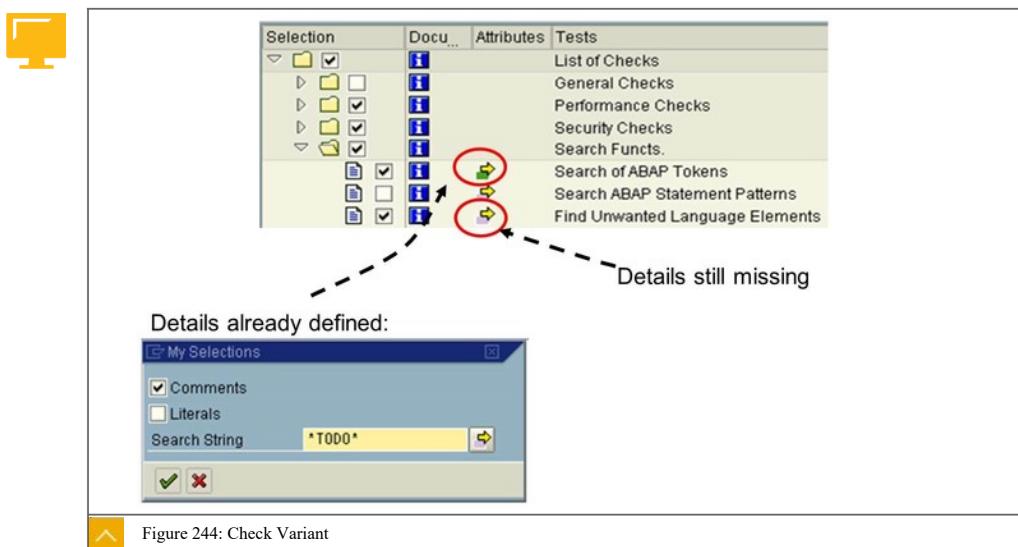


## Hint:

You can create check variants, object sets, and inspections either as private or public. To switch between these two categories, use the button that is placed in the input field. Note that only you can use your private objects whereas public objects are available to all users of the system.

A check variant consists of one or more check categories, which in turn consist of one or more single checks (inspections). You can parameterize these single checks by using a keyword or an indicator for a specific partial aspect of the check. In most cases, single checks investigate a specific object type, such as the "Check of Table Attributes", which only examines DDIC tables.

#### Check Variant



#### Check Categories

The system assigns single checks to various check categories. The most important check categories are as follows:

##### General Checks

General checks contain data formatting, such as the list of table names from SELECT statements.

##### Performance Checks

Performance checks contain checks for performance and resource usage, such as an analysis of the WHERE condition for SELECT or UPDATE and DELETE, SELECT statements that bypass the table buffer, and low-performance access to internal tables.

##### Security Checks

Security checks contain checks of critical statements, cross-client queries, and insufficient authority checks.

##### Syntax Check and Generation

Syntax check and generation contain the ABAP syntax check, an extended program check, and generation.

##### Programming Conventions

Programming conventions contain checks of naming conventions.

##### Search Functions

Search functions contain searches for tokens (words) and statements in ABAP source code.



#### LESSON SUMMARY

You should now be able to:

- Describe the Code Inspector
- Use the Code Inspector

## Unit 11

### Learning Assessment

1. Which of the following statements about the Code Inspector are true?

Choose the correct answers.

- A You can only use the DEFAULT check variant.
- B You can create a check variant to define the details of what to check.
- C Standard inspection is carried out when check is performed through context menu from the Object Navigator.
- D You can define check variants, object sets, and inspections using transaction SCI.

2. Which aspects does the Code Inspector take into consideration when examining a program?

Choose the correct answers.

- A Syntax check
- B Typical semantic errors (for example, AUTHORITY-CHECK statement without subsequent SY-SUBRC check)
- C Performance (for example, nested SELECT statements)
- D Security (for example, cross-client data accesses)
- E Formatting of the source code (for example, indenting of the program lines within loops)

## Unit 11

### Learning Assessment - Answers

1. Which of the following statements about the Code Inspector are true?

Choose the correct answers.

- A You can only use the DEFAULT check variant.
- B You can create a check variant to define the details of what to check.
- C Standard inspection is carried out when check is performed through context menu from the Object Navigator.
- D You can define check variants, object sets, and inspections using transaction SCI.

You are correct! When you initiate a check from inside the ABAP Editor, the Code Inspector uses a default check variant with a predefined set of checks to be performed. You can overwrite this default check variant by creating a new check variant. Calling the Code Inspector from the context menu of the object list will make use of the DEFAULT variant. To define individual checks (check variants, object sets, inspections), start the Code Inspector with transaction code SCI. Read more in the lesson, Improving the Quality of ABAP Code with the Code Inspector, Task: Definition of Check Variants, Object Sets, and Inspections, in the course BC400 (Unit 11, Lesson 1) or TAW10 Part I (Unit 14, Lesson 1).

2. Which aspects does the Code Inspector take into consideration when examining a program?

Choose the correct answers.

- A Syntax check
- B Typical semantic errors (for example, AUTHORITY-CHECK statement without subsequent SY-SUBRC check)
- C Performance (for example, nested SELECT statements)
- D Security (for example, cross-client data accesses)
- E Formatting of the source code (for example, indenting of the program lines within loops)

You are correct! The most important check categories are as follows: General Checks, Performance Checks, Security Checks, Syntax Check and Generation, Programming Conventions, Search Functions. Read more in the lesson, Improving the Quality of ABAP Code with the Code Inspector, Task: Check Categories, in the course BC400 (Unit 11, Lesson 1) or TAW10 Part I (Unit 14, Lesson 1).

