BC405

Programming ABAP Reports

SAP NetWeaver

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Date _____

Training Center _____

Instructors _____

_____

Education Website _____

Participant Handbook

Course Version: 92

Course Duration: 5 Day(s)

Material Number: 50099916

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

Internal Use SAP Partner Only

Copyright

Copyright © 2011 SAP AG. 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 AG. The information contained herein may be changed without prior notice.

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

Trademarks

- Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.
- IBM®, DB2®, OS/2®, DB2/6000®, Parallel Sysplex®, MVS/ESA®, RS/6000®, AIX®, S/390®, AS/400®, OS/390®, and OS/400® are registered trademarks of IBM Corporation.
- ORACLE® is a registered trademark of ORACLE Corporation.
- INFORMIX®-OnLine for SAP and INFORMIX® Dynamic ServerTM are registered trademarks of Informix Software Incorporated.
- UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.
- Citrix®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc.
- HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- JAVA® is a registered trademark of Sun Microsystems, Inc.
- JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
- SAP, SAP Logo, R/2, RIVA, R/3, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo and mySAP.com are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

Disclaimer

THESE MATERIALS ARE PROVIDED BY SAP ON AN "AS IS" BASIS, AND SAP EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR APPLIED, INCLUDING WITHOUT LIMITATION WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THESE MATERIALS AND THE SERVICE, INFORMATION, TEXT, GRAPHICS, LINKS, OR ANY OTHER MATERIALS AND PRODUCTS CONTAINED HEREIN. IN NO EVENT SHALL SAP BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR PUNITIVE DAMAGES OF ANY KIND WHATSOEVER, INCLUDING WITHOUT LIMITATION LOST REVENUES OR LOST PROFITS, WHICH MAY RESULT FROM THE USE OF THESE MATERIALS OR INCLUDED SOFTWARE COMPONENTS.

Internal Use SAP Partner Only

About This Handbook

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

Typographic Conventions

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

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

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Icons in Body Text

The following icons are used in this handbook.

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

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Contents

Course Overview ..... **vii**
Course Goals vii
Course Objectives viii

Unit 1: Introduction **1**
Reporting – an Introduction 2

Unit 2: Selection Screen **19**
Generating and Designing a Selection Screen 20
Input Checks and Variants 46
Selection Screen: Modifications at Runtime 58

Unit 3: ALV: Usage **73**
Using the ALV to Display Data in Table Form 74

Unit 4: ALV: Creation **83**
ALV in the Container: Preparing the Container Screen 84
ALV Grid Control: Architecture 104

Unit 5: ALV: Design **131**
ALV Grid Control: Layout Variants 132
ALV Grid Control: Layout 140
ALV Grid Control: Field Catalog 161

Unit 6: ALV: Events and Methods **185**
ALV Grid Control: Events 187
ALV Grid Control: Further Methods 219
ALV Grid Control: Context Menus 233

Unit 7: Logical Databases **249**
Logical Databases: Introduction 250
Logical Databases: Subobjects and Data Retrieval 255

Unit 8: Data Retrieval Without Logical Databases **281**
Evaluating Multiple Database Tables 282
Further Additions to the SELECT Statement 295

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit 9: Call Programs with Data Transfer 315

Calling Programs and Passing Data 316

Unit 10: Background Processing 345

Background Processing 346

Unit 11: ALV Object Model (Optional) 357

Creating and Displaying the ALV OM 359
ALV OM: Creating in the Container 376
ALV OM: Fetching and Modifying Subobjects of a Main Class 405
ALV OM: Adjusting the Display 413
ALV OM: Defining Column Properties 428
ALV OM: Basic Functions and Layout Variants 446
ALV OM: Output Appearance 465
ALV OM: Functions, Events, and Interactive Elements 497
ALV OM: Print Design and the Header and Footer 538

Appendix 1: Detail Lists 561

Appendix 2: Customer GUI Status for Interactive Lists 573

Index 587

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Course Overview

Participating in this course enables you to use the ABAP programming language to create complex programs for reporting.

The following aspects of reporting are covered:

- Creating and checking a complex selection screen
- Data output with SAP List Viewer (ALV) and associated aspects
  - Creating an ALV
  - Programming initial display settings
  - Handling an event (such as the double-click)
- Retrieving data from the database and associated aspects
  - Using logical databases (that is, using central data retrieval programs)
  - Self-programmed data retrieval of more than one database table with the SELECT command
- Calling programs and transactions and passing of context data

Target Audience

This course is intended for the following audiences:

- ABAP consultants
- Project members
- Program maintenance specialists

Course Prerequisites

Required Knowledge

- SAPTEC – SAP Netweaver Basics
- BC400 – ABAP Workbench Foundations
- Initial ABAP programming experience

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Course Goals

This course will prepare you to:

- Differentiate between the different reporting techniques
- Program selection screens
- Use the SAP List Viewer (ALV) for data output
- Use logical databases to retrieve data from the database
- Retrieve data with Open SQL
- Call programs and pass context data to them

Course Objectives

After completing this course, you will be able to:

- Design a selection screen with complex screen elements
- Create an SAP List Viewer (ALV)
- Define the initial layout of an ALV
- Define additional columns for an ALV
- Work with events and methods to extend the functions of an ALV
- Read data from the database using logical databases
- Read data from the database using your own OPEN SQL accesses
- Call executable programs from an ALV

SAP Software Component Information

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

- SAP NetWeaver 6.0

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit 1

Introduction

Internal Use SAP Partner Only

Unit Overview

This unit consists of the “Introduction” lesson, which provides you with an overview of the most important reporting solution methods.

Unit Objectives

After completing this unit, you will be able to:

- List the different types of reports
- State advantages of using the ALV rather than a list
- Create a list with the help of the WRITE statement
- Work with include programs

Unit Contents

Lesson: Reporting – an Introduction 2
Exercise 1: Reporting – Introduction..... 13

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: Reporting – an Introduction

Lesson Overview

This lesson is the introduction to the course. It provides an overview of the most important solutions for programming reports. It presents the two relevant output media for reports: the ALV Grid Control and the classic list created by WRITE. Working with include programs is also explained here.

Lesson Objectives

After completing this lesson, you will be able to:

- List the different types of reports
- State advantages of using the ALV rather than a list
- Create a list with the help of the WRITE statement
- Work with include programs

Business Example

You are working as a programmer for ABC Travel. Your manager wants to know what would be a modern output format for a report, so that users can access a large number of interactive functions.

Standard Reports, Queries, Your Own ABAP Reports

Before you solve a reporting task by programming your own ABAP program, you should first check whether there are less time-consuming resources available. Individual programming is often the most time-consuming option.

Checklist Before Starting Individual Programming

- Can the function be performed by a standard report (delivered by SAP)?
- Or does a standard report at least cover the majority of the requirement?
- Can you meet the task by using a report tool?
- If you are programming your own ABAP program: Can you use a logical database (a data retrieval program that is available around the globe)?

If you believe that the time required to develop your own ABAP program is less than to find a standard report or to familiarize yourself with a report tool, you may prefer to create an individual solution anyway. However, as a basic principle, it is important that programmers do not start to write their own ABAP program without checking the alternatives. To be genuinely free in choosing the solution method, it is important for programmers to know how they can effectively search for a standard report, and which reporting tools are available to them.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Standard Reports

Standard reports are reports that SAP has created and delivered to customers as part of the standard system. Some of these reports can be used very flexibly.

Procedure for Searching for Suitable Standard Reports

1. *Easy Access → SAP menu → Information Systems*
2. Then open either the subtree for the desired application or the *General Report Selection* subtree.
3. Search the open subtree for suitable reports (read documentation and try reports).

Reporting Tools

There are now a considerable number of reporting tools (also known as query tools) in SAP systems. In particular, tools of this type that are not restricted to specific applications could be considered (almost) as equivalent replacements for an individually programmed ABAP program.

Generally available reporting tools

1. SAP Query
2. InfoSet Query
3. QuickViewer

There is also *SAP NetWeaver Business Intelligence* . This is a separate system, independent of the real-time SAP system, in which data is grouped by aspects that are typical for data warehouses.

The above reporting tools are extremely effective. A programmer should be familiar with them at least to some extent, to be able to determine when these tools reach the limits of their capabilities.

Your Own ABAP Programs with Logical Databases

If you have selected individual programming as your solution method, you should check whether logical databases can be used for data retrieval. Logical databases are globally available data retrieval programs. The database access, authorization checking, and the selection screen are centrally implemented in them. If a programmer uses logical databases, he or she therefore does not need to deal with these aspects of his or her program.

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Comparison of the Reporting Solution Methods

Comparison of the Reporting Solution Methods 1

Reporting Tool	Purpose	Advantages	Limitations
Standard reports	Provision of solutions for the most common reporting requirements	Immediately usable No development effort	Limited flexibility You cannot choose output fields arbitrarily
InfoSet Query	Intuitively operable, general SAP reporting tool for creating your own reports Allows the creation of evaluations for all areas of human resources	Clear interface Very easy to operate No programming effort required Integrates with SAP Query, queries can be additionally processed with SAP Query Can be included in roles with the appropriate InfoSet	InfoSets and user groups need to be defined in SAP Query to be able to use the InfoSet Query Not possible to display multi-row lists
SAP Query	General SAP reporting tool for creating your own reports Individual definition of user groups, InfoSets, and Queries	Very flexible No programming effort required Queries can be made available in the SAP Easy Access Menu Provides extensive options for aggregating data, performing calculations, and for graphical display Allows the display of multi-row lists	Restricted to data from the SAP system Requires a higher training investment than other solutions

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Reporting Tool	Purpose	Advantages	Limitations
		Allows you to define a basic list and multiple statistics and ranked lists for each Query	
Your own ABAP programs	Implement the customer's own requirements	Entire scope of the ABAP language is available The ABAP Workbench provides an easy-to-use development environment	Provides a solution with total freedom Programmers can cause more damage (bad runtime behavior, incorrect data is retrieved, or data is incorrectly interpreted)

ALV or List?

There are essentially two output formats available to display the report results.

ALV

The SAP List Viewer (ALV) is a tool for displaying mass data in the form of a table or tree. The ALV offers programmers a large number of standard functions, making the ALV a better alternative to a classic list for any report purpose.

This course only handles the ALV for output in table form. You can choose from two programming models: The ALV Grid Control (available as of *SAP R/3 4.6*) and the ALV Object Model (as of *SAP NetWeaver 2004*).

“Classic” ABAP List

This is the older of the two output formats. This format also allows you to create lists with excellent appearance and interactive functions. However, the programmer needs to implement almost all of the interactive functions himself or herself. The “classic” ABAP list does not meet the requirements of accessibility.

In place of the ALV types for displaying information in table form, you should present the characteristics of the ALV Grid Control compared with the classical list:

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Advantages and Disadvantages of ALV Grid Control and ABAP List

Point of View	ALV Grid Control	ABAP List
Accessible	✔	✘
Detailed display can be implemented	✔	✔
Can be generated in the background	✔	✔
Interactive sorting is a standard function	✔	✘
Interactive aggregation is a standard function	✔	✘
Interactive generation of subtotals is a standard function	✔	✘
Interactive hiding is a standard function	✔	✘
Excel in place is a standard function	✔	✘
Show and hide columns, change column width, change column order (are all standard functions)	✔	✘
Implement customer-defined interactive functions	✔	✔
Display of multi-row data records	✘	✔

Accessibility

Accessibility is used here to mean the availability of information and the suitability of software for people with disabilities. For a number of years, public and private organizations have been working to make information technology (IT) and therefore also software available for people with disabilities (such as blind people or those with visual impairments). In May 2001, with Paragraph 508 of the Rehabilitation Act, the USA became the first country to pass a law that obliges public organizations to purchase only IT that is “accessible”.

In the course of this development, SAP formulated design and programming guidelines with the aim that SAP software is accessible. In particular, it must be possible to operate the software using only the keyboard, since many people with disabilities cannot follow the movements of the mouse pointer. Being able to operate the software with the keyboard only is an absolute requirement for many supporting technologies, such as screen readers.

In the context of SAP Reporting, the following applies: The ALV Grid Control is more accessible than the classic ABAP list.

© 2011 SAP AG. All rights reserved.

Demonstrations, Copy Templates, and Solutions

Naming Conventions for Training Programs

Package BC405 with programs following these naming conventions:	
Demos	BC405_xxx_D...
Copy template	BC405_xxx_T...
Solutions	BC405_xxx_S...
xxx	Abbreviations for individual units

Abbreviations for individual units:

Abbreviations for individual units:

INTRO	Unit 1:	Introduction
SCREEN	Unit 2:	Selection Screen
USAGE	Unit 3:	ALV: Usage
CREATE	Unit 4:	ALV: Creation
DESIGN	Unit 5:	ALV: Design
EVENT	Unit 6:	ALV: Events and Methods
LOGDB	Unit 7:	Logical Databases
SELECT	Unit 8:	Data Retrieval Without Logical Databases
CALL	Unit 9:	Call Programs with Data Transfer
BACKGR	Unit 10:	Background Processing
OM	Unit 11:	ALV Object Model (Optional)
OTHER	Appendix:	Additional Topics

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Generating a List with the WRITE Statement

```
REPORT bc405_fold_list_creation.

DATA: gs_spfli TYPE spfli.

START-OF-SELECTION.

  SELECT carrid connid cityfrom cityto
    FROM spfli
    INTO (gs_spfli-carrid, …

    WRITE: / gs_spfli-carrid
             gs_spfli-connid
      ...
  ENDSELECT.
  IF sy-subrc <> 0. ... ENDIF
```

```
DEMO: Generating a List                    1

AA 0017 NEW YORK          SAN FRANCISCO
AA 0064 SAN FRANCISCO     NEW YORK
AZ 0555 ROME              FRANKFURT
AZ 0788 ROME              TOKYO
          :                    :
```

Figure 1: Generating a List

The first `WRITE` statement in an ABAP program triggers list generation. The output is directed to a list buffer. As soon as the buffer has been completely built, the system generates the screen from the list buffer.

As a standard function, the system generates two header lines (standard header). The first header line contains the program title from the program attributes in the upper left corner and the page number in the upper right corner. The second header line consists of an unbroken line. Both header lines remain in the window when you scroll.

When printing lists, the first line appears as follows:

- Top left: System date
- Top right: Page number
- In the middle: List header

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
WRITE [AT] [/<pos(len)>] <f> <option1> <option2> ...  .
```

```
REPORT bc405_fold_write ... .

* constants for positions of outputs
CONSTANTS: gc_pos TYPE i VALUE 12,
           gc_len TYPE i VALUE 10,    "sflight-fldate
           ...

WRITE AT: / sy-vline,
            gc_pos(gc_len)  gs_sflight-fldate
                            COLOR COL_KEY,
                            ...
             (gc_len_pri)   gs_sflight-price
                            CURRENCY gs_sflight-currency,
             (gc_len_cur)   gs_sflight-currency.
                            ...
```

Figure 2: WRITE statement: General syntax

WRITE Statement Additions:

Addition	Description
NO-GAP	Suppresses output of spaces after the <f> field.
NO-ZERO	If the value of field <f> equals zero, only spaces are output. If the field is of type C or N, spaces replace leading zeros.
DD/MM/YY	The various date formats determine only whether the year number has two or four digits, and which separators are used. The order of day, month, and year is exclusively user-dependent.
CURRENCY <currency_key>	Formats currency fields in accordance with the rules defined in the *TCURX* table
UNIT <unit_key>	Formats quantity fields in accordance with the rules defined in the *T006* table
USING EDIT MASK <mask>	Output according to formatting template
UNDER <g>	The output starts at the column in which field <g> was output.
LEFT-JUSTIFIED	Left-aligned output (default for types C, N, D, T, and X)
CENTERED	Centered output within the output length
RIGHT-JUSTIFIED	Right-justified output (default for all number fields: I, P and F)

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Working with Include Programs

Include programs simplify the organization of programming. Source code for certain aspects is stored in certain includes, which are identified by unique names. This simplifies distributed development and reuse.

The `INCLUDE <include>.` statement includes includes in a framework program. Includes can be nested.

The following naming conventions apply for includes:

Naming Conventions for Includes

Program Type	Naming Convention
Main program	*ZDEMO_00*
Top include (global data definitions)	*ZDEMO_00_TOP*
PBO modules	*ZDEMO_00_O01* (sequence number from 0 to 99)
PAI modules	*ZDEMO_00_I01* (sequence number from 0 to 99)
Subroutines	*ZDEMO_00_F01* (sequence number from 0 to 99)
ABAP events (*START-OF-SELECTION*, and so on)	*ZDEMO_00_E01* (sequence number from 0 to 99)

Copying Programs with Includes

To copy programs with includes, proceed as follows:

In the Object Navigator, place the cursor on the copy template and use the right mouse button to select *Copy*.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

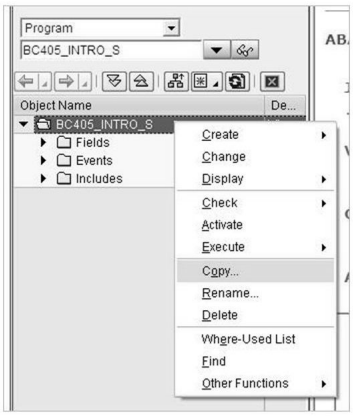


Figure 3: Copying Includes (1)

Enter the name of the new main program on the dialog box that appears.

Figure 4: Copying Includes (2)

Then set at least the following three checkmarks:

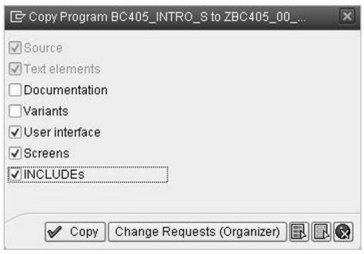


Figure 5: Copying Includes (3)

The following applies for the dialog box that follows: Select all the lines and in the column on the right side, specify the name of each new program along with its correct addition; for example, *TOP* (top include), *E01* (event include), *F01* (subroutine include).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

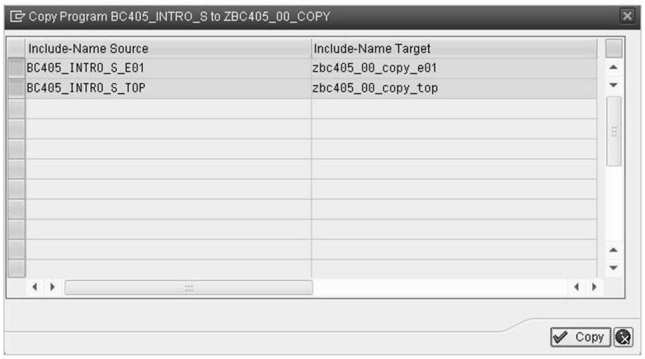


Figure 6: Copying Includes (4)

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 1: Reporting – Introduction

Exercise Objectives

After completing this exercise, you will be able to:

- Create executable programs with includes
- Generate a list with WRITE
- Copy executable programs with includes

Business Example

You are a programmer for ABC Travel. To ensure that you have no problems during the maintenance of programs with includes that is due to start shortly, your manager asks you to create a practice program with a top include and an event include. You are then to copy this program, including its includes, for practice.

Task:

Create an executable program with a top include and an event include. In this program, display data about flights from the database view *DV_FLIGHTS*. Then copy the program including all includes.

1. Create the package *ZBC405_##*.
2. Create the executable program *ZBC405_##_SOL*. Assign it to the package *ZBC405_##*. Then create a top include and an event include for the program (note the naming conventions) and include the includes in your framework program. (Tip: It is best to create the includes in the framework program using forward navigation.)
3. Now retrieve flight data from the *DV_FLIGHTS* database view using a `SELECT-ENDSELECT` loop and output it as a list using `WRITE`.

   Tip: A database view is a summary of two or more tables. For read access, you can essentially program as if only one table were involved.
4. Now copy your program together with the includes. Call the copy *ZBC405_##_COPY*. Name the copies of the includes in accordance with the appropriate naming conventions. Note that it is always the active version of a program that is copied.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 1: Reporting – Introduction

Task:

Create an executable program with a top include and an event include. In this program, display data about flights from the database view *DV_FLIGHTS*. Then copy the program including all includes.

1. Create the package *ZBC405_##*.
   a) Start transaction SE80.
   b) Choose *Package* and specify the name of the package. Choose *Display*.
   c) Answer the question "Create Object?" by choosing *Yes*. Specify the attributes of the package in the following dialog box.
   d) Save the package in the transport request that your instructor specifies.
2. Create the executable program *ZBC405_##_SOL*. Assign it to the package *ZBC405_##*. Then create a top include and an event include for the program (note the naming conventions) and include the includes in your framework program. (Tip: It is best to create the includes in the framework program using forward navigation.)
   a) Create the program *ZBC405_##_SOL*. Insert the following source code:
   ```
   REPORT zbc405_##_sol

   INCLUDE zbc405_##_sol_top.
   INCLUDE zbc405_##_sol_e01.
   ```
   b) Create the include programs using forward navigation. Optional: Move the `REPORT` statement from the framework program to the TOP include.
3. Now retrieve flight data from the *DV_FLIGHTS* database view using a `SELECT-ENDSELECT` loop and output it as a list using `WRITE`.

Continued on next page

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Tip: A database view is a summary of two or more tables. For read access, you can essentially program as if only one table were involved.

a) Define a work area of type *DV_FLIGHTS* in the TOP include.

```
DATA: gs_flight TYPE dv_flights.
```

b) In the E01 include, implement database access and list output at the `START-OF-SELECTION` program event. To create the `WRITE` command, use the pattern function (*Pattern → Write → Structure*).

```
*&---------------------------------------------------------------------*
*&  Include           BC405_INTRO_S1_E01
*&---------------------------------------------------------------------*
START-OF-SELECTION.
  SELECT
      * FROM  dv_flights
      INTO gs_flight.

    WRITE: / gs_flight-carrid,
             gs_flight-connid,
             gs_flight-fldate,
             gs_flight-countryfr,
             gs_flight-cityfrom,
             gs_flight-airpfrom,
             gs_flight-countryto,
             gs_flight-cityto,
             gs_flight-airpto,
             gs_flight-seatsmax,
             gs_flight-seatsocc.

  ENDSELECT.
```

4. Now copy your program together with the includes. Call the copy *ZBC405_##_COPY*. Name the copies of the includes in accordance with the appropriate naming conventions. Note that it is always the active version of a program that is copied.

   a) Copy your main program *ZBC405_##_SOL*. Select it in the navigation window of transaction SE80, then press the right mouse button and choose *Copy*.

   b) In the dialog box that appears, select all the subobjects (especially Includes). Choose *Copy*.

   c) In the following dialog box, select every listed include, and specify the name for the copy of the include to be created.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- List the different types of reports
- State advantages of using the ALV rather than a list
- Create a list with the help of the WRITE statement
- Work with include programs

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- List the different types of reports
- State advantages of using the ALV rather than a list
- Create a list with the help of the WRITE statement
- Work with include programs

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Unit 2

Selection Screen

Internal Use SAP Partner Only

Unit Overview

This unit repeats (from BC400) the ABAP commands for generating a selection screen. More additions and techniques are introduced. ABAP events that belong to the selection screen (such as `AT SELECTION-SCREEN`) are discussed. Creating variants and dynamically changing a selection screen are also covered.

Unit Objectives

After completing this unit, you will be able to:

- Explain the purpose of selection screens
- Create input fields with PARAMETERS
- Create input fields with SELECT-OPTIONS
- Design the selection screen
- Define additional selection screens
- Define subscreens
- Define a tabstrip within a selection screen
- Implement input checks for selection screens
- Create variants for a selection screen
- Create buttons on the selection screen
- Program modifications of the selection screen at runtime

Unit Contents

Lesson: Generating and Designing a Selection Screen 20
Exercise 2: Generating and Designing a Selection Screen..... 37
Lesson: Input Checks and Variants 46
Procedure: Creating a Variant with Dynamic Date Calculation..... 51
Exercise 3: Input Checks and Variants 53
Lesson: Selection Screen: Modifications at Runtime 58
Exercise 4: Selection Screen: Modifications at Runtime..... 63

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson: Generating and Designing a Selection Screen

Lesson Overview

This lesson shows you how to create and design a selection screen, and how to check input. It concludes by showing you how screen elements can be changed at runtime.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the purpose of selection screens
- Create input fields with PARAMETERS
- Create input fields with SELECT-OPTIONS
- Design the selection screen
- Define additional selection screens
- Define subscreens
- Define a tabstrip within a selection screen

Business Example

You are working as a programmer for ABC Travel. Your manager wants to see flight data for ABC Airways. It is now your task to write a program with a user-friendly selection screen to fulfill your manager's requirements.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Selection Screen Overview

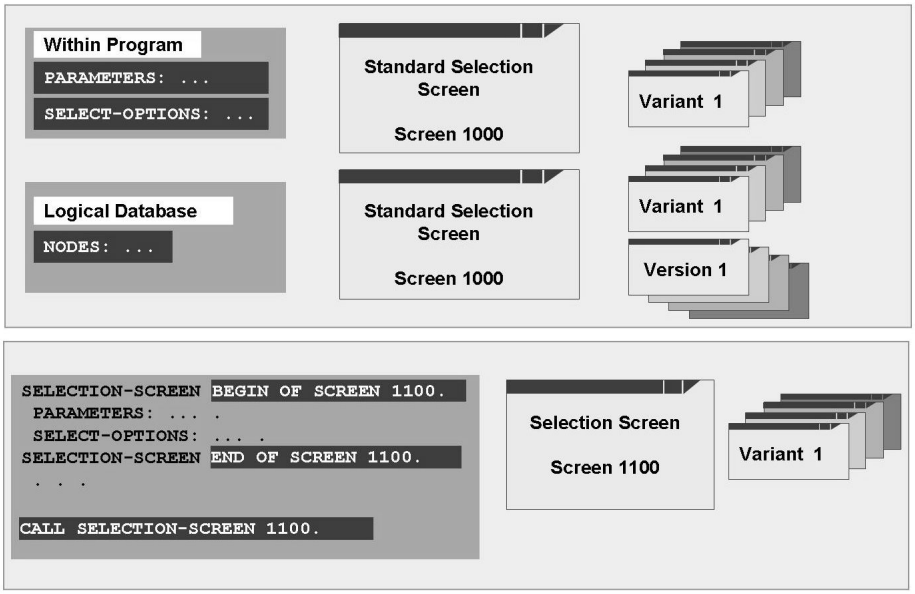


Figure 7: Selection Screen: Overview

Selection screens serve as the interface between the program and the user, and allow limitation of the amount of data to be read from the database.

You can use the declarative language elements `PARAMETERS` and `SELECT-OPTIONS` to generate a standard selection screen (screen 1000) with input-ready fields.

Logical databases supply selection screens whose concrete appearance is dependent on the specified node name (`NODES <name>`). (For more information, see the "Logical Databases" unit.) Selection screen versions offer a subset of the standard selection screen.

As well as the standard selection screen, you can use `SELECTION-SCREEN BEGIN OF ...` to generate additional selection screens, and `CALL SELECTION-SCREEN` to call them.

Create variants to save selection screen values that are used frequently. The variant can be used again at any time. If a report with a selection screen runs in the background, a variant is always required.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Creating Fields in a Selection Screen

```
PARAMETERS <f>
 [TYPE <type>] [LIKE <f1>] [DECIMALS <n>] [MEMORY ID <pid>]
 [OBLIGATORY][DEFAULT <value>][LOWER CASE][VALUE CHECK]
 [AS CHECKBOX]
 [RADIOBUTTON GROUP <grp>] [MODIF ID <mod>].
```

```
REPORT bc405_screen_d2.
... .
PARAMETERS: pa_carr  TYPE sflight-carrid,
            pa_name AS CHECKBOX DEFAULT 'X',
            pa_curr AS CHECKBOX DEFAULT 'X',
            pa_lim_1 RADIOBUTTON GROUP lim DEFAULT 'X',
            pa_lim_2 RADIOBUTTON GROUP lim,
            pa_lim_3 RADIOBUTTON GROUP lim.
CONSTANTS gc_mark VALUE 'X'.

* Check if any checkbox has been selected
IF pa_name EQ gc_mark. ... . ENDIF.
IF pa_curr EQ gc_mark. ... . ENDIF.

* Check which radiobutton has been selected
CASE gc_mark.
  WHEN pa_lim_1. ... .
  WHEN pa_lim_2. ... .
  WHEN pa_lim_3. ... .
ENDCASE.
```

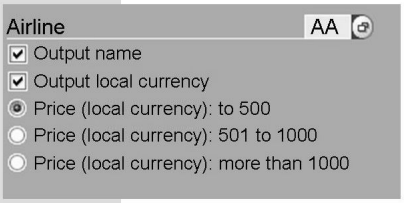


Figure 8: Declaring Fields with PARAMETERS

The `PARAMETERS` statement is a declarative language element. As in the case of the `DATA` statement, you can declare the parameters with `TYPE` or `LIKE`. The system generates input-ready fields on the selection screen. You can maintain selection texts by choosing the menu path *Goto → Text Elements → Selection Texts*.

Hint: The names of the `PARAMETERS` fields can have a maximum of **eight** characters.

With the `DEFAULT <value>` addition, you can set a default value for a `PARAMETERS` statement. If you assign a `MEMORY ID <pid>`, the system retrieves the current value from the SAP memory and outputs it on the screen automatically. The SAP memory is a user-specific memory area in which simple data values (such as the key for an airline) is stored for SET/GET parameters for the duration of a user session. SET/GET parameters are identified by a name (maximum 20 characters long). You can find out the name of a relevant SET/GET parameter by choosing *Help (F1) → Technical Information* in an output field to call the *Technical Information*. In the *Parameter ID* field, you will find the name of the SET/GET parameter as it is proposed in the data element being used

If you declare mandatory fields with the `OBLIGATORY` addition, users cannot leave the selection screen until values have been entered in these fields.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

By default, the automatic ABAP Dictionary check is deactivated for report selection screens. However, if you refer to a Dictionary object for which a foreign key is set up, you can use `VALUE CHECK` to trigger a validity check of the input.

You can also define parameters as checkboxes (AS CHECKBOX). With this definition, you create a single character field that contains **`Space`** or "X". You can evaluate the content of the checkboxes in `IF/ENDIF` structures.

You can also generate a group of radio buttons for the selection screen using the addition `RADIOBUTTON GROUP <grp>`. Only **one** radio button in a group can be active; it can be evaluated in the program processing. You can evaluate the contents of radio buttons using `CASE/ENDCASE` control structures. The radio button you chose has the value "X", all others have the value (`space`). If you have not assigned a value to any of the radio buttons, the system will automatically select the first radio button when it displays the selection screen.

Hint: The name `<grp>` of a radio button group can have a maximum of **four** characters.

You can use the `MODIF ID <mod>` addition to simplify dynamic modifications to the selection screen.

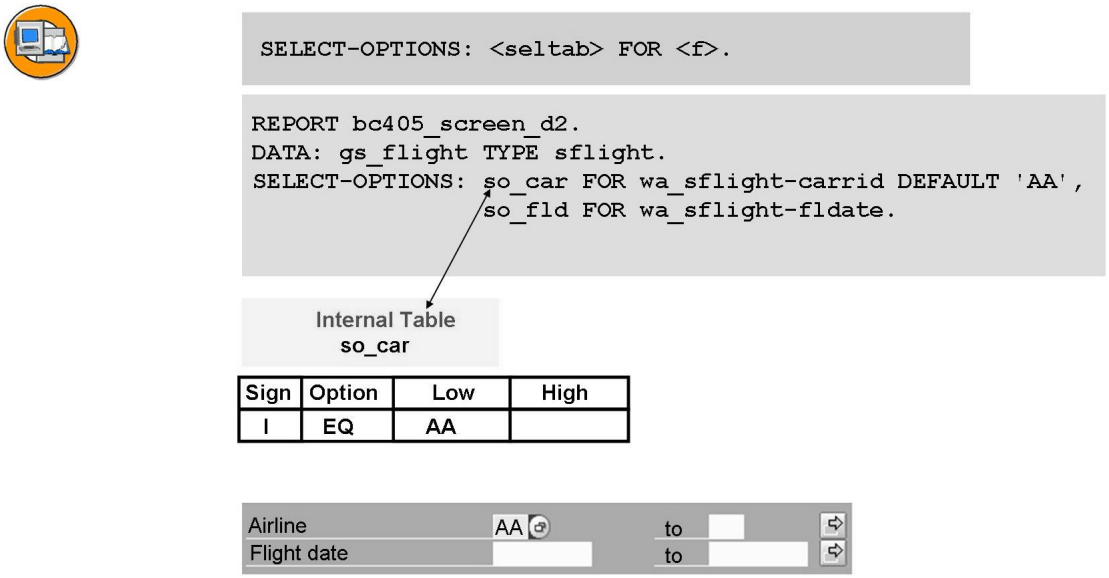


Figure 9: Selections with SELECT-OPTIONS

`SELECT-OPTIONS` is a declarative language element with the same naming restriction of eight characters as `PARAMETERS`. Unlike `PARAMETERS`, `SELECT-OPTIONS` does allow the use of a value range and complex selections instead of a single input field.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The SELECT-OPTIONS keyword generates an internal table <seltab> with a standard structure and a header line. The table has four columns: sign (indicator meaning including or only), option (relational operator), low (lower limit) and high (upper limit). You can maintain the selection text in exactly the same way as with PARAMETERS, by choosing the path *Goto → Text Elements → Selection Texts*.

Use the addition FOR to specify the field against which the system should check the selection input. This field must be declared, for example in a DATA or NODES statement. Both limit fields (high and low) inherit the attributes of the reference field.

Each line of the selection table <seltab> formulates a condition using one of the following relational operators. The following values are possible:

Possible Values for SIGN and OPTION

	Value	Meaning
SIGN	**I**	Include
	E	Exclude
OPTION	**EQ**	Equal
	NE	Not Equal
	LE	Less or Equal
	LT	Less Than
	GE	Greater or Equal
	GT	Greater Than
	BT	Between
	NB	Not Between
	CP	Contains Pattern
	NP	Contains Pattern not

The selection set S is a union of all includes (I_1, ..., I_n) minus the union of all excludes (E_1, ..., E_m). If no values are entered in the SELECT-OPTIONS object, the system returns all rows from the database ("nothing is everything").

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

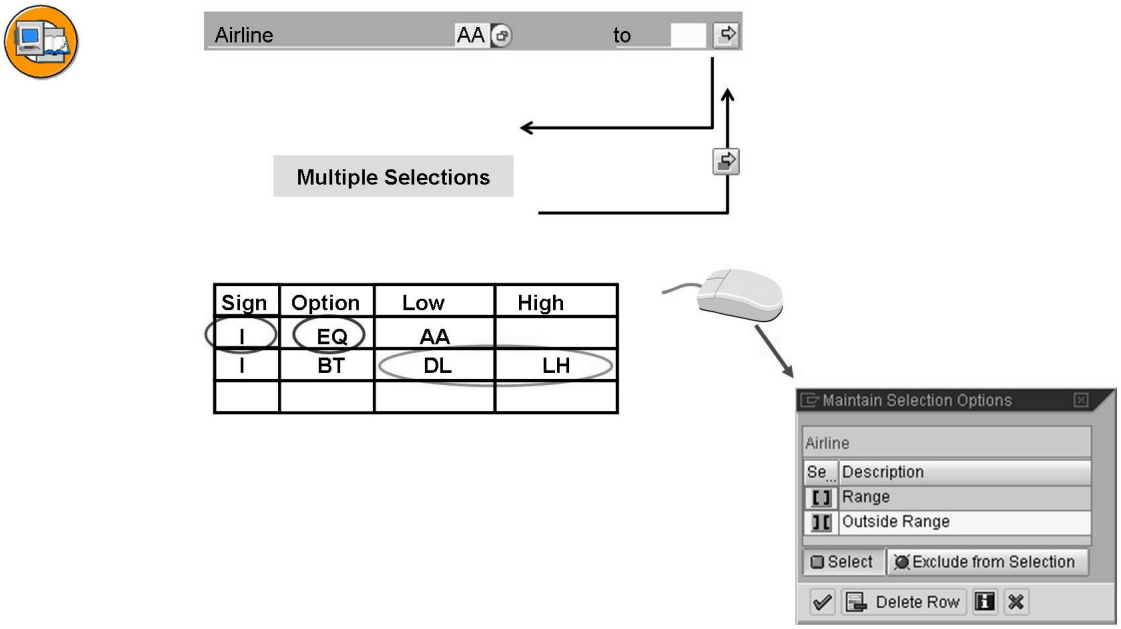


Figure 10: SELECT-OPTIONS with Multiple Selection Criteria

If the user makes entries in a `SELECT-OPTIONS` object, the system automatically fills the internal table.

To change the default values for the table fields `sign` and `option`, choose *Edit → Selection Options*. The system offers all suitable options for this selection. If the traffic signal icon is green during *Select*, there is an **I** (inclusive) in the SIGN column; a red light indicates **E** (exclusive).

To delete a table entry, choose *Edit → Delete Selection Row*.

Every selection criterion can be used to make multiple selections unless defined otherwise. If multiple selections are present, the color of the arrow changes from gray to green.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

SELECT-OPTIONS <seltab> FOR <f>	
DEFAULT <value>	SIGN <x> OPTION <xx>
DEFAULT <value1> TO <value2>	
MEMORY ID <pid>	
LOWER CASE	
OBLIGATORY	
NO-EXTENSION	
NO INTERVALS	
MODIF ID <mod>	

Figure 11: Syntax of the SELECT-OPTIONS Statement

`SELECT-OPTIONS` statement additions:

- You can use `DEFAULT` to set default values for the parameters `low` or `low` and `high`. You can use `OPTION` and `SIGN` to set default values for `option` and `sign` that differ from the normal default values.
- If you assign a `MEMORY ID <pid>`, the system retrieves the current value from the SAP memory, automatically outputs it to the screen, and writes it back again when you exit the screen (including any changes you might have made).
- `LOWER CASE` suppresses the conversion of the input to uppercase, as long as the domain allows it, or the typing was made for ABAP type **C**.
- `OBLIGATORY` generates an obligatory field. A question mark appears in the input field in the selection screen, and the user must enter a value.
- `NO-EXTENSION` suppresses the possibility of multiple selections.
- `NO INTERVALS` suppresses the upper interval limit (`<seltab>-high`) on the selection screen. You can use the additional screen *Multiple Selection* to enter intervals.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Dynamic Preassignment of the Selection Screen

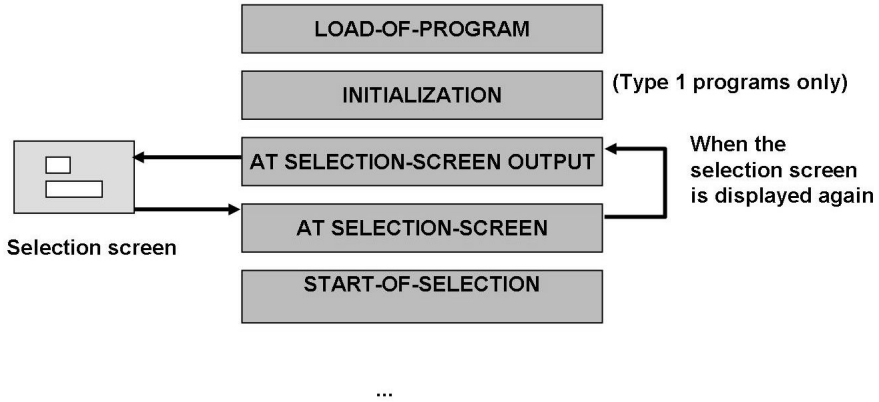


Figure 12: ABAP Events in the Environment of the Selection Screen

The default value of a `PARAMETERS` field or of a `SELECT-OPTIONS` object can be overwritten in the ABAP events `LOAD-OF-PROGRAM` and `INITIALIZATION`. These events are therefore appropriate for a dynamic preassignments of the input fields displayed on a selection screen.

`AT SELECTION-SCREEN OUTPUT` is processed immediately before and `AT SELECTION-SCREEN` immediately after the selection screen is displayed. Since you can display the selection screen multiple times (for *ENTER*, for example), the system can also process those event blocks multiple times.

```
INITIALIZATION.
```

```
REPORT bc405_screen_d4.
...
INITIALIZATION.

  MOVE:  gc_mark TO pa_all.

  MOVE: 'I'  TO so_carr-sign,
        'BT' TO so_carr-option,
        'AA' TO so_carr-low,
        'LH' TO so_carr-high.
  APPEND so_carr.
  CLEAR so_carr.
  MOVE: 'E'  TO so_carr-sign,
        'EQ' TO so_carr-option,
        'DL' TO so_carr-low.

  APPEND so_carr.
...
```

Figure 13: Initializing the Selection Screen

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

The system only processes the INITIALIZATION event once in each executable program. You can deliver default values for the selection screen fields of a report or of a logical database during this event. You can use the F1 help (technical help) to determine the names of the selection fields if a logical database is connected with the program.

```
DATA: gs_booking TYPE sbook.
SELECT-OPTIONS: so_car FOR gs_booking-carrid.
```

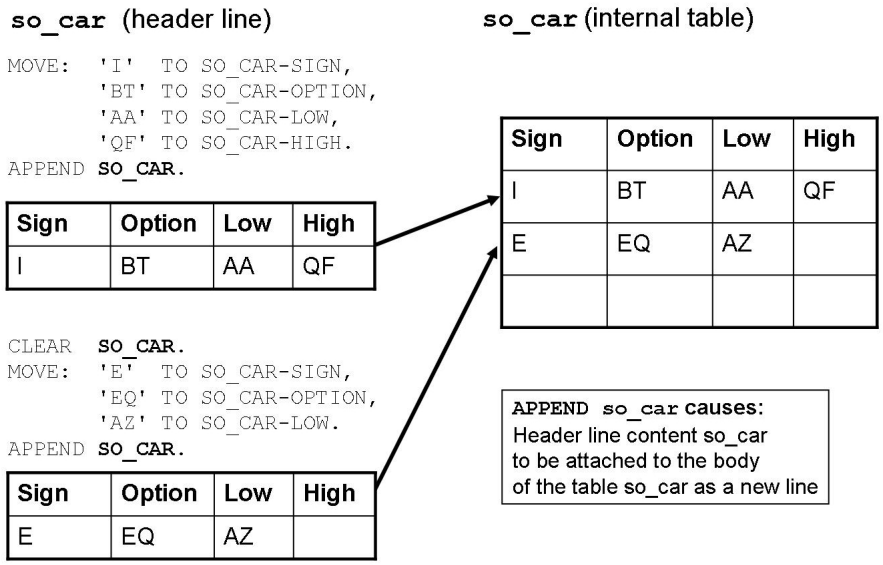


Figure 14: Filling an Internal Table with a Header Line

The SELECT-OPTIONS <sel_opt> FOR <data_object> statement always generates one internal table with a header line. This actually creates two data objects: a table body named <sel_opt> and a work area (the header line), also named <sel_opt>. Fill in the <sel_opt> work area with suitable values, then use the APPEND <sel_opt> command to add it to the table body as a separate line.

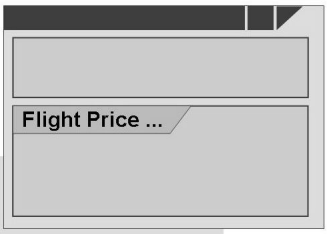
You can also provide values for the selection screen at the AT SELECTION-SCREEN OUTPUT event. This event occurs immediately before the selection screen is displayed. However, this event may be processed multiple times (for example, if the user clicks **Enter** on the selection screen). In this case, the user input is overwritten by the values provided at the AT SELECTION-SCREEN OUTPUT event.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Designing the Selection Screen

```
SELECTION-SCREEN BEGIN OF BLOCK <block>
WITH FRAME    TITLE <text>
SELECTION-SREEN END OF BLOCK <block>
```




```
REPORT bc405_sreen_d5a.
...
SELECTION-SCREEN BEGIN OF BLOCK carr WITH FRAME.
 SELECT-OPTIONS: so_carr FOR wa_sflight-carrid.
SELECTION-SCREEN END OF BLOCK carr.

SELECTION-SCREEN BEGIN OF BLOCK limit WITH FRAME TITLE text-001.

 PARAMETERS:     pa_lim_1  RADIOBUTTON GROUP lim DEFAULT 'X',
                 pa_lim_2  RADIOBUTTON GROUP lim,
                 pa_lim_3  RADIOBUTTON GROUP lim.

SELECTION-SCREEN END OF BLOCK limit.
...
```

Figure 15: Designing the Selection Screen I

You can use the `SELECTION-SCREEN` statement to design the structure of the selection screen.

You can use the `BEGIN OF BLOCK <block>` addition to group logically-related screen elements and `WITH FRAME` to frame them. You can nest frames to a depth of five levels.

For accessibility reasons, you should assign a heading to the block. The title can be declared as either `text-<xxx>` or as a field name of up to eight characters. In the second case, you set the text at runtime in the `INITIALIZATION` event. Before you design a selection screen, you should familiarize yourself with the guidelines for screen design shown in transaction BIBS.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

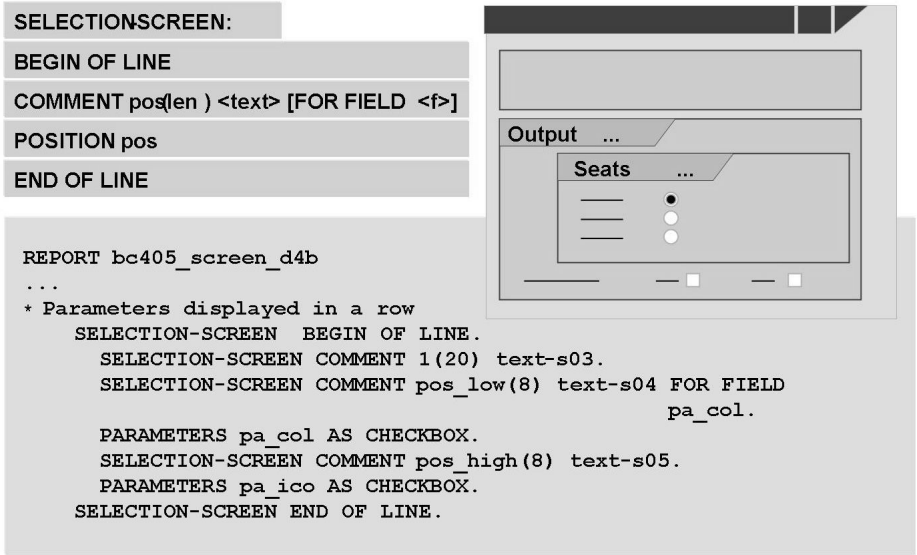


Figure 16: Designing the Selection Screen II

You can display multiple parameters and comments in the same output line. To do this, you need to enclose them between `SELECTION-SCREEN BEGIN OF LINE` and `SELECTION-SCREEN END OF LINE`.

The `COMMENT` addition enables you to include text in the line. Comment texts must always have a format (position and output length). You can define the position using an integer literal or `pos_low` and `pos_high`. The latter two of these are the positions of the upper and lower limits of the field `SELECT-OPTIONS` on the selection screen. The output length is always defined as an integer literal.

The `COMMENT ... FOR FIELD <f>` addition causes the system to automatically display the F1 help of the <f> field for both the comment text and the parameter. If you hide the parameter (selection variant: attribute *invisible*), the comment text is also hidden.

You can use `POSITION <pos>` to position the cursor on a line for the next output (only in `BEGIN OF LINE ... END OF LINE`). Here, `<pos>` is an integer.

Empty lines can be defined via the addition `SKIP <n>`, where `<n>` is an integer.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Creating Additional Selection Screens

```
REPORT
bc405_screen_d6
 ...
SELECTION-SCREEN
  BEGIN OF SCREEN 1100.
    PARAMETERS: pa_cus AS CHECKBOX,
                pa_agy AS CHECKBOX.
SELECTION-SCREEN
  END OF SCREEN 1100.
...

AT SELECTION-SCREEN ON pa_add.
  IF pa_add = 'X'.
    CALL SELECTION-SCREEN 1100
      STARTING AT 5 5
      ENDING AT 50 10.
    IF sy-subrc <> 0.
       LEAVE TO SCREEN 1000.
    ENDIF.
  ENDIF.
```

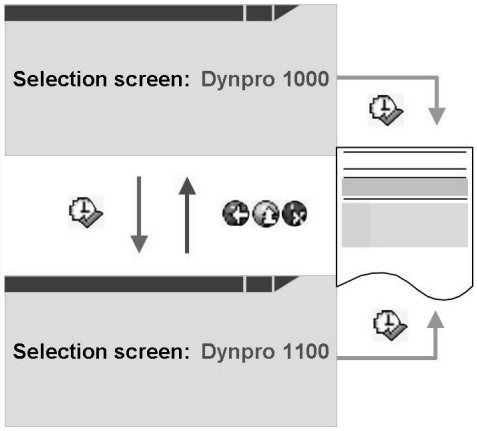


Figure 17: CALL SELECTION-SCREEN

You can work with several selection screens in one program. The standard selection screen always has the number 1000. You can define other selection screens using `SELECTION-SCREEN BEGIN OF SCREEN <nnnn>` and `SELECTION-SCREEN END OF SCREEN <nnnn>`. Within the `BEGIN ... END` statements, you use `SELECT-OPTIONS` and `PARAMETERS` to declare the required selections. Call this selection screen using `CALL SELECTION-SCREEN <nnnn>`. A selection screen should have a screen number higher than 1000.

If the user performs any action on the called selection screen, the event `AT SELECTION-SCREEN` is processed completely. In this loop, the system field `sy-dynnr` equals the number of the called selection screen. Then the program continues with the statement that follows the call. The system field `sy-subrc` contains 0 if the user chooses *Execute* or F8. If the user chooses *Cancel* or *Exit*, `sy-subrc` contains the value 4.

You can supply the additional selection screens with default values at `INITIALIZATION`.

Use the system field `sy-dynnr` in the selection screen events to determine which screen is processed.

You can also call a selection screen as a modal dialog box. Use the following syntax to do this:

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
CALL SELECTION-SCREEN <nnnn> STARTING AT
<left_col> <upper_row> ENDING AT <right_col>
<lower_row>.
```
`<left_col>` and `<upper_row>` are the coordinates of the top left corner of the screen, `<right_col>` and `<lower_row>` are the coordinates of the bottom right corner.

Tab Pages on the Selection Screen

You can also display tabstrips on selection screens. Tabstrips allow a better logical grouping of the fields and therefore a clearer structuring of large selection screens.

- **Concise arrangement of large selection screens**
- **Logical field groupings**

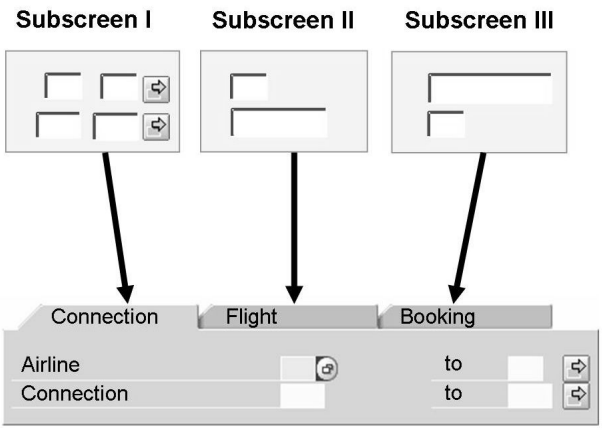


Figure 18: Tabstrip on the Selection Screen

For a selection screen with tabstrips, you define:

- A subscreen area on the selection screen for the tabstrip
- The individual tab titles
- Selection screens as subscreens for individual tab titles.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
DATA: gs_spfli TYPE spfli.

SELECTION-SCREEN BEGIN OF SCREEN 101 AS SUBSCREEN.
   SELECT-OPTIONS: so_car FOR gs_spfli-carrid,
                   so_con FOR gs_spfli-connid.
SELECTION-SCREEN END OF SCREEN 101.

SELECTION-SCREEN BEGIN OF SCREEN 102 AS SUBSCREEN.
   PARAMETERS:     ...
SELECTION-SCREEN END OF SCREEN 102.

SELECTION-SCREEN BEGIN OF SCREEN 103 AS SUBSCREEN.
   PARAMETERS:     ...
SELECTION-SCREEN END OF SCREEN 103.
```

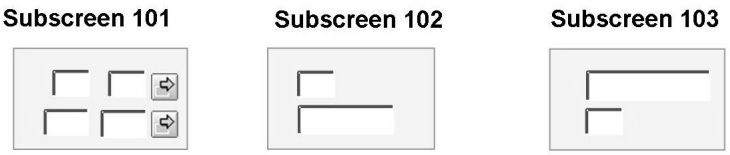


Figure 19: Selection Screens as Subscreens

Since selection screens can be defined as subscreens, you can include them on any other screen, especially on tab pages. The system processes the `AT SELECTION-SCREEN OUTPUT` and `AT SELECTION-SCREEN` ABAP events for each subscreen in addition to processing the surrounding selection screen. Work with the system variables `sy-dynnr` to determine which screen (or selection screen or subscreen) is currently being processed.

You can define a selection screen as a subscreen as follows:

```
SELECTION-SCREEN BEGIN OF SCREEN <dynnr> AS SUBSCREEN
[NESTING LEVEL <m>] [NO INTERVALS].
....
SELECTION-SCREEN END OF SCREEN <dynnr>.
```

Optional additions:

`[NESTING LEVEL <m>]`: The nesting level `NESTING LEVEL` further reduces the size of the subscreen. In this way, you can prevent scroll bars appearing if you are using the subscreen in a tabstrip control on the selection screen and the tabstrip already has a frame. If the tabstrip does not have a frame, use `NESTING LEVEL 0`. For each frame around the tabstrip control, increase `NESTING LEVEL` by 1.

`[NO INTERVALS]`: This option hides the `HIGH` fields for all selection criteria defined with `SELECT-OPTIONS` on this screen.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

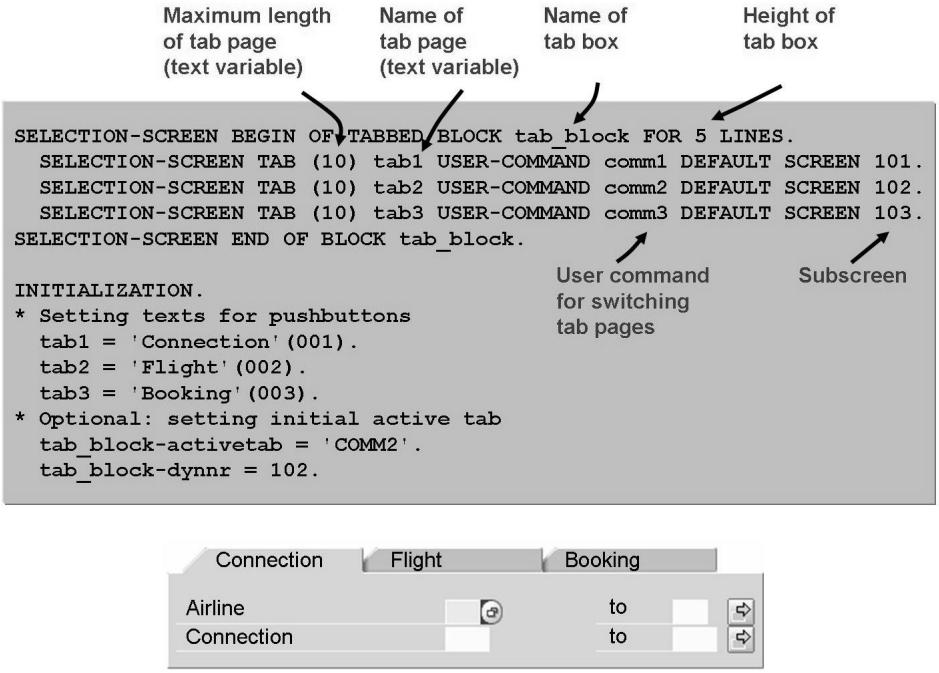


Figure 20: Defining Tabstrips on the Selection Screen

You can define a subscreen area for a tabstrip on a selection screen as follows:

```
SELECTION-SCREEN BEGIN OF TABBED BLOCK <blockname>
FOR <n> LINES.
```

```
SELECTION-SCREEN END OF BLOCK <blockname>
```

The height of the subscreen area in lines is defined by <n>.

You can define a single tab title as follows:

```
SELECTION-SCREEN TAB (max_length) <name>
USER-COMMAND <ucomm> [DEFAULT [PROGRAM <prog>] SCREEN <dynnr>].
```

`TAB` is the name of a variable containing the text to be displayed. This variable is part of the tabstrip control, so you do not have to declare it.

If the text to be displayed on the tab title is shorter than specified via `max_length`, the length of the tab title is optimized. However, if the text to be displayed on the tab title exceeds the maximum length, it is truncated.

If you specify the `DEFAULT` addition, you also need to specify the `SCREEN` addition. The `PROGRAM` addition is optional. You only need it if the screen comes from another program.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

You can delay the specification of the connection between tab titles and selection screens until runtime. You can also change an existing assignment at runtime. To do this, fill the `<blockname>` structure. It is automatically created for every tabstrip block. The structure has the same name as the tabstrip block and contains the fields `prog`, `dynnr`, and `activetab`.

For the `AT SELECTION-SCREEN` event, the `sy-ucomm` system variable contains the user command declared for the tab page currently selected (`<ucomm>`). The `<blockname>` structure contains information about the selection screen currently displayed. `AT SELECTION-SCREEN` is processed twice: First for the selection screen on the currently selected tab page, then for the selection screen `1000`.

For the `AT SELECTION-SCREEN OUTPUT` event, the `<blockname>` structure contains information about the next active tab page and the selection screen related to this page. `AT SELECTION-SCREEN OUTPUT` is processed twice: First for the selection screen `1000`, then for the selection screen on the next visible tab page.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 2: Generating and Designing a Selection Screen

Exercise Objectives

After completing this exercise, you will be able to:

- Define selection options
- Define parameters
- Design the selection screen
- Preassign the selection screen fields (optional)

Business Example

You are working as a programmer for ABC Travel. Your task is to add several fields to the selection screen of an existing report, create a tabstrip, and adjust data retrieval accordingly.

Task 1:

Enhance your program *ZBC401_##_SOL* or copy the model solution *BC405_INTRO_S1* from the previous exercise.

Model solution for the task: *BC405_SCREEN_S1A*

Create selection screen

1. Create a `SELECT-OPTION` for the airline. Take the lower interval value from SAP Memory.

   Create a `SELECT-OPTION` for the flight connection number.

   Create a selection criterion for the flight date. Suppress the multiple selection option for the flight date.

2. Create a group of three radio buttons with the following functionality:

Button	Function
1	Read **all flights**.
2	Read only **domestic flights**.
3	Read only **international flights**.

   The default setting is to be that international flights are read.

   Create a frame around the radio button group.

3. Ensure that only the data records requested are read from the database.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

To do this, supplement the `WHERE` clause of the `SELECT` statement with the conditions that result from the selections.

Hint: You need three separate `SELECT` statements with different `WHERE` conditions. You can map the national/international condition directly on the database: `countryto = dv_flights~countryfr` or `countryto <> dv_flights~countryfr`. You use the tilde (~) to address the database field.

4. **Optional:** Program a complex preassignment for the airline, such as AA to QF, but not AZ.
5. Activate and test your program.

Task 2:

Integrate tab pages

Model solution for the task: *BC405_SCREEN_S1B*.

1. Create selection screen 1100 as a subscreen and display the selection criteria for the airline and connection number on this screen.

   Create selection screen 1200 as a subscreen and display the selection criterion for the flight date there.

   Create selection screen 1300 as a subscreen and integrate the radio buttons there.
2. Create a tabstrip on the selection screen.

   Display each subscreen on its own tab with the following tab titles:

Tab	Tab title	Subscreen
1	Flight connections	1100
2	Flight date	1200
3	Flight type	1300

3. **Optional:** Ensure that the system displays the center tab page the first time it displays the selection screen.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 2: Generating and Designing a Selection Screen

Task 1:

Enhance your program *ZBC401_##_SOL* or copy the model solution *BC405_INTRO_S1* from the previous exercise.

Model solution for the task: *BC405_SCREEN_S1A*

Create selection screen

1. Create a `SELECT-OPTION` for the airline. Take the lower interval value from SAP Memory.

   Create a `SELECT-OPTION` for the flight connection number.

   Create a selection criterion for the flight date. Suppress the multiple selection option for the flight date.

   a) See the solution at the end of the exercise.

      To find the appropriate SET/GET parameters for the airline, double-click the reference variable where the select-option is declared, then double-click the type (dv_flights), and then the data element (S_CARR_ID) of the *CARRID* field. Finally, in the data element display, go to to the *Further Characteristics* tab page.

2. Create a group of three radio buttons with the following functionality:

Button	Function
1	Read **all flights**.
2	Read only **domestic flights**.
3	Read only **international flights**.

   The default setting is to be that international flights are read.

   Create a frame around the radio button group.

   a) See the solution at the end of the exercise.

3. Ensure that only the data records requested are read from the database.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

To do this, supplement the `WHERE` clause of the `SELECT` statement with the conditions that result from the selections.

Hint: You need three separate `SELECT` statements with different `WHERE` conditions. You can map the national/international condition directly on the database: `countryto = dv_flights~countryfr` or `countryto <> dv_flights~countryfr`. You use the tilde (~) to address the database field.

a) See the solution at the end of the exercise.

4. **Optional:** Program a complex preassignment for the airline, such as AA to QF, but not AZ.

   a) See the solution at the end of the exercise.

5. Activate and test your program.

   a) Press **`Ctrl + F3`** and then **`F8`**.

Task 2:

Integrate tab pages

Model solution for the task: *BC405_SCREEN_S1B*.

1. Create selection screen 1100 as a subscreen and display the selection criteria for the airline and connection number on this screen.

   Create selection screen 1200 as a subscreen and display the selection criterion for the flight date there.

   Create selection screen 1300 as a subscreen and integrate the radio buttons there.

   a) See the solution at the end of the exercise.

2. Create a tabstrip on the selection screen.

   Display each subscreen on its own tab with the following tab titles:

Tab	Tab title	Subscreen
1	Flight connections	1100
2	Flight date	1200
3	Flight type	1300

   a) See the solution at the end of the exercise.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

3. **Optional:** Ensure that the system displays the center tab page the first time it displays the selection screen.

   a) See the solution at the end of the exercise.

Result

Main Program

```
REPORT bc405_screen_s1b.

INCLUDE bc405_screen_s1b_top.
INCLUDE bc405_screen_s1b_e01.
```

TOP Include

```
* Workarea for data fetch
DATA: gs_flight type dv_flights.

* Constant for CASE statement
CONSTANTS gc_mark VALUE 'X'.

* Selections for connections
SELECTION-SCREEN BEGIN OF SCREEN 1100 AS SUBSCREEN.
SELECT-OPTIONS: so_car FOR gs_flight-carrid MEMORY ID car,
                so_con FOR gs_flight-connid.
SELECTION-SCREEN END OF SCREEN 1100.

* Selections for flights
SELECTION-SCREEN BEGIN OF SCREEN 1200 AS SUBSCREEN.
SELECT-OPTIONS so_fdt FOR gs_flight-fldate NO-EXTENSION.
SELECTION-SCREEN END OF SCREEN 1200.

* Output parameter
SELECTION-SCREEN BEGIN OF SCREEN 1300 AS SUBSCREEN.
SELECTION-SCREEN BEGIN OF BLOCK radio WITH FRAME.
PARAMETERS: pa_all RADIOBUTTON GROUP rbg1,
            pa_nat RADIOBUTTON GROUP rbg1,
            pa_int RADIOBUTTON GROUP rbg1 DEFAULT 'X'.
SELECTION-SCREEN END OF BLOCK radio.
SELECTION-SCREEN END OF SCREEN 1300.

SELECTION-SCREEN BEGIN OF TABBED BLOCK airlines
                 FOR 5 LINES.
SELECTION-SCREEN TAB (20) tab1 USER-COMMAND conn
  DEFAULT SCREEN 1100.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
SELECTION-SCREEN TAB (20) tab2 USER-COMMAND date
  DEFAULT SCREEN 1200.
SELECTION-SCREEN TAB (20) tab3 USER-COMMAND type
  DEFAULT SCREEN 1300.
SELECTION-SCREEN END OF BLOCK airlines .
```

E01 Include

```
*&---------------------------------------------------------------------*
*&   Event INITIALIZATION
*&---------------------------------------------------------------------*
INITIALIZATION.
* Initialize select-options for CARRID" OPTIONAL
  MOVE: 'AA' TO so_car-low,
        'QF' TO so_car-high,
        'BT' TO so_car-option,
        'I'  TO so_car-sign.
  APPEND so_car.

  CLEAR so_car.
  MOVE: 'AZ' TO so_car-low,
        'EQ' TO so_car-option,
        'E'  TO so_car-sign.
  APPEND so_car.

* Set texts for tabstrip pushbuttons
  tab1 = 'Connections'(tl1).
  tab2 = 'Date'(tl2).
  tab3 = 'Type of flight'(tl3).

* Set second tab page as initial tab
airlines-activetab = 'DATE'.
airlines-dynnr     = '1200'.


*&---------------------------------------------------------------------*
*&   Event START-OF-SELECTION
*&---------------------------------------------------------------------*
START-OF-SELECTION.
* Checking the output parameters
  CASE gc_mark.
    WHEN pa_all.
* Radiobutton ALL is marked
      SELECT * FROM dv_flights INTO gs_flight
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
        WHERE carrid IN so_car
        AND   connid IN so_con
        AND   fldate IN so_fdt.

        WRITE: / gs_flight-carrid,
                 gs_flight-connid,
                 gs_flight-fldate,
                 gs_flight-countryfr,
                 gs_flight-cityfrom,
                 gs_flight-airpfrom,
                 gs_flight-countryto,
                 gs_flight-cityto,
                 gs_flight-airpto,
                 gs_flight-seatsmax,
                 gs_flight-seatsocc.
      ENDSELECT.

    WHEN pa_nat.
* Radiobutton NATIONAL is marked
      SELECT * FROM dv_flights INTO gs_flight
        WHERE carrid   IN so_car
        AND   connid   IN so_con
        AND   fldate   IN so_fdt
        AND   countryto = dv_flights~countryfr.

        WRITE: / gs_flight-carrid,
                 gs_flight-connid,
                 gs_flight-fldate,
                 gs_flight-countryfr,
                 gs_flight-cityfrom,
                 gs_flight-airpfrom,
                 gs_flight-countryto,
                 gs_flight-cityto,
                 gs_flight-airpto,
                 gs_flight-seatsmax,
                 gs_flight-seatsocc.
      ENDSELECT.

    WHEN pa_int.
* Radiobutton INTERNATIONAL is marked
      SELECT * FROM dv_flights INTO gs_flight
        WHERE carrid    IN so_car
        AND   connid    IN so_con
        AND   fldate    IN so_fdt
        AND   countryto <> dv_flights~countryfr.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
        WRITE: / gs_flight-carrid,
                 gs_flight-connid,
                 gs_flight-fldate,
                 gs_flight-countryfr,
                 gs_flight-cityfrom,
                 gs_flight-airpfrom,
                 gs_flight-countryto,
                 gs_flight-cityto,
                 gs_flight-airpto,
                 gs_flight-seatsmax,
                 gs_flight-seatsocc.
    ENDSELECT.

ENDCASE.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Explain the purpose of selection screens
- Create input fields with PARAMETERS
- Create input fields with SELECT-OPTIONS
- Design the selection screen
- Define additional selection screens
- Define subscreens
- Define a tabstrip within a selection screen

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: Input Checks and Variants

Lesson Overview

In this lesson, you learn how input checks are implemented for the selection screen. It also shows you how to make only selected fields ready for input in the case of a warning or error message.

Variants are also introduced in this lesson. These allow you to repeatedly reuse values entered once on the selection screen.

Lesson Objectives

After completing this lesson, you will be able to:

- Implement input checks for selection screens
- Create variants for a selection screen

Business Example

You are working as a programmer for ABC Travel. Your manager wants to have meaningful information about ABC Airways flights to a particular city. If the user chooses a city to which ABC Airways does not fly, the selection screen is to be displayed again with an error message. To do this, you need to write a program with a user-friendly selection screen that contains the corresponding input check.

Input Checks

AT SELECTION-SCREEN	
ON <f>	ON <seltab>
ON RADIOBUTTON GROUP <grp>	ON BLOCK <block>

```
REPORT bc405_screen_d7.
...

*  Check selection criterion SO_DEPT
 AT SELECTION-SCREEN ON so_dept.
   IF ( so_dept-low LT '060000' AND so_dept-high LT '060000' )
   OR ( so_dept-low GE '220000' AND so_dept-high GE '220000' )
   AND city_fr EQ 'FRANKFURT'.
     MESSAGE e046(bc405).
   ENDIF.
...
```

Figure 21: Input checks: AT SELECTION-SCREEN

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

The evaluation of screen input takes place in the `AT SELECTION-SCREEN` event. If an error message or warning message is displayed during this event, the system makes all selection screen fields ready for input.

When you use the `ON <f> bzw. ON <seltab>` addition, you refer to specific selections. In case of errors, only these selections are made input-ready again.

The `AT SELECTION-SCREEN ON BLOCK <block>` event is available for you to check entry combinations of a logical group. Fields in this block are made ready for input when an error message is issued.

The `AT SELECTION-SCREEN ON END OF <seltab>` event belongs to the PAI processing of the selection screen for multiple selections.

You can perform input checks for the selection criteria of a logical database used and for your own program-specific selections.

```
AT SELECTION-SCREEN
  ON HELP-REQUEST FOR [<param>|<sel_opt>]
  ON VALUE-REQUEST FOR [<param>|<sel_opt-low>|<sel_opt-high>]
```

```
REPORT  bc405_screen_d9
...
SELECT-OPTIONS: so_dept FOR spfli-deptime DEFAULT '080000' TO '220000'
                NO-EXTENSION.
...
* Help request for SO_DEPT (F1 help).
  AT SELECTION-SCREEN ON HELP-REQUEST FOR so_dept.
     CALL SCREEN 100 STARTING AT 30 03
                     ENDING   AT 70 10.
```

Figure 22: Field Help and Input Help (F1 and F4)

You can program your own F1 help (field help) and F4 help (input help) for the events `AT SELECTION-SCREEN ON HELP-REQUEST` and `AT SELECTION-SCREEN ON VALUE-REQUEST`. The F1 help can refer to individual parameters or a selection table (`seltab`). If you specify the name of a `SELECT-OPTIONS` table, you receive help for both `seltab-low` and `seltab-high`.

You must specify the lower interval field or the upper interval field as a reference for an F4 help for a selection option.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

After positioning the cursor and using the F1 or F4 key, the system triggers the appropriate event.

The addition of `ON VALUE-REQUEST` displays the button for the F4 input help for the corresponding field.

Hint: F1 and F4 help are normally stored in the Dictionary. This has the advantage for the programmer that he or she does not need to deal with the actual data retrieval for the input help and the transferring of the selected value to the input field.

For the user, F1 and F4 help of this type provides a uniform, cross-program design.

Variants

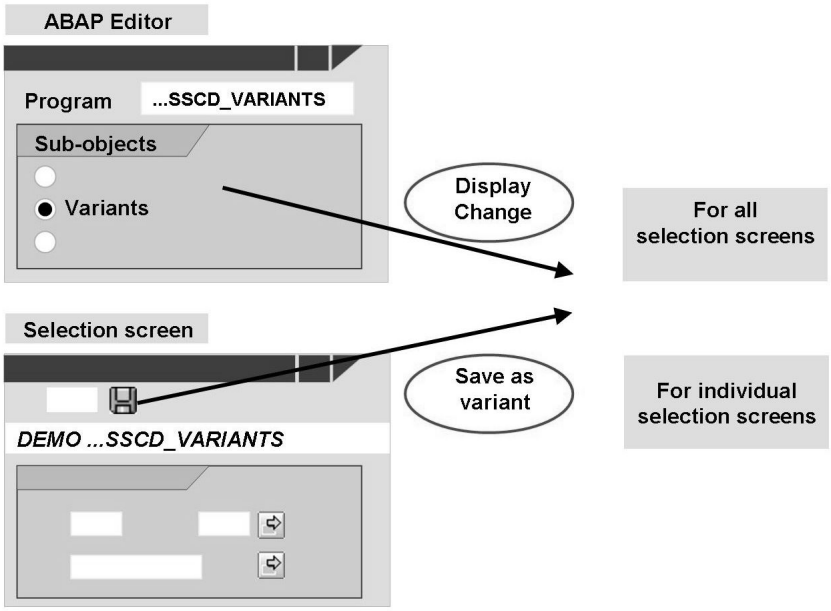


Figure 23: Selection Screen: Variants I

Users can create any number of (selection screen) variants for a program. The variants are uniquely assigned to the program and cannot be shared with other programs.

Creating variants makes sense when you frequently start a program with the same selection subset or run the program in the background.

If you select *Start using variant* in the program attributes, the system forces the user to specify a variant when executing the program.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

If the program uses several selection screens, you can choose to create a variant for all the selection screens or individually for each selection screen.

Naming conventions and transporting variants

`SAP&xxx` are supplied with the SAP system

`CUS&xxx` are created by the customer

Variants that follow these naming conventions are client-independent and will automatically be transported along with the report. If these naming conventions are not followed, an entry for a request (task) must be added to the object list. To do this, choose *Goto → Variants* in the ABAP Editor to switch to the variant maintenance and enter the variants to be transported there by choosing *Utilities → Transport Request*.

Selection Screen: Variants II

Attributes	
Variant:	Only for background processing
	Protect variant
	Display only in catalog
	System variant (automatic transport)
Field attributes:	Obligatory field
	Deactivate SET/GET parameters
	Save field without values
	Selection variable
	Hide To field
	Hide field
	Protect field

You have to assign a name and a description to each variant. By default, variants are available for both online and background processing. You can also define a variant exclusively for use with background processing.

You can protect your variant against changes by other users. If you select *Only display in catalog*, this variant will not be displayed in the general value help (F4), but only in the catalog. The variant catalog is part of the ABAP Workbench and therefore normally only accessible to ABAP developers but not to the normal users of a program. You can call the variants catalog (an overview of all existing variants), starting from the ABAP Editor, by choosing *Goto → Variants* and then *Variants → Catalog*.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The *Type* of a selection is defined in its declaration (type `S` for `SELECT-OPTIONS`, type `P` for `PARAMETERS`). If you select *Selections protected*, then the field(s) will not be ready for input. You can use the *Hide* attribute to hide selections that are not required, therefore making the selection screen clearer.

If you use *selection variables*, there are essentially three ways of supplying your selections with values at runtime:

- From the `TVARVC` table (`T` type): You can permanently define variables with values in this table; the system uses the variable names to retrieve the value at runtime.
- You can use dynamic date calculation (type `D`) to supply date fields with values, such as the day's date or the current month.
- With user-specific variables (`B` type), here you must declare the selection with the `MEMORY ID <pid>` addition.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Creating a Variant with Dynamic Date Calculation

Hint: This describes the procedure for *SAP NetWeaver 7.0*. Although the idea of dynamic date calculation was similar in earlier releases, the procedure was slightly different.

1. Create a variant and switch to the *Variant Attributes*.
2. Use F4 help to define the type of selection variable (*Selection Variable* attribute), for dynamic date calculation **D**.
3. Use F4 to define the name of the variable (*Name of Variable* attribute), such as *Current Month* with dynamic date calculation for a time interval.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 3: Input Checks and Variants

Exercise Objectives

After completing this exercise, you will be able to:

- Initialize and check the selection screen
- Create variants

Business Example

Extend your flight data report: If the user chooses *Domestic Flights*, you must also specify a country to limit the number of flights. Program a corresponding input check for the selection screen.

Task:

Enhance your program *ZBC401_##_SOL* or copy the model solution *BC405_SCREEN_S1B* from the previous exercise.

Model solution for the task: *BC405_SCREEN_S2*.

Add an additional parameter and an input check to the selection screen.

1. On the third tab page, create an additional parameter for the country code that references `gs_flight-countryfr` and requires the user to enter a value there if he or she selects domestic flights. For such a case, enhance the `WHERE` clause of the relevant *SELECT*.
2. Output error message 003 of message class *BC405* if the user has selected the *Domestic Flights* option and the input parameter for the country is initial. In case of error, only the radio button group and the country parameter should be ready for input.
3. Create a variant **`V##`**, where ## is your group number. Use dynamic date calculation (selection variable) when doing so, and set up the flight date so that the current month is used.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 3: Input Checks and Variants

Task:

Enhance your program *ZBC401_##_SOL* or copy the model solution *BC405_SCREEN_S1B* from the previous exercise.

Model solution for the task: *BC405_SCREEN_S2*.

Add an additional parameter and an input check to the selection screen.

1. On the third tab page, create an additional parameter for the country code that references `gs_flight-countryfr` and requires the user to enter a value there if he or she selects domestic flights. For such a case, enhance the `WHERE` clause of the relevant *SELECT*.
   a) See the solution at the end of the task.
2. Output error message 003 of message class *BC405* if the user has selected the *Domestic Flights* option and the input parameter for the country is initial. In case of error, only the radio button group and the country parameter should be ready for input.
   a) See the solution at the end of the task.
3. Create a variant **V##**, where ## is your group number. Use dynamic date calculation (selection variable) when doing so, and set up the flight date so that the current month is used.
   a) Create the variant *V##*, where ## is your group number, from the selection screen by choosing **Save as Variant**.
   b) Select the *Flight Date* input field on selection screen 1200.
   c) Choose **D** for the *Selection Variable* attribute for the selected input field.
   d) Choose the value **Current Month** for the Name of Variable attribute for the selected input field.

Result

Framework Program

```
REPORT bc405_screen_s2.

INCLUDE bc405_screen_s2_top.
INCLUDE bc405_screen_s2_e01.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

TOP Include

```
* Workarea for data fetch
DATA: gs_flight TYPE dv_flights.

* Constant for CASE statement
CONSTANTS gc_mark VALUE 'X'.

* Selections for connections
SELECTION-SCREEN BEGIN OF SCREEN 1100 AS SUBSCREEN.
SELECT-OPTIONS: so_car FOR gs_flight-carrid,
                so_con FOR gs_flight-connid.
SELECTION-SCREEN END OF SCREEN 1100.