## UNIT 12

# ABAP Development Tools for SAP NetWeaver

### Lesson 1

Describing ABAP Development Tools for SAP NetWeaver

343

### Lesson 2

Creating an ABAP Project in Eclipse

346

### UNIT OBJECTIVES

- Describe Eclipse-based ABAP development
- Create an ABAP project in Eclipse

## Unit 12

### Lesson 1

# Describing ABAP Development Tools for SAP NetWeaver

#### LESSON OVERVIEW

This lesson explains the working of the ABAP Debugger for elementary data objects.

#### Business Example

As an ABAP programmer, you can develop ABAP programs with Eclipse-based ABAP development tools as an alternative to using the ABAP Workbench. For this reason, you require the following knowledge:

- An understanding of the ABAP development tools for SAP NetWeaver



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

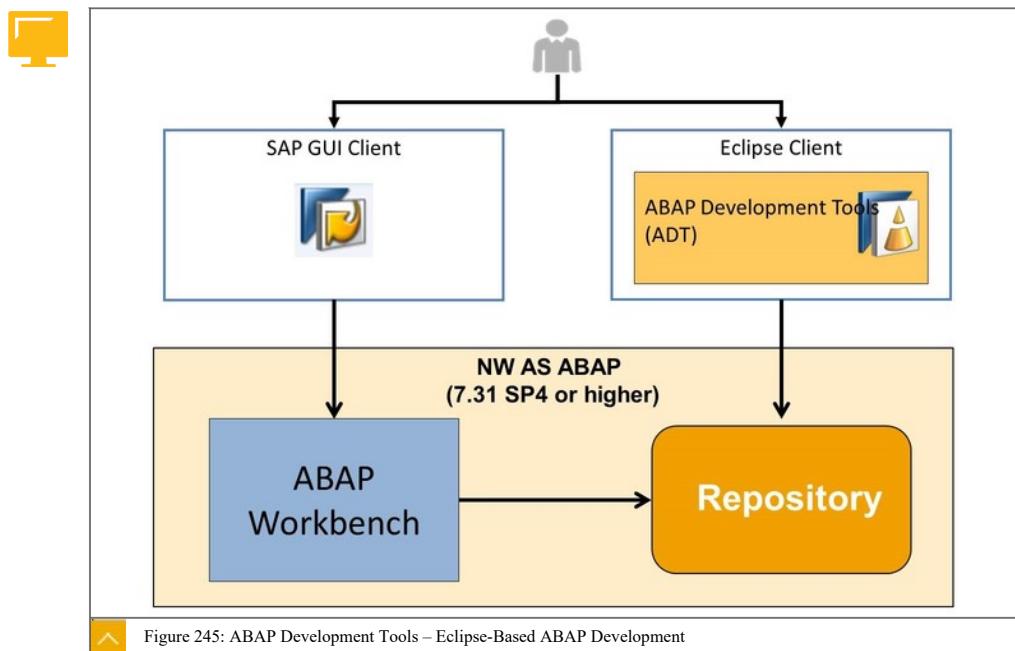
- Describe Eclipse-based ABAP development

#### Eclipse-Based ABAP Development

##### Features of ABAP Development Tools



- ABAP Development Tools (ADTs) are an alternative to the ABAP Workbench. ADTs provide the following features:
  - A new ABAP development experience on top of the Eclipse platform.
  - An open platform for developing new ABAP-related tools.
  - A set of open, language-independent and platform-independent application programming interfaces (APIs) that developers can use to build new custom tools for the ABAP environment.



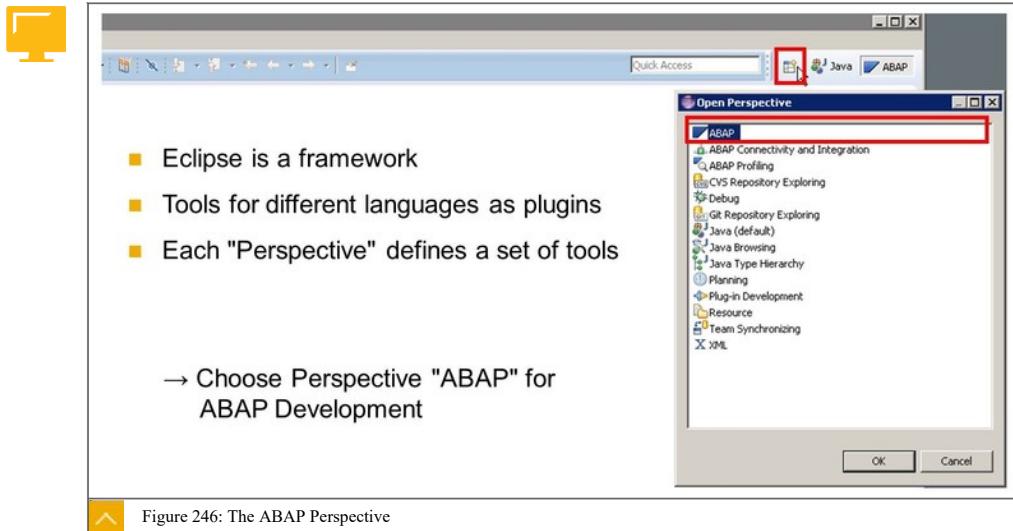
#### Benefits of Eclipse

With ABAP Development Tools in Eclipse, you can capitalize on the usability, speed, and flexibility of Eclipse while also benefitting from proven ABAP Workbench features. It is the best of both worlds. ADTs improve developer productivity by offering better refactoring functionality, code completion, auto-insertion, and code templates. They also include an invaluable Quick Fix feature, and they are easy to navigate.