* Selections for flights
SELECTION-SCREEN BEGIN OF SCREEN 1200 AS SUBSCREEN.
SELECT-OPTIONS so_fdt FOR gs_flight-fldate NO-EXTENSION.
SELECTION-SCREEN END OF SCREEN 1200.

* Output parameter
SELECTION-SCREEN BEGIN OF SCREEN 1300 AS SUBSCREEN.

SELECTION-SCREEN BEGIN OF BLOCK param.
SELECTION-SCREEN BEGIN OF BLOCK radio WITH FRAME.
PARAMETERS: pa_all RADIOBUTTON GROUP rbg1,
            pa_nat RADIOBUTTON GROUP rbg1,
            pa_int RADIOBUTTON GROUP rbg1 DEFAULT 'X'.

SELECTION-SCREEN END OF BLOCK radio.
PARAMETERS: pa_ctry LIKE gs_flight-countryfr.
SELECTION-SCREEN END OF BLOCK param.
SELECTION-SCREEN END OF SCREEN 1300.
```

E01 Include

```
AT SELECTION-SCREEN ON BLOCK param.

* check country for national flights is not empty
  CHECK pa_nat = 'X' AND pa_ctry = space.
  MESSAGE e003(bc405).


START-OF-SELECTION.
...
* Radiobutton NATIONAL is marked
      SELECT * FROM dv_flights INTO gs_flight
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
WHERE carrid IN so_car
  AND connid IN so_con
  AND fldate IN so_fdt
  AND countryto = dv_flights~countryfr
  AND countryto = pa_ctry.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Implement input checks for selection screens
- Create variants for a selection screen

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: Selection Screen: Modifications at Runtime

Lesson Overview

This lesson shows you how you can change the properties of a selection screen element (such as an input field) at runtime, for example from “ready for input and output” to “ready for output”. In this context, the lesson also explains how to create buttons on the selection screen.

Lesson Objectives

After completing this lesson, you will be able to:

- Create buttons on the selection screen
- Program modifications of the selection screen at runtime

Business Example

You are working as a programmer for ABC Travel. Your manager wants you to change the report for the flight data so that a number of selection fields are only shown if they are required, and are not always visible. You have been given the task of performing this conversion.

Generating Buttons on the Selection Screen

```
* Selection screen structure with user command
TABLES sscrfields.

* Pushbutton
SELECTION-SCREEN
      PUSHBUTTON /pos_low(12) gv_txt USER-COMMAND on.

* Pushbutton text
INITIALIZATION.
  gv_txt = 'Display details'(s01).
```

Figure 24: Generating Buttons on the Selection Screen

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

You can generate buttons on the selection screen. Use the statement `SELECTION-SCREEN PUSHBUTTON ... .` to do this. Note that, by default, the button is not positioned in a new line. To insert an explicit line break, combine your position specification with a slash (/), for example /1(20).

You need to specify a name and a user command for the button.

You need to maintain the text that is to appear on the button separately. You do this using text symbols, which you assign to the button during `INITIALIZATION`. Alternatively, you can also use the ICON_CREATE function module to generate a text and/or an icon.

If the user chooses the button, the user command assigned to it is triggered and placed in the `sscrfields-ucomm` field. You therefore need to include the statement `TABLES sscrfields` in the program.

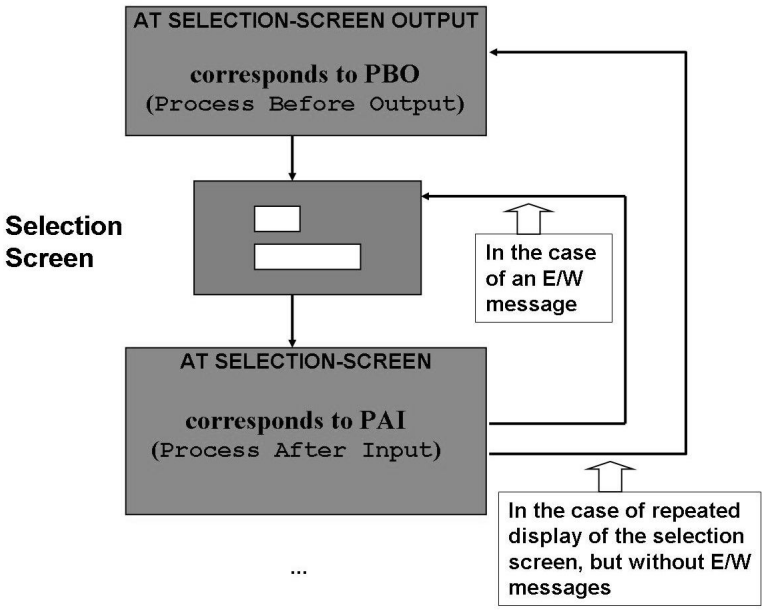


Figure 25: ABAP Events for the Selection Screen Versus Dynpro Events

Selection screens are special dynpros generated by the system that are generated only on the basis of ABAP statements. In the case of normal dynpro programming, there are two times for each dynpro at which the process logic can be implemented:

- **Process Before Output (PBO)** immediately before the dynpro is displayed
- **Process After Input (PAI)** immediately after the dynpro has been displayed

The equivalents to these events for the selection screen are:

- **Process Before Output (PBO):** `AT SELECTION-SCREEN OUTPUT`
- **Process After Input (PAI):** `AT SELECTION-SCREEN`

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

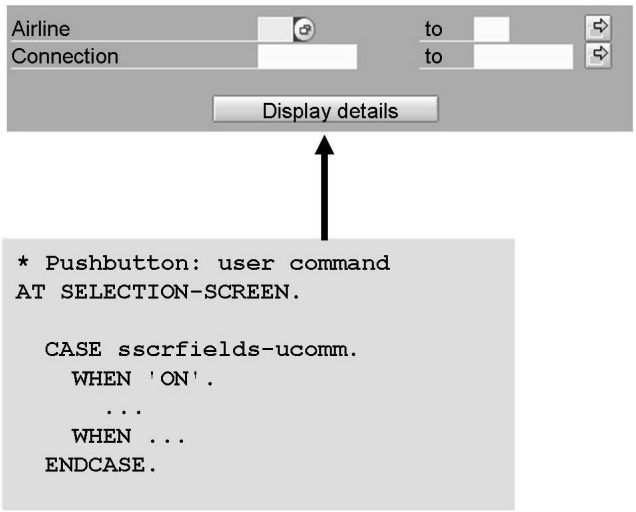


Figure 26: Evaluating Buttons on the Selection Screen

At the PAI time for the selection screen, the user command is placed in the `sscrfields-ucomm` field. The PAI time is represented by the `AT SELECTION-SCREEN` event.

In this way, you can determine which button was chosen by the user, and react accordingly in the application. The best way to implement this is using a case distinction in a `CASE` statement.

Changing the Selection Screen at Runtime

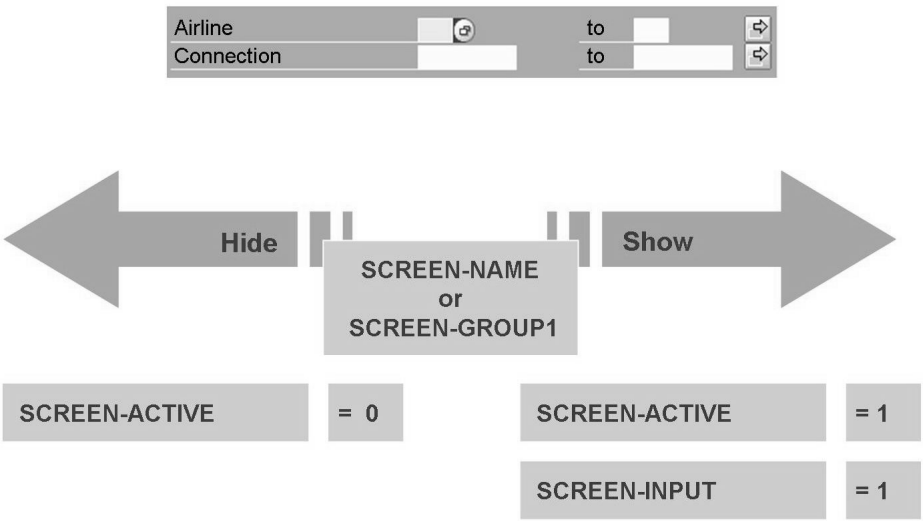


Figure 27: Showing and Hiding Selection Fields

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Every field of the selection screen has an entry in the system table SCREEN. You can change the contents of this table.

You can hide selection fields by assigning the value 0 to the attribute SCREEN-ACTIVE. You control which fields are hidden using either the name of the field or the group name.

For the ready for input status of fields, you must assign 0 or 1 to the screen-input field.

```
DATA: gs_booking TYPE sbook.
SELECT-OPTIONS:
  so_car FOR gs_booking-carrid,
  so_con FOR gs_booking-connid MODIF ID abc.


* Screen modification
AT SELECTION-SCREEN OUTPUT.

  LOOP AT SCREEN.
    IF screen-name  CS 'SO_CON' OR
       screen-group1 = 'ABC'.
      screen-active = 0.
      MODIFY SCREEN.
    ENDIF.
  ENDLOOP.
```

SCREEN-NAME	SO_CON
SCREEN-GROUP1	ABC
SCREEN-ACTIVE	1. 0
SCREEN-INPUT	1. 0

Figure 28: LOOP AT SCREEN

Every field on the selection screen has a name, such as so_car-low and so_car-high for the low and high field of a SELECT-OPTIONS so_car.

You can address these fields within LOOP AT SCREEN ... ENDLOOP using their names (field screen-name) and show and hide them on the selection screen at runtime. The showing and hiding must be performed at the PBO time of the selection screen. This time is represented by the AT SELECTION-SCREEN OUTPUT event.

So that you do not need to address each field individually, you can combine selections into modification groups. You assign the name of a modification group to the PARAMETERS or SELECT-OPTIONS statement with the MODIF ID <xxx> addition.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 4: Selection Screen: Modifications at Runtime

Exercise Objectives

After completing this exercise, you will be able to:

- Define buttons on a selection screen
- Evaluate buttons
- Dynamically show and hide selections

Business Example

You are a programmer for ABC Travel. The users of your program for flight data reporting want, if necessary, to be able to select flights by departure and arrival cities. To avoid making the selection screen cluttered, the selection options should only be shown if they are required by choosing a button.

Task:

Extend your program *ZBC405_##_SOL* or copy the program *BC405_SCREEN_S2*.

Model solution: *BC405_SCREEN_S3*

1. Create two more select-options on subscreen 1100: *City of Departure* and *City of Arrival* (with a reference to `gs_flight-cityfrom` / `gs_flight-cityto`). Place a frame around the selections and assign a frame title. Adjust the database accesses.

2. A button is to be shown below the select-option `so_con` (connection number). By default, this is to display the text “Hide Details”.

   If the user chooses the “Hide Details” pushbutton, the system should hide *City of Departure* and *City of Arrival*. The pushbutton should now display “Show Details”.

   If the user chooses the pushbutton in its “Show Details” state, the hidden fields should become visible again.

   Hint:

   You need to use the events `INITIALIZATION`, `AT SELECTION-SCREEN`, and `AT SELECTION-SCREEN OUTPUT`.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 4: Selection Screen: Modifications at Runtime

Task:

Extend your program *ZBC405_##_SOL* or copy the program *BC405_SCREEN_S2*.

Model solution: *BC405_SCREEN_S3*

1. Create two more select-options on subscreen 1100: *City of Departure* and *City of Arrival* (with a reference to `gs_flight-cityfrom` / `gs_flight-cityto`). Place a frame around the selections and assign a frame title. Adjust the database accesses.

   a) See the solution at the end of the task.

2. A button is to be shown below the select-option `so_con` (connection number). By default, this is to display the text "Hide Details".

   If the user chooses the "Hide Details" pushbutton, the system should hide *City of Departure* and *City of Arrival*. The pushbutton should now display "Show Details".

   If the user chooses the pushbutton in its "Show Details" state, the hidden fields should become visible again.

   Hint:

   You need to use the events `INITIALIZATION`, `AT SELECTION-SCREEN`, and `AT SELECTION-SCREEN OUTPUT`.

   a) See the solution at the end of the task.

Result

Framework Program

```
REPORT bc405_screen_s3.

INCLUDE bc405_screen_s3_top.
INCLUDE bc405_screen_s3_e01.
```

TOP Include

```
* Workarea for data fetch
DATA: gs_flight TYPE dv_flights.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
* Constant for CASE statement
CONSTANTS gc_mark VALUE 'X'.
* Data objects for dynamic screen modifications
TABLES sscrfields.
DATA gv_switch TYPE n VALUE '1'.

* Selections for connections
SELECTION-SCREEN BEGIN OF SCREEN 1100 AS SUBSCREEN.
SELECT-OPTIONS: so_car FOR gs_flight-carrid,
                so_con FOR gs_flight-connid.
SELECTION-SCREEN SKIP 2.
SELECTION-SCREEN PUSHBUTTON pos_low(20) gv_txt USER-COMMAND details.
SELECTION-SCREEN SKIP.
SELECTION-SCREEN BEGIN OF BLOCK bl_det WITH FRAME TITLE text-tld.
  SELECT-OPTIONS:
    so_start FOR gs_flight-cityfrom MODIF ID det,
    so_dest  FOR gs_flight-cityto   MODIF ID det.
SELECTION-SCREEN END OF BLOCK bl_det.
SELECTION-SCREEN END OF SCREEN 1100.

* Selections for flights
SELECTION-SCREEN BEGIN OF SCREEN 1200 AS SUBSCREEN.
SELECT-OPTIONS so_fdt FOR gs_flight-fldate NO-EXTENSION.
SELECTION-SCREEN END OF SCREEN 1200.

* Output parameter
SELECTION-SCREEN BEGIN OF SCREEN 1300 AS SUBSCREEN.
SELECTION-SCREEN BEGIN OF BLOCK param WITH FRAME TITLE text-tl3.
SELECTION-SCREEN BEGIN OF BLOCK radio WITH FRAME.
PARAMETERS:
  pa_all RADIOBUTTON GROUP rbg1,
  pa_nat RADIOBUTTON GROUP rbg1,
  pa_int RADIOBUTTON GROUP rbg1 DEFAULT 'X'.
SELECTION-SCREEN END OF BLOCK radio.
PARAMETERS: pa_ctry LIKE gs_flight-countryfr.
SELECTION-SCREEN END OF BLOCK param.
SELECTION-SCREEN END OF SCREEN 1300.

SELECTION-SCREEN BEGIN OF TABBED BLOCK airlines
  FOR 10 LINES.
SELECTION-SCREEN TAB (20) tab1 USER-COMMAND conn
  DEFAULT SCREEN 1100.
SELECTION-SCREEN TAB (20) tab2 USER-COMMAND date
  DEFAULT SCREEN 1200.
SELECTION-SCREEN TAB (20) tab3 USER-COMMAND type
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
  DEFAULT SCREEN 1300.
SELECTION-SCREEN END OF BLOCK airlines.
```

E01 Include

```
*&---------------------------------------------------
*&   Event INITIALIZATION
*&---------------------------------------------------
INITIALIZATION.
* Initialize select-options for CARRID" OPTIONAL
  MOVE: 'AA' TO so_car-low,
        'QF' TO so_car-high,
        'BT' TO so_car-option,
        'I'  TO so_car-sign.
  APPEND so_car.

  CLEAR so_car.
  MOVE: 'AZ' TO so_car-low,
        'EQ' TO so_car-option,
        'E'  TO so_car-sign.
  APPEND so_car.

* Set texts for tabstrip pushbuttons
  tab1 = 'Connections'(tl1).
  tab2 = 'Date'(tl2).
  tab3 = 'Type of flight'(tl3).

* Set second tab page as initial tab
  airlines-activetab = 'DATE'.
  airlines-dynnr     = '1200'.

* Text for pushbuttons
  gv_txt = 'Hide details'(p02).


*&---------------------------------------------------
*&   Event AT SELECTION-SCREEN ON BLOCK PARAM
*&---------------------------------------------------
AT SELECTION-SCREEN ON BLOCK param.    " OPTIONAL

* check country for national flights is not empty
  CHECK pa_nat = 'X' AND pa_ctry = space.
  MESSAGE e003(bc405).
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
*&---------------------------------------------------
*&   Event at selection-screen output
*&---------------------------------------------------
AT SELECTION-SCREEN OUTPUT.
  CASE sy-dynnr.

    WHEN 1100.
      LOOP AT SCREEN.
        IF screen-group1 = 'DET'.
          screen-active = gv_switch.
          MODIFY SCREEN.
        ENDIF.
      ENDLOOP.

      IF gv_switch = '1'.
        gv_txt = text-p02.
      ELSE.
        gv_txt = text-p01.
*       clear additional select-options to avoid unwanted
*       influence at selection from database
        REFRESH: so_start,
                 so_dest.
      ENDIF.
  ENDCASE.


*&---------------------------------------------------
*&   Event at selection-screen
*&---------------------------------------------------
AT SELECTION-SCREEN.
* Evaluate pushbutton command
  CASE sscrfields-ucomm.
    WHEN 'DETAILS'.
      CHECK sy-dynnr = 1100.
      IF gv_switch = '1'.
        gv_switch = '0'.
      ELSE.
        gv_switch = '1'.
      ENDIF.
  ENDCASE.


*&---------------------------------------------------
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
*&   Event START-OF-SELECTION
*&---------------------------------------------------
START-OF-SELECTION.

* Checking the output parameters
  CASE gc_mark.
    WHEN pa_all.
*     Radiobutton ALL is marked
      SELECT * FROM dv_flights INTO gs_flight
        WHERE carrid   IN so_car
        AND   connid   IN so_con
        AND   fldate   IN so_fdt
        AND   cityfrom IN so_start
        AND   cityto   IN so_dest.
        WRITE: / gs_flight-carrid,
                 gs_flight-connid,
                 gs_flight-fldate,
                 gs_flight-countryfr,
                 gs_flight-cityfrom,
                 gs_flight-airpfrom,
                 gs_flight-countryto,
                 gs_flight-cityto,
                 gs_flight-airpto,
                 gs_flight-seatsmax,
                 gs_flight-seatsocc.
      ENDSELECT.

    WHEN pa_nat.
*     Radiobutton NATIONAL is marked
      SELECT * FROM dv_flights INTO gs_flight
        WHERE carrid   IN so_car
        AND   connid   IN so_con
        AND   fldate   IN so_fdt
        AND   cityfrom IN so_start
        AND   cityto   IN so_dest
        AND   countryto = dv_flights~countryfr
        AND   countryto = pa_ctry.

        WRITE: / gs_flight-carrid,
                 gs_flight-connid,
                 gs_flight-fldate,
                 gs_flight-countryfr,
                 gs_flight-cityfrom,
                 gs_flight-airpfrom,
                 gs_flight-countryto,
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
                gs_flight-cityto,
                gs_flight-airpto,
                gs_flight-seatsmax,
                gs_flight-seatsocc.
      ENDSELECT.

    WHEN pa_int.
*     Radiobutton INTERNAT is marked
      SELECT * FROM dv_flights INTO gs_flight
        WHERE carrid    IN so_car
        AND   connid    IN so_con
        AND   fldate    IN so_fdt
        AND   cityfrom  IN so_start
        AND   cityto    IN so_dest
        AND   countryto <> dv_flights~countryfr.

        WRITE: / gs_flight-carrid,
                 gs_flight-connid,
                 gs_flight-fldate,
                 gs_flight-countryfr,
                 gs_flight-cityfrom,
                 gs_flight-airpfrom,
                 gs_flight-countryto,
                 gs_flight-cityto,
                 gs_flight-airpto,
                 gs_flight-seatsmax,
                 gs_flight-seatsocc.
      ENDSELECT.
  ENDCASE.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Create buttons on the selection screen
- Program modifications of the selection screen at runtime

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Explain the purpose of selection screens
- Create input fields with PARAMETERS
- Create input fields with SELECT-OPTIONS
- Design the selection screen
- Define additional selection screens
- Define subscreens
- Define a tabstrip within a selection screen
- Implement input checks for selection screens
- Create variants for a selection screen
- Create buttons on the selection screen
- Program modifications of the selection screen at runtime

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Unit 3

ALV: Usage

Internal Use SAP Partner Only

Unit Overview

Since the introduction of the SAP List Viewer (ALV), the classic ABAP list for displaying tabular data should no longer be used, because it is more complex to program and less convenient for the user.

The ALV for displaying non-hierarchical lists is an intuitive tool. Only the basic functions are defined here.

Unit Objectives

After completing this unit, you will be able to:

- Describe the major functions of ALV
- Operate the ALV (from the user view)

Unit Contents

Lesson: Using the ALV to Display Data in Table Form..... 74

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: Using the ALV to Display Data in Table Form

Lesson Overview

This will familiarize you with the basic functions of the SAP List Viewer (ALV), or to be more precise, the form used to display simple, two-dimensional tables (without hierarchy levels).

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the major functions of ALV
- Operate the ALV (from the user view)

Business Example

You are a programmer for ABC Travel. Management wants you to present all the functions of the ALV (from the user view). So you must familiarize yourself with the functions of the ALV by testing everything out carefully.

Using the ALV to Display Data

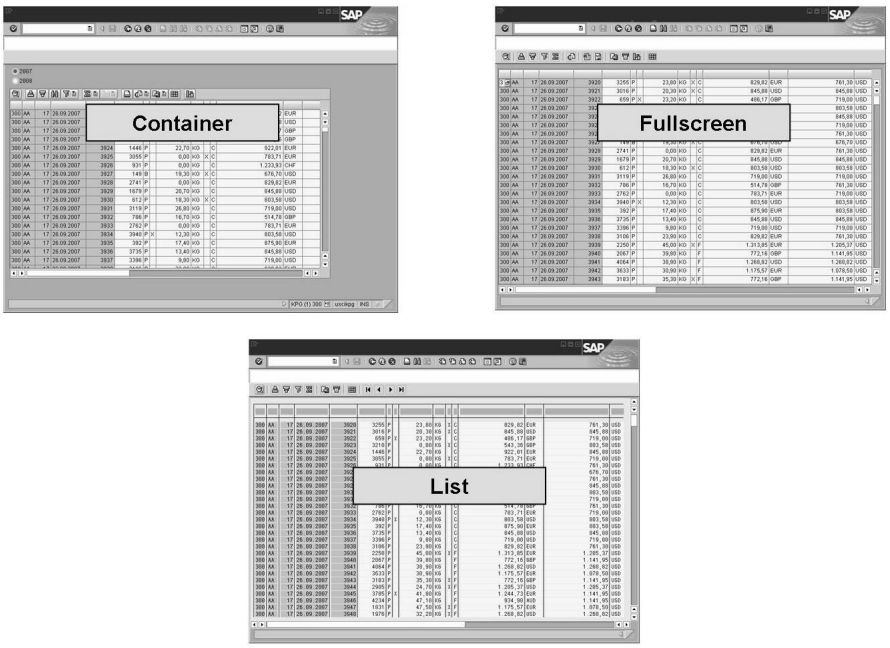


Figure 29: ALV: Overview

The **SAP List Viewer** is used in many standard reports and reporting tools (such as SAP Query). ABAP programmers can install it in their own programs.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

ALV is typically used as an abbreviation; it goes back to the old name (**A**BAP **L**ist **V**iewer).

The ALV has a number of interactive standard functions that list users frequently use (such as *Print*). As a developer, you have the option of hiding, adjusting, or extending those standard functions.

The ALV Object Model (ALV OM), which was introduced for *SAP NetWeaver 2004*, offers three output forms for non-hierarchical lists: full screen, embedded in a container (filling only a part of a subscreen container), or the classic ABAP list. The ALV is always embedded in a container in the ALV Grid Control version (introduced for *SAP R/3 4.6*).

In many respects, you operate it in a similar way. Show the ALV in the container as an example.

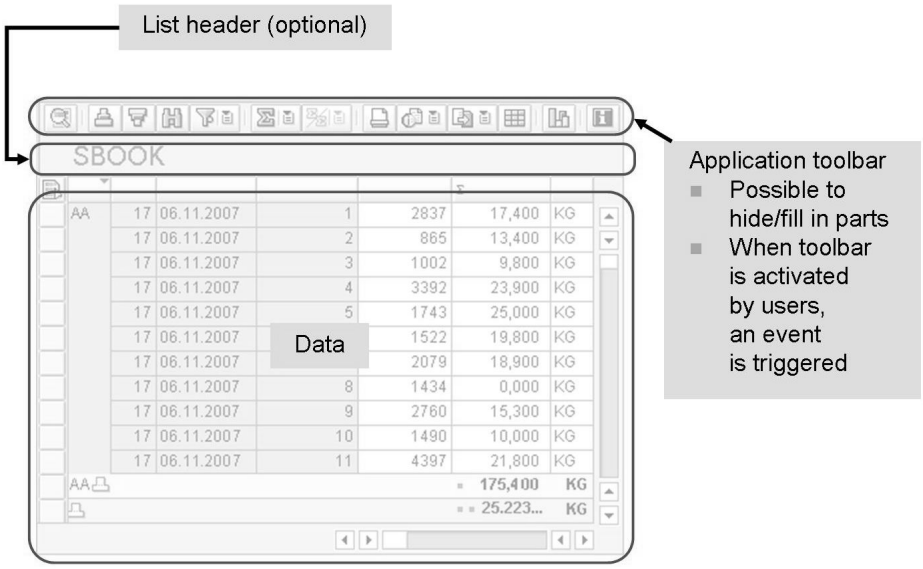


Figure 30: ALV Screen Layout (in Container)

The screen layout of the container ALV is composed of three areas:

- The application toolbar is displayed in the header area. The calling program can hide individual functions, or even the entire application toolbar.
- The calling program can display a single-line list header under the application toolbar.
- Below the header is the area that displays list data (known as the data area).

The data area itself has a structure of its own.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

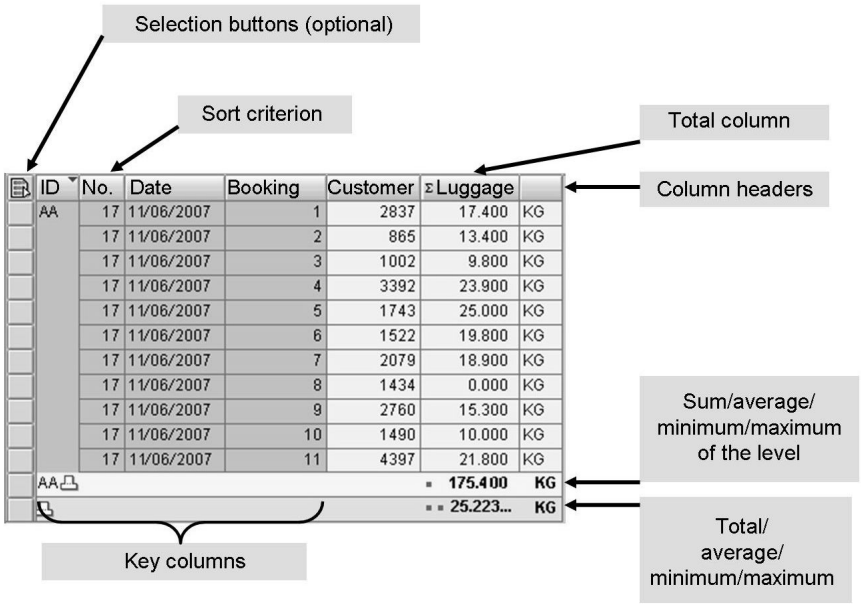


Figure 31: Data Area Layout

The data is passed from the calling program as a table.

If the data table contains more rows and columns than can be displayed in the control area, horizontal and vertical scroll bars appear automatically.

Columns can be divided into lead columns and non-lead columns. Lead columns have a different color from non-lead columns. They also have a fixed position and therefore remain unchanged when the user scrolls horizontally through the list. Furthermore, you can make any column in your data table a lead column from the calling program.

You can select (depending on the implementation in the program): Cells and groups of cells, whole rows, whole columns, or whole rows and columns. If *Line Selection* is configured from the calling program, the system automatically displays a column (the first column in the data area) with pushbuttons that allow the user to select whole lines.

If a column is used as a sort criterion, a small red triangle will indicate that.

Columns that are totaled for have a sigma sign (Σ) in their header.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Display details for this row in the list in a dialog box

Sort ascending (by selected column, or in the order chosen by the user)

Sort descending (by selected column, or in the order chosen by the user)

Search list

Set filter (restrict data displayed using conditions chosen by the user)

Total: aggregate functions for selected, numerical columns

Subtotal for control levels

Print

Views (including print preview)

Export (HTML, spreadsheet, word processing, local file, etc.)

Change layout using dialog

Display graphic

Figure 32: Standard Functions

The following functions can be offered as standard in the application toolbar, so that you do **not** need to implement them yourself:

- *Details:* Displays the data for the selected row in a modal dialog box.
- *Sort ascending/descending:* Displays the list content sorted by the selected column or columns. Users can specify as many columns as they want, and the order in which they are to used for sorting, in a dialog box.
- *Search:* Users can specify search terms and search order (columns or rows).
- *Set filter:* Restricts the dataset display based on any filter criteria the user chooses.
- *Total:* Users can choose to display the total, arithmetic mean, maximum value, or minimum value for the numeric columns they selected.
- *Subtotal:* Displays subtotals for a data group (which the user defines by selecting a non-numeric column).
- *Print:* Opens the standard print dialog box.
- *Views: Output List* displays a print preview.
- *Export:* Allows users to open other installed applications with the data.
- *Layout:* Allows users to set up other display variants temporarily in a single dialog box, along with the standard functions already described.
- *Graphic:* Opens a graphic editor with the data from the list. Users can choose other processing options by double-clicking.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

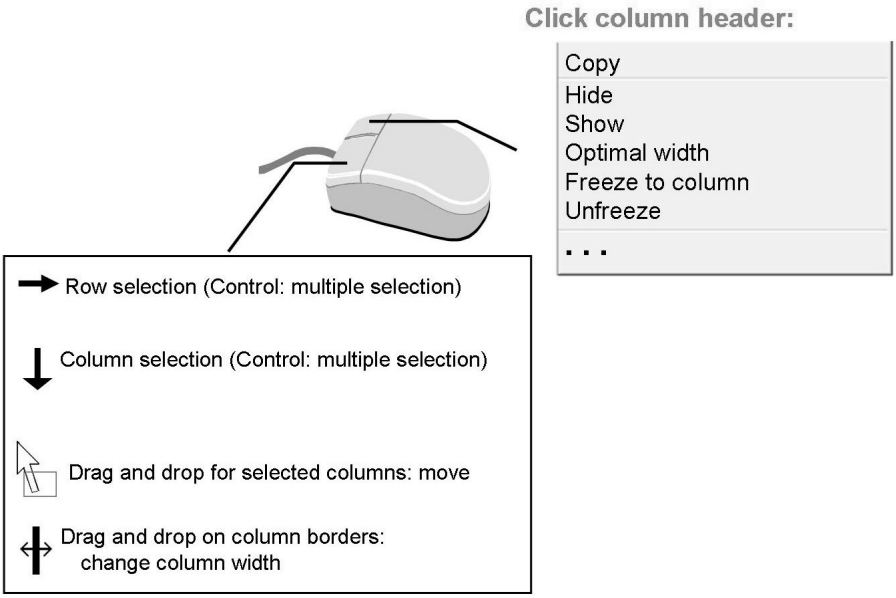


Figure 33: Using the Mouse

Functions of the **left mouse button**:

- You select columns by left-clicking the column header. You select several columns by holding down the control key on the keyboard at the same time. Click the column header again to deselect it.
- You select a line by clicking it.
- You move selected columns by holding down the left mouse button and dragging the column to the required position (drag & drop).
- You change the column width by holding down the left mouse button and dragging a column margin in the header line to the required position (Drag&Drop).

The mouse pointer changes according to the function you choose.

Important functions of the **right mouse button (context menu)** when you click a column header:

- You can hide selected columns for a clearer layout. By choosing *Show*, you can choose to show the required columns again in a dialog box.
- If you choose the *Optimum Width* function, the system will adjust all of the columns displayed to their optimum width.
- You can define a selected column as the last fixed column: When you scroll across, only columns to the right of the selected column are scrolled through. This is particularly useful when you have a wide list in which not all columns can be displayed on the screen at once. It means that you can still see the most important columns all the time.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

1. Select numerical column(s)
2. Form total(s)
3. Possibly select non-numerical column(s) for control levels
4. Create subtotals
5. Instead of 3: Make entries in a dialog box:

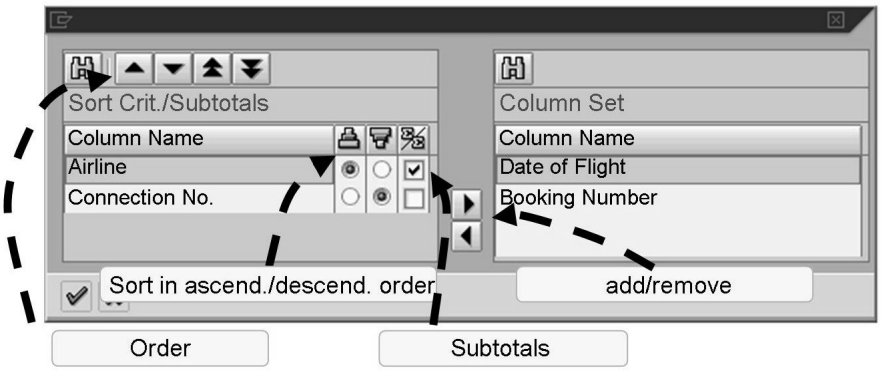


Figure 34: Subtotals

To generate subtotals in the ALV:

1. Select the column or columns for which you want to create totals.
2. Click the pushbutton that has the sigma symbol. This makes the pushbutton for subtotals visible.
3. If you do not require complex subtotals, select the columns for which you want to create subtotals. They must be non-numerical columns. Then when you create subtotals, the list is sorted in ascending order according to the fields in the columns you selected and subtotals are created for all the columns you selected.
4. You can then create the subtotals by clicking the corresponding pushbutton.
5. For more complex subtotals, leave out step three. When you click the subtotals pushbutton, a dialog box appears in which the columns that you want to use for sorting are shown on the left and the rest of the columns are shown on the right. You move columns from one side to the other by double-clicking them, for example. On the left-hand side, you can determine for each column in the ALV whether you want to sort in ascending or descending order and whether you want to create subtotals. Moreover, you can influence the sort hierarchy by using the pushbuttons with the arrows (top-left) to move sort criteria up or down.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Describe the major functions of ALV
- Operate the ALV (from the user view)

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Describe the major functions of ALV
- Operate the ALV (from the user view)

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Unit 4

ALV: Creation

Internal Use SAP Partner Only

Unit Overview

In this unit, you will learn how to embed an ALV into a custom container on a dynpro.

Unit Objectives

After completing this unit, you will be able to:

- Create a screen in which the ALV is to be embedded
- Program the screen flow logic for screen processing
- Create a GUI status with the Menu Painter
- Process a user command in a PAI module
- Explain the SAP Control Framework for GUI Controls
- Generate and link objects
- Display data in the ALV Grid Control

Unit Contents

Lesson: ALV in the Container: Preparing the Container Screen 84
Exercise 5: ALV Grid Control: Preparing the Container Screen 97
Lesson: ALV Grid Control: Architecture 104
Exercise 6: ALV Grid Control: Architecture 123

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV in the Container: Preparing the Container Screen

Lesson Overview

In this unit, you learn how to program a container screen so that it can incorporate an ALV.

Lesson Objectives

After completing this lesson, you will be able to:

- Create a screen in which the ALV is to be embedded
- Program the screen flow logic for screen processing
- Create a GUI status with the Menu Painter
- Process a user command in a PAI module

Business Example

You are a programmer at ABC Travel. The manager wants to see data in an ALV. This ALV is to appear on a screen in such a way that other screen elements (such as input fields) can also be on that screen. To make this possible, you have to create the container screen in a first step and implement its flow logic.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Control Framework

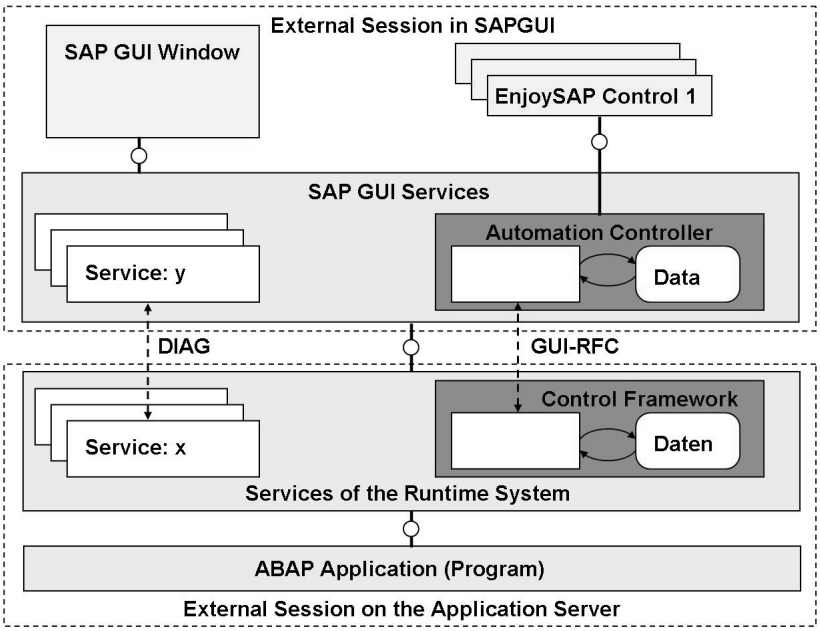


Figure 35: Control Framework Architecture: Overview

The graphical user interface in the *SAP* system is based on *SAP GUI* windows (screens, from the technical viewpoint of the programmer). Every user dialog in the system is implemented using dynpros of application programs.

The communication between the application program and the dynpro at runtime takes place using various services of the runtime system (ABAP processor and dynpro processor) and using services in the *SAP GUI* (managing the windows and their pop-up levels, data transfer, and so on).

Since *SAP R/3 4.5*, you can also use screen elements other than those provided in the *Screen Painter* in *SAP GUI* windows. These additional screen elements are known as controls. Controls are standalone binary software components that are reusable. As a developer, you can integrate one or more controls in your user dialogs and use the functions they make available.

The communication between the control and the application program is different from that of the "classic" screen elements. As of *SAP R/3 4.6A*, communication for all controls is performed in the same way using the Control Framework (CFW). The Control Framework is implemented using special services in the *SAP GUI* and the runtime system on the application server (the Automation Controller on the presentation server and the CFW service on the application server).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

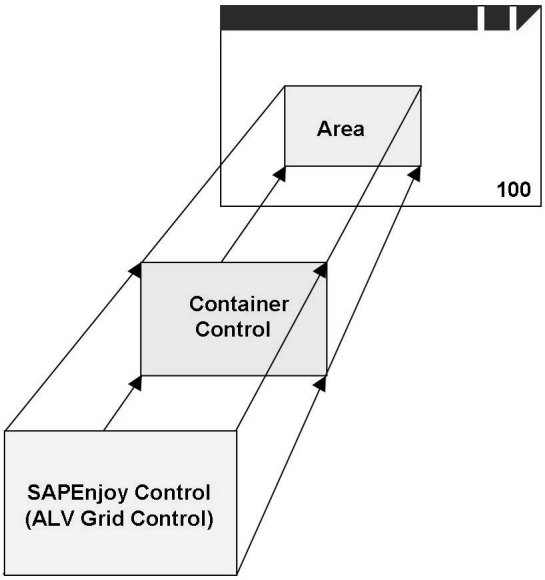


Figure 36: Graphic Illustration of the Objects

You cannot create a standalone instance of a control on the presentation server. You must always have two partners there: a screen and a special GUI Control, also known as a container control.

Graphically considered, the connection of a GUI Control to a dynpro is nothing other than the “interleaving” of the areas concerned:

- An area is reserved on the screen to display a control.
- A container control is visualized in this area.
- The area of the container control itself acts as the basis for the visualization of the GUI Control, such as an ALV Grid Control.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Creating a Screen

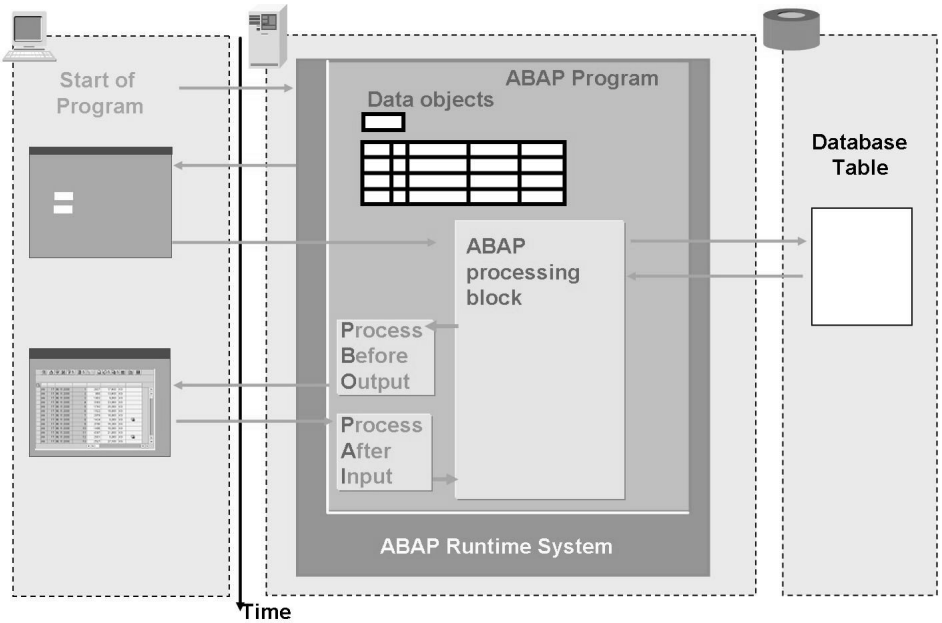


Figure 37: Screens as carriers for the ALV

The main steps of programming a screen are set out below. Let us look at a simple dialog program with a selection screen as its initial screen and a screen for displaying an ALV.

When the program starts, the system loads its program context and prepares memory space for the program's data objects. The selection screen is displayed.

The user enters data and chooses *Execute* (F8) to continue the program.

Within a processing block, data is read from the database and transferred to an internal table in accordance with the selection screen.

The processing logic calls a screen. Then, a processing block (PBO = **P**rocess **B**efore **O**utput) for that screen is triggered first. Within the PBO, the system also prepares the ALV.

Any user action on the screen (Enter, choosing a menu entry, clicking a pushbutton, etc.) returns the control to the runtime system; this triggers a processing block for the screen, which is always processed sequentially immediately after a user action (PAI = **P**rocess **A**fter **I**nput).

However, there are specific ALV user actions that do not necessarily trigger a PAI/PBO cycle.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

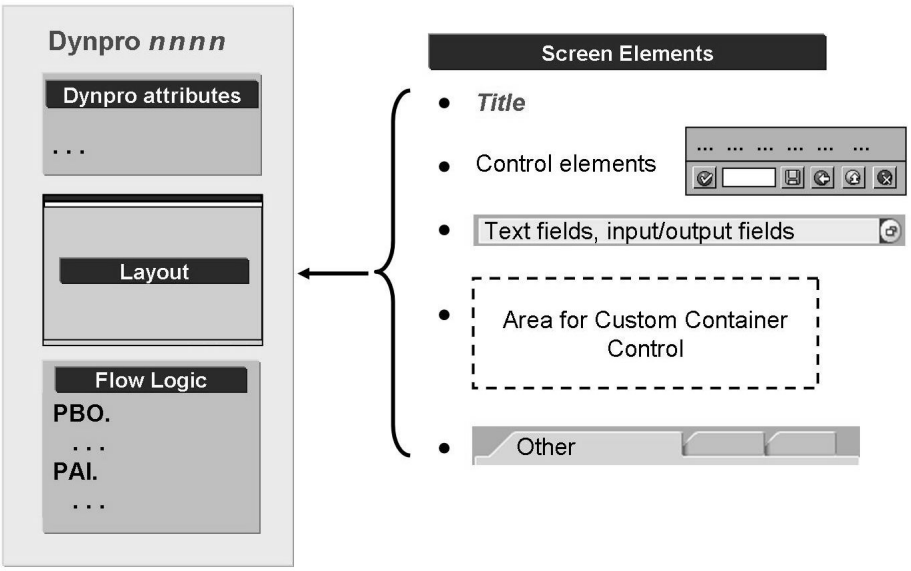


Figure 38: Screen Components

The term "Screen" refers to the combination of a screen display and its flow logic. It consists of the following four parts:

- Screen attributes
- Screen display (layout)
- Element list
- Flow logic. The flow logic contains the so-called flow logic code (not an ABAP statement).

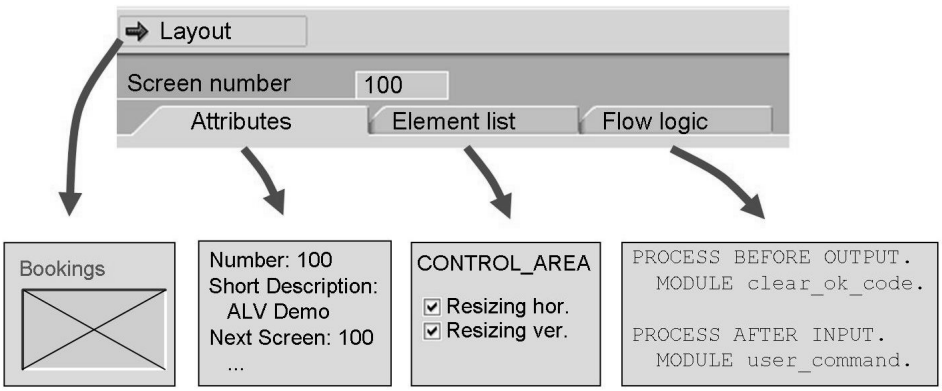


Figure 39: Creating a Screen

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

When you create a screen, you must:

- Determine the basic screen attributes
- Determine the screen layout (in the graphical Screen Painter)
- Set the field properties (element list)
- Program the flow logic

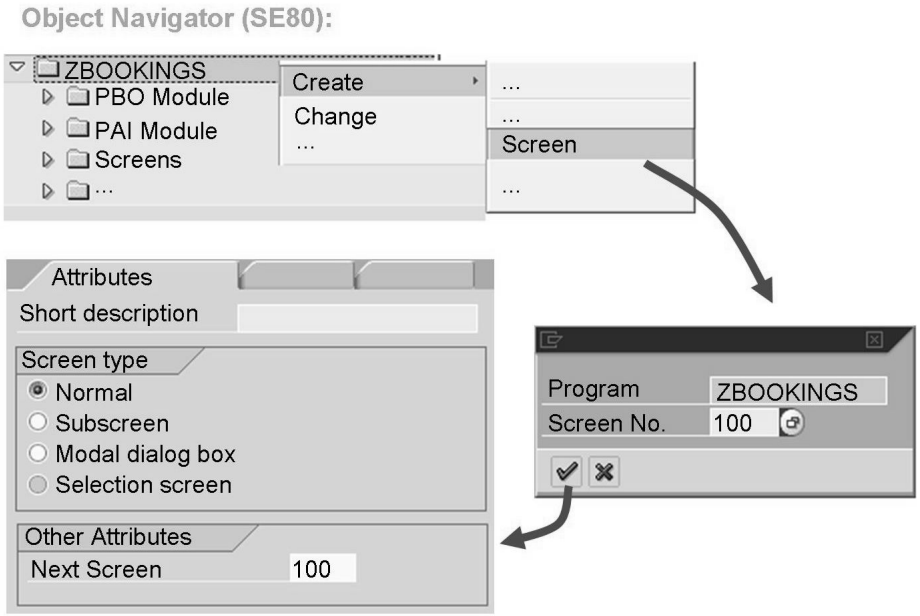


Figure 40: Creating a Screen: Screen Attributes

If you want to create a screen starting from the object list of the Object Navigator (transaction SE80), choose *Create* → *Screen* from the program's context menu in the navigation area.

When you create a screen, you first have to enter its attributes. Enter a screen number, a short text, and a screen type. You will usually use the screen type *Normal*. You can specify the number of the next screen in the *Next screen* field.

If you enter 0 (or no value) for the next screen, the system resumes processing from the point at which the screen was called once it finishes processing the screen itself.

Once you have entered the screen number itself, the screen is displayed again after a user action (unless programmed otherwise). This is the standard value for simple applications with the ALV in a container.

You can also create a screen by double-clicking the screen number <nnnn> in the ABAP command `CALL SCREEN <nnnn>` in the ABAP Editor.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

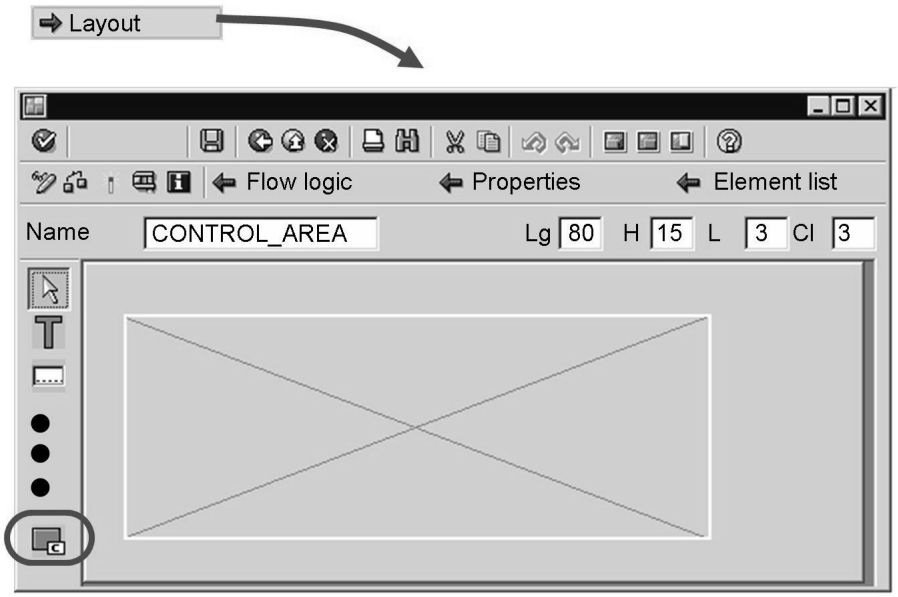


Figure 41: Creating a Screen: Layout

To reserve an area on a screen for a GUI Control, use the graphical *full screen editor* of the *Screen Painter* (*Layout* button).

In the toolbar on the left next to the editing area, there is an icon with the text *Custom Control*. Operation:

- Choose the *Custom Control* button (usually with the left mouse button). Go to the editing area of the screen and define the size and position of the area on the screen. To do this, proceed as follows:

  You determine the upper left corner of the area by pressing the select button of your mouse. Hold down the mouse button and drag the mouse diagonally to the right and down, to the desired lower right corner of the area to be created. As soon as you let go of the mouse button, the lower right corner is fixed.

- You can change the size and position of the area by drag and drop at any time using the drag points that appear at the edge of the area.

Enter a name for the newly created screen element (above: *Control_Area*).

Delete a screen element by first selecting it with the mouse and then pressing the Delete button. You can also use the temporary storage functions (Copy, Cut, Paste).

Move screen elements by holding down the left mouse button and dragging them to a new position.

Select more than one screen element by holding down the left mouse button and using the Control key on your keyboard.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

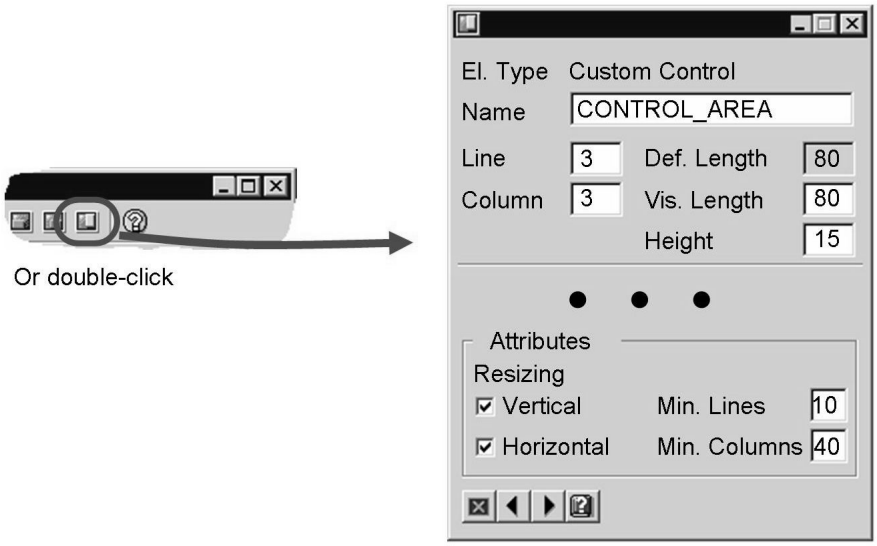


Figure 42: Creating a Screen: Details for one element

To determine details for a screen element, first double-click it. A dialog box will open where you can make the entries you wish.

Use the two attributes *Resizing vertical* and *Resizing horizontal* to define whether the area is also to be rescaled, if necessary, if the size of the screen changes. When you set these attributes, you can also set the minimum values for the area using the attributes *Min. Lines* and *Min. Columns*. The upper limit for the size of the area is defined by the created size.

Maintain details for several screen elements in the element list. You can open this by clicking the button with the green rectangle.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Programming the Screen Flow Logic for Screen Processing

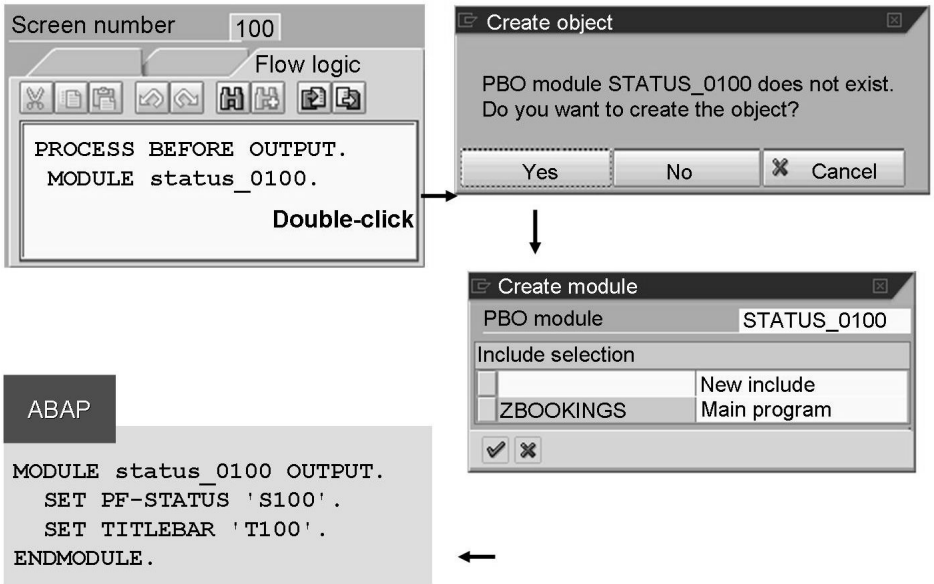


Figure 43: Creating a Screen: Flow Logic

Screens have their own set of commands that you use in the flow logic for the PBO and PAI events. You cannot program ABAP directly in the flow logic. Instead, you encapsulate ABAP commands in modules (screen command `MODULE`).

Please note that a module you want to call for the PBO event must be defined by the `MODULE ... OUTPUT` statement; a module defined by the `MODULE ... INPUT` statement can only be called for a PAI event.

You can create a module by double-clicking the module name in the flow logic editor. You can write all modules into the framework program. However, it is more useful to write all PAI modules and all PBO modules into a separate Include.

To create a module starting from the object list of the Object Navigator, create a *PBO module* or *PAI module* as a new development object in your program.

You can call a module from more than one screen of the same program. You can then keep processing separate by using the screen number (`sy-dynnr`) if necessary.

Creating a GUI status

Unlike a list or a selection screen, a standard screen is not suitable for user interaction. Therefore you must introduce this yourself by creating a GUI status and setting this for the PBO of a screen.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

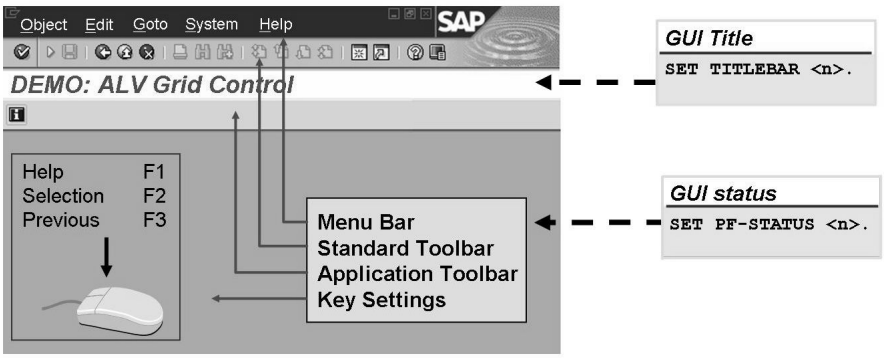


Figure 44: Overview: User Interface

A **GUI status** is made up of a menu bar, a standard toolbar, an application toolbar, and of function key settings.

Menus are located at the top of the screen. Menus can also contain submenus. The *System* and *Help* menus can be found on every screen in the system and always contain the same functions. Neither of these menus can be changed or hidden.

The **standard toolbar** contains icons for frequently used functions. Their arrangement, appearance and assignments to the toolbar's function codes is preset in the SAP system. Inactive functions are grayed out.

The **application toolbar** allows the user to choose frequently-used functions by clicking the corresponding button.

A **function key setting** defines the function keys assigned to specific functions such as the *Details* function.

A screen interface also needs to have a title.

All of a program's GUI titles and statuses taken together make up its user interface. Whenever you add a new status, you must generate the user interface anew.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

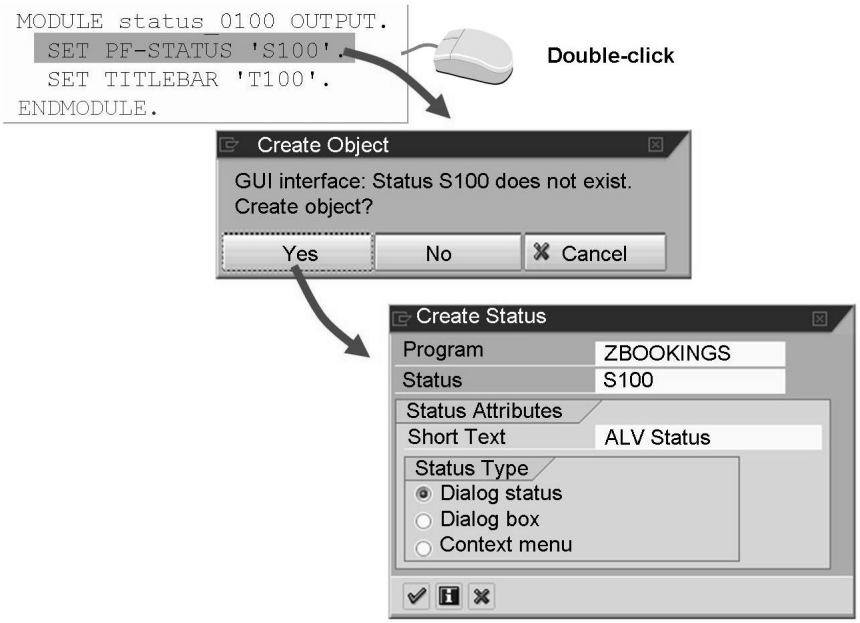


Figure 45: Creating a GUI Status

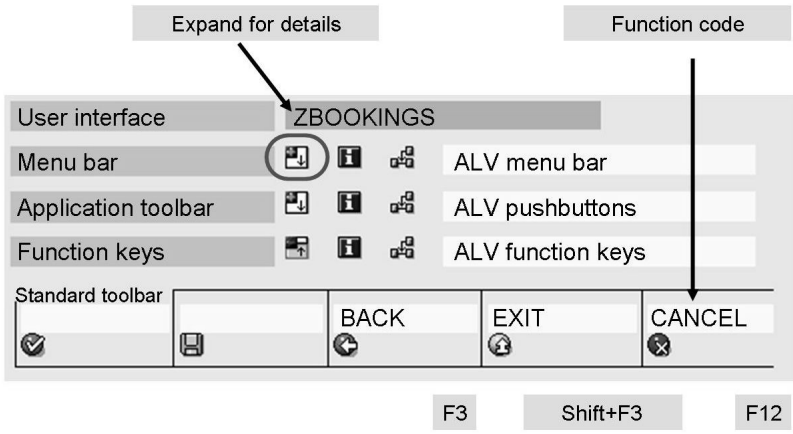
The easiest way to create a status is by forward navigation (that is, by double-clicking the name of a status) from a PBO module.

For the ABAP command `SET PF-STATUS`, the name of a status must be set in single quotation marks.

The status type indicates the technical attributes of the status. You can choose between a dialog status (status for fullscreen), or a dialog box status (for use with modal dialog boxes). Context menus are special collections of functions to be displayed when the user clicks the right-hand mouse button.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.



Do not forget to activate!

Figure 46: Possible Minimal Status for an ALV Application

If you are only using a screen as a basis for an ALV in the container, it is often sufficient to define a minimal status. At least the functions *Back, Cancel*, and *Exit* should be activated. It is good programming style to include these functions in the standard toolbar (so that they are automatically assigned to the appropriate function keys) and to integrate them in the menu.

Every time you change an interface element, you must activate the program interface anew for the changes to take effect.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Processing a User Command From a Screen

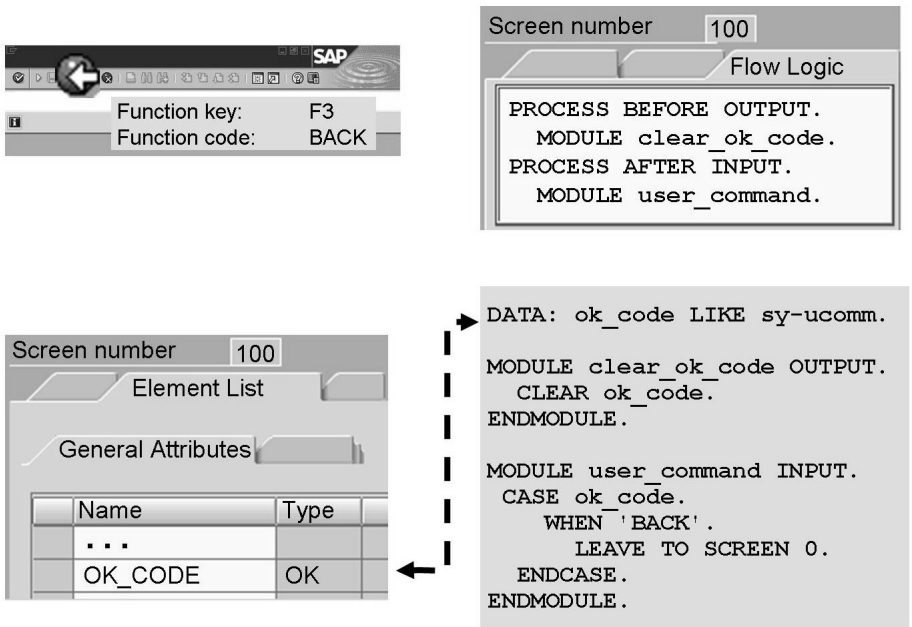


Figure 47: Processing the Function Code

When the user triggers a function of the type “ ” (space) from the interface (using a pushbutton, menu entry, or function key), the system places the relevant function code in the command field of the screen.

To allow you to process this field in PAI, you must assign a name to the field, which you enter in the element list in the Screen Painter. Then create a field of the same name in your ABAP program and assign a type to it with reference to the system variable `sy-ucomm`. The automatic field transport at the beginning of PAI then ensures that the function code is available in this field for processing and can be evaluated by ABAP means within a PAI module.

If you want to return to the point where the screen was called, program the `LEAVE TO SCREEN 0` command into the PAI module. If you want to leave the program, write: `LEAVE PROGRAM`.

To avoid the function code leading to unexpected processing steps on the next screen (ENTER does not usually change the command field), you should initialize the field during every dialog step. One of the possibilities is to reset the field in a PBO module.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 5: ALV Grid Control: Preparing the Container Screen

Exercise Objectives

After completing this exercise, you will be able to:

- Prepare the dynpro for an ALV Grid Control
- Program simple command processing

Business Example

You are a programmer for ABC Travel. The users of your program for flight data reporting want, on the one hand, to have the flight data displayed in an ALV, and, on the other hand, to have input fields available for their own flight bookings. It is now your task first to program a container screen including flow logic for the ALV.

Task:

Perform the following task:

1. Create the program **`ZBC405_##_ALV`** without a TOP include. Type: 1 (Executable program).

   Model Solution: *BC405_CREATE_S1*

2. Read data from the *SFLIGHT* table

   Create an internal table *gt_flights* with reference to the database table *SFLIGHT* and an appropriate work area (*gs_flight*).

   Define a selection screen with two *select-options* for the Airline (*carrid*) and Connection Number (*connid*) fields of the table *SFLIGHT*.

   For `START-OF-SELECTION`, read the appropriate values from the *SFLIGHT* table in your internal table *gt_flights*. Use an array fetch technique to do this.

3. Create a screen 100 to be called after accessing the database. Tip: This is best achieved by forward navigation from your coding. Ensure that the dynpro type *Normal* is selected in the dynpro properties, and that the next screen is set to 100.

4. Create a custom control area in screen 100 using the graphical *Screen Painter*, and assign the following properties to it:

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Attribute	Value
Name	*MY_CONTROL_AREA*
Resizing vertical	On
Resizing horizontal	On
Visible length	100 (default)
Height	25 (default)

5. Using forward navigation in the flow logic of screen 100, create a PBO module with the name *STATUS_0100*.

   Create the module as part of the framework program. From this module, create a status *DYN* of type *Dialog Status*. Provide the functions *Back, Cancel*, and *Exit* with function codes (such as *BACK*, *CANCEL*, and *EXIT*) in the *Menu Painter*.

   To be able to query the OK code, you need to specify the entry provided for it in the element list of the dynpro (usually with *ok_code*). You also need a field with the same name in the ABAP coding, which you need to assign a type with reference to *sy-ucomm*.

   Also create a PAI module *user_command_0100* (either in the framework program or in a new include), and program the command processing there. In the case of all three commands (*BACK*, *CANCEL*, and *EXIT*), the selection screen is to be displayed again.

   Test whether the screen call and return to the editor function correctly.

   Optional: Make it possible for the user to also end the program by making a menu entry.

6. Create and set a title

   Create a title bar called ***T100*** from the PBO module *STATUS_0100* by forward navigation. Use “Flights”, for example, as the title.

Hint: The next exercise will handle data output. When testing your program, an empty screen should appear after the selection screen, which you can leave again using the functions of the standard toolbar.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Solution 5: ALV Grid Control: Preparing the Container Screen

Task:

Perform the following task:

1. Create the program **ZBC405_##_ALV** without a TOP include. Type: 1 (Executable program).

   Model Solution: *BC405_CREATE_S1*

   a) Self-explanatory

2. Read data from the *SFLIGHT* table

   Create an internal table *gt_flights* with reference to the database table *SFLIGHT* and an appropriate work area (*gs_flight*).

   Define a selection screen with two *select-options* for the Airline (*carrid*) and Connection Number (*connid*) fields of the table *SFLIGHT*.

   For `START-OF-SELECTION`, read the appropriate values from the *SFLIGHT* table in your internal table *gt_flights*. Use an array fetch technique to do this.

   a) See result at the end.

3. Create a screen 100 to be called after accessing the database. Tip: This is best achieved by forward navigation from your coding. Ensure that the dynpro type *Normal* is selected in the dynpro properties, and that the next screen is set to 100.

   a) See result at the end.

4. Create a custom control area in screen 100 using the graphical *Screen Painter*, and assign the following properties to it:

Attribute	Value
Name	*MY_CONTROL_AREA*
Resizing vertical	On
Resizing horizontal	On
Visible length	100 (default)
Height	25 (default)

   a) See result at the end.

Continued on next page

 © 2011 SAP AG. All rights reserved.

5. Using forward navigation in the flow logic of screen 100, create a PBO module with the name *STATUS_0100*.

   Create the module as part of the framework program. From this module, create a status *DYN* of type *Dialog Status*. Provide the functions *Back, Cancel*, and *Exit* with function codes (such as *BACK*, *CANCEL*, and *EXIT*) in the *Menu Painter*.

   To be able to query the OK code, you need to specify the entry provided for it in the element list of the dynpro (usually with *ok_code*). You also need a field with the same name in the ABAP coding, which you need to assign a type with reference to *sy-ucomm*.

   Also create a PAI module *user_command_0100* (either in the framework program or in a new include), and program the command processing there. In the case of all three commands (*BACK*, *CANCEL*, and *EXIT*), the selection screen is to be displayed again.

   Test whether the screen call and return to the editor function correctly.

   Optional: Make it possible for the user to also end the program by making a menu entry.

   a) See result at the end.

6. Create and set a title

   Create a title bar called ***T100*** from the PBO module *STATUS_0100* by forward navigation. Use “Flights”, for example, as the title.

   Hint: The next exercise will handle data output. When testing your program, an empty screen should appear after the selection screen, which you can leave again using the functions of the standard toolbar.

   a) See result at the end.

Result

Main program

```
REPORT bc405_create_s1.

* work area and internal table for ALV data
DATA: gt_flights TYPE TABLE OF sflight,
      gs_flight  TYPE          sflight.
DATA  ok_code    LIKE          sy-ucomm.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
SELECT-OPTIONS: so_car FOR gs_flight-carrid,
                so_con FOR gs_flight-connid.


***************************************************
*ABAP events
***************************************************
START-OF-SELECTION.
  SELECT * FROM sflight
    INTO TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

  CALL SCREEN 100.


***************************************************
*PBO Modules
***************************************************
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.               " clear_ok_code  OUTPUT

*---------------------------------------------------
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'DYN'.
  SET TITLEBAR 'T1'.
ENDMODULE.                 " STATUS_0100  OUTPUT

***************************************************
*PAI Modules
***************************************************
*---------------------------------------------------
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK' OR 'CANCEL' OR 'EXIT'.
      SET SCREEN 0.
  ENDCASE.
ENDMODULE.             " USER_COMMAND_0100  INPUT
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Screen 100: flow logic

```
PROCESS BEFORE OUTPUT.
  MODULE clear_ok_code.
  MODULE status_0100.

PROCESS AFTER INPUT.
  MODULE user_command_0100.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Create a screen in which the ALV is to be embedded
- Program the screen flow logic for screen processing
- Create a GUI status with the Menu Painter
- Process a user command in a PAI module

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV Grid Control: Architecture

Lesson Overview

Creating an ALV Grid Control

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the SAP Control Framework for GUI Controls
- Generate and link objects
- Display data in the ALV Grid Control

Business Example

You are working as a programmer for ABC Travel. The manager wants flight data displayed with an ALV.

Generating and Linking Objects

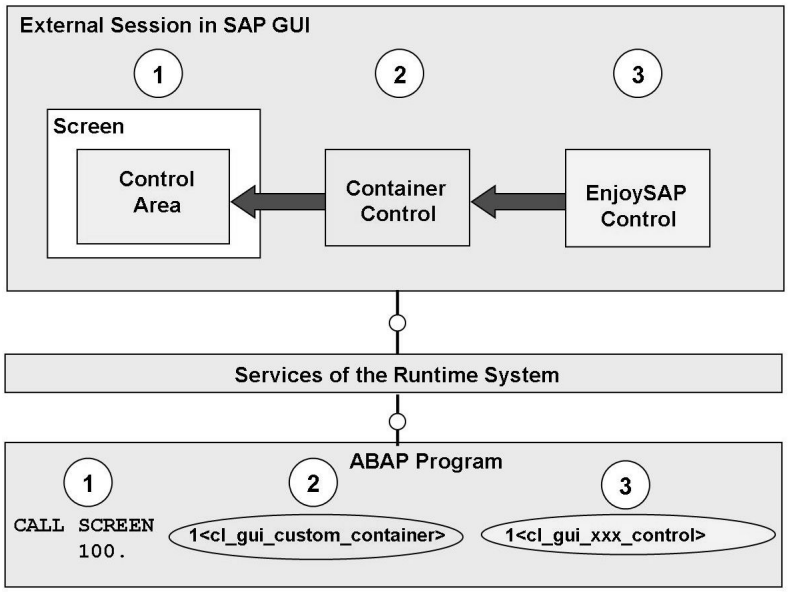


Figure 48: Controls and Dynpros

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

After creating a dynpro including the container control, the GUI Control can now be connected to this container:

- In your program, you assign an instance of the container control class, (here: **cl_gui_custom_container**) to the reserved area on the screen.
- The container control instance is assigned the instance of the actual GUI Control.

The presentation server control is integrated into the dynpro concept with the first two steps (PBO, PAI processing and resize behavior).

Since every presentation server control has a representative instance in the ABAP program, the program communicates with two ABAP Objects instances and at least one dynpro. (In total, four instances are generated, two in the ABAP program and two at the presentation server.)

There are various options for connecting a container instance to a dynpro. For more information, see the appendix.

The following now shows what you need to do to be able to perform the three specified steps in an ABAP program.

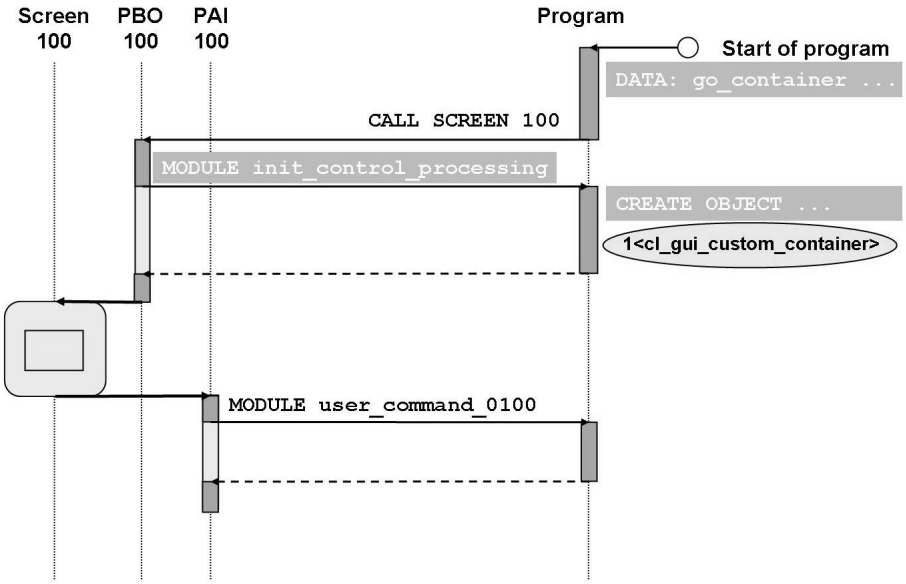


Figure 49: Creating a Container Control Instance: Process Flow

You then need to generate an instance of a container. You should do this in your program before the screen on which the GUI Control is to appear is displayed at the presentation server. The processing time PBO of the dynpro that contains the reserved area is suitable for this.

To make your program easier to follow, place all control programming in a separate module (in this example: init_control_processing.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

To be able to create a container control instance, you also require a reference variable for your instance.

The individual steps are explained in more detail in the following.

```
DATA: <variable_name> "     Name of the reference variable
      TYPE REF TO     "     Specification of the variable as an object reference
      <class_name>.   "     Name of a class
```

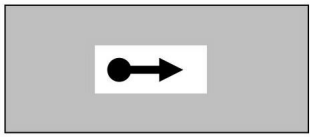


Figure 50: Syntax: Defining a Reference Variable

If your program is to use instances of classes (object-oriented programming), you require pointers to the objects (instances) generated and administered in the main memory of the SAP Web AS system. These pointers are administered in your ABAP program using a reference variable.

You generate the reference variable in your program using the statement `DATA` and assign a name for your reference variable. The field is typed using `TYPE REF TO <class_name>`. The addition `TYPE REF TO` means that the field can include a pointer to an instance. You need to specify the name of a class after `TYPE REF TO`. The reference variable can then contain pointers to instances of this class. For more information about reference variables (static types, dynamic types, and so on), see the keyword documentation for the `DATA` statement.

A reference variable is typed to a class. This class can be locally or globally defined. The classes for the GUI Controls are globally defined, that is, they were created with the *Class Builder* and are therefore globally visible.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
DATA: <object_name> TYPE REF TO <class_name>.
CREATE OBJECT <object_name>
  EXPORTING
    <parameter_name_1> = <program_value_1>
      ...
    <parameter_name_n> = <program_value_n>
  EXCEPTIONS
    <exception_name_1> = <return_code_1>
      ...
    <exception_name_m> = return_code_m>.
```

Interface of the Constructor (Method CONSTRUCTOR)

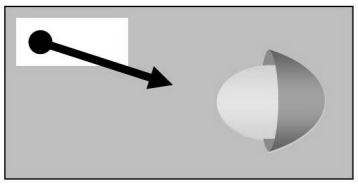


Figure 51: Syntax: Creating Instances

Using the `CREATE OBJECT <object_name>` ABAP statement, create an instance of the class for which the `<object_name>` reference variable has been assigned a type.

If the class for which you want to generate an instance has a method with the name CONSTRUCTOR, this method is automatically executed when the instance is generated. The CONSTRUCTOR method can only have `IMPORTING` parameters and exceptions (`EXCEPTIONS`) as an interface. The interface must also be specified in the `CREATE OBJECT` statement. When you do so, you need to assign values to all of the mandatory `IMPORTING` parameters, which do not have any default values.