ADTs allow you to connect to multiple ABAP systems and provide session failover, which reduces the impact of lost connections. ADTs also enable cross-platform development by integrating ABAP and non-ABAP development tools in a single powerful integrated development environment. In addition, ADTs are tightly integrated with SAP HANA studio, SAP UI5 tools, and JAVA.

#### The ABAP Perspective

Eclipse allows development in many different programming languages. For different languages it contains different combinations of tools, called **perspectives**. If you have ADTs installed, you can switch to the ABAP perspective.



### LESSON SUMMARY

You should now be able to:

- Describe Eclipse-based ABAP development

## Unit 12

### Lesson 2

## Creating an ABAP Project in Eclipse

### LESSON OVERVIEW

This lesson explains how to create an ABAP project using Eclipse-based development tools.

#### Business Example

As an ABAP programmer, you can develop ABAP projects using Eclipse-based development tools as an alternative to using the ABAP Workbench. For this reason, you require the following knowledge:

- An understanding of how to create ABAP projects using Eclipse-based development tools.



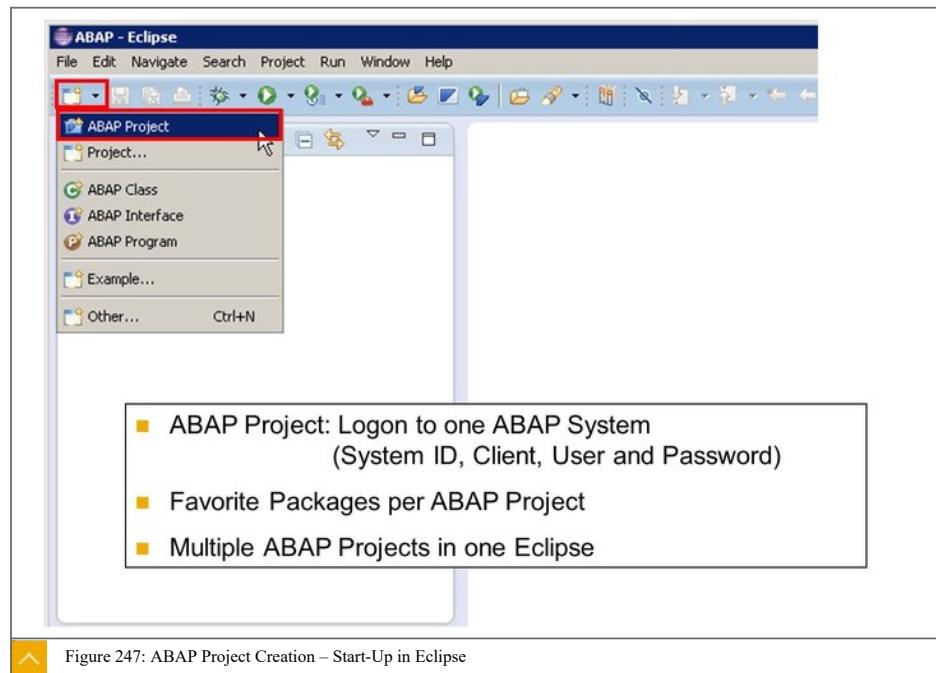
### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create an ABAP project in Eclipse

#### Project Creation

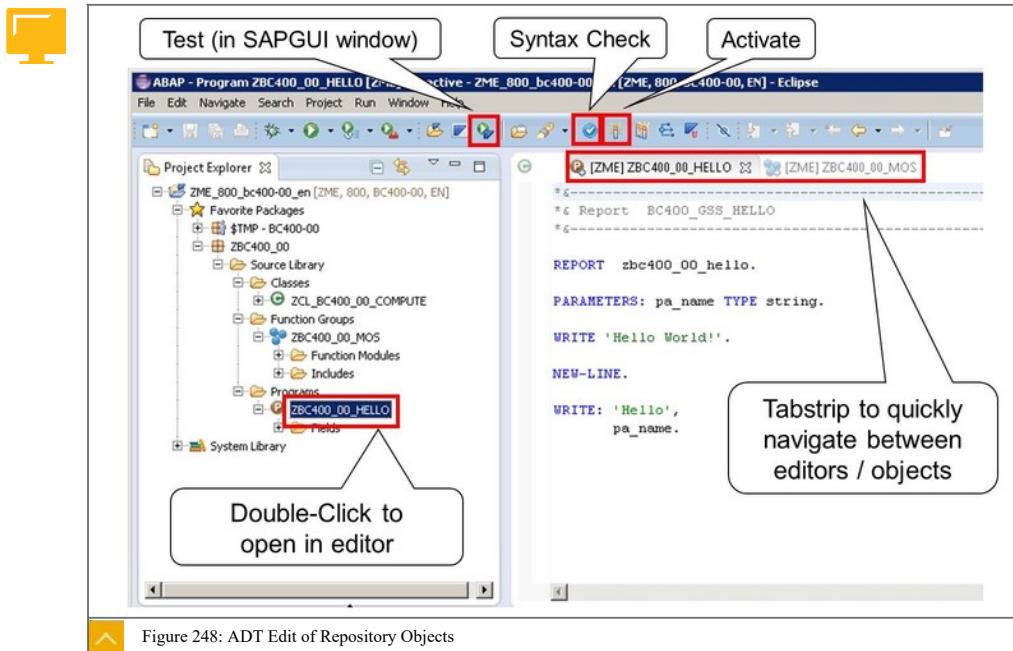
Before you can work with ABAP development tools, you have to connect to a system. This is done by creating an ABAP project. Afterwards, you can open the various Repository objects with the available editors.



When you use ABAP development tools, you log onto an SAP back-end system and work directly with its Repository objects. In this sense, the development process is exactly the same as when you use the ABAP Workbench—there is no check-out and check-in of the objects.

The ABAP project serves as a container for the development objects that are stored in a particular ABAP back-end system, and contains the logon parameters for the system logon: System, Client, User, and language. You must be logged on to the system to edit an object there. Within the project, you can access any development object in the Repository. However, to make it easier to manage the objects, you can set up favorite packages for each project.

### Repository Object Editing



**Project Explorer** provides a hierarchical view on the repository that is very similar to the navigation area of the ABAP Workbench. Under each project you can maintain a list of favorite packages. By expanding the node **System Library** you get a list of all packages.

To open a specific repository in its respective editor, double-click it.

The editor is shown on the right-hand side of the ABAP perspective. For each new object, a new tab is added to the tabstrip. This simplifies the navigation between editors and objects.

### Editors in the ABAP Development Tools

- Eclipse features
- Eclipse-like look & feel
- Available for some Repository Objects (e.g. Programs, Classes Web Dynpro, etc.)

- No new features
- SAP GUI Look & Feel
- Used where no native editor is available (e.g. Transactions, Dictionary Objects, Screens, etc.)

**Native Eclipse Editors**

```

REPORT zbc400_00_hello.

PARAMETERS: pa_name TYPE string.

WRITE 'Hello World!',  
|  
NEW-LINE.  
  
WRITE: 'Hello',  
      pa_name.
  
```

**Integrated SAP GUI Editors**

ZBC400\_00\_HELLO

Display Report Transaction

Transaction code: ZBC400\_00\_HELLO  
Package: ZBC400\_00

Transaction text: Demo: Transaction  
Program: ZBC400\_00\_HELLO  
Selection screen: 1000

Figure 249: Native Editors or Integrated SAP GUI Editors

### Types of Editors

- There are two types of editors in the ABAP development tools:
  - Editors for which there is a native Eclipse implementation  
Example: ABAP Editor, an Eclipse editor
  - Editors that are displayed in an in-place SAP graphical user interface (GUI)  
Example: ABAP transaction editor with the classic SAP GUI visualization appearing within the Eclipse environment

Generally speaking, there is no requirement to use ABAP development tools for ABAP Development, and each developer is free to choose whether to use ABAP development tools or the classic ABAP Workbench. The functions and features of Eclipse in general and the ABAP development tools in particular lend themselves particularly well to object-oriented programming; among them are various built-in refactoring functions and support for unit testing.

### Additional Resources for ABAP Development Tools for SAP NetWeaver



Table 7: Additional Resources for ABAP Development Tools for SAP NetWeaver

<b>Resource</b>	<b>Online Location</b>
ABAP development tools	<ul style="list-style-type: none"> <li>• SAP Software Download Center at <a href="http://support.sap.com/swdc">http://support.sap.com/swdc</a></li> <li>• SAP Development Tools for Eclipse at <a href="http://tools.hana.ondemand.com">http://tools.hana.ondemand.com</a></li> <li>• SAP Community Network at <a href="http://scn.sap.com/community/abap/eclipse">http://scn.sap.com/community/abap/eclipse</a></li> </ul>
Documentation and installation support for SAP Development Tools for Eclipse	<p>SAP Community Network pages:</p> <ul style="list-style-type: none"> <li>• <a href="http://scn.sap.com/community/abap/eclipse">http://scn.sap.com/community/abap/eclipse</a></li> <li>• <a href="http://scn.sap.com/community/abap/hana">http://scn.sap.com/community/abap/hana</a></li> </ul>



### LESSON SUMMARY

You should now be able to:

- Create an ABAP project in Eclipse

## Unit 12

### Learning Assessment

1. Which of the following statements are true?

Choose the correct answers.

- A** ABAP Development Tools (ADTs) are an alternative to the ABAP Workbench.
- B** ADTs allow you to connect to one ABAP system at a time and provide session failover, which reduces the impact of lost connections.
- C** ADTs enable cross-platform development by integrating ABAP and non-ABAP development tools in a single powerful integrated development environment.
- D** Eclipse allows development in many different programming languages. For each language, it contains the same combination of tools.

2. Only ABAP development tools should be used for ABAP development.

Determine whether this statement is true or false.

- True
- False

## Unit 12

### Learning Assessment - Answers

1. Which of the following statements are true?

Choose the correct answers.

**A** ABAP Development Tools (ADTs) are an alternative to the ABAP Workbench.

**B** ADTs allow you to connect to one ABAP system at a time and provide session failover, which reduces the impact of lost connections.

**C** ADTs enable cross-platform development by integrating ABAP and non-ABAP development tools in a single powerful integrated development environment.

**D** Eclipse allows development in many different programming languages. For each language, it contains the same combination of tools.

You are correct! ABAP Development Tools (ADTs) are an alternative to the ABAP Workbench. ADTs also enable cross-platform development by integrating ABAP and non-ABAP development tools in a single powerful integrated development environment. In addition, ADTs are tightly integrated with SAP HANA studio, SAPUI5 tools, and JAVA. Read more in the lesson, Describing ABAP Development Tools for SAP NetWeaver, Task: Benefits of Eclipse, in the course BC400.

2. Only ABAP development tools should be used for ABAP development.

Determine whether this statement is true or false.

True

False

You are correct! As an ABAP programmer, you can develop ABAP programs with Eclipse-based ABAP development tools as an alternative to using the ABAP Workbench. Read more in the lesson, Describing ABAP Development Tools for SAP NetWeaver, Task: Business Example, in the course BC400.