The syntax of the interface for the call is the same as for function modules:

- The names of the interface parameters of the method are in the `EXPORTING` part on the left, and the literals or program variables (fields) are on the right of the equal sign.
- You can assign whole numbers as desired to the exceptions listed in the interface These are then in the field `sy-subrc` after the method has been executed.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* data declarations
DATA: go_container
        TYPE REF TO cl_gui_custom_container.
```

my_container

```
* PBO of screen containing CONTROL_AREA
MODULE init_control_processing OUTPUT.
  IF go_container IS INITIAL.
*   create container object and link to screen area
CREATE OBJECT go_container
  EXPORTING
    container_name = 'MY_CONTROL_AREA'
  EXCEPTIONS
    others = 1.
    IF sy-subrc NE 0.
      MESSAGE a...
    ENDIF.
    ...
  ENDIF.
```

my_container

Figure 52: Example: Creating a Container Control Instance

To create a reference variable for your container control instance, use `DATA` to create a variable (above: `go_container`), which is specified using `TYPE REF TO cl_gui_custom_container`.

You generate a container control instance at the presentation server by using the ABAP statement `CREATE OBJECT` to generate an instance of the class *CL_GUI_CUSTOM_CONTAINER* as a representative object.

Using the `IMPORTING` parameter *container_name*, you transfer the name of the area reserved for the control on the screen (above: `MY_CONTROL_AREA`).

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

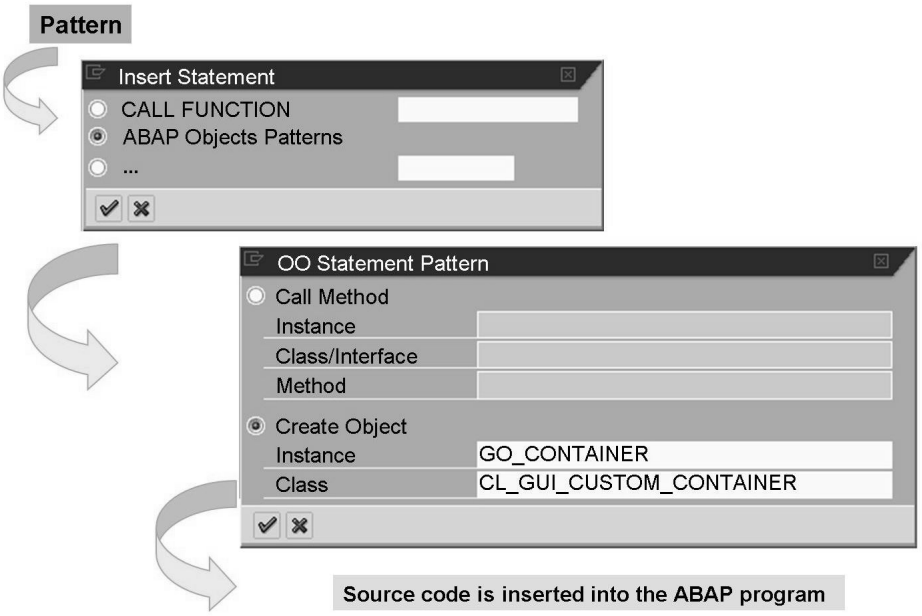


Figure 53: Inserting CREATE OBJECT by Pattern

Use the ABAP Objects pattern to insert the `CREATE OBJECT` statement.

On the *Insert Statement* initial screen, choose the *ABAP Objects Patterns* option.

On the next screen, choose the *Create Object* option. Specify the reference variable for the object to be created under *Instance*, and the class of the object to be created under *Class*.

The statement `CREATE OBJECT`, including all parameters and exceptions of the constructor method, is then inserted in the ABAP source code. Optional parameters and exceptions are inserted as comments.

In this way, you can avoid typing errors and omissions.

You then need to complete the actual parameters and possibly program the exception handling.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

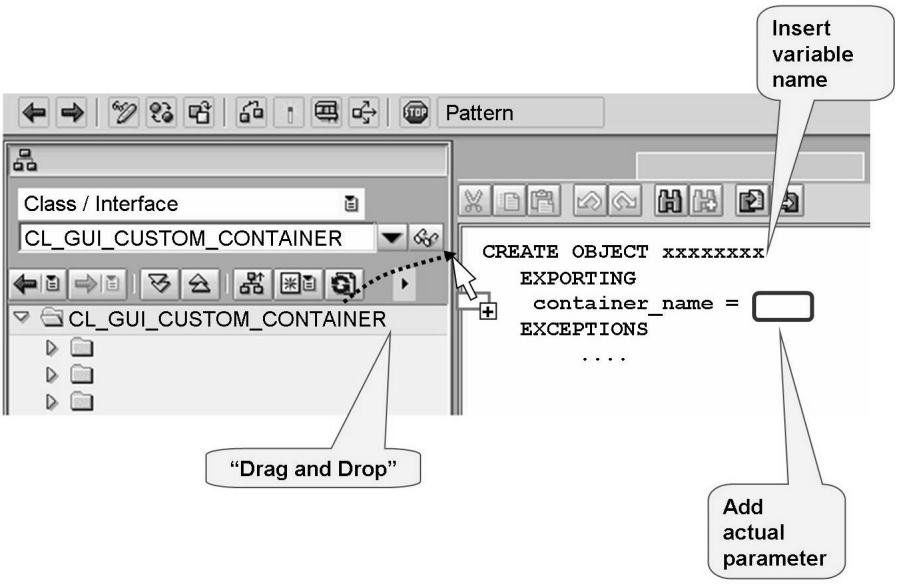


Figure 54: Inserting CREATE OBJECT by Drag and Drop

You can also use the *drag-and-drop* function of the ABAP Workbench to insert the statement `CREATE OBJECT` for a global class.

You can display the global class (in this example: *cl_gui_custom_container*) in the navigation area and transfer it to the editing area using drag-and-drop.

The ABAP statement `CREATE OBJECT`, including the interface parameters and exceptions, is inserted at the current cursor position in the source code of the program. Optional parameters and exceptions are inserted as comments.

You then need to complete the actual parameters and possibly program the exception handling.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

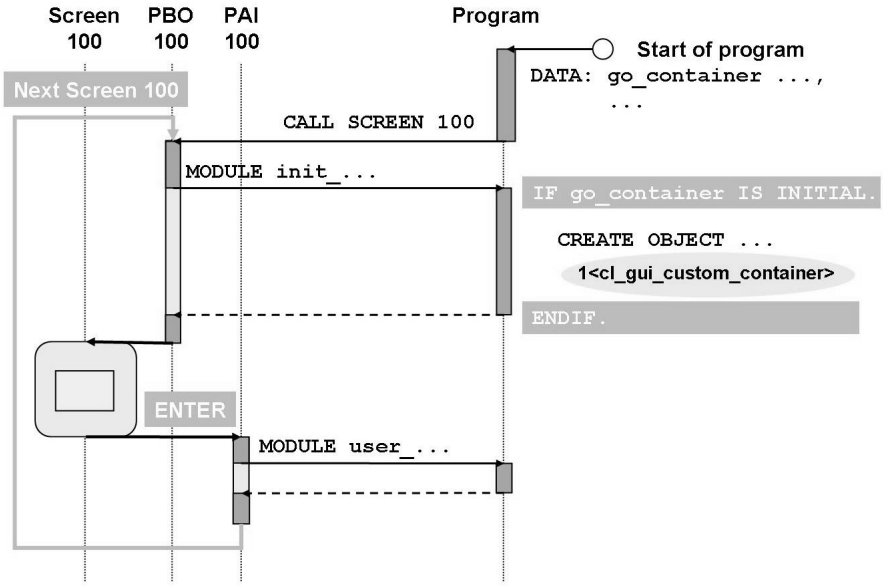


Figure 55: Avoiding Multiple Generation of Controls

If you package the control-specific processing in a PBO module, you need to ensure that an instance is only generated the first time the dynpro is called. Otherwise there is a danger that a new object will be generated every time the dynpro is processed:

- The container screen usually has itself statically as the next screen.
- Each action on the screen that triggers PAI (such as **`Enter`**) means that the dynpro is processed again after PAI has been processed.
- If method calls were performed for PAI operations at the presentation server control, their results would not be visible in the *SAP GUI* window after the dialog step, since a new instance would be linked with the container screen at every PBO.

You can stop unwanted instances from being created by requesting the initial contents for the container reference (above: `my_container`).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

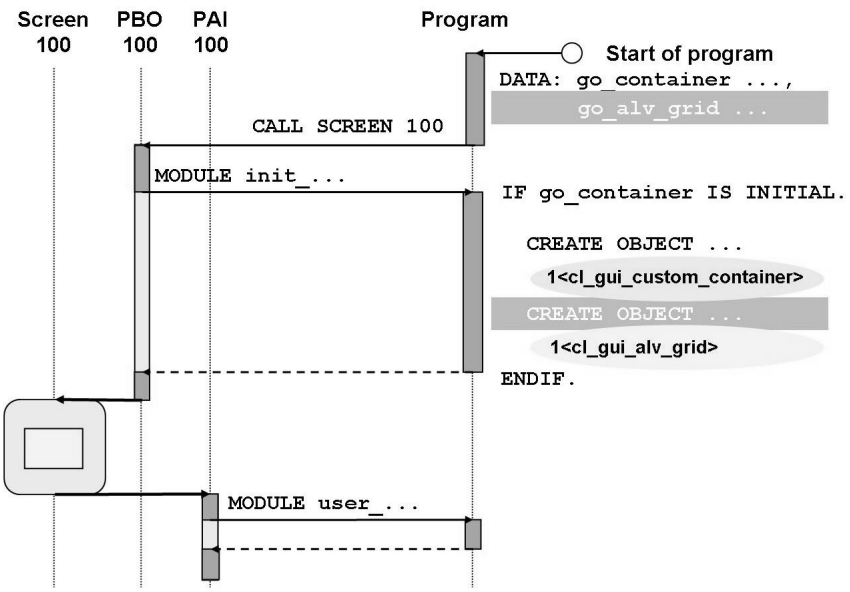


Figure 56: Creating a GUI Control Instance: Process Flow

To generate an instance of the ALV Grid Control, you follow a procedure similar to the one for the container control instance.

The instance of the ALV Grid Control must be created after the container control instance, but before the *SAPGUI* window is sent to the presentation server.

```
* data declarations
DATA: go_container     TYPE REF TO cl_gui_custom_container,
      go_alv_grid      TYPE REF TO cl_gui_alv_grid.
```

```
MODULE init_control_processing OUTPUT.

  IF go_container IS INITIAL.
    ...
*   create ALV Grid Control object and link to container
    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others = 1.

    IF sy-subrc NE 0.
      MESSAGE a...
    ENDIF.
  ENDIF.
```

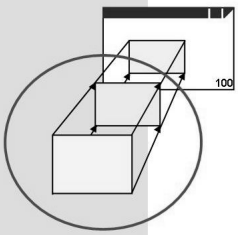


Figure 57: Creating an ALV Grid Instance

To create an ALV Grid Control instance, you need a reference variable declared with reference to class *cl_gui_alv_grid* (above: `my_alv_grid`).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Assign a reference to the previously generated container control instance to the parameter *i_parent* at the constructor.

If an error occurs when creating the instance, you should react with a termination message.

The instance of the ALV Grid Control is connected to the container and therefore to the container screen using the object reference passed in the interface.

To display the ALV Grid Control in full screen mode (that is, the control covers the entire screen), assign a value to the export parameter *i_parent* as follows:

```
i_parent = cl_gui_custom_container=>screen0.
```

In this case, you require neither a control area on the screen, nor a container control.

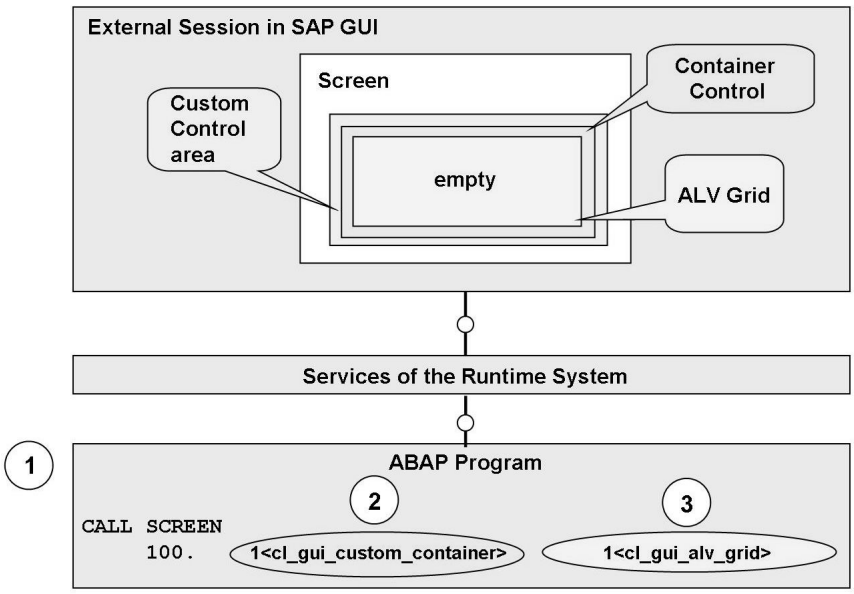


Figure 58: Result: Screen, Container and ALV Grid Control

As a result of the steps you have taken, you have created an ALV Grid Control as a screen element. However, you will only see a frame in the *SAPGUI* window as the controls do not display anything. We still need to supply the presentation server controls with data that they can display.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
MODULE exit_commands_0100 INPUT.
  CASE ok_code.
    WHEN 'EXIT'.                " leave program
      PERFORM free_control_resources.
      LEAVE PROGRAM.
    WHEN 'CANCEL'.              " cancel screen processing
      PERFORM free_control_resources.
      LEAVE TO SCREEN 0.
    ...
ENDMODULE.

FORM free_control_resources.
  CALL METHOD go_alv_grid->free.
  CALL METHOD go_container->free.
  FREE: go_alv_grid, go_container.
ENDFORM.
```

Figure 59: Releasing Control Instances

By default, resources that the control occupies at the presentation server are automatically released at the end of the program.

However, you can also explicitly release these resources by calling the instance method *free*.

Every control wrapper has this method (inherited from CL_GUI_OBJECT).

You should call this method in every case in which the presentation server controls are no longer required in the course of your program.

By releasing the object references (ABAP statements `CLEAR` or `FREE`), you ensure that the resources for the ABAP Objects instances (representative objects) generated in your program are released in the working memory the next time the *Garbage Collector* runs.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Displaying Data with the ALV Grid Control

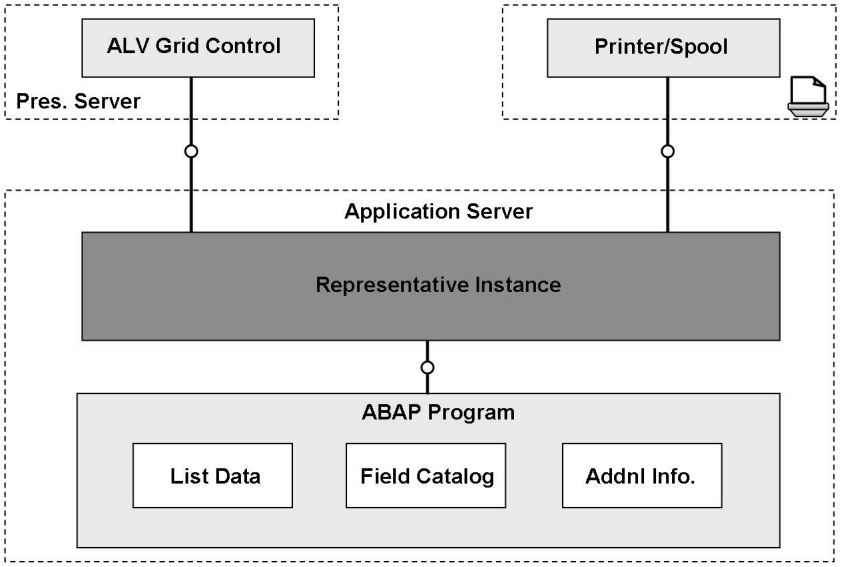


Figure 60: The Proxy Instance of the ALV Grid Control

As with the other GUI Controls, a representative instance of the class *cl_gui_alv_grid* packages the technical details of the communication with the instance of the ALV Grid Control on the presentation server.

The representative instance also communicates with two other partners:

- The ABAP programs as its user
- The spool system or printer to output the data as an ABAP print list

The communication using the chain *ABAP program - representative instance - ALV Grid Control on the presentation server* takes place interactively at runtime.

The (interactive) standard functions that are provided for screen output and the generation of the ABAP print list are therefore also implemented in the class *cl_gui_alv_grid*.

On the other hand, the calling program needs to provide all of the information that it would not make sense to package in the representative instance. This is the data to be displayed itself, the rules that are to define how this data is to be displayed (field catalog) and additional information, if certain user settings are to be predefined by the program. Special data objects need to be held in the program for this.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

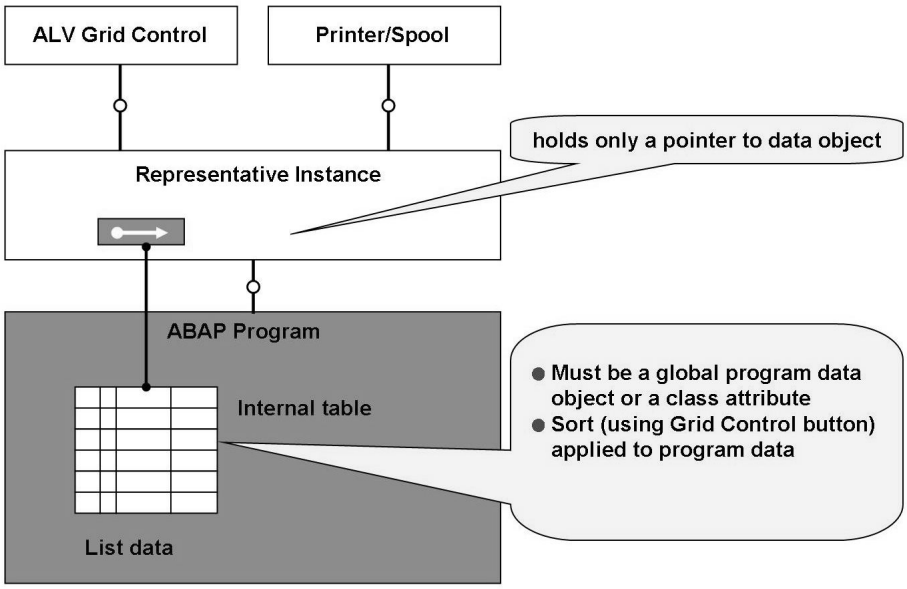


Figure 61: Data Retention for the List

The data to be displayed must be passed to the representative object as an internal table (standard table type).

The representative instance **does not** have a copy of the data, but rather manages only a **reference** to the passed internal table.

All interactive actions of the instance on the presentation server (sorting, filtering, ...) are performed by the representative instance on the internal table in the calling program. This means that the retention period of the internal table that contains the list data must be at least as long as the retention period of the representative instance.

Sort processes in the ALV Grid Control change the status of the internal table in the calling program. All other actions only read the data.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

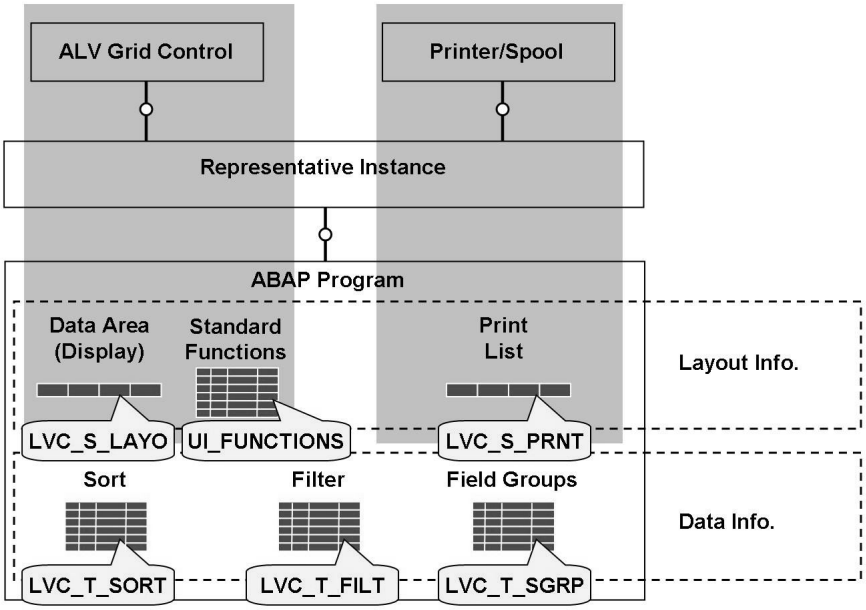


Figure 62: Additional Information in Detail

You can use the specified global data types to define data objects in the calling program and therefore pass the additional information to the representative instance.

- Display settings on the presentation server:
  - You can pass settings for the data area using a structure.
  - You can use an internal table to hide functions of the application toolbar.
- You can control output properties for the print list using a structure.
- You can use internal tables to pass information about sort criteria and filter options.

Usually, you will not manually create the filter and sort criteria, since these can be interactively changed by the user. The representative instance automatically changes the associated internal tables at user interactions. Sort and filter criteria are usually handled as elements of display variants (stored and read).

If your table to be displayed has a large number of fields, the dialog boxes for field selection (column selection) displayed during interactions quickly become unclear. You can therefore use the field catalog to combine fields into logical groups (field groups). You can then use an internal table to pass texts for these field groups to the representative instance.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
go_alv_grid->set_table_for_first_display(
* EXPORTING
*   i_structure_name      =   Global structure type for field catalog
*   is_variant            =
*   i_save                =   Managing display variants
*   i_default             =
*   is_layout             =   Structure for displaying control
*   is_print              =   Structure for displaying print list
*   it_special_groups     =   Internal table with field group texts
*   it_toolbar_excluding  =   Internal table with inactive functions
  CHANGING
    it_outtab             =   Internal table with list data
*   it_fieldcatalog       =   Internal table for field catalog
*   it_sort               =   Internal table with initial sort criteria
*   it_filter             =   Internal table with initial filter criteria
  EXCEPTIONS
    ...).
```

Figure 63: Passing List Data and Additional Information

You can use the method *set_table_for_first_display* to pass the list data, the field catalog, and other additional information to the representative instance.

You pass the list data to be displayed using the parameter *it_outtab*.

If you pass the name of a global structure type to the parameter *i_structure_name*, the representative instance automatically generates the corresponding field catalog. Only the columns with the same names are then displayed from the data table. However, there must be a column with the same name in the data table for every component of the structure.

You set the configuration options of the display variants for the user with the parameters *is_variants* and *i_save*. You can use the *i_default* parameter to set an existing display variant as the initial variant.

You can pass default settings for control and print list output using the parameters *is_layout* and *is_print*.

The names of field groups defined in the field catalog are passed using the parameter *it_special_groups*.

You can use the parameter *it_toolbar_excluding* to pass the names of the standard functions of the application toolbar that are to be hidden.

If you want to change the automatically created field catalog, or create it completely yourself, you need to pass a corresponding internal table to the parameter *it_fieldcatalog*.

You can pass default settings for filtering and sorting the data to be displayed using the parameters *it_filter* and *it_sort*.

The highlighted parameters are considered in more detail below. For more information about the other parameters, see the online documentation.

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
go_alv_grid->set_table_for_first_display(
      EXPORTING
        i_structure_name      = 'SBOOK'
      CHANGING
        i_structure_name      = gt_bookings
      EXCEPTIONS
        it_outtab             = 1 ).
```

At least the display data from the internal table and information about formatting must be passed.

The simplest way to pass the information about formatting is using a link to a Dictionary structure in the parameter `i_structure_name`.

Figure 64: Minimum Provision

The parameter *it_outtab* for the internal table containing the display data is obligatory.

You can pass the information for formatting with various parameters. The simplest variant is to link to a Dictionary structure of the same type with the parameter *i_structure_name*. Other options are discussed in the “Field Catalog” unit.

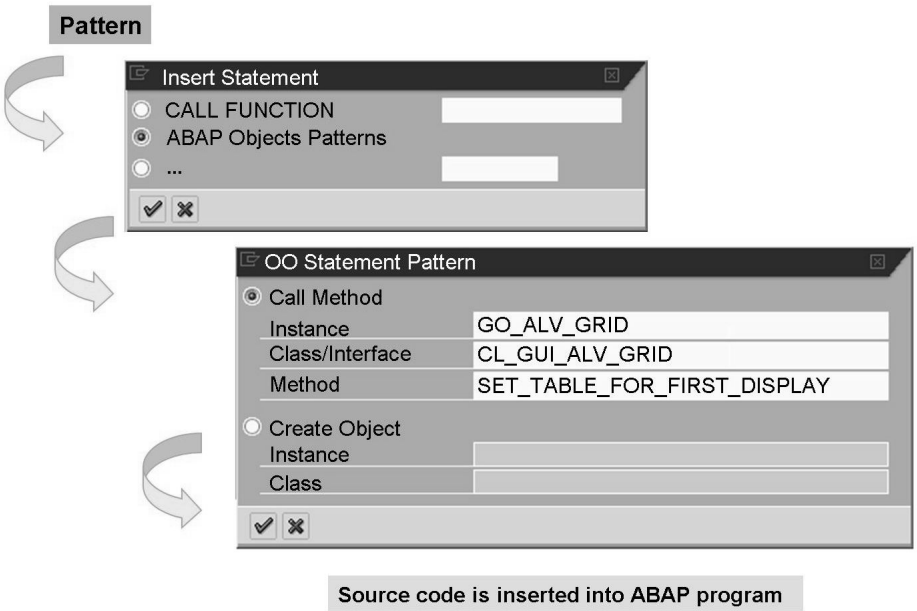


Figure 65: Inserting CALL METHOD by Pattern

To implement a method call of a global class, use either the drag-and-drop function or the ABAP Objects pattern.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The creation of the statement `CALL METHOD` with the ABAP Objects pattern is described in the following:

On the *Insert Statement* initial screen, choose the *ABAP Objects Patterns* option.

On the next screen, choose the *Call Method* option. Specify

- For *Instance*: the reference variable for the object for which the method is to be used
- For *Class/Interface*: the class of the object (in this example, *cl_gui_alv_grid*)
- For *Method*: the method names (in this example, *set_table_for_first_display*)

The statement `CALL METHOD`, including all parameters and exceptions, is then inserted at the cursor position in the ABAP source code. Optional parameters and exceptions are inserted as comments.

In this way, you can avoid typing errors and omissions.

You then need to complete the actual parameters and possibly program the exception handling.

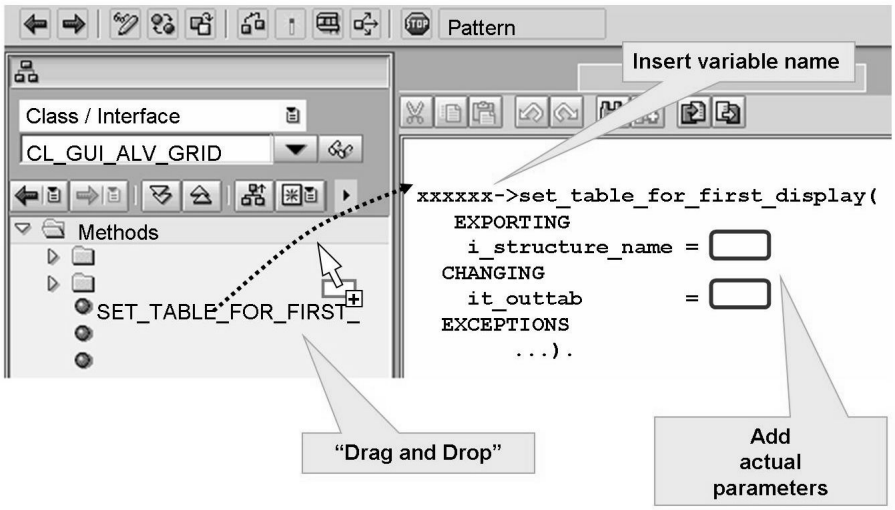


Figure 66: Inserting CALL METHOD by Drag and Drop

Using drag-and-drop, you can display the global class (in this example, *cl_gui_alv_grid*) in the navigation area of the *ABAP Workbench* and drag the method into the editing area by holding down the left mouse button.

The ABAP statement `CALL METHOD`, including the interface parameters and exceptions, is inserted at the current cursor position in the source code of the program. Optional parameters and exceptions are inserted as comments.

You then need to add the names of the reference variables instead of "xxxxxxx", and to complete the actual parameters and possibly program the exception handling.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
go_alv_grid->refresh_table_display(

*   is_stable      =   row/column position remains stable during refresh
*   i_soft_refresh =   only refresh data, not sort criteria, and so on
  ).
```

```
DATA <st_name> TYPE lvc_s_stbl.
...
<st_name>-row =  row-related yes/no
<st_name>-col =  column-related yes/no
```

Prompts the representative instance to send

- The contents of the data table
- If necessary, the additional information again

→ **Display on the presentation server is refreshed**

Figure 67: Refreshing the Display

You can use the method *refresh_table_display* to prompt the representative instance to send the list data and, if necessary, also the additional information to the instance on the presentation server again.

You can use the parameter *i_soft_refresh* to specify if the current filter and sort criteria are kept and applied (assign the value " ") or if the current filter and sort criteria are kept but not applied (assign the value "X") .

You can use the fields *row* and *col* of a structure of the global data type *lvc_s_stbl* to specify that possible scroll positions with regard to the rows or columns are retained when refreshing (assign the value "X" to each field). Pass the corresponding filled structure to the parameter *is_stable*.

Hint: If you change the row structure of the data table, you need to call the method *set_table_for_first_display* again, since the field catalog must be created again.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 6: ALV Grid Control: Architecture

Exercise Objectives

After completing this exercise, you will be able to:

- Include an ALV Grid in a screen
- Display data in an ALV Grid Control

Business Example

Task:

Finish your program for displaying the flights using the ALV Grid Control.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_CREATE_S2*

1. Expand your own program from the last exercise (*ZBC405_##_ALV*).

   (If you are not completely finished with the last task or if you would like to have a "fresh" basis on which to begin, you can copy the model solution of the last task (*BC405_CREATE_S1*) under a new name *ZBC405_##_ALV_2*. In this case, save the program in the *ZBC405_##* package in the request given by your trainer.)

2. In the previous exercise, you created a custom control area. Now create a custom control container and an ALV Grid for it.

   Define a reference variable based on an object of the *CL_GUI_CUS-TOM_CONTAINER* class and an object of the *CL_GUI_ALV_GRID* class.

   Suggested name: *go_container* and *go_alv_grid*.

3. Implement a module in the PBO event block of screen 100 in which the objects are to be created (suggested name: *create_and_transfer*). For coding the command, use the *CREATE OBJECT* model key. Valuate the obligatory parameters: *container_name* for the Custom Container Control (enter the name of the Custom Control Area), *parent* for the ALV Grid Control.

   Evaluate each of the exceptions for the calls and compile these using *OTHERS*. If an error occurs, call message 010 from the message class BC405_408 as a termination message. Only create the objects when the reference variable *go_container* is in the initial state for the container.

4. Transfer all of the data from the internal table to be displayed to the ALV Grid Control using the *set_table_for_first_display.* method. Supply these methods with actual display data and with a description of the structure of the internal table (*i_structure_name* parameter). Output in case of an error

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

or a termination message (number 012 of the *BC405_408* message class). Implement this data transfer in the PBO module from the previous part of the exercise.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Solution 6: ALV Grid Control: Architecture

Task:

Finish your program for displaying the flights using the ALV Grid Control.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_CREATE_S2*

1. Expand your own program from the last exercise (*ZBC405_##_ALV*).

   (If you are not completely finished with the last task or if you would like to have a "fresh" basis on which to begin, you can copy the model solution of the last task (*BC405_CREATE_S1*) under a new name *ZBC405_##_ALV_2*. In this case, save the program in the *ZBC405_##* package in the request given by your trainer.)

   a) See result at the end.

2. In the previous exercise, you created a custom control area. Now create a custom control container and an ALV Grid for it.

   Define a reference variable based on an object of the *CL_GUI_CUSTOM_CONTAINER* class and an object of the *CL_GUI_ALV_GRID* class.

   Suggested name: *go_container* and *go_alv_grid*.

   a) See result at the end.

3. Implement a module in the PBO event block of screen 100 in which the objects are to be created (suggested name: *create_and_transfer*). For coding the command, use the *CREATE OBJECT* model key. Valuate the obligatory parameters: *container_name* for the Custom Container Control (enter the name of the Custom Control Area), *parent* for the ALV Grid Control.

   Evaluate each of the exceptions for the calls and compile these using *OTHERS*. If an error occurs, call message 010 from the message class BC405_408 as a termination message. Only create the objects when the reference variable *go_container* is in the initial state for the container.

   a) See result at the end.

4. Transfer all of the data from the internal table to be displayed to the ALV Grid Control using the *set_table_for_first_display.* method. Supply these methods with actual display data and with a description of the structure of the internal table (*i_structure_name* parameter). Output in case of an error or a termination message (number 012 of the *BC405_408* message class). Implement this data transfer in the PBO module from the previous part of the exercise.

   a) See result at the end.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Result

Main program

```
*&---------------------------------------------------------------------*
*& Report  BC405_CREATE_S2                                            *
*&---------------------------------------------------------------------*
REPORT  bc405_create_s2.

* work area and internal table for ALV data
DATA: gt_flights TYPE TABLE OF sflight,
      gs_flight  TYPE          sflight.
DATA  ok_code    LIKE sy-ucomm.

DATA: go_alv_grid  TYPE REF TO cl_gui_alv_grid,
      go_container TYPE REF TO cl_gui_custom_container.

SELECT-OPTIONS: so_car FOR gs_flight-carrid,
                so_con FOR gs_flight-connid.

************************************************************************
*ABAP events
************************************************************************
START-OF-SELECTION.
  SELECT * FROM sflight
    INTO TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

  CALL SCREEN 100.

************************************************************************
*PBO modules
************************************************************************
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.                 " clear_ok_code  OUTPUT

*----------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'DYN'.
  SET TITLEBAR 'T1'.
ENDMODULE.                 " STATUS_0100  OUTPUT

*----------------------------------------------------------------------*
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    go_alv_grid->set_table_for_first_display(
      EXPORTING
        i_structure_name = 'SFLIGHT'
      CHANGING
        it_outtab        = gt_flights
      EXCEPTIONS
        OTHERS           = 1 ).
    IF sy-subrc <> 0.
      MESSAGE a012(bc405_408).
    ENDIF.

  ENDIF.
ENDMODULE.                 " create_and_transfer  OUTPUT

*********************************************************************
*PAI modules
*********************************************************************
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK' OR 'EXIT' OR 'CANCEL'.
      SET SCREEN 0.
  ENDCASE.
ENDMODULE.                 " USER_COMMAND_0100  INPUT
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Screen 100: flow logic

```
PROCESS BEFORE OUTPUT.

  MODULE clear_ok_code.
  MODULE status_0100.
  MODULE create_and_transfer.

PROCESS AFTER INPUT.

  MODULE user_command_0100.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Explain the SAP Control Framework for GUI Controls
- Generate and link objects
- Display data in the ALV Grid Control

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Create a screen in which the ALV is to be embedded
- Program the screen flow logic for screen processing
- Create a GUI status with the Menu Painter
- Process a user command in a PAI module
- Explain the SAP Control Framework for GUI Controls
- Generate and link objects
- Display data in the ALV Grid Control

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Unit 5

ALV: Design

Internal Use SAP Partner Only

Unit Overview

Just showing the standard ALV to the user is often not enough. There are plenty of possibilities to change the design of an ALV.

Unit Objectives

After completing this unit, you will be able to:

- Work with layout variants in an ALV Grid Control
- Create the layout structure of the ALV Grid Control
- Set colors of lines and cells in the ALV Grid Control
- Hide standard toolbar functions
- Explain the purpose of the field catalog
- Add columns to the ALV Grid Control or change columns of the ALV

Unit Contents

Lesson: ALV Grid Control: Layout Variants 132
Exercise 7: ALV Grid Control: Layout Variants 135
Lesson: ALV Grid Control: Layout 140
Exercise 8: ALV Grid Control: Layout 151
Lesson: ALV Grid Control: Field Catalog 161
Exercise 9: ALV Grid Control: Field Catalog 175

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson: ALV Grid Control: Layout Variants

Lesson Overview

Layout Variants (SAP Web AS 6.20)

Lesson Objectives

After completing this lesson, you will be able to:

- Work with layout variants in an ALV Grid Control

Business Example

You are working as a programmer for ABC Travel. Every user is to be able to create and save his or her own layout variants. You are to implement this.

Working with Layout Variants

Management Mode	Assign values to parameters in the method SET_TABLE_FOR_FIRST_DISPLAY	
	IS_VARIANT	I_SAVE
Change (Default)	space	space
Change and select existing	<st_name>	space
Change, save, and select existing: • Only user-related • Only cross-user • Both user-related and cross-user	<st_name>	 'U' 'X' 'A'

```
DATA <st_name> TYPE disvariant,
...
<st_name>-report =
<st_name>-handle =
```

Program name (usually sy-cprog)
Logical name (only if there are multiple instances per program)

Figure 68: Defining the Management Mode of the Layout Variants

You can use the parameter *is_variant* and *i_save* to determine what options for variant management are offered to the user.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

There are three modes that you can define using the two parameters:

- *The user can change the current layout variant*:

  To set this mode, assign *space* to both parameters (default setting).

- *The user can change the current layout variant and select existing variants*:

  To set this mode, pass a structure of the global type *disvariant* to the parameter *is_variant*, in which the field *report* has been assigned the program name (*sy-cprog*), and assign the value *space* to the parameter *i_save*.

  If there are multiple instances in your calling program, you also need to control the variant modes separately using freely-definable names, which you can assign to them using the field *handle*.

- *The user can change the current layout variant and save and manage existing variants*:

  To do this, assign the parameter *is_variant* as for the previous mode. You can use the value you assign to the parameter *i_save* to determine the way in which the user can manage the layout variants:

  "U" The user can only save user-related layout variants.

  "X" The user can save cross-user layout variants.

  "A" The user can save both user-related and cross-user layout variants.

```
* Save variants
DATA: gs_variant TYPE disvariant,
      gv_save    TYPE c LENGTH 1.

* Fill variant structure
* At least the report name has to be filled
  gv_variant-report = sy-cprog.