## UNIT 13

# SAP Standard Software Adjustments

Lesson 1

Adjusting the SAP Standard Software

353

### UNIT OBJECTIVES

- Explain terms for adjusting the SAP standard software
- Describe options for adjusting SAP standard software
- Describe enhancement types

## Unit 13

### Lesson 1

# Adjusting the SAP Standard Software

#### LESSON OVERVIEW

This lesson provides an overview of the options for adjusting the SAP standard software to your company's requirements.

#### Business Example

You want to make adjustments to the SAP standard software but want to avoid making modifications which might have disadvantages. For this reason, you require the following knowledge:

- An understanding of terms, such as original, copy, correction, repair, customizing, modification, and enhancement
- An understanding of the available options for adjusting the SAP standard software to your company's requirements
- An understanding of the disadvantages of modifications and the advantages of SAP enhancements
- An understanding of the various enhancement types

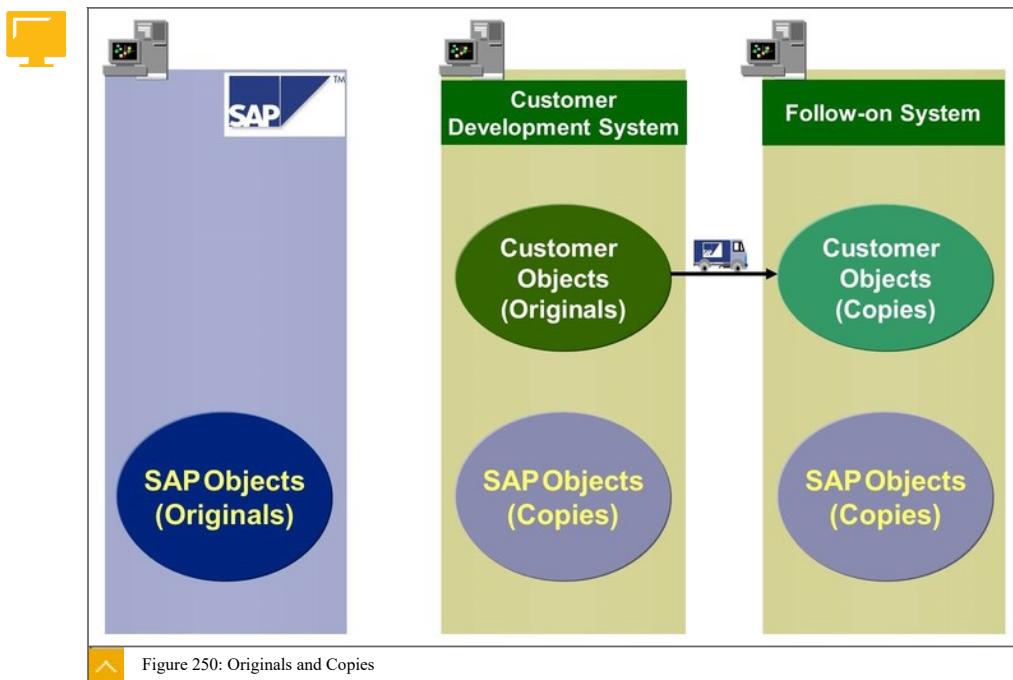


#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain terms for adjusting the SAP standard software
- Describe options for adjusting SAP standard software
- Describe enhancement types

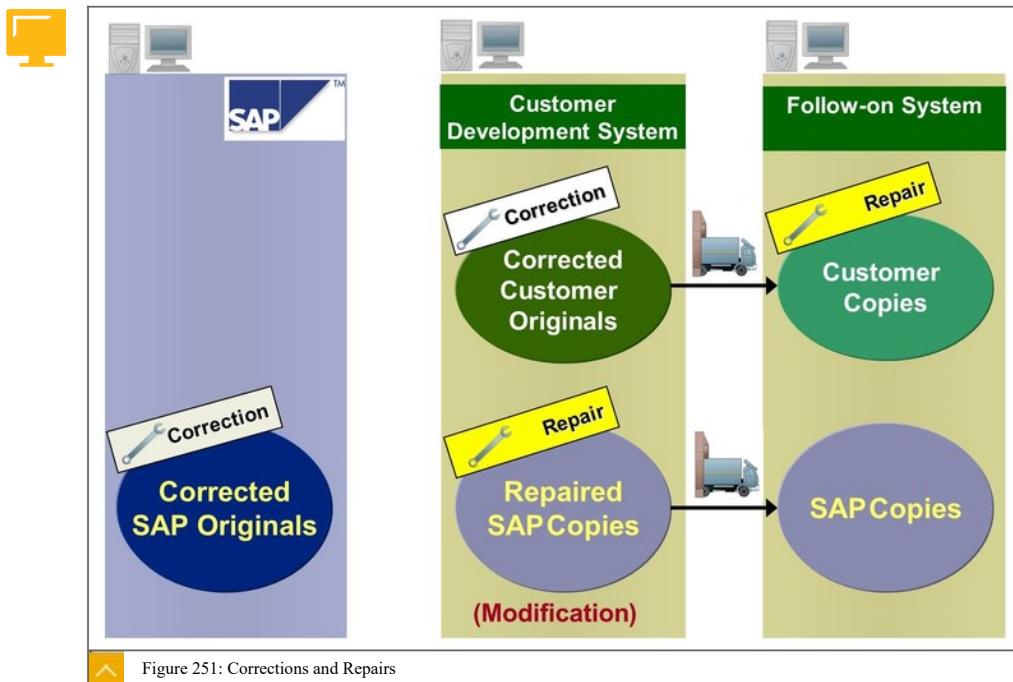
### Basic Terms of the SAP Software Adjustment



In the SAP system where it was originally developed, a Repository object is an original object. In another system where that Repository object is transported, it exists only as a copy. The original system (the birthplace) of a Repository object is noted in the object's attributes.

Best practice for development is to make changes only to the original system's objects; those changes then propagate through transport to subsequent systems. This ensures that Repository objects are consistent across all systems. It is possible to change copies, but this is a dangerous practice, because systems can become inconsistent if the changes are not also made to the original objects. Originals can never be overwritten in transports, so any changes that are made first on a subsequent system must be made manually on the original system.

## Corrections and Repairs



## Comparison of Terms

• **Correction**

Changing an original Repository object is called a correction. You can make corrections in development or correction-type tasks.

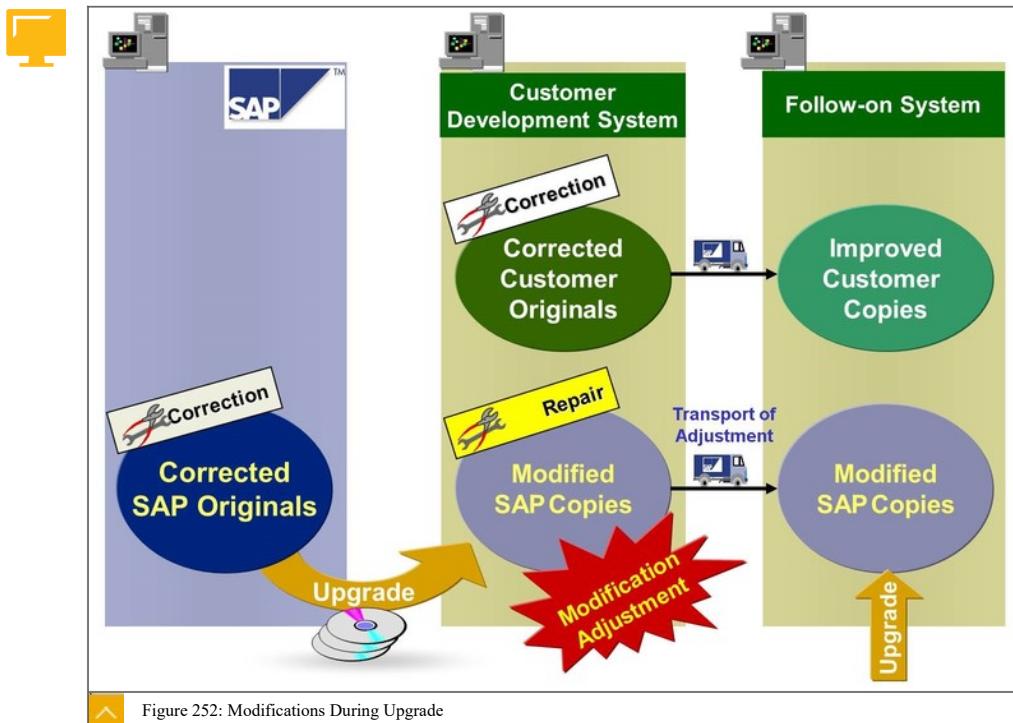
• **Repair**

By contrast, changes to a copy are called repairs. You can make repairs in repair-type tasks. After a customer object has been repaired (for example, to fix a problem in the production system), the change needs to be reproduced in the original to ensure that the systems remain consistent.

• **Modification**

Repairing an SAP repository in a customer system is also called a modification.

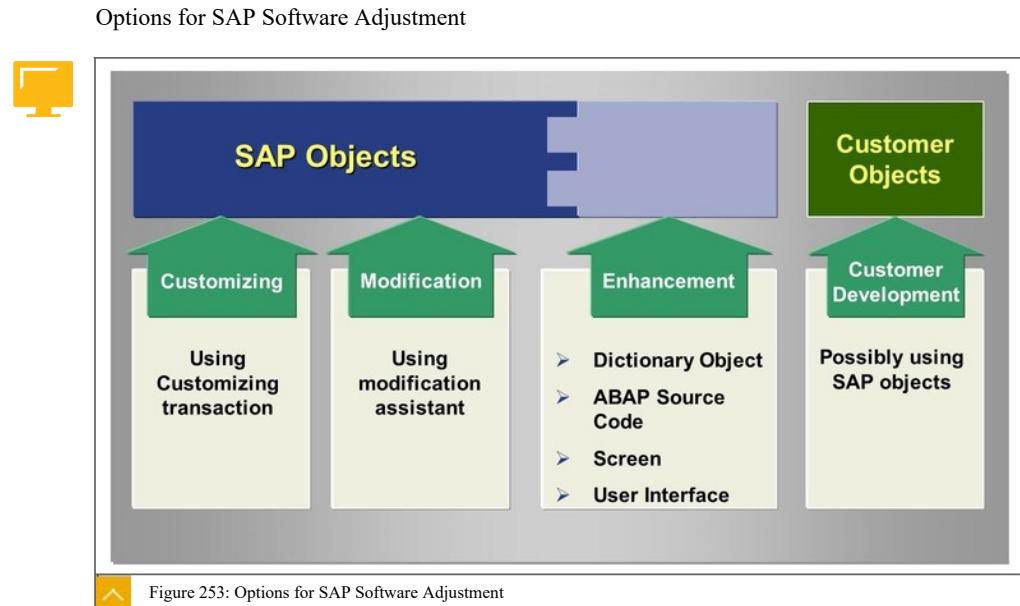
## Modifications During Upgrade



During an upgrade, any modified SAP objects in your system might be overwritten by new SAP-supplied versions of those objects. You will need to make a modification adjustment to get your modifications back into the system. When doing this, you must compare your old, modified version with the newly supplied version and copy earlier changes to the SAP standard into the new version.