* Determine saving options ('A' = all types)
  gv_save = 'A'.

go_alv_grid->set_table_for_first_display(
   EXPORTING
     i_structure_name = 'SBOOK'
     is_variant       =  gs_variant
     i_save           =  gv_save
     i_default        = 'X'

   CHANGING
     it_outtab        =  gt_bookings ).
```

Figure 69: Saving Layout Variants

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

About the namespace of the variants:

- Customers' cross-user variants (known as standard variants) must start with /
- SAP's cross-user variants start with a digit
- User-dependent variants must start with a letter

Variants are client-specific.

Please note: Only standard layout variants can be transported. To do this, call the layout management, select the desired variants, and choose *Layout → Transport.*

When saving a layout variant, you can designate it as the default variant if the parameter *i_default* of the method *set_table_for_first_display* has the value “X”. (This value is the default value.)

For a user to be permitted to create or change standard layouts, or to set them as the default setting, he or she requires the entry **23** in the *ACTVT* field of the *S_ALV_LAYO* authorization object.

Procedure:

```
DATA: gv_variant   TYPE disvariant.
PARAMETERS: pa_lv  TYPE disvariant-variant.
...
gv_variant-report  = sy-cprog.
gv_variant-variant = pa_lv.
...
go_alv_grid->set_table_for_first_display(
    EXPORTING
      ...
      is_variant       = gv_variant
      ... ).
```

Figure 70: Passing a Layout Variant to the Selection Screen

It may be necessary to pass the layout variant to the selection screen of the report (for example, in the case of execution in the background).

To do this, a parameter is created on the selection screen that is assigned a type with *disvariant-variant*. The name of the layout variant is passed with this parameter.

The content of the parameter is assigned to the field *variant* of the *is_variant* parameter before data is displayed in the ALV.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Exercise 7: ALV Grid Control: Layout Variants

Exercise Objectives

After completing this exercise, you will be able to:

- Allow the user to save layout variants
- Allow the user to select a layout variant on the selection screen

Business Example

You are working as a programmer for ABC Travel. The users would like to create layout variants.

Task:

Allow users to create their own layouts (layout variants).

Program: *ZBC405_##_ALV*

Model Solution: *BC405_DESIGN_S1*

1. Expand your own program from the last exercise.

   (If you are not completely finished with the last task or if you would like to have a "fresh" basis on which to begin, you can copy the model solution of the last task (*BC405_CREATE_S2*) under a new name *ZBC405_##_ALV_3*. In this case, save the program in the *ZBC405_##* package in the request given by your trainer.)

2. Allow the end users to create and save layout variants for the ALV Grid, for both user-related and cross-user variants.

3. Create a layout variant (suggested name: LV1) in which at least one field is summed (for example, using *Occupied seats economy class)* and in which there are subtotals for at least one column.

4. Create an input field on the selection screen with which the end user can determine the layout variant before calling the ALV Grid. (This is primarily useful for background processing.)

   Assign the type *disvariant-variant* to the input field.

   Ensure that the selected variant is passed to the ALV Grid.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 7: ALV Grid Control: Layout Variants

Task:

Allow users to create their own layouts (layout variants).

Program: *ZBC405_##_ALV*

Model Solution: *BC405_DESIGN_S1*

1. Expand your own program from the last exercise.

   (If you are not completely finished with the last task or if you would like to have a "fresh" basis on which to begin, you can copy the model solution of the last task (*BC405_CREATE_S2*) under a new name *ZBC405_##_ALV_3*. In this case, save the program in the *ZBC405_##* package in the request given by your trainer.)

   a) See result at the end.

2. Allow the end users to create and save layout variants for the ALV Grid, for both user-related and cross-user variants.

   a) See result at the end.

3. Create a layout variant (suggested name: LV1) in which at least one field is summed (for example, using *Occupied seats economy class)* and in which there are subtotals for at least one column.

   a) See result at the end.

4. Create an input field on the selection screen with which the end user can determine the layout variant before calling the ALV Grid. (This is primarily useful for background processing.)

   Assign the type *disvariant-variant* to the input field.

   Ensure that the selected variant is passed to the ALV Grid.

   a) See result at the end.

Result

Main program

```
REPORT  bc405_design_s1.

* work area and internal table for ALV data
DATA: gt_flights TYPE TABLE OF sflight,
      gs_flight  TYPE          sflight.
```

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
DATA  ok_code LIKE sy-ucomm.

DATA: go_container TYPE REF TO cl_gui_custom_container,
      go_alv_grid  TYPE REF TO cl_gui_alv_grid.
DATA: gv_variant   TYPE        disvariant.

SELECT-OPTIONS: so_car FOR gs_flight-carrid,
                so_con FOR gs_flight-connid.
SELECTION-SCREEN SKIP.
PARAMETERS: pa_lv TYPE disvariant-variant.


***********************************************************************
*ABAP events
***********************************************************************
START-OF-SELECTION.
  SELECT * FROM sflight
    INTO TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

  CALL SCREEN 100.


***********************************************************************
*PBO Modules
***********************************************************************
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.                 " clear_ok_code  OUTPUT

*----------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'DYN'.
  SET TITLEBAR 'T1'.
ENDMODULE.                 " STATUS_0100  OUTPUT

*----------------------------------------------------------------------*
MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
```

Continued on next page

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    gv_variant-report  = sy-cprog.
    gv_variant-variant = pa_lv.

    go_alv_grid->set_table_for_first_display(
      EXPORTING
        i_structure_name = 'SFLIGHT'
        is_variant       = gv_variant
        i_save           = 'A'
      CHANGING
        it_outtab        = gt_flights
      EXCEPTIONS
        OTHERS           = 1 ).
    IF sy-subrc <> 0.
      MESSAGE a012(bc405_408).
    ENDIF.

  ENDIF.
ENDMODULE.                 " create_and_transfer  OUTPUT


***********************************************************************
*PAI Modules
***********************************************************************
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK' OR 'EXIT' OR 'CANCEL'.
      SET SCREEN 0.
  ENDCASE.
ENDMODULE.                 " USER_COMMAND_0100  INPUT
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Work with layout variants in an ALV Grid Control

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV Grid Control: Layout

Lesson Overview

The layout of the ALV grid can be influenced by the programmer in many different ways. Among other things, you will learn different ways in which to manipulate the layout of the entire ALV (for example selecting), to add special columns or change cell colors.

Lesson Objectives

After completing this lesson, you will be able to:

- Create the layout structure of the ALV Grid Control
- Set colors of lines and cells in the ALV Grid Control
- Hide standard toolbar functions

Business Example

You are a programmer at ABC Travel. You have been requested to display some lines in the ALV output in red, depending on the content. Other lines are to be displayed in orange, again depending on the content. As usual, it is up to you to complete this difficult task.

Filling and Passing-On the Layout Structure

```
DATA: gs_layout TYPE lvc_s_layo.

* Fill layout structure
  gs_layout-grid_title = 'ALV Grid Title'(002).
  gs_layout-zebra      = 'X'.
  gs_layout-cwidth_opt = 'X'.

go_alv_grid->set_table_for_first_display(
   EXPORTING
     i_structure_name = 'SBOOK'
     is_layout        = gs_layout
   CHANGING
     it_outtab        = gt_bookings ).
```

Figure 71: Filling and Passing-On the Layout Structure

You can change the layout of your ALV Grid Control, for example, select the title, create a striped (zebra) pattern for the lines, or optimize the column width.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

To do this, create the <*gs_layout*> work area in the program, with reference to the Dictionary structure *lvc_s_layo* and fill the relevant fields, such as <*gs_layout*>*-grid_title*, <*gs_layout*>*-cwidth_opt* or <*gs_layout*>*-zebra*.

Then, pass on the work area to the *is_layout* parameter of the *set_table_for_first_display* method.

Structure Type `lvc_s_layo` is globally defined in DIC.

• grid_title	Text of the title row
• zebra	Alternating row colors
• no_headers	Hide the column headers
• sel_mode	Selection modes
• cwidth_opt	Optimized column widths
• no_toolbar	Hide the toolbar
• totals_bef	Display totals before the first row
• info_fname	Field name of the output field for row color
• ctab_fname	Field name of the output table for cell colors

Figure 72: Key Fields of the Layout Structure

lvc_s_layo is a global structure type. The following fields are available, among others:

- *grid_title*: a title with maximum 70 characters.
- *zebra:* if "X", the list is displayed with a striped pattern.
- *no_headers*: If "X", no column headers are displayed.
- *sel_mode*: Setting that determines how many cells, lines, and columns can be selected.
- *cwidth_opt*: If "X", a column widths are optimized. The column heading and the cell contents are then displayed.
- *info_fname*: Additional data table field that contains information on the required line colors.
- *ctab_fname*: Additional data table field that contains information on the required cell colors.
- For more fields and details, refer to the following sections or to the online documentation.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
DATA: gs_layout TYPE lvc_s_layo.
gs_layout-sel_mode = 'A'.
```

Value	Possible Selections	Selection Column
space	See 'B'	No
'A'	• Multiple columns • Multiple rows	Yes
'B'	• Multiple columns • Single rows	No
'C'	• Multiple columns • Multiple rows	No
'D'	• Multiple columns • Multiple rows • Any cell selection	Yes

Figure 73: Selection of Cells, Columns, and Lines

Depending on the value you enter in the *sel_mode* field in the layout structure, users can select individual or several lines or cells.

To select several entries, select the **`Ctrl`** key and keep the mouse button pressed.

To make a selection from one entry to another, select the **`Shift`** key and keep the mouse button pressed.

```
* add column to data table
TYPES: BEGIN OF gty_flight.
         INCLUDE TYPE sflight.
TYPES:   light   TYPE c LENGTH 1,
       END OF gty_flight.
DATA: gt_flights TYPE TABLE OF gty_flight,
      gs_flight  TYPE          gty_flight.

* set value for icon
LOOP AT gt_flights INTO gs_flight.
  IF gs_flight-seatsocc < 5.
    gs_flight-light = '1'.
  ELSEIF gs_flight-seatsocc < 100.
    gs_flight-light = '2'.
  ELSE.
    gs_flight-light = '3'.
  ENDIF.
  MODIFY gt_flights FROM gs_flight.
ENDLOOP.

* pass the name of the exception column to the layout structure
gs_layout-excp_fname = 'LIGHT'.

go_alv_grid->set_table_for_first_display(
    EXPORTING i_structure_name = 'SFLIGHT'
              is_layout        = gs_layout... ).
```

```
* alternative display:
gs_layout-excp_led = 'X'.
```

	AA	17	0
	AA	64	5
	LH	400	124

Figure 74: Threshold Value Display: Exceptions

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

You can use exceptions to display whether threshold values have been achieved, exceeded, or not achieved, on a line-by-line basis. Follow the instructions below:

Add a one-digit character field to your data table structure.

Fill this field for each row of your database table depending on the desired threshold value: 0 = gray, 1 = red, 2 = yellow, 3 = green, space = nothing.

Enter the name of the exception column in the *excp_fname* field of the layout structure, and pass the layout structure to the is_layout *parameter of the* set_table_for_first_display *method.*

If you enter “X” in the *excp_led* field of the layout structure, colored LEDs are displayed instead of the normal traffic light icons.

```
* data needed to set sorting order
DATA: gt_sorting_criteria  TYPE lvc_t_sort,
      gs_sorting_criterion TYPE lvc_s_sort.

gs_sorting_criterion-fieldname = 'ORDER_DATE'.
*  gs_sorting_criterion-spos   = 1.
APPEND gs_sorting_criterion TO gt_sorting_criteria.

clear gs_sorting_criterion.
gs_sorting_criterion-fieldname = 'CUSTOMID'.
gs_sorting_criterion-up        = 'X'.
*  gs_sorting_criterion-spos   = 2.
APPEND gs_sorting_criterion TO gt_sorting_criteria.

go_alv_grid->set_table_for_first_display(
    EXPORTING  i_structure_name = 'SBOOK'
    CHANGING   it_outtab        = gt_bookings
               it_sort          = gt_sorting_criteria
    EXCEPTIONS OTHERS           = 1 ).
```

Figure 75: Sort Criteria for First List Output

Data in the ALV Grid Control can be dynamically sorted by users.

To sort data in a certain way at the very beginning, you must enter an internal table for the *it_sort* parameter of the *set_table_for_first_display* method. Use the *lvc_t_sort* type for this table.

In this internal table, you must insert a line for each field that will be used as a sort criterion.

- Enter the column name in *fieldname*.
- If you use more than one sort criterion, in each case, you must enter the sequence in which the sort criterion will be evaluated in the *spos* field. Alternatively, you can also insert the fields in the required sequence in the sort criteria's internal table.
- Enter “X” in the *up* field if you wish to sort the entries in ascending order.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Colors of Lines and Cells

```
TYPE-POOLS col. " allow symbolic names for colors
DATA gs_layout TYPE lvc_s_layo.

TYPES: BEGIN OF gty_booking.
         INCLUDE TYPE sbook.
TYPES:   color TYPE c LENGTH 4,
       END OF gty_booking.
DATA: gt_bookings TYPE TABLE OF gty_booking,
      gs_booking  TYPE          gty_booking.

gs_booking-color = 'C' && col_negative && '10'.
MODIFY gt_bookings FROM gs_booking
  TRANSPORTING color WHERE ... .

* pass the name of the color field to the layout structure
gs_layout-info_fname = 'COLOR'.

go_alv_grid->set_table_for_first_display(
  EXPORTING
    i_structure_name = 'SBOOK'
    is_layout        = gs_layout
  CHANGING
    it_outtab        = gt_bookings ).
```

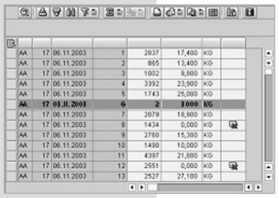


Figure 76: Changing the Line Color

If the field catalog is to be set up automatically using a global structure type, it is enough to maintain another field for the color values in the data table. This field should be the last component of the data table's line type. If the original global structure type is used as the line type for the ALV Grid Control, the color values are not displayed as a column on the screen. Instead, only the color required for the line is used.

The `INCLUDE TYPE` (or `INCLUDE STRUCTURE`) statement provides a simple way of defining the necessary line type for the data table locally in the program.

Fill the color column of your data table. As a rule, comply with the following convention: "**C**`<color_constant>`**10**". In this context, `<color_constant>` represents the eight color constants of the COL type group that you must incorporate into your program, for example using the `TYPE-POOLS col` statement.

In the *info_fname* field, enter the component name of your color column for a structure of the global type *lvc_s_layo* and pass the structure to the parameter *is_layout* of the method *set_table_for_first_display*.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
wa_sbook-color = 'C' && <color_constant>
                     && <intensified>
                     && <inverse>.
```

Possible values:

`<color_constant>`	**Color number from the table**
`<intensified>`	**1 or 0**
`<inverse>`	**1 or 0**

No.	Color	INTENSIFIED	INTENSIFIED OFF	INVERSE
0	COL_BACKGROUND	0123456789	0123456789	0123456789
1	COL_HEADING	0123456789	0123456789	0123456789
2	COL_NORMAL	0123456789	0123456789	0123456789
3	COL_TOTAL	0123456789	0123456789	0123456789
4	COL_KEY	0123456789	0123456789	0123456789
5	COL_POSITIVE	0123456789	0123456789	0123456789
6	COL_NEGATIVE	0123456789	0123456789	0123456789
7	COL_GROUP	0123456789	0123456789	0123456789

Figure 77: Color Options

The value of the color field is made up of four components:

- The letter “C”
- The number of one of the eight colors
- 1 (intensified display) or 0 (intensified display off)
- 1 (inversion of foreground and background color) or 0 (no inversion)

Examples: C111, C710, C501.

For an overview of colors, refer to the online documentation for the ABAP command `FORMAT`.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

- **Extend the data table with an internal field color table**
- **This internal table contains an entry for every field of the data row to be highlighted.**

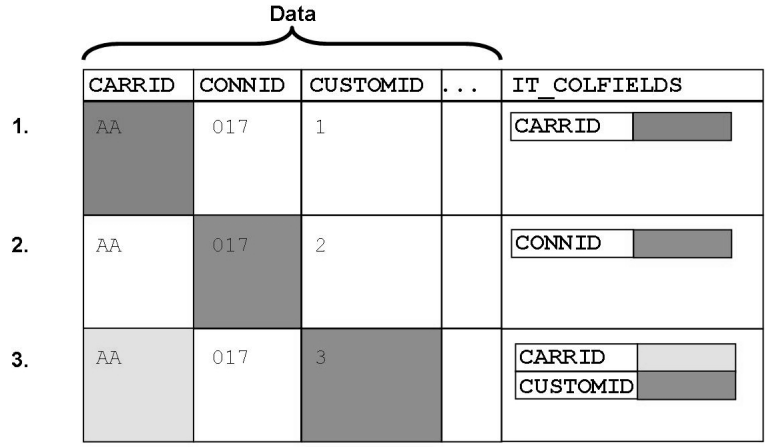


Figure 78: Changing the Cell Color (1)

To highlight individual cells using colors, the row type of the output table (data table) must be expanded by one internal table. The contents of this internal table is not output but included only for the layout.

In this internal table, for each data record in the data table, list the fields that must be highlighted with color and the color code for each. Each field to be highlighted requires an entry in this internal table.

A cell color has a higher priority than a row color.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

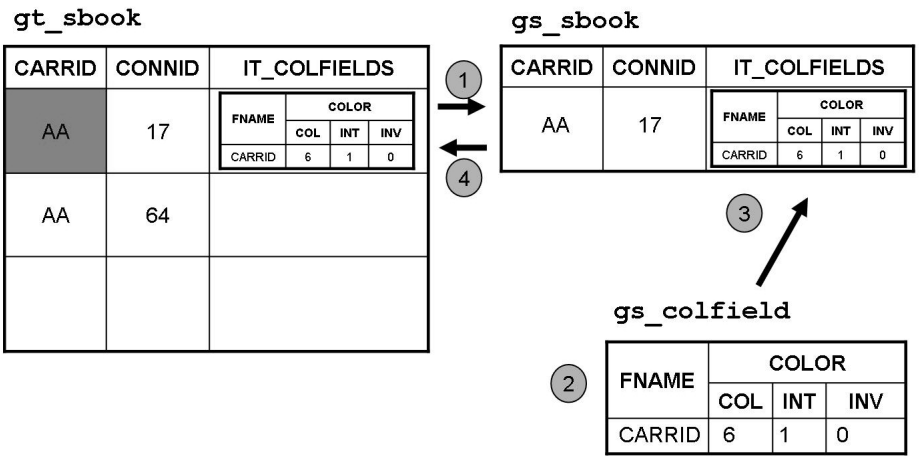


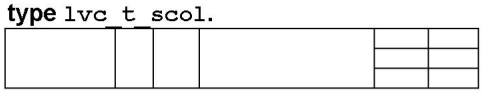
Figure 79: Changing the Cell Color (2)

1. Looping the internal table *gt_sbook* in the *gs_sbook* structure. Different cells can be colored for each data record. If cells must only be colored for one data record, then a `READ TABLE` is sufficient.
2. Filling the *gs_colfield* help field. It has two fields:
   - *fname*: The name of the field to be colored.
   - *color*: The color code to be used, compiled from the three subfields *col*, *int* and *inv*.
3. Appending the *gs_colfield* help field to the internal table *gs_sbook-it_colfields* (which contains information about the cell color combinations.)
4. Writing back the *gs_sbook* work area into the internal data table using `MODIFY`.

You must complete steps two and three for each field to be colored.

The steps required to color by cell:

- **Extend the row structure of the data table with an internal color table of type `lvc_t_scol`.**



- **Within a loop for the data table, define for every record which cells are to be highlighted with which colors.**
- **Assign the name of the field color table to the field `ctab_fname` of the layout structure.**
- **Pass the layout structure to the method `set_table_for_first_display`.**

Figure 80: Changing the Cell Color (3)

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* allow symbolic names for colors, e.g. col_normal
TYPE-POOLS col.

* layout structure needed for set_table_for_first_display
DATA gs_layout TYPE lvc_s_layo.

* data table needs to have an extra internal table to keep those
* fields that need to be emphasized
TYPES: BEGIN OF gty_booking.
         INCLUDE      TYPE sbook.
TYPES:   it_colfields TYPE lvc_t_scol,
       END OF gty_booking.
DATA: gt_bookings TYPE TABLE OF gty_booking,
      gs_booking  TYPE          gty_booking.

* work area is required in order to fill the internal table
* with the fields/colors
gs_colfield LIKE LINE OF gs_sbook-it_colfields.
```

Figure 81: Changing the Cell Color (4)

Expand the output table (data table) by adding an internal table of the *lvc_t_scol* type. (The contents of this internal table is not output but included only for the layout.)

The row type for this internal table contains the following fields:

- *fname*: The name of the field in which the data row is to be highlighted.
- *color*: The color setting for this field.
  - *col*: One of the eight color constants of the **col** type group.
  - *int*: 1 means intense, 0 means not intense
  - *inv*: 1 means inverted (the foreground and background colors are switched), 0 means not inverted.
  - *nokeycol*: If the *nokeycol* field is set, you can change the color of key fields.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* loop at data table and find the smokers
* set color red for the field 'SMOKER'
LOOP AT gt_bookings INTO gs_booking
     WHERE smoker = 'X'.
  gs_colfield-fname     = 'SMOKER'.
  gs_colfield-color-col = col_negative.
  gs_colfield-color-int = '1'.
  gs_colfield-color-inv = '0'.
  APPEND gs_colfield TO gs_booking-it_colfields.
  MODIFY gt_bookings FROM gs_booking
                     TRANSPORTING it_colfields.
ENDLOOP.

* pass the name of the internal table with the
* fields/colors to the layout structure
gs_layout-ctab_fname = 'IT_COLFIELDS'.

* call normal display method
go_alv_grid->set_table_for_first_display(
  EXPORTING  i_structure_name = 'SBOOK'
             is_layout        = gs_layout
  CHANGING   it_outtab        = gt_bookings ).
```

Figure 82: Changing the Cell Color (5)

It is optimal for you to determine for each data record whether fields are to be highlighted using color and to loop this using you data table.

You can assign the names of the internal tables which contain color field information to the *ctab_fname* field of the layout structure.

Transfer the layout structure and the output table using the *set_table_for_first_display* method.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Hiding Standard Functions

Information about possible function codes:

```
* define internal table for functions to be excluded
DATA: gt_excluded_functions TYPE ui_functions.

* list functions and menus that should be excluded
APPEND cl_gui_alv_grid=>mc_fc_filter TO gt_excluded_functions.
APPEND cl_gui_alv_grid=>mc_mb_sum    TO gt_excluded_functions.

* pass internal table to method
go_alv_grid->set_table_for_first_display(
  EXPORTING
    i_structure_name     = 'SBOOK'
    it_toolbar_excluding = gt_excluded_functions
  CHANGING
    it_outtab            = gt_bookings
  EXCEPTIONS
    others               = 1 ).
```

Figure 83: Hiding Standard Functions

Depending on the application and the data displayed in the ALV Grid Control, only some standard functions are useful. You can display either only a subset of the standard functions, or hide the entire toolbar.

To hide **individual functions**, define an internal table of the *ui_functions* type. Fill the internal table with the name of the functions that are to be kept inactive. Inactive functions on the application toolbar are not displayed. Inactive functions that are assigned a menu entry, on the other hand, are grayed out.

The possible function codes are defined as constants of the *cl_gui_alv_grid* class. The class constants for function codes all have the prefix **MC_FC_**. The prefix **MC_MB_** represents an entire menu in the toolbar.

Finally, pass the table to the standard functions when you call the *set_table_for_first_display* method.

To completely deactivate **all functions** on the application toolbar, you can use the *MC_FC_EXCL_ALL* class constant.

To hide the **entire toolbar**, only set the *no_toolbar* parameter of the layout structure to “X” and pass this structure to *set_table_for_first_display*.

We will look at the (event-driven) **addition** of functions to the application toolbar at a later stage.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Exercise 8: ALV Grid Control: Layout

Exercise Objectives

After completing this exercise, you will be able to:

- Control the layout of your ALV Grid Control

Business Example

The user of your program wishes, among other things, to be able to indicate multiple rows at the same time and for flights of the current month to be highlighted using color.

Task:

Adjust the layout of your program.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_DESIGN_S2*

1. Expand your program from the last exercise or copy from the *BC405_DESIGN_S1* program.
2. **Preparation**

   You need a field of type *lvc_s_layo*. This field is used to handle the appropriate layout parameters to the *SET_TABLE_FOR_FIRST_DISPLAY* method.
3. **Small Visual Changes**

   Set the text *Flights* as the title for the ALV Grid Control.

   In the ABAP Dictionary, find out the names of the fields in the *lvc_s_layo* structure that are used for suppressing the horizontal and vertical grid lines. Set them appropriately.

   Optional: Make sure that the ALV title can be translated.
4. Allow the user to select multiple columns and multiple rows.

   Hint: Multiple columns/rows can be indicated by using the left mouse button while holding down the Ctrl key.
5. **Highlight Rows**

   Mark the rows for all flights for the current month in a particular color.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

To do this, you must change your simple definition of the internal database table *gt_flights* and your *gs_flight* work area: Define the work area using the syntax `BEGIN OF ... END OF ...`.

Insert the *SFLIGHT* dictionary structure and add a character field (suggested name: **color**) with a length of 4.

Now assign a type to the internal table with reference to the structure that you have just created.

You to not have to change anything at the data retrieval (complete the `INTO CORRESPONDING FIELDS` addition if necessary). After data retrieval, you must change the contents of the internal table:

For flights in the current month, put a color of your choice into the *color* field. This is not to be highlighted but inverted.

Hint: Include the *col* type group so that your code can be read. Descriptive constants such as *col_normal* are also available.

6. **Traffic Light Icon**

Add a graphical indicator ("traffic light icon") for the airline. Use exceptions to do this. It is necessary for you to expand the data structure of the internal table *gt_sflight* again by adding a field of the type *c* (suggested name: **light**).

The condition for the traffic light color: If no bookings are available in economy class exist for a flight (`seatsocc = 0`), the traffic light should be red. For a value of less then 50, it should be yellow and otherwise it should be green.

7. **Optional**: Highlight Cells

Highlight the cells containing the aircraft type in the ALV – but only if they correspond to an aircraft type of your choice.

Hint: Using the transaction SE16 (Data Browser), you can determine which plane types are currently available in the *SAPLANE* table.

Expand the data structure of the internal table *gt_flights* as well as your output area *gs_flight*:

Attach a new field of type *lvc_t_scol* (suggested name ***it_field_colors***). Create a corresponding work area (suggested name ***gs_field_color***). This work area has to have the same structure type as a row of the internal table *it_field_colors*.

For each row of the *gt_flights* data table, check whether the plane is of the desired type.

Continued on next page

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

If the aircraft type matches yours, assign the fields of the work area *gs_field_color* appropriately. Use the *col_positive* color. Attach the filled work area to the internal table *gs-flight-it_field_colors*.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 8: ALV Grid Control: Layout

Task:

Adjust the layout of your program.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_DESIGN_S2*

1. Expand your program from the last exercise or copy from the *BC405_DESIGN_S1* program.

   a) Self-explanatory

2. **Preparation**

   You need a field of type *lvc_s_layo*. This field is used to handle the appropriate layout parameters to the *SET_TABLE_FOR_FIRST_DISPLAY* method.

   a) See result at the end.

3. **Small Visual Changes**

   Set the text *Flights* as the title for the ALV Grid Control.

   In the ABAP Dictionary, find out the names of the fields in the *lvc_s_layo* structure that are used for suppressing the horizontal and vertical grid lines. Set them appropriately.

   Optional: Make sure that the ALV title can be translated.

   a) See result at the end.

4. Allow the user to select multiple columns and multiple rows.

   Hint: Multiple columns/rows can be indicated by using the left mouse button while holding down the Ctrl key.

   a) See result at the end.

5. **Highlight Rows**

   Mark the rows for all flights for the current month in a particular color.

   To do this, you must change your simple definition of the internal database table *gt_flights* and your *gs_flight* work area: Define the work area using the syntax `BEGIN OF ... END OF ...`.

   Insert the *SFLIGHT* dictionary structure and add a character field (suggested name: **color**) with a length of 4.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Now assign a type to the internal table with reference to the structure that you have just created.

You to not have to change anything at the data retrieval (complete the `INTO CORRESPONDING FIELDS` addition if necessary). After data retrieval, you must change the contents of the internal table:

For flights in the current month, put a color of your choice into the *color* field. This is not to be highlighted but inverted.

Hint: Include the *col* type group so that your code can be read. Descriptive constants such as *col_normal* are also available.

a) See result at the end.

6. **Traffic Light Icon**

Add a graphical indicator ("traffic light icon") for the airline. Use exceptions to do this. It is necessary for you to expand the data structure of the internal table *gt_sflight* again by adding a field of the type *c* (suggested name: **light**).

The condition for the traffic light color: If no bookings are available in economy class exist for a flight (`seatsocc = 0`), the traffic light should be red. For a value of less then 50, it should be yellow and otherwise it should be green.

a) See result at the end.

7. **Optional**: Highlight Cells

Highlight the cells containing the aircraft type in the ALV – but only if they correspond to an aircraft type of your choice.

Hint: Using the transaction SE16 (Data Browser), you can determine which plane types are currently available in the *SAPLANE* table.

Expand the data structure of the internal table *gt_flights* as well as your output area *gs_flight*:

Attach a new field of type *lvc_t_scol* (suggested name ***it_field_colors***). Create a corresponding work area (suggested name ***gs_field_color***). This work area has to have the same structure type as a row of the internal table *it_field_colors*.

For each row of the *gt_flights* data table, check whether the plane is of the desired type.

Continued on next page

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

If the aircraft type matches yours, assign the fields of the work area *gs_field_color* appropriately. Use the *col_positive* color. Attach the filled work area to the internal table *gs-flight-it_field_colors*.

a) See result at the end.

Result

Main program

```
REPORT  bc405_design_s2.

TYPE-POOLS: col.

* work area type for ALV data and color information
TYPES: BEGIN OF gty_flight.
         INCLUDE         TYPE sflight.
TYPES:   color           TYPE c LENGTH 4,
         light           TYPE c LENGTH 1, "lights ind. booking status
         it_field_colors TYPE lvc_t_scol, "for cell highlighting
       END OF gty_flight.

DATA: gt_flights TYPE TABLE OF gty_flight,
      gs_flight  TYPE          gty_flight.

DATA: ok_code LIKE sy-ucomm.

DATA: go_alv_grid  TYPE REF TO cl_gui_alv_grid,
      go_container TYPE REF TO cl_gui_custom_container.

DATA: gs_variant TYPE disvariant.

* data needed for layout
DATA: gs_layout      TYPE lvc_s_layo,
      gs_field_color TYPE lvc_s_scol.

SELECT-OPTIONS: so_car FOR gs_flight-carrid,
                so_con FOR gs_flight-connid.
SELECTION-SCREEN SKIP.
PARAMETERS: pa_lv TYPE disvariant-variant.


***********************************************************************
*ABAP events
***********************************************************************
START-OF-SELECTION.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
  SELECT * FROM sflight
    INTO CORRESPONDING FIELDS OF TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

  LOOP AT gt_flights INTO gs_flight.

* set indicator for flights of current month
    IF gs_flight-fldate(6) = sy-datum(6).
      gs_flight-color = 'C' && col_negative && '01'.
    ENDIF.

* set icon for bookings status
    IF gs_flight-seatsocc = 0.
      gs_flight-light = 1.
    ELSEIF gs_flight-seatsocc < 50.
      gs_flight-light = 2.
    ELSE.
      gs_flight-light = 3.
    ENDIF.

* highlight specific aircraft
    IF gs_flight-planetype = '747-400'.
      gs_field_color-fname     = 'PLANETYPE'.
      gs_field_color-color-col = col_positive.
      gs_field_color-color-int = 1.
      gs_field_color-color-inv = 0.
      gs_field_color-nokeycol  = 'X'.
      APPEND gs_field_color TO gs_flight-it_field_colors.
    ENDIF.

    MODIFY gt_flights FROM gs_flight
           TRANSPORTING color light it_field_colors.
  ENDLOOP.

  CALL SCREEN 100.


***********************************************************************
*PBO modules
***********************************************************************
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.                 " clear_ok_code  OUTPUT
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
*---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'DYN'.
  SET TITLEBAR 'T1'.
ENDMODULE.                 " STATUS_0100  OUTPUT

*---------------------------------------------------------------------*
MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    gs_variant-report  = sy-cprog.
    gs_variant-variant = pa_lv.

    gs_layout-grid_title = 'Flights'(h01).
    gs_layout-no_hgridln = 'X'.
    gs_layout-no_vgridln = 'X'.

* field that contains information on row color
    gs_layout-info_fname = 'COLOR'.

* internal table that contains information on cell color
    gs_layout-ctab_fname = 'IT_FIELD_COLORS'.

*field that contains information on exception (indicator)
    gs_layout-excp_fname = 'LIGHT'.

* multiple row and column selection
    gs_layout-sel_mode   = 'A'.
```

Continued on next page

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
    go_alv_grid->set_table_for_first_display(
      EXPORTING
        i_structure_name = 'SFLIGHT'
        is_variant       = gs_variant
        i_save           = 'A'
        is_layout        = gs_layout
      CHANGING
        it_outtab        = gt_flights
      EXCEPTIONS
        OTHERS           = 1 ).
    IF sy-subrc <> 0.
      MESSAGE a012(bc405_408).
    ENDIF.

  ENDIF.
ENDMODULE.                 " create_and_transfer  OUTPUT


*********************************************************************
*PAI Modules
*********************************************************************
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK' OR 'CANCEL' OR 'EXIT'.
      SET SCREEN 0.
  ENDCASE.
ENDMODULE.                 " USER_COMMAND_0100  INPUT
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Create the layout structure of the ALV Grid Control
- Set colors of lines and cells in the ALV Grid Control
- Hide standard toolbar functions

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV Grid Control: Field Catalog

Lesson Overview

Using the field catalog, it is possible to determine column properties.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the purpose of the field catalog
- Add columns to the ALV Grid Control or change columns of the ALV

Business Example

You are working as a programmer for ABC Travel. Some additional data that is only calculated at runtime is now to be displayed for the flight information. It is your task to add a new column to the ALV.

Purpose of the Field Catalog

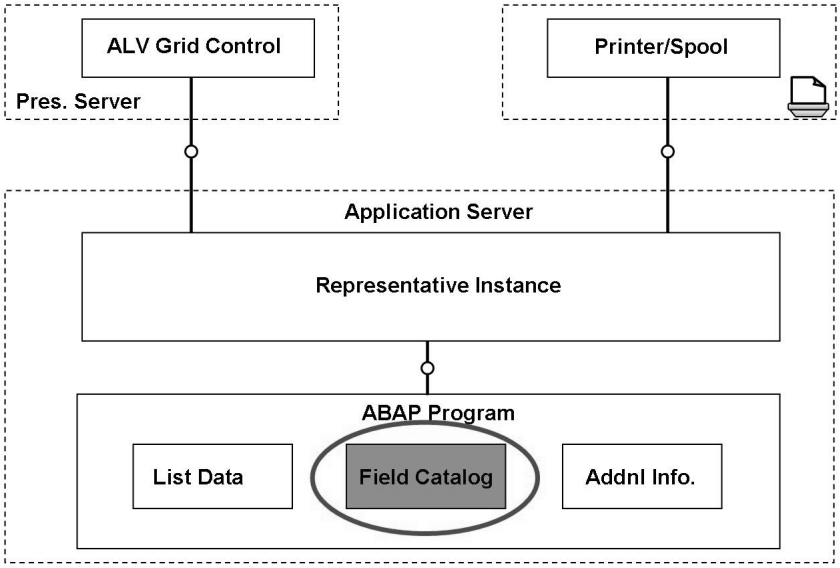


Figure 84: ALV Grid Control: Field Catalog

The **field catalog** is a format description of the display area for data.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

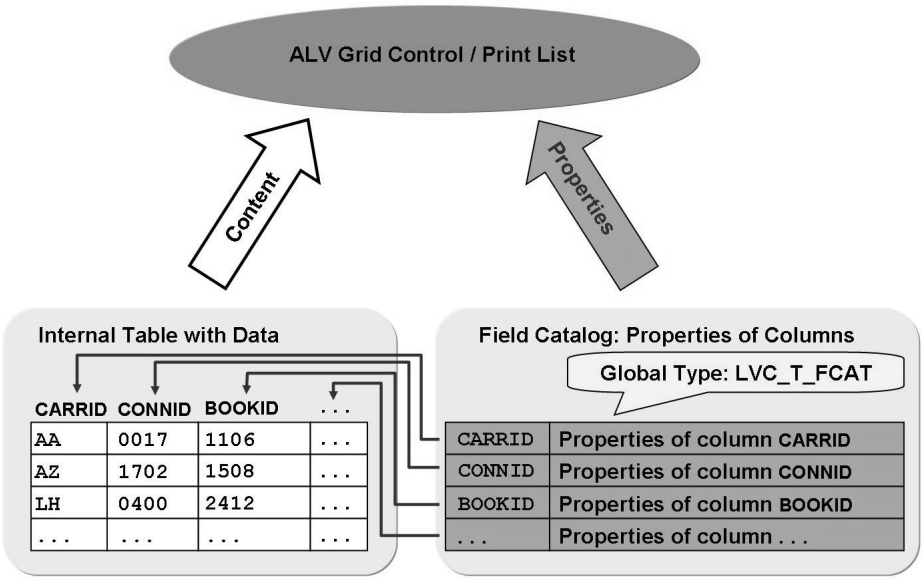


Figure 85: Function of the Field Catalog

The internal table that contains the data to be displayed can have a freely-definable row type. So that the representative instance can display the data transferred to it as desired on the screen output or when creating a print list, it requires corresponding information, the **field catalog**.

You can have the field catalog automatically generated by the representative instance. For row types that are defined in the ABAP Dictionary, you only need to pass the name of the structure in the ABAP Dictionary to the representative instance for this.

Alternatively, or additionally, you can, however, also pass this display information to the representative instance using an additional internal table. For simplicity, as of this point, **this additional table** will be known as the field catalog. The global data type of this internal table is LVC_T_FCAT. Your row type is called LVC_S_FCAT.

Examples of typical situations in which you need to pass a field catalog are:

- Although the internal table containing the data has a row type with Dictionary reference, the display is to differ from it, for example with regard to column position or title.
- The internal table has (only or additionally) columns that are not contained in a Dictionary structure.

The field catalog needs to contain a row for every column of the data table that differs from an underlying Dictionary structure or is not contained in the Dictionary. This row must contain the technical properties and other formatting information for the column.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Adjusting the Grid Structure Without the Field Catalog

- For many requirements, there are alternatives to using the field catalog:
  - Additional / different columns

    → Use a different global structure type;
  - Hide columns / different order

    → Use standard display variables;
  - Display only some of the columns left-aligned

    → Use the INCLUDE TYPE statement for local row type of the data table;
  - Generate the data table dynamically

    (as of *SAP Web Application Server 6.10*)

    → Transfer the structure type as field symbol.

Before you define and fill a field catalog, and pass it to the representative instance, you can consider whether you can implement your requirements another way. In some cases, it is simpler, more flexible, or even both for you to implement the alternative solutions listed above.

Some of the above solution approaches also offer additional advantages, such as central maintainability, reusability possibilities, and so on.

The INCLUDE TYPE statement has existed since *SAP Web Application Server 6.10*. You should use it to include *Types* (Dictionary types, ABAP types, self-defined types) *directly*. Use INCLUDE STRUCTURE to include *types of data fields* in your program. (For reasons of backward compatibility, you can also continue to use INCLUDE STRUCTURE to include types.)