Such a comparison can be very time-consuming and is error-prone. It is therefore advantageous to use an SAP enhancement instead of a modification. Enhancements provide the option of adjusting SAP functionality but are not dependent on release version and thus require much less maintenance.

To ensure consistency between the development system and the subsequent systems, you should make the modification adjustment in the development system and then transport the adjusted objects to subsequent systems (by releasing the request you used).



#### Adjustment Options for SAP Standard Software to Meet Customer Requirements

The options for adjusting the SAP standard software to meet customer requirements are as follows:

##### **Customer developments**

You can develop your own Repository objects under the customer name space. This might be necessary when there is customer-required functionality that is not available in the SAP standard release.

##### **Customizing**

You can set certain system properties and functions by means of appropriate maintenance transactions. SAP has built such adjustments into the software design, and customizing is a mandatory part of setting up a new SAP system.

##### **Enhancements**

SAP Repository objects can be adjusted without modifications by means of enhancements. Enhancements are release-independent and do not require any adjustments. However, not all customer requirements are covered by the available enhancement options.

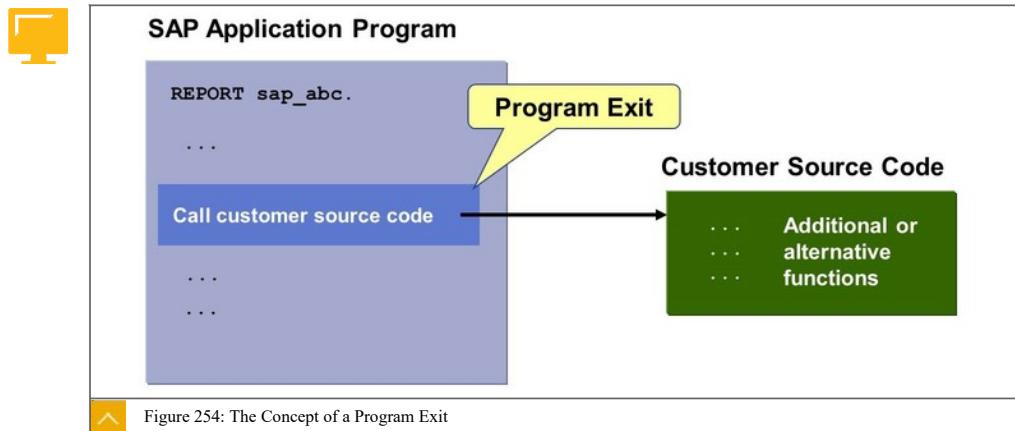
##### **Modifications**

Modifications should be made when there is no appropriate enhancement option in the system but the changed functionality is a customer requirement. Modifications impose a significant maintenance workload due to the ongoing necessity for adjustments. To help organize the modification workload and to facilitate any subsequent adjustments, the Modification Assistant has been integrated in the system since SAP R/3 4.6b.

For detailed information about the Modification Assistant, refer to the online documentation for the ABAP Workbench (changes to the SAP standard software).

## Program Exit

### The Concept of a Program Exit



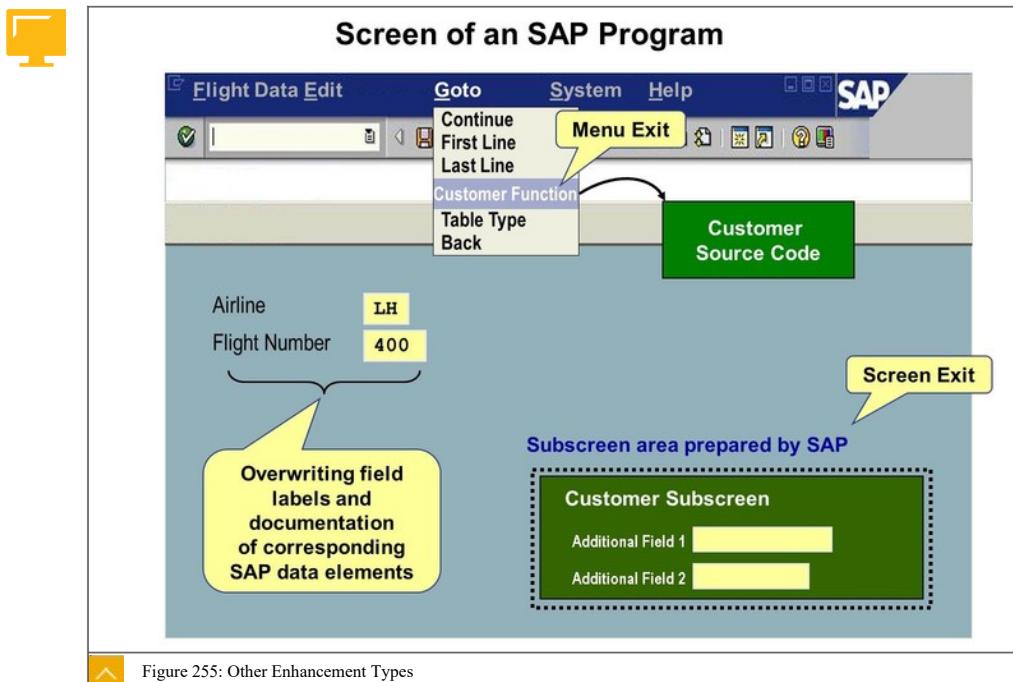
SAP has implemented program exits in some SAP programs, which are places in the code where a customer can attach his or her own source code. This code is processed at runtime, which provides the ability to run customer-created functions without changing the SAP program itself. Program exits are an extremely powerful type of enhancement.