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

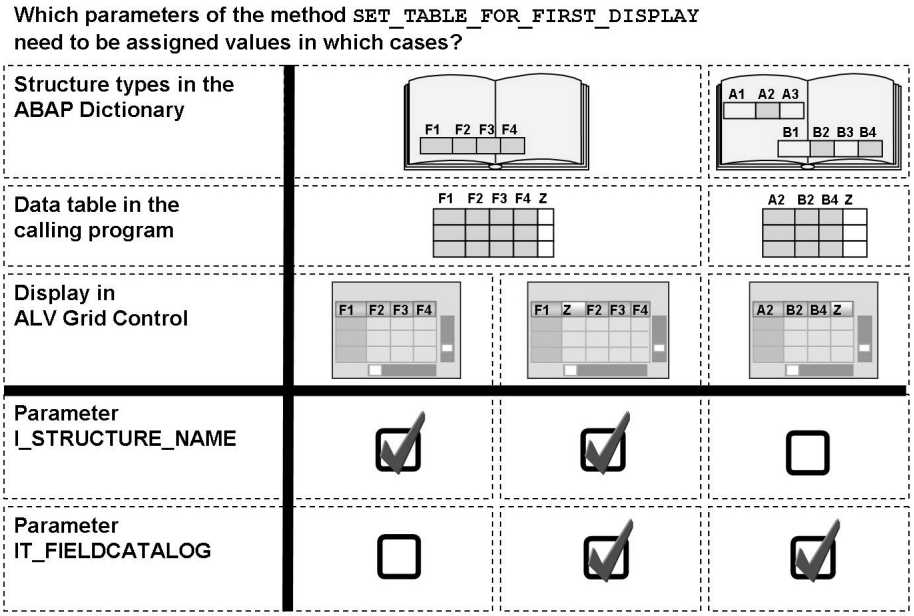


Figure 86: Options for Creating a Grid

Field catalogs are generated differently, depending on the row type used for the data table. There are three situations that can occur:

Case 1:

All fields of a global structure type exist in the data table with the same names. Exactly these are also to be output.

In this case, you can have the field catalog automatically generated by the representative instance. You only need to pass the name of the global structure type. Columns of the data table that are not defined in the global structure type are not displayed.

Case 2:

All fields of a global structure occur in the data table with the same names, however adjustments are to be made to the specifications from the ABAP Dictionary, or additional columns are to be output, or both.

In this case, you need to create rows in the field catalog for the columns to be changed or added.

Case 3:

The row type of the data table contains either no references to the ABAP Dictionary or only to individual global structure fields or data elements.

In this case, the field catalog needs to be completely created by the calling program.

Cases 2 and **3** are explained in more detail in the following.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

What entries need to be made in the field catalog and in which cases?

Properties:
- Unit formatting
- Output properties
- Formatting of column contents
- Texts
- Technical column properties
- Other fields

Column in data table contains reference to global structure field

Column in data table contains reference to global structure field, but adjust definition

Column in data table is of a locally defined type

FIELDNAME	REF_FIELD	REF_TABLE	...
✓	✓	✓	☐
✓	✓	✓	✓
✓	☐	☐	✓

Field Catalog

Figure 87: Overview of the Structure of the Field Catalog

The situations shown here will now refer only to **Cases 2** and **3** described above.

You create a field catalog in the calling program as follows:

- Define an internal table for the field catalog based on the global table type **lvc_t_fcat** and a structure with compatible rows as the work area.
- Fill the fields of the work area for each affected column of your data table and add it to the internal table for the field catalog.

Assign the name of a column from the data table to the field *fieldname*. This field assigns a row in the field catalog to a column of the data table.

The other fields of the field catalog can be divided into two groups with regard to their use when creating a field catalog:

- References to global types (structure fields in the ABAP Dictionary) are created using the fields *ref_field* and *ref_table*.
- All other fields of the field catalog contain values for the properties of the columns.

If you assign values to the fields *ref_table* and, if necessary, *ref_field*, all type definitions are transferred from the specified structure fields in the ABAP Dictionary. You can overwrite these by assigning values to **individual** fields of the latter group.

If you do not assign values to either *ref_field* or *ref_table*, you **must** assign values, **if necessary for all** fields, of the latter group.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

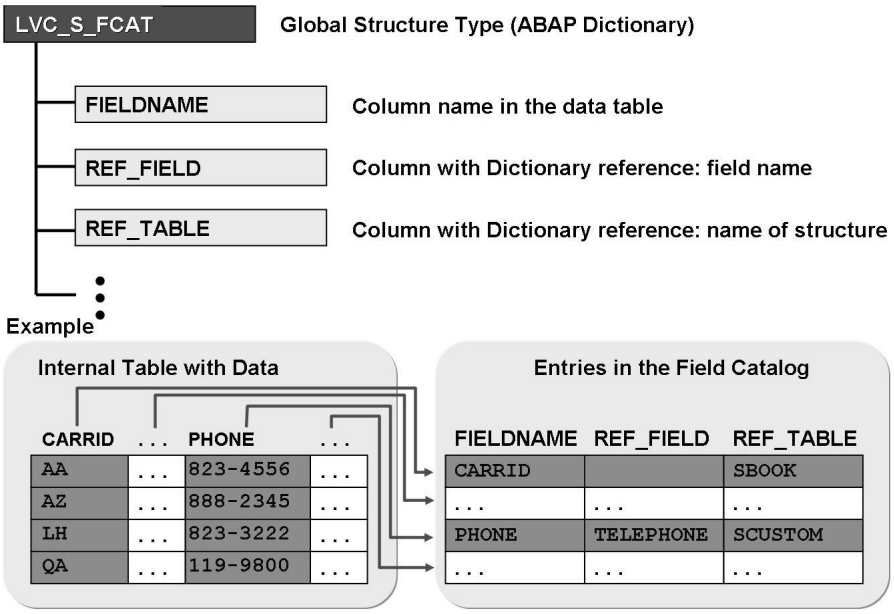


Figure 88: Assigning Types to Columns with Dictionary References

The *fieldname* column in the field catalog contains the column name of the data table. Every column of the data table for which you want to control formatting must have a row in the field catalog.

If you want to refer to a field of a global structure **with the same name**, it is sufficient to assign the name of the structure to the field *ref_table*. You only need to add the field name from the structure to the *ref_field* field if the column name of the data table and the structure field name are different.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Adding or Changing Columns

For booking data:

- Additional column “telephone”
- Additional column “booking canceled” (as icon)
- Hide original “booking canceled” column

```
TYPES: BEGIN OF gty_booking.
         INCLUDE        TYPE sbook.
TYPES:   phone          TYPE scustom-telephone,
         cancelled_icon TYPE icon-id, "char 4
       END OF gty_booking,

DATA: gt_bookings TYPE TABLE OF gty_booking.

* field catalog and appropriate work area
DATA:
  gt_field_cat  TYPE lvc_t_fcat,
  gs_field_cat  TYPE lvc_s_fcat.
```

Output Table

Field Catalog

Figure 89: Example: Adding and Changing Columns (1)

In the example above, the data table consists of fields (columns) of the type SBOOK and two others. You need to describe these in the field catalog, along with the column that is to be hidden.

```
gs_field_cat-fieldname = 'PHONE'.
gs_field_cat-ref_field = 'TELEPHONE'.
gs_field_cat-ref_table = 'SCUSTOM'.
gs_field_cat-col_pos   = 4.
APPEND gs_field_cat TO gt_field_cat.

CLEAR gs_field_cat.
gs_field_cat-fieldname = 'CANCELLED_ICON'.
gs_field_cat-icon      = 'X'.
gs_field_cat-coltext   = 'Cancelled'(h05).
APPEND gs_field_cat TO gt_field_cat.

CLEAR gs_field_cat.
gs_field_cat-fieldname = 'CANCELLED'.
gs_field_cat-no_out    = 'X'.
APPEND gs_field_cat TO gt_field_cat.
```

Fill field catalog for additional fields

```
go_alv_grid->set_table_for_first_display(
  EXPORTING   i_structure_name = 'SBOOK'
  CHANGING    it_outtab        =  gt_bookings
              it_fieldcatalog  =  gt_field_cat ).
```

Figure 90: Example: Adding and Changing Columns (2)

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

The above example shows a semi-automatic field description: Part of the field description comes from the Dictionary structure (SBOOK), while another part is explicitly defined in the field catalog (*gt_fieldcats*).

The field catalog (internal table) is filled in the program and is passed on together with the name of the Dictionary structure during the method call. The information is merged accordingly in method *set_table_for_first_display*.

Note that the fields of the data table *gt_bookings* that cannot be read from the SBOOK table also need to filled meaningfully.

If you are constructing the field catalog semi-automatically or manually, you should check the consistency. While pressing the shift key, double-click with the right mouse button in an area of the ALV grid that contains no data (for example the header).

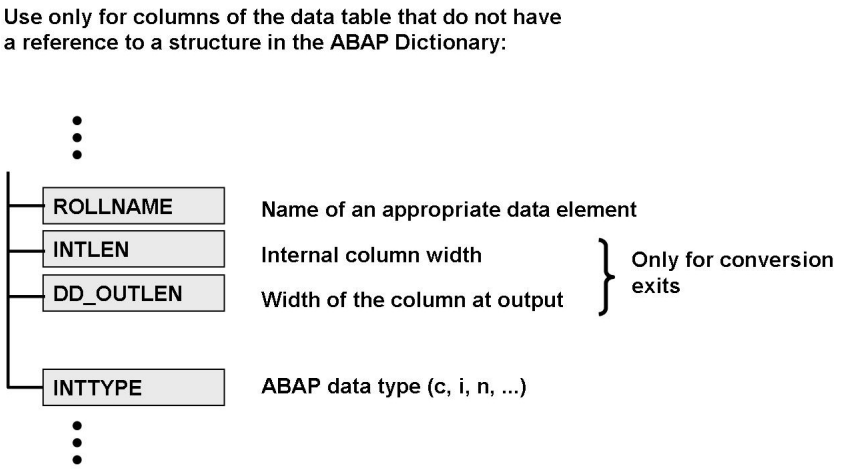


Figure 91: Assigning a Type to a Column Without Reference to a Dictionary Structure

rollname: Using this, you specify the data element that is used for the F1 help and the text for the column headers, selection screens and quick info text (if it is not already explicitly set).

If you have not filled either the field *ref_field* or the field *ref_table*, there is not an adequate reference to the ABAP Dictionary. In this case, the fields shown here allow you to format the relevant columns for output.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

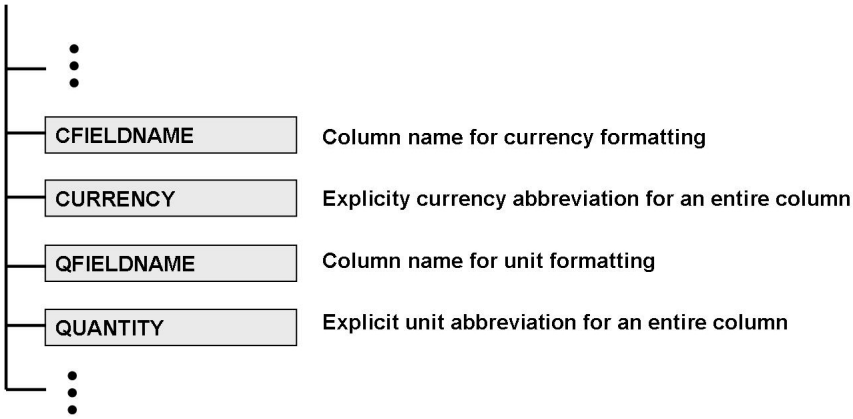


Figure 92: Formatting Amounts Appropriately for Units

Currency formatting:

- If the contents of a column are to be formatted in accordance with the formatting options of a currency unit that is specified in another column of the data table, specify the name of this column in the field *cfieldname*.
- If all values of a column are to be formatted in accordance with a particular currency, specify the relevant currency abbreviation in the *currency* column.
- If the *currency* field has a value, entries in the *cfieldname* are ignored.

Quantity formatting:

- If the contents of a column are to be formatted in accordance with the formatting options of a unit that is specified in another column of the data table, specify the name of this column in the field *qfieldname*.
- If all values of a column are to be formatted in accordance with a particular unit, specify the relevant unit abbreviation in the *quantity* column.
- If the *quantity* field has a value, entries in the *qfieldname* are ignored.

For more information, see the online documentation.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

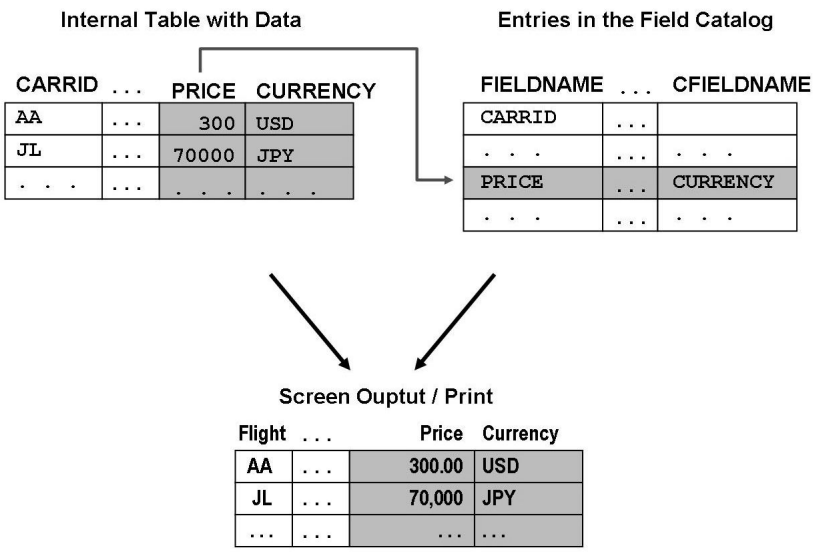


Figure 93: Example: Appropriate Formatting for Currency Columns

In the example shown here, the data table contains the two columns *price* (the amount to be paid for a flight) and *currency* (currency for the contents of *price*).

By making the entry "CURRENCY" in the *cfieldname* field of the field catalog for the *price* column, you determine that its contents are to be formatted in accordance with the currency specified in the *currency* column.

(Example: "USD"→ two decimal places, "JPY"→ no decimal places.)

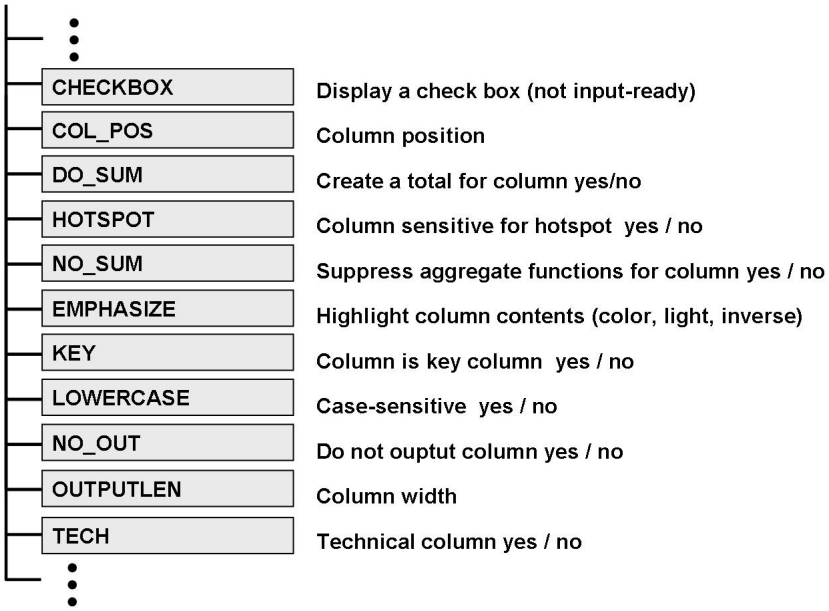


Figure 94: Output Properties of a Column

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Field **col_pos**:

To calculate the column position, hidden columns need to be taken into account, but exceptions do not. Without explicit positioning, fields that are manually added to the field catalog are positioned before the other fields of the data table. Exceptions are always displayed in the first column.

Field **emphasize**:

To highlight with a default color, set “X” here.

Alternatively, you can define the color yourself by passing a four-digit numerical code in accordance with the following convention:

C<col_constant> {0|1}{0|1}. **<col_constant>** stands for a color constant of the type group **col**. The second digit specifies whether the color is to be displayed intensively. “1” as the third digit means an inverted display. Example: C300, C401.

Field **no_out**:

The column is hidden on the list (screen or print), but appears in the field selection dialogs of the standard functions (sorting, filtering, and so on).

Field **tech**:

The column is not displayed in the list or in the field selection dialogs.

For more information, see the online documentation.

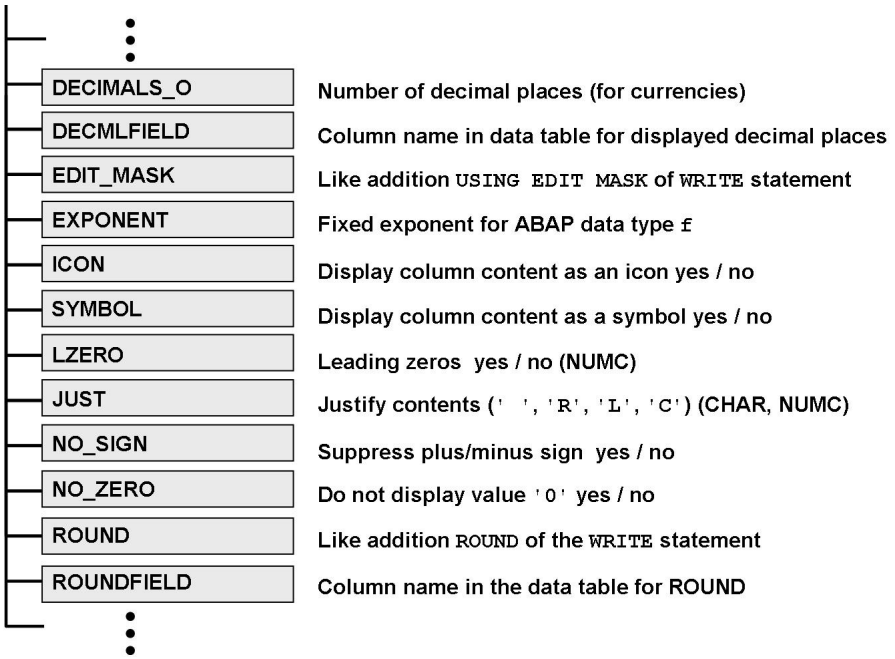


Figure 95: Formatting Properties of Column Contents

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

In the same way as for the formatting options for currency fields, you can use the two fields *decmlfield* and *decimals_o* to define the numbers of decimal places.

These entries affect numerical content.

If you want to output a column as an icon, set the field *icon* of the layout structure to “X”. Run the report SHOWICON to obtain an overview of icons. Note that not all icons can be printed.

If you want to output a column as a symbol, set the field *symbol* of the layout structure to “X”. Run the report SHOWSYMB to obtain an overview of symbols.

For more information, see the online documentation.

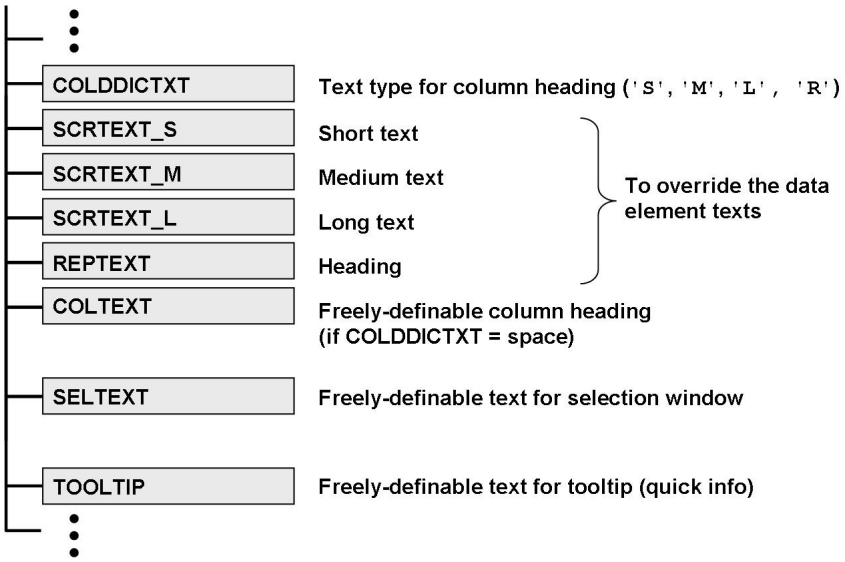


Figure 96: Texts

You only need to make entries in the fields listed here if a column has no reference to the ABAP Dictionary or you want to explicitly define or override a text.

You can use the value assigned to the field *colddictxt* to determine which of the four data element texts are to be used as the *column header* (**“S”** = **s**hort, **“M”** = **m**iddle, **“L”** = **l**ong, **“R”** = heading). These texts are in the fields *scrtext_s, scrtext_m scrtext_l*, and *reptext*, or (if there is no Dictionary reference) can be set there.

If *colddictxt* has an initial value, the text of the field *coltext* is set as the column header.

If *coltext* also has an initial value, an appropriate text is automatically used from the four data element texts, depending on the width of the column.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The **texts in selection screens** (for example with a filter) behave in the following way: The entry in the *seltext* field has the highest priority. If it is missing, the longest of the four data element texts is used. If none of these four texts exists, the contents of the field *tooltip* is used, or finally, the field name itself.

With *tooltip*, you set the text that is displayed if the mouse cursor hovers over the column header (*quick info*). If there is no entry in *tooltip*, the longest of the four data element texts is used. If none of these four texts exist, the field *coltext* is used.

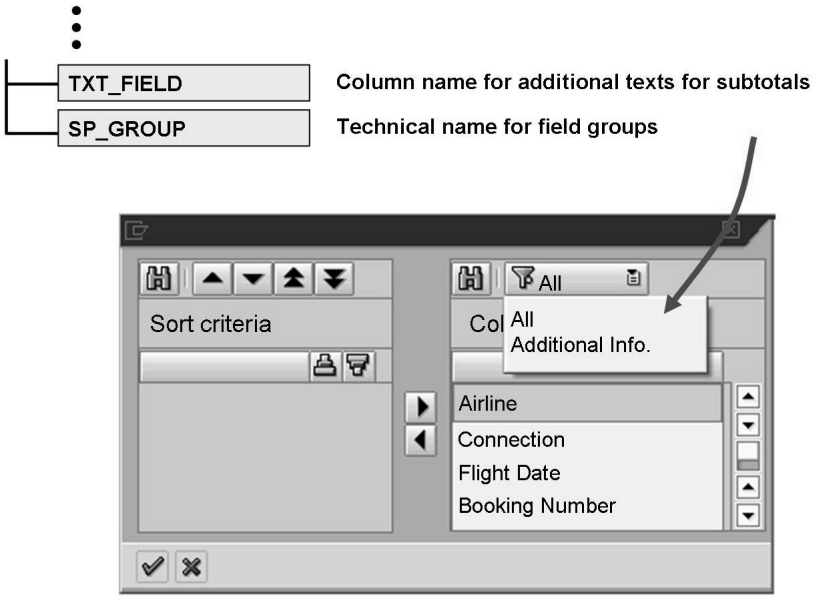


Figure 97: Other Properties of a Column

Field **sp_group**:

You combine multiple fields into a group with the group key. In various dialog boxes (such as the selection of the sort fields), the user can then display all columns or only those of a specific group.

To be able to work with group texts, you also need to fill an internal table of type *lvc_t_sgrp* for the field group texts and pass it using the parameter *it_special_groups* of the method *set_table_for_first_display*:

```
DATA gt_groups TYPE lvc_t_sgrp.
DATA gs_group  TYPE lvc_s_sgrp.

gs_group-sp_group = 'INFO'.
gs_group-text     = text-inf.
APPEND gs_group TO gt_groups.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Field **txt_field**:

You can use this field to obtain additional texts for subtotals.

Example:

Column A contains works numbers, column B contains the associated long texts. On the list, the associated long texts are also to be displayed for subtotals of the works numbers.

The name of column B must then be in the field *txt_field* for column A.

(That is: `<struc_name>-txt_field = 'B'.`)

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 9: ALV Grid Control: Field Catalog

Exercise Objectives

After completing this exercise, you will be able to:

- Control column properties
- Build the field catalog semi-automatically

Business Example

These users would like some properties of individual columns to be changed in your program, for example, a different column header or continuous summation.

In addition, the users want additional data that does not appear in the database to appear in ALV. To solve this problem, you have to create an additional column and display it in the ALV.

Task:

Change the properties of individual columns using the field catalog. Add a new column to the ALV.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_DESIGN_S3*

1. Expand your program from the last exercise or copy the model solution of the last task (*BC405_DESIGN_S2*).
2. Define properties of existing columns

   Create a field called *gt_field_cats* of the type *lvc_t_fcat* and an appropriate work area *gs_field_cat* of type *lvc_s_fcat*. Pass the field catalog to the method *set_table_for_first_display*.

   Add an entry to the field catalog so that the column containing the occupied seats in Economy Class is totaled by default.

   The column containing the current booking totals is to be hidden. Extend the field catalog appropriately.

   The traffic light icon that graphically displays the utilization is to have a descriptive (and translatable) heading.
3. Add additional columns to the data table and display them in the ALV

   Expand the row type of the data table so that a further field with the name *changes_possible* of the *icon-id* type will be included.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Fill this field for each data record as follows: If the flight date is in the past it should receive the *icon_space* value, in other cases, it should be *icon_okay.* So that the symbolic names of the icons are known, you need to insert the following syntax into your program:

```
TYPE-POOLS: icon.
```

Add the new column to the automatically generated field catalog as follows:

Add a new entry to the field catalog so that the new column is output in fifth position, with the (translatable) heading *Changes Possible* and the quick info (tooltip) *Are changes still possible?*

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 9: ALV Grid Control: Field Catalog

Task:

Change the properties of individual columns using the field catalog. Add a new column to the ALV.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_DESIGN_S3*

1. Expand your program from the last exercise or copy the model solution of the last task (*BC405_DESIGN_S2*).

   a) Self-explanatory

2. Define properties of existing columns

   Create a field called *gt_field_cats* of the type *lvc_t_fcat* and an appropriate work area *gs_field_cat* of type *lvc_s_fcat*. Pass the field catalog to the method *set_table_for_first_display*.

   Add an entry to the field catalog so that the column containing the occupied seats in Economy Class is totaled by default.

   The column containing the current booking totals is to be hidden. Extend the field catalog appropriately.

   The traffic light icon that graphically displays the utilization is to have a descriptive (and translatable) heading.

   a) See result at the end.

3. Add additional columns to the data table and display them in the ALV

   Expand the row type of the data table so that a further field with the name *changes_possible* of the *icon-id* type will be included.

   Fill this field for each data record as follows: If the flight date is in the past it should receive the *icon_space* value, in other cases, it should be *icon_okay.* So that the symbolic names of the icons are known, you need to insert the following syntax into your program:

   ```
   TYPE-POOLS: icon.
   ```

   Add the new column to the automatically generated field catalog as follows:

   Add a new entry to the field catalog so that the new column is output in fifth position, with the (translatable) heading *Changes Possible* and the quick info (tooltip) *Are changes still possible?*

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

a) See result at the end.

Result

Main program

```
REPORT  bc405_design_s3.

TYPE-POOLS: col, icon.

* work area for ALV data and color information
TYPES: BEGIN OF gty_flight.
         INCLUDE          TYPE sflight.
TYPES:   color            TYPE c LENGTH 4,
         light            TYPE c LENGTH 1, "booking status
         it_field_colors  TYPE lvc_t_scol, "for cell highlighting
         changes_possible TYPE icon-id,
       END OF gty_flight.

DATA: gt_flights TYPE TABLE OF gty_flight,
      gs_flight  TYPE          gty_flight.

DATA: ok_code LIKE sy-ucomm.

DATA: go_alv_grid  TYPE REF TO cl_gui_alv_grid,
      go_container TYPE REF TO cl_gui_custom_container.

DATA: gs_variant TYPE disvariant.

* data needed for layout
DATA: gs_layout      TYPE lvc_s_layo,
      gs_field_color TYPE lvc_s_scol.

* field catalog
DATA: gt_field_cat  TYPE lvc_t_fcat,
      gs_field_cat  TYPE lvc_s_fcat.

SELECT-OPTIONS: so_car FOR gs_flight-carrid,
                so_con FOR gs_flight-connid.
SELECTION-SCREEN SKIP.
PARAMETERS: pa_lv TYPE disvariant-variant.


***********************************************************************
*ABAP events
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
***********************************************************************
START-OF-SELECTION.
  SELECT * FROM sflight
    INTO CORRESPONDING FIELDS OF TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

  LOOP AT gt_flights INTO gs_flight.

* set indicator for flights of current month
    IF gs_flight-fldate(6) = sy-datum(6).
      gs_flight-color = 'C' && col_negative && '01'.
    ENDIF.

* set icon for bookings status
    IF gs_flight-seatsocc = 0.
      gs_flight-light = 1.
    ELSEIF gs_flight-seatsocc < 50.
      gs_flight-light = 2.
    ELSE.
      gs_flight-light = 3.
    ENDIF.

* highlight specific aircraft
    IF gs_flight-planetype = '747-400'.
      gs_field_color-fname = 'PLANETYPE'.
      gs_field_color-color-col = col_positive.
      gs_field_color-color-int = 1.
      gs_field_color-color-inv = 0.
      gs_field_color-nokeycol  = 'X'.
      APPEND gs_field_color TO gs_flight-it_field_colors.
    ENDIF.

* set indicator for flights in the past
    IF gs_flight-fldate < sy-datum.
      gs_flight-changes_possible = icon_space.
    ELSE.
      gs_flight-changes_possible = icon_okay.
    ENDIF.

    MODIFY gt_flights FROM gs_flight
                      TRANSPORTING color light
                                   it_field_colors
                                   changes_possible.
  ENDLOOP.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
  CALL SCREEN 100.


***********************************************************************
*PBO modules
***********************************************************************
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.                 " clear_ok_code  OUTPUT

*---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'DYN'.
  SET TITLEBAR 'T1'.
ENDMODULE.                 " STATUS_0100  OUTPUT

*---------------------------------------------------------------------*
MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    gs_variant-report = sy-cprog.
    IF NOT pa_lv IS INITIAL.
      gs_variant-variant = pa_lv.
    ENDIF.

* define layout
    gs_layout-grid_title = 'Flights'(h01).
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
    gs_layout-no_hgridln = 'X'.
    gs_layout-no_vgridln = 'X'.

* field that contains information on row color
    gs_layout-info_fname = 'COLOR'.

* internal table that contains information on cell color
    gs_layout-ctab_fname = 'IT_FIELD_COLORS'.

* field that contains information on exception (indicator)
    gs_layout-excp_fname = 'LIGHT'.

* multiple row and column selection
    gs_layout-sel_mode   = 'A'.

* fill field catalog
    gs_field_cat-fieldname = 'SEATSOCC'.
    gs_field_cat-do_sum    = 'X'.
    APPEND gs_field_cat TO gt_field_cats.

    CLEAR gs_field_cat.
    gs_field_cat-fieldname = 'PAYMENTSUM'.
    gs_field_cat-no_out    = 'X'.
    APPEND gs_field_cat TO gt_field_cats.

    CLEAR gs_field_cat.
    gs_field_cat-fieldname = 'LIGHT'.
    gs_field_cat-coltext   = 'Utilization'(h02).
    APPEND gs_field_cat TO gt_field_cats.

    CLEAR gs_field_cat.
    gs_field_cat-fieldname = 'CHANGES_POSSIBLE'.
    gs_field_cat-col_pos   = 5.
    gs_field_cat-coltext   = 'Changes possible'(h03).
    gs_field_cat-tooltip   = 'Are changes possible?'(t01).
    APPEND gs_field_cat TO gt_field_cats.

    go_alv_grid->set_table_for_first_display(
      EXPORTING
        i_structure_name = 'SFLIGHT'
        is_variant       = gs_variant
        i_save           = 'A'
        is_layout        = gs_layout
      CHANGING
        it_fieldcatalog  = gt_field_cats
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
        it_outtab        = gt_flights
      EXCEPTIONS
        OTHERS           = 1 ).
    IF sy-subrc <> 0.
      MESSAGE a012(bc405_408).
    ENDIF.

  ENDIF.
ENDMODULE.                 " create_and_transfer  OUTPUT


*********************************************************************
*PAI modules
*********************************************************************
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK' OR 'CANCEL' OR 'EXIT'.
      SET SCREEN 0.
  ENDCASE.
ENDMODULE.                 " USER_COMMAND_0100  INPUT
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Explain the purpose of the field catalog
- Add columns to the ALV Grid Control or change columns of the ALV

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Work with layout variants in an ALV Grid Control
- Create the layout structure of the ALV Grid Control
- Set colors of lines and cells in the ALV Grid Control
- Hide standard toolbar functions
- Explain the purpose of the field catalog
- Add columns to the ALV Grid Control or change columns of the ALV

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 6

ALV: Events and Methods

Unit Overview

The ALV raises events for a number of user interactions (for example, clicking in a table cell or on a toolbar element). The program making use of the ALV can react on these events if appropriate event handler methods have been registered on these events. In addition, the ALV framework offers methods to read or change ALV properties.

This unit explains how to create, implement, and register event handler methods and how to use context menus with the ALV. In addition, an overview of important ALV methods is given.

Unit Objectives

After completing this unit, you will be able to:

- Describe how events are triggered and how they can be handled
- List suitable events for a particular task
- Create and register an event handler method
- Add customer functions to the toolbar
- Define a handler method for print list events such as TOP-OF-PAGE
- Find suitable methods for a task
- Implement a method call in an ALV program
- Use context menus
- Create context menus for ALV Grid Controls
- Process functions from context menus

Unit Contents

Lesson: ALV Grid Control: Events.....187
Exercise 10: ALV Grid Control: Events I (Double-Click) 205
Exercise 11: ALV Grid Control: Events II (Page Headers in Print Lists).....211
Lesson: ALV Grid Control: Further Methods 219
Exercise 12: ALV Grid Control: Events III (Expanding the Toolbar and Evaluating Your Own Function Codes) 223

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson: ALV Grid Control: Context Menus.....233
Exercise 13: ALV Grid Control: Context Menus.....239

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV Grid Control: Events

Lesson Overview

Some user activities, such as a double-click or pressing a pushbutton, trigger events that must be handled appropriately.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how events are triggered and how they can be handled
- List suitable events for a particular task
- Create and register an event handler method
- Add customer functions to the toolbar
- Define a handler method for print list events such as TOP-OF-PAGE

Business Example

You are working as a programmer for ABC Travel. It is now your task to handle a double-click in your ALV so that detailed information (such as bookings) is displayed in a separate ALV Grid for the selected row.

Control Framework

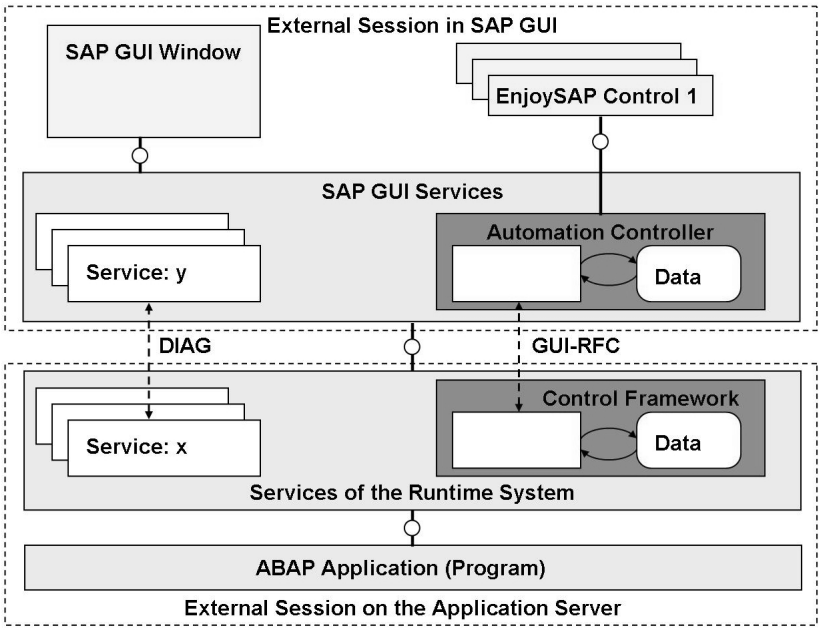


Figure 98: Control Framework Architecture: Revision

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The graphical user interface in the SAP R/3 System is based on SAP GUI windows (screens from the technical view of the programmer). Every user dialog in the system is implemented using dynpros of application programs.

The communication between the application program and the dynpro at runtime takes place using various services of the runtime system (ABAP processor and dynpro processor) and using services in the SAP GUI (managing the windows and their pop-up levels, data transfer, and so on).

Since *SAP R/3 4.5*, you can also use screen elements other than those provided in the Screen Painter in SAP GUI windows. These additional screen elements are known as controls. Controls are standalone binary software components that are reusable. As a developer, you can integrate one or more controls into your user dialogs and use the functions they make available.

The communication between the control and the application program is different from that of the "classic" screen elements. As of *SAP R/3 4.6A*, communication for all controls is performed in the same way using the Control Framework (CFW). The Control Framework uses special services in the SAP GUI and the runtime system on the application server (presentation server: Automation Controller; application server: CFW service).

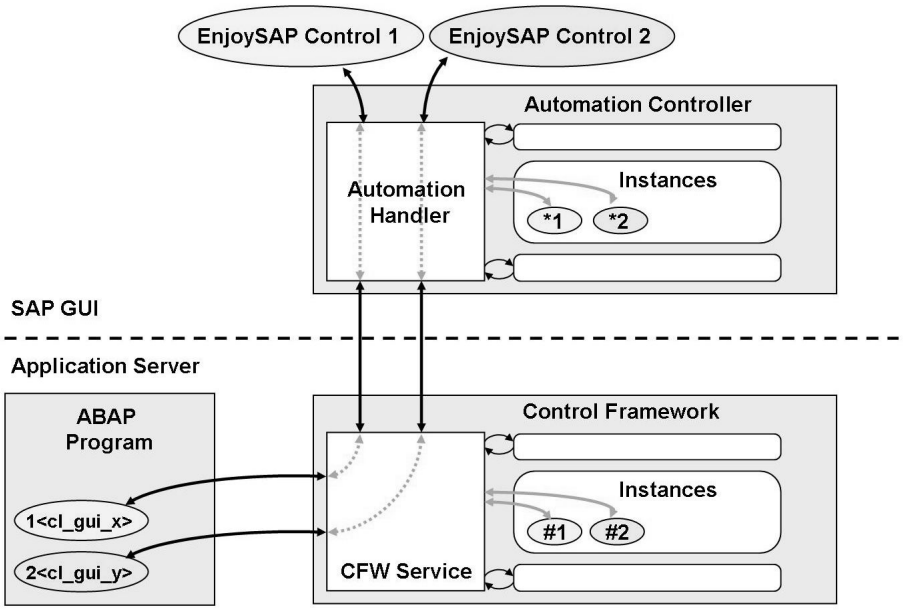


Figure 99: Controls and ABAP Objects

Every GUI Control with which an application program communicates represents an object (an instance of a class) on the presentation server in the sense of an object-oriented programming language. The communication with the instances therefore uses method calls and events of the objects.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

As of *SAP R/3 4.6*, there are representative classes (proxy classes) for all GUI Controls in the class library, which access the Control Framework in the same way and package the Control-specific communication. (These are referred to as Control wrappers.)