### Available Program Exits



- SAP provides the following types of program exits:
  - User Exits
  - Customer Exits
  - Business Transaction Events (BTEs)
  - Business Add-Ins (BAdIs)

## Other Enhancement Types



## Other Enhancement Types

- In addition to program exits, the following enhancement types exist:

- Menu exit

Some SAP menus contain menu exits, which work similarly to program exits; customers can link their own code to these menus.

- Screen exit

Some screens contain SAP subscreen areas in which you can integrate your own screens.

## Changes to SAP Data Elements in the ABAP Dictionary

- You can make the following changes to SAP data elements in the ABAP Dictionary:

- Overwriting the field documentation

You can overwrite the SAP field documentation that is displayed when you press F1 with your own texts.

- Overwriting the field labels

You can overwrite the field labels of an SAP data element with your own texts.

The enhancement types already discussed generally involve introducing code which runs at various places in the SAP application.

### Additional Enhancement Techniques



- Append structure to database tables

You can attach an append structure with additional fields to most transparent tables. Using that structure, you can implement additional columns in the database tables.

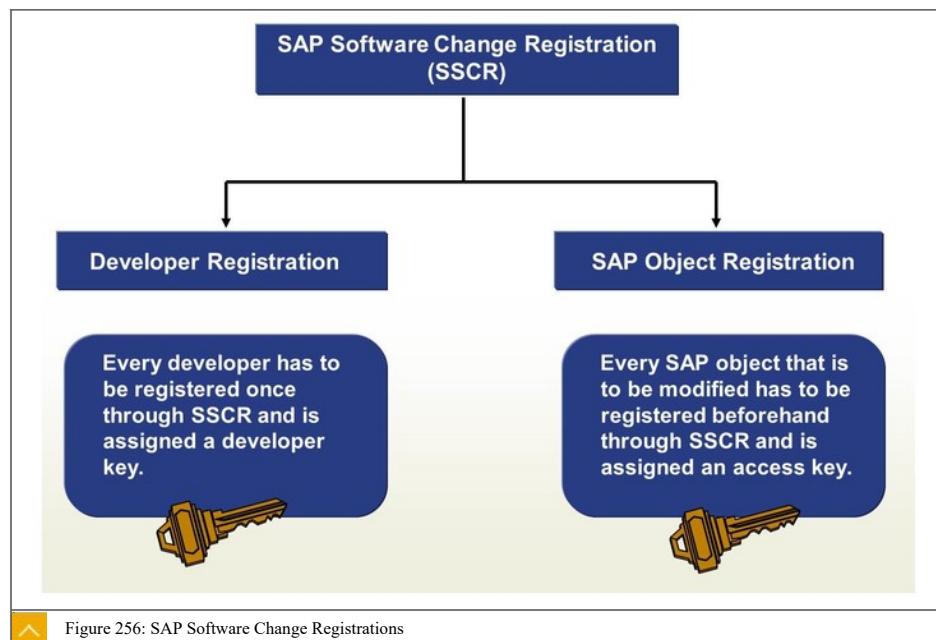
- Append structure to dictionary structure types

You can attach an append structure with additional fields to most dictionary structure types.

- Append structure to database views

You can attach an append structure with additional fields to most database views. This allows you to select additional fields from the base tables given from the view definition but you cannot add new base tables.

### SAP Software Change Registrations



Whenever a developer creates or changes a Repository object for the first time, the system will prompt them for their developer key. The developer can obtain this key by means of a corresponding one-time SAP Software Change Registration (SSCR) developer registration. The developer key is linked to the developer's user ID and the license number of the SAP system.

The developer will need an object-related access key (object key) for every SAP Repository object that is to be modified. They can get these keys by means of SSCR object registration. When you register, specify the Repository object name, the object type, and the SAP system license number and its release.



### LESSON SUMMARY

You should now be able to:

- Explain terms for adjusting the SAP standard software
- Describe options for adjusting SAP standard software
- Describe enhancement types

## Unit 13

### Learning Assessment

1. What is a repair of an SAP repository object in a customer system called?

Choose the correct answer.

- A Modification adjustment
- B Customer exit
- C Modification
- D Correction

2. Which of the following are the options for adjusting the SAP software to customer requirements?

Choose the correct answers.

- A Customizing
- B Enhancements
- C Development

3. Which of the following are enhancement types?

Choose the correct answers.

- A Menu Exit
- B Business Application Programming Interface (BAPI)
- C Screen Exit
- D Customer Event Exit

## Unit 13

### Learning Assessment - Answers

1. What is a repair of an SAP repository object in a customer system called?

Choose the correct answer.

- A Modification adjustment
- B Customer exit
- C Modification
- D Correction

You are correct! Repairing an SAP repository in a customer system is also called a modification. Read more in the lesson, Adjusting the SAP Standard Software, Task: Comparison of Terms, in the course BC400.

2. Which of the following are the options for adjusting the SAP software to customer requirements?

Choose the correct answers.

- A Customizing
- B Enhancements
- C Development

You are correct! Customizing, Enhancements and Customer developments are options for adjusting the SAP software. Read more in the lesson, Adjusting the SAP Standard Software, Task: Adjustment Options for SAP Standard Software to Meet Customer Requirements, in the course BC400.

3. Which of the following are enhancement types?

Choose the correct answers.

**A** Menu Exit

**B** Business Application Programming Interface (BAPI)

**C** Screen Exit

**D** Customer Event Exit

You are correct! In addition to program exits, the following enhancement types exist:  
Menu Exits and Screen Exits. Read more in the lesson, Adjusting the SAP Standard Software, Task: Other Enhancement Types, in the course BC400.