When you use a GUI control in an ABAP program, there are actually two objects involved: an instance in the ABAP program, and an instance of the control at the presentation server. The communication between the ABAP program and a GUI Control is performed as follows:

- The ABAP program always communicates only with the representative object (method calls, operations on the attributes). The representative object performs the associated operations on the presentation server instances using the CFW.
- Actions that come from the presentation server control are forwarded to the representative object using the CFW. The representative object then communicates with the ABAP program.

The CFW maintains a list of the active instances with which it is communicating both at the presentation server and on the application server.

Event Triggering and Event Handling

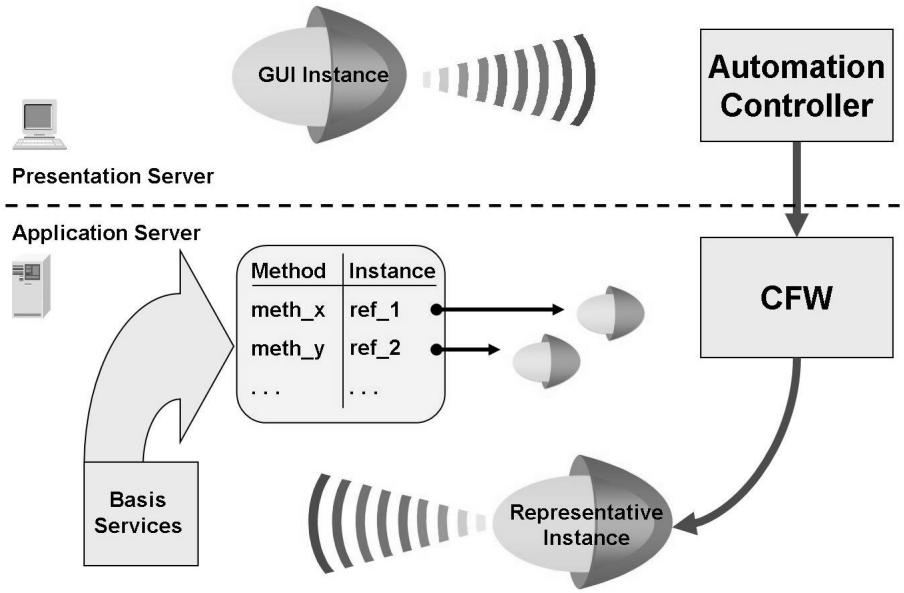


Figure 100: Control Events: Overview

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

When an event is passed to a presentation server control, a range of partner levels are involved:

- SAP GUI/Automation Controller: The event triggered by the presentation server control is received by the Automation Controller in the SAP GUI. The Automation Controller forwards the information about an event that it receives to the Basis Services of the runtime system as a special function code (and additional data).
- The Basis Services forward the information to the representative object in the ABAP program.
- The representative object in the ABAP program then triggers an event in the sense of ABAP Objects, thus informing the components registered for the event. These components then handle the event with an appropriate method (handler method).

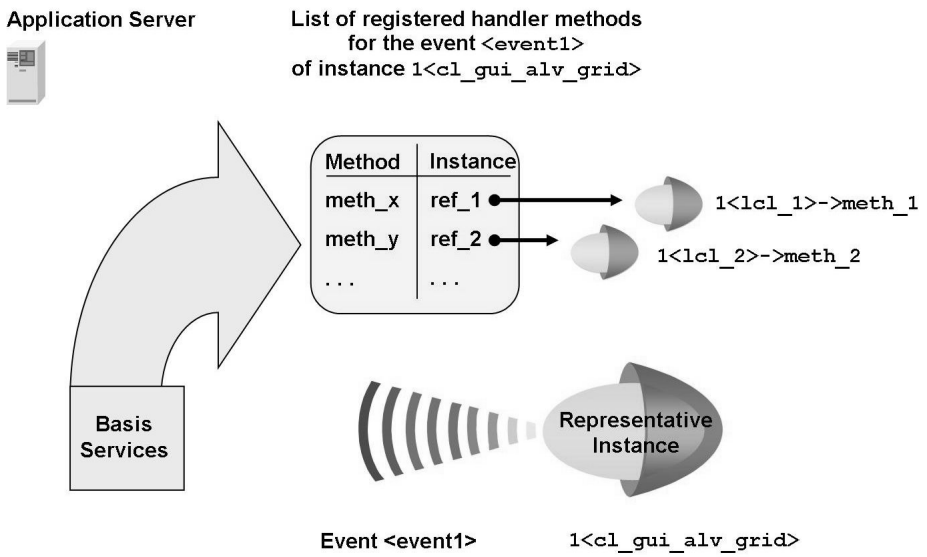


Figure 101: Processing ABAP Objects Events

An object can announce that its state has changed by triggering events.

Other objects can contain handler methods that are executed when the event occurs.

For the handler methods of the instances to be run, these instances need to be entered into a list of the object that can trigger the event.

At the point when the event is triggered, a Basis Service then ensures – on the basis of this list – that the registered methods are executed.

Unlike explicit method calls, where the caller has control and knows the method called, the trigger of the event does not “know” “who” will handle the event in this case. This applies both when the event is defined and when the event occurs at runtime.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

If you want to receive and handle events of a GUI Control in your program, you first need to find out what events the object can trigger. The associated class for representative objects of the control can provide you with this information.

For information about the representative classes, see the online documentation for the respective controls.

Finding Out Information About Events

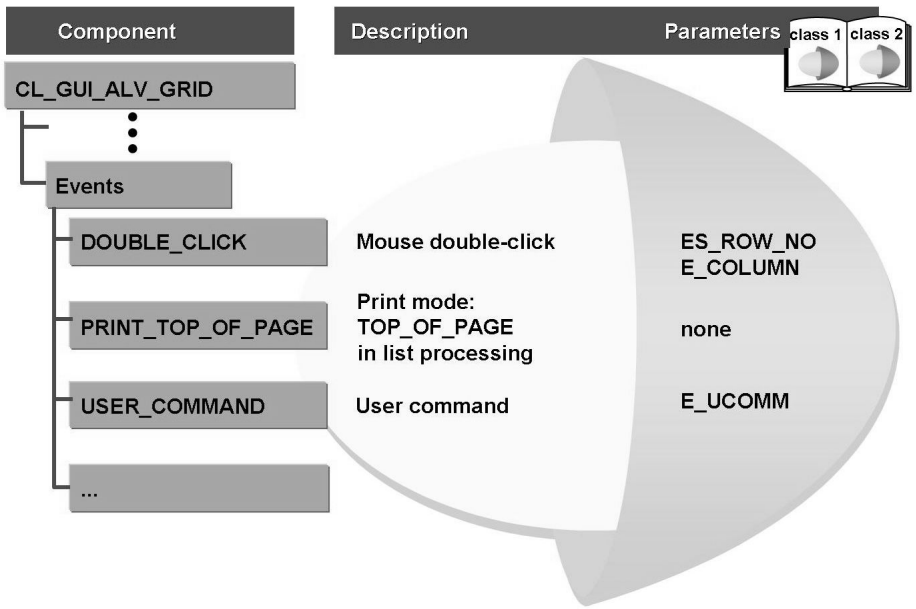


Figure 102: Events of the ALV Grid Control

The ALV Grid Control has a large number of events (more than 25).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

These are handled below as examples:

- **double_click**

  The event is triggered after a mouse double-click.

- **print_top_of_list, print_top_of_page, print_end_of_page, print_end_of_list**

  These events are triggered when outputting the print list.

- **toolbar**

  This event is triggered when the ALV Grid Control is created or explicitly with a method. It allows the changing of the toolbar.

- **user_command**

  This event is triggered at self-defined function codes (your own buttons in the toolbar or context menu). It allows you to implement your own functions.

For information about other events and more information about these events, see the online documentation.

Creating and Registering an Event Handler Method

```
CLASS <class_name> DEFINITION.
  [PUBLIC| PROTECTED| PRIVATE] SECTION.
    [CLASS-]METHODS:
       <method_name> FOR EVENT <event_name> OF <class_name>
           IMPORTING <p1> ... <pn> [SENDER],
       <method_name1> ...
ENDCLASS.
CLASS <class_name> IMPLEMENTATION.
  METHOD <method_name>.
    ...           " method implementation
  ENDMETHOD.
ENDCLASS.
```

Figure 103: Syntax: Handler Method for an Event

Any class can contain handler methods for events of other classes.

Event handler methods require the addition FOR EVENT **<event_name>** OF **<class_name>** in the definition part. This specifies that this method can react to events **<event_name** of instances of the class **<class_name>**.

The names of event handler methods should be structured as follows:

on_**<event_name>**, where **<event_name>** is the name of the event.

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

The interface of an event handler method can only contain formal parameters that were defined when the corresponding event was defined. The properties of the parameters are also taken from the event. However, the event handler method does not have to accept all of the parameters passed in the RAISE EVENT statement.

Events always have an implicit parameter called **sender**, which you can also receive by listing it in the IMPORTING addition. This parameter allows handlers of instance events to access the trigger.

A static handler method is declared with CLASS-METHODS ... . It can run without an instance of the class needing to be generated.

The definition of an event handler method in a class means that the instances of the class and the class itself are, in principle, capable of handling the event in question.

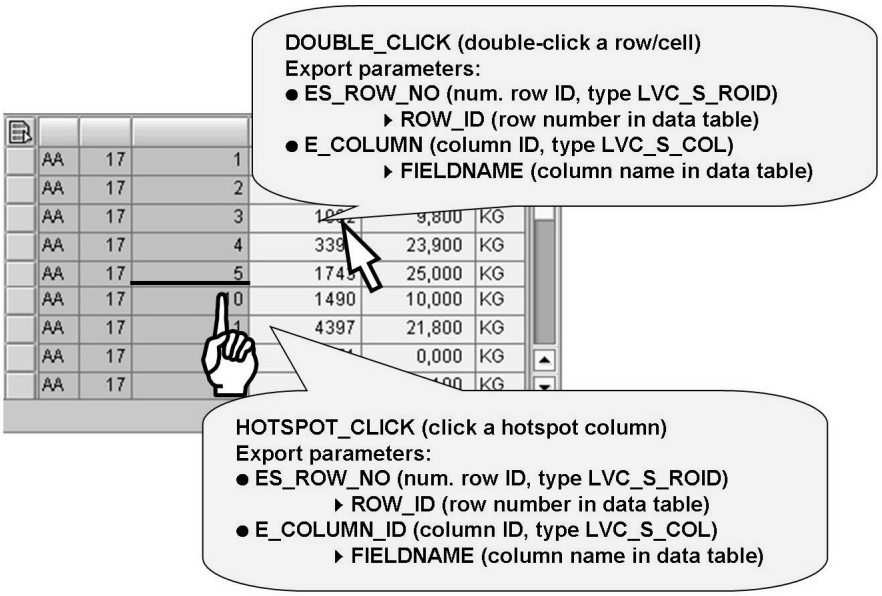


Figure 104: ALV Grid Control Events Through Mouse Operations

If the user “double-clicks” the data area, the event **double_click** is triggered in the calling program.

The export parameters contain information about the data table. Note that both of these are structures.

- **es_row_no:**

  You can find the number of the row of the internal table to which the “double-clicked” row in the data area belongs in the **row_id** field.

- **e_column:**

  You can find the name of the column of the internal table to which the “double-clicked” row in the data area belongs in the **fieldname** field.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

If a column has been identified as a hotspot, if a user clicks the mouse in this column of the data area, the event **hotspot_click** is triggered in the calling program. The export parameters are filled in the same way as for the **double_click** event.

For more detailed information about these or other mouse operation events, see the online documentation.

```
DATA: go_alv_grid TYPE REF TO cl_gui_alv_grid.
...
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: on_double_click
         FOR EVENT double_click
         OF cl_gui_alv_grid
         IMPORTING es_row_no e_column.
   ...
ENDCLASS.

CLASS lcl_event_handler IMPLEMENTATION .
  METHOD on_double_click.
    ...
    MESSAGE i016(bc408)
        WITH es_row_no-row_id
             e_column-fieldname.
  ENDMETHOD.
ENDCLASS.
```

cl_gui_alv_grid has event double_click

Static method on_double_click can react to the double_click event

Implementing the static method on_double_click

It is not necessary to instantiate an object.

Figure 105: Syntax Example: Handlers as Class Methods

A handler method can be either a class method (static method of a local class) or an instance method of an object.

We will first consider the class method. In this case, the local class can be compared with a repository in which the methods are stored. The significant difference to the instance method (see the next page) is that you do not need to instantiate an object for the local class to use the method.

To create a handler object for an event, first define a local class. This class has a public method (in the PUBLIC SECTION), to which an event can react. Note that you must use the CLASS-METHODS statement for this.

After you have defined the class, you need to implement it. You store the source code that is to be run when the event is triggered within the implementation of the handler method.

The method receives the information provided by the event about the mouse position at the double-click and generates an information message that displays the row and field of the mouse click.

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
DATA: go_alv_grid TYPE REF TO cl_gui_alv_grid.
  ...

CLASS lcl_event_handler DEFINITION .
  PUBLIC SECTION.
    METHODS on_double_click
            FOR EVENT double_click
            OF cl_gui_alv_grid ...
  ...
ENDCLASS .

CLASS lcl_event_handler IMPLEMENTATION.
  METHOD on_double_click.
   ...
  ENDMETHOD.
ENDCLASS .
...
DATA: go_handler TYPE REF TO lcl_event_handler.
   ...
CREATE OBJECT go_alv_grid.
CREATE OBJECT go_handler.
```

cl_gui_alv_grid has the event double_click

Instance method on_double_click can react to double_click

Implementation of the instance method on_double_click

Create a reference variable for the handler object

Instances of classes: cl_gui_alv_grid and lcl_event_handler

Figure 106: Syntax Example: Handlers as Instance Methods

To generate an instance method, use the METHODS statement when defining the class (as opposed to CLASS-METHODS for the class method).

Then implement the method in exactly the same way as the class method.

Afterwards, create a reference variable with reference to your local class.

Finally, generate a corresponding object.

```
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: on_double_click
      FOR EVENT double_click OF cl_gui_alv_grid
      ...
ENDCLASS.

CLASS lcl_event_handler IMPLEMENTATION.
  METHOD on_double_click.
    ...
  ENDMETHOD.
ENDCLASS.
   .
   .
   .
CREATE OBJECT go_alv_grid   ...

SET HANDLER lcl_event_handler=>on_double_click
            FOR go_alv_grid.
```

What is the reaction?

Who reacts to whose events, how, and as of when?

Figure 107: Linking the Representative Instance and the Handler Method

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

In order for an event handler method to react to a triggered event, you must define at runtime the trigger to which it should react.

The SET HANDLER statement links a list of handler methods with the event triggers.

In the above example, the static (cross-instance) method **on_double_click** of the class **lcl_event_handler** is to be called if the instance to which the object reference **alv** refers triggers the event **double_click**.

For further information about registering handler methods for events, refer to the keyword documentation for the `SET HANDLER` statement.

In the example shown here, a static method is used for the handling. If an instance method is to be used, the syntax is:

```
SET HANDLER lcl_event_handler_object->on_double_click
            FOR go_alv_grid.
```

Observe the component selector -> (for static methods =>). This statement connects the handler method to the ALV Grid Control.

Implementing Customer Functions

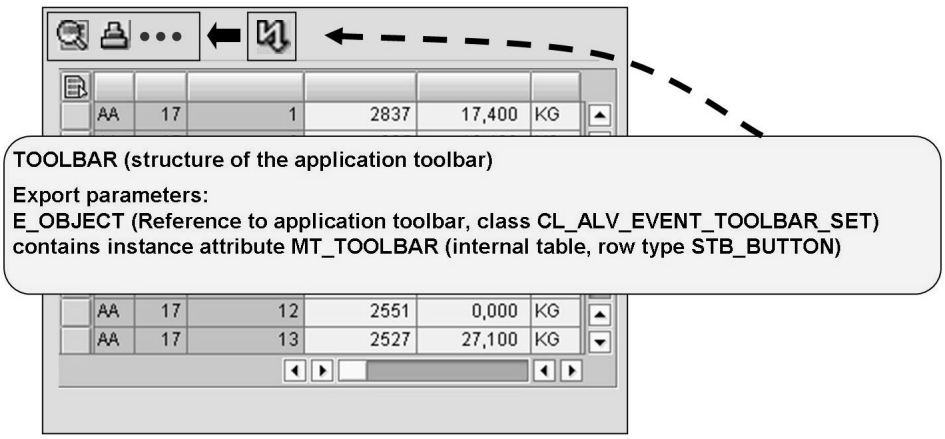


Figure 108: Event at the Creation of the Application Toolbar

Before the standard toolbar is integrated into the ALV Grid Control, the representative instance triggers the event **toolbar**, that is, during the execution of the methods **set_table_for_first_display** and **refresh_table_display**.

During the lifetime of the representative instance, you can prompt an additional triggering of this event by calling the method **set_toolbar_interactive**.

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

To add elements to the standard toolbar, you therefore need to implement a handler method for the event **toolbar** of the class **cl_gui_alv_grid**.

```
CLASS ... DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS on_toolbar
                  FOR EVENT toolbar
                  OF cl_gui_alv_grid
                  IMPORTING e_object.
ENDCLASS.
CLASS ... IMPLEMENTATION.
  METHOD on_toolbar.
    DATA ls_button TYPE stb_button.
    ls_button-function  =
    ls_button-icon      =
    ls_button-quickinfo =
    ls_button-butn_type =
    ls_button-disabled  =
    ls_button-text      =
    ls_button-checked   =
    INSERT ls_button
           INTO TABLE e_object->mt_toolbar.
  ENDMETHOD.
ENDCLASS.
...
SET HANDLER ...=>on_toolbar FOR go_alv_grid.
```

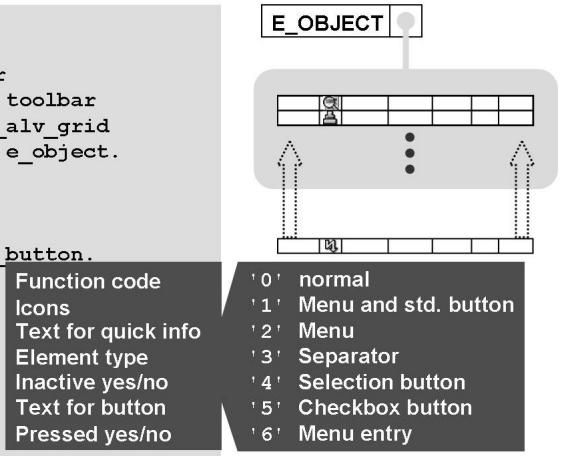


Figure 109: Extending the Standard Toolbar

The **toolbar** event has a pointer to an instance of the class **cl_alv_event_toolbar_set** as an export parameter. This instance of a toolbar contains, in turn, the public attribute **mt_toolbar**. This is an internal table with the row type **stb_button**. You therefore need to fill this internal table in the handler method to add additional buttons or other elements.

If the user chooses an element of the type “1” or “2”, the representative instance triggers the event **menu_button** with the parameter **e_object** of the class **cl_ctmenu**. Therefore, to add a menu to the toolbar, you also need to implement a handler method in which you create the menu in the same way as a context menu. You can differentiate between multiple menus by querying the function code.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
CLASS ... DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS on_user_command
                  FOR EVENT user_command
                  OF cl_gui_alv_grid
                  IMPORTING e_ucomm.
ENDCLASS.
CLASS ... IMPLEMENTATION.
  METHOD on_user_command.
    CASE e_ucomm.
      WHEN 'FC1'.
        ...
      WHEN 'FC2'.
        ...
    ENDCASE.
  ENDMETHOD.
ENDCLASS.
...
SET HANDLER ...=>on_user_command
  FOR go_alv_grid.
```

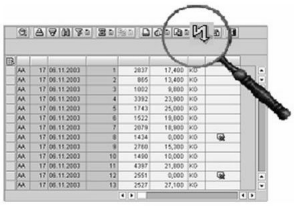


Figure 110: Handling a Function

If the user chooses a function element that has been added to the standard toolbar, the representative instance triggers the event **user_command**. This contains the function code as the export parameter **e_ucomm**.

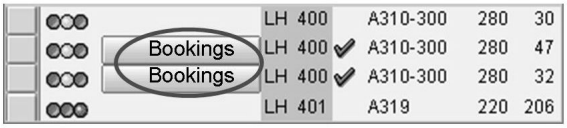
Therefore, to handle the additional function, you need to implement a handler method for this event. You can differentiate between the function elements by querying the function code.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Handling Button Clicks




```
TYPES: BEGIN OF gty_flight.
         INCLUDE   TYPE sflight.
TYPES:   btn_text  TYPE c LENGTH 10, "text displ. on button
         ct                TYPE lvc_t_styl, for buttons
       END OF gty_flight.
DATA: gt_flights TYPE TABLE OF gty_flight,
      gs_flight  TYPE          gty_flight,
      gs_ct      TYPE          lvc_s_styl.  "help structure

gs_ct-fieldname = 'BTN_TEXT'.
gs_ct-style     = cl_gui_alv_grid=>mc_style_button.
APPEND gs_ct TO gs_flight-ct.

gs_flight-btn_text = 'Bookings'(t11).
...
gs_layout-stylefname = 'CT'.
```

Figure 111: Layout: Buttons in the List Line

There are two ways to display a field in the ALV Grid as a button:

- Either this attribute is specified for an entire column in the field catalog (using the parameter **style**).
- Or you specify it for individual fields. To do this, add an internal table of the type **lvc_t_styl** to the data structure, in which you define for each data record which fields of the data record are to be displayed as a button.

In the above example, a separate field (**btn_text**) has been defined for the structure **gs_flight**, in which a button is displayed or is not displayed, depending on other data. (You can, however, also use existing fields of the data structure as long as they do not have F4 help.)

An entry for each field that is represented as a button is created in the internal table of type **lvc_t_styl** (in this example: **ct**). The name of the output field is specified with **fieldname**, the class constant **cl_gui_alv_grid -> mc_style_button** is specified with **style**.

Finally, the name of the internal table of the type **lvc_t_styl** that contains the information about the buttons is specified in the layout structure.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* define method
CLASS: lcl_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: on_button_click
                   FOR EVENT button_click OF cl_gui_alv_grid
                   IMPORTING es_row_no.
ENDCLASS.
* implement method
CLASS: lcl_handler IMPLEMENTATION.
  METHOD on_button_click.
    READ TABLE gt_flights INTO gs_flight
      INDEX es_row_no-row_id.
     ...
  ENDMETHOD.
ENDCLASS.

* register method as handler
SET HANDLER lcl_handler=>on_button_click FOR go_alv_grid.

CALL METHOD go_alv_grid->set_table_for_first_display ...
```

Figure 112: Handling the button_click Event

If a user presses a button in a cell, this triggers the event **button_click**. The event returns the row and column of the button click. An associated handler method could, for example, implement a context-dependent display of detailed as a reaction.

Handling Print List Events

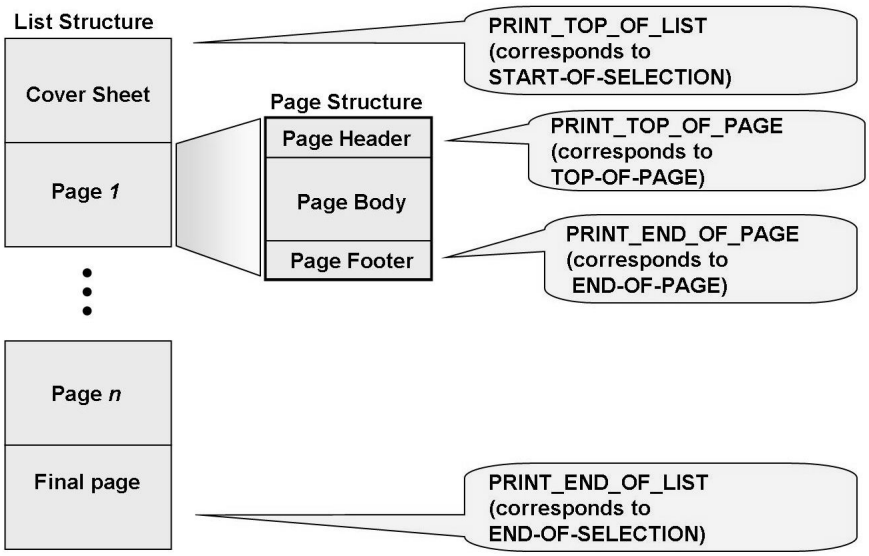


Figure 113: Print List Events

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The print list automatically generated by the representative instance can be extended by the calling program.

Use the above events to do this. By using these events, you can implement the same logical process steps as are familiar from the “classic” ABAP list layout.

Thus, in the event handler methods to be implemented, you can use the usual statements for ABAP list output as well as their many additions **`(write, skip, uline`**, among others).

You can also use the **subtotal_text** event to define additional texts for subtotals across columns whose output has been suppressed. For more information, see the online documentation.

Procedure:

1. **“Give approval” to set the behavior at control level change in the print list**

```
DATA: gs_print TYPE lvc_s_prnt.
...
gs_print-grpchgedit = 'X'.
go_alv_grid->set_table_for_first_display(
  EXPORTING
    ...
    is_print         = gs_print
...).
```

2. **Interactive sorting in ALV Grid and / or print list**

Figure 114: Print List: Page Break for Group Change (1)

To have a page break at a control level change in the print list, you first need to set an indicator in the print structure that the user can format the control level change in the print list. This is controlled by the field **grpchgedit** of the transfer parameter **is_print** (structure of type **lvc_s_prnt**). This is the programmer's task.

The program user then sorts in the ALV Grid or in the print list by the fields for which he or she wants group processing.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

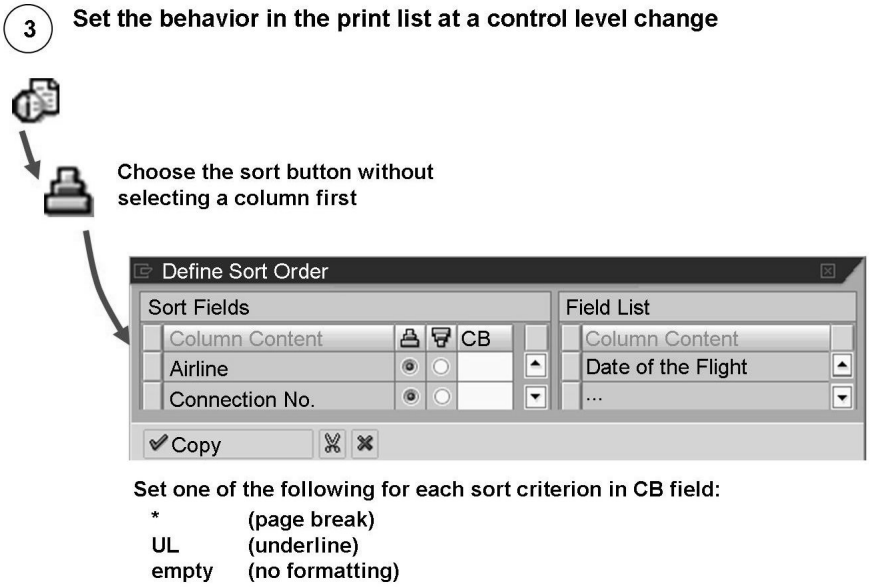


Figure 115: Print List: Page Break for Group Change (2)

In the print list (list display), the user calls the “Define Sort Order” dialog box. He or she does this by choosing the Sort button without first selecting a column.

In the dialog box, the user can then set the following in the *CB* field:

*	Page break
UL	Underlined
empty	No formatting

The solution presented only works in interactive mode, not in background processing. For background processing, you need to specify the settings for sort order and for control level formatting in the parameter **it_sort** of the method **set_table_for_first_display**. To do this, assign * or UL to the field GROUP in **it_sort**.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

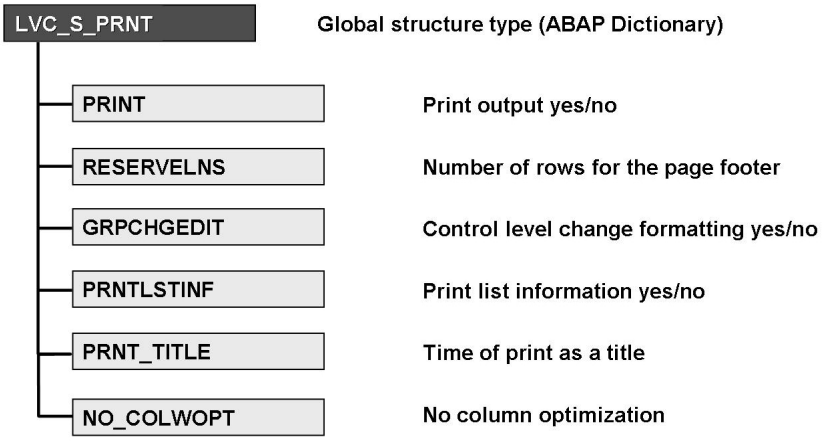


Figure 116: Display Information for the Print List

You can make settings for the print list using the fields **print, reservelns, grpchgedit**, and **prntlstinf** of a structure of the global type **lvc_s_prnt**. To do this, assign a structure of this type to the parameter **is_print** of the method **set_table_for_first_display**.

If you assign the value "X" to the **print** field, you can use the ALV Grid Control to print without first displaying the list on screen: The passed data table is formatted at once as an ABAP print list and sent to the print spool. Interactive screen output is not processed.

You can use the field **reservelns** to define the number of rows required for the page footer.

If you assign the value "X" to the field **prntlstinf**, information about the sort order, totals, and defined filters are printed, together with data statistics, at the start of the list.

prnt_title: Specifies the time at which the grid title is to be printed (0/default: before the event **print_top_of_list;** 1: after the event **print_top_of_list;** 2: before the event **print_top_of_page;** 3: after the event **print_top_of_page).**

no_colwopt: If "X", no column width optimization occurs.

The other fields of the structure type are **not** used for the ALV Grid Control.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Exercise 10: ALV Grid Control: Events I (Double-Click)

Exercise Objectives

After completing this exercise, you will be able to:

- Handle the "mouse double-click" event on a line of the ALV Grid Control, for example, recognize and react to each individually.
- Implement handler methods for ALV Grid Control events
- Register handler methods

Business Example

When double-clicking a row of the ALV, the users of your program would like to receive an I message with the total number of bookings for this row (this flight). (This information is not present in the ALV. Only the total number of bookings per class is available, for example, first, business and economy.)

Task:

Expand the program in such a way that by double-clicking a row, the total number of bookings for this row is output with an information message.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_EVENT_S1*

1. Expand your program from the last exercise or copy the model solution of the last exercise (*BC405_DESIGN_S3*).
2. Define and implement an event handler for the *double_click* event of the ALV Grid Control. To do this, proceed as follows:

   Familiarize yourself with the interface of the *double_click* event of the class of the ALV Grid Control.

   Define a local class (suggested name: *lcl_event_handler*) which contains a static method for the evening handling of the *double_click* event for instances of the *cl_gui_alv_grid* class (suggested name: *on_double_click*).

   Implement the method *on_double_click* in the implementation part of the local class. The method should contain the following logical steps:

Continued on next page

 © 2011 SAP AG. All rights reserved.

No.	System response
1.	Find out which data record was selected
2.	Calculate the total number of bookings
3.	Display an information message

Proceed as follows:

Read the connection data record selected from the internal table *it_sflight* using the `READ TABLE ... INDEX ...` statement. (If an error occurs, you can use dialog message 075 of the message class *BC405_408*. Display an information message for the user and end the handler method.)

Add together the bookings from all three booking classes (First, Business, and Economy) for the selected flight. Output the total number of bookings with an I message (syntax variant `MESSAGE <text> TYPE 'I'`).

Register your event handler method for the ALV Grid Control instance for the basic list.

Activate and test your program.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Solution 10: ALV Grid Control: Events I (Double-Click)

Task:

Expand the program in such a way that by double-clicking a row, the total number of bookings for this row is output with an information message.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_EVENT_S1*

1. Expand your program from the last exercise or copy the model solution of the last exercise (*BC405_DESIGN_S3*).

   a) Self-explanatory

2. Define and implement an event handler for the *double_click* event of the ALV Grid Control. To do this, proceed as follows:

   Familiarize yourself with the interface of the *double_click* event of the class of the ALV Grid Control.

   Define a local class (suggested name: *lcl_event_handler*) which contains a static method for the evening handling of the *double_click* event for instances of the *cl_gui_alv_grid* class (suggested name: *on_double_click*).

   Implement the method *on_double_click* in the implementation part of the local class. The method should contain the following logical steps:

No.	System response
1.	Find out which data record was selected
2.	Calculate the total number of bookings
3.	Display an information message

   Proceed as follows:

   Read the connection data record selected from the internal table *it_sflight* using the `READ TABLE ... INDEX ...` statement. (If an error occurs, you can use dialog message 075 of the message class *BC405_408*. Display an information message for the user and end the handler method.)

   Add together the bookings from all three booking classes (First, Business, and Economy) for the selected flight. Output the total number of bookings with an I message (syntax variant `MESSAGE <text> TYPE 'I'`).

   Register your event handler method for the ALV Grid Control instance for the basic list.

Continued on next page

 © 2011 SAP AG. All rights reserved.

Activate and test your program.

a) See result at the end.

Result

Main program

```
REPORT  bc405_event_s1.
...

***********************************************************************
*Local class for ALV event handling
***********************************************************************
CLASS: lcl_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: on_doubleclick FOR EVENT double_click
                                  OF cl_gui_alv_grid
                                  IMPORTING es_row_no.
ENDCLASS.                    "lcl_handler DEFINITION

*---------------------------------------------------------------------*
CLASS: lcl_handler IMPLEMENTATION.
  METHOD on_doubleclick.
    DATA: lv_bookings_total   TYPE i,
          lv_bookings_total_c TYPE c LENGTH 10,
          lv_message_text     TYPE c LENGTH 60.

    READ TABLE gt_flights INTO gs_flight INDEX es_row_no-row_id.
    IF sy-subrc NE 0.
      MESSAGE i075(bc405_408).
      EXIT.
    ENDIF.

    lv_bookings_total = gs_flight-seatsocc +
                        gs_flight-seatsocc_b +
                        gs_flight-seatsocc_f.

* built message text
    lv_bookings_total_c = lv_bookings_total.
    lv_message_text = 'Total number of bookings:'(m01) &&
                      lv_bookings_total_c

    MESSAGE lv_message_text TYPE 'I'.
  ENDMETHOD.                     "on_doubleclick
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
ENDCLASS.                    "lcl_handler IMPLEMENTATION


***********************************************************************
*ABAP events
***********************************************************************
*----------------------------------------------------------------------*
START-OF-SELECTION.
...
  CALL SCREEN 100.

***********************************************************************
*PBO modules
***********************************************************************
...
*----------------------------------------------------------------------*
MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <>.
      MESSAGE a010(bc405_408).
    ENDIF.

* register event handler
    SET HANDLER lcl_handler=>on_doubleclick  FOR go_alv_grid.

    gs_variant-report = sy-cprog.
    IF NOT pa_lv IS INITIAL.
      gs_variant-variant = pa_lv.
    ENDIF.
...
ENDMODULE.                 " create_and_transfer OUTPUT
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
***********************************************************************
*PAI modules
***********************************************************************
...
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 11: ALV Grid Control: Events II (Page Headers in Print Lists)

Exercise Objectives

After completing this exercise, you will be able to:

- Handle the "page break in print list" event in order to output page headers.

Business Example

Your users would like a special header to be output for the beginning of each page in the print list.

Task:

Output a special page header at the beginning of each page in the print list.

Program *ZBC405_##_ALV*.

Model Solution: *BC405_EVENT_S2*

1. Expand your program from the last exercise or copy the model solution of the last task (*BC405_EVENT_S1*).
2. For each page, output a page header with the following data:

   "Flight data" as fixed text, page number and current date. The entire text should be displayed in the *col_heading* color. Output a dotted line for the following rows.

   Define a static method to handle the *print_top_of_page* event (suggested name: *on_print_top*).

   Implement the method.

   Output the fields listed above using the `WRITE` statement.

   Register the methods in the ALV Grid Control for the basic list.
3. Allow the users to interactively set up a page break in the print list for each control level change.

   Create a suitable data object which can supply the *is_print* parameter when you call the *set_table_for_first_display* method. Provide a suitable value for the *grpchgedit* field of the data object.
4. Maintain the *prntlstinf* field for the *is_print* parameter so that information regarding sort, subtotals, defined filters and a data statistic are printed at the beginning of the list. (The correct value is "X".)

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

5. **Optional:** Output the airlines and connections on the print list for which data was retrieved. To do this, use the *print_top_of_list* event and output the contents of the selection options.

   Proceed as follows:

   Define a static method to handle the *print_top_of_list* event (suggested name: *on_print_tol*).

   Implement the method: Program a loop using both internal tables which contain the selections. Output all fields of the tables using the *WRITE* statement.

   Register the methods in the ALV Grid Control.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 11: ALV Grid Control: Events II (Page Headers in Print Lists)

Task:

Output a special page header at the beginning of each page in the print list.

Program *ZBC405_##_ALV*.

Model Solution: *BC405_EVENT_S2*

1. Expand your program from the last exercise or copy the model solution of the last task (*BC405_EVENT_S1*).

   a) Self-explanatory

2. For each page, output a page header with the following data:

   "Flight data" as fixed text, page number and current date. The entire text should be displayed in the *col_heading* color. Output a dotted line for the following rows.

   Define a static method to handle the *print_top_of_page* event (suggested name: *on_print_top*).

   Implement the method.

   Output the fields listed above using the `WRITE` statement.

   Register the methods in the ALV Grid Control for the basic list.

   a) See result at the end.

3. Allow the users to interactively set up a page break in the print list for each control level change.

   Create a suitable data object which can supply the *is_print* parameter when you call the *set_table_for_first_display* method. Provide a suitable value for the *grpchgedit* field of the data object.

   a) See result at the end.

4. Maintain the *prntlstinf* field for the *is_print* parameter so that information regarding sort, subtotals, defined filters and a data statistic are printed at the beginning of the list. (The correct value is "X".)

   a) See result at the end.

5. **Optional:** Output the airlines and connections on the print list for which data was retrieved. To do this, use the *print_top_of_list* event and output the contents of the selection options.

   Proceed as follows:

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Define a static method to handle the *print_top_of_list* event (suggested name: *on_print_tol*).

Implement the method: Program a loop using both internal tables which contain the selections. Output all fields of the tables using the *WRITE* statement.

Register the methods in the ALV Grid Control.

a) See result at the end.

Result

Main program

```
REPORT bc405_event_s2.
...
DATA: go_alv_grid  TYPE REF TO cl_gui_alv_grid,
      go_container TYPE REF TO cl_gui_custom_container.

DATA: gs_variant TYPE disvariant,
      gs_print   TYPE lvc_s_prnt.

* data needed for layout
DATA: gs_layout      TYPE lvc_s_layo,
      gs_field_color TYPE lvc_s_scol.

...
***********************************************************************
*Local class for ALV event handling
***********************************************************************
CLASS: lcl_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: on_doubleclick FOR EVENT double_click
                                  OF cl_gui_alv_grid
                                  IMPORTING es_row_no,
                   print_top FOR EVENT print_top_of_page
                             OF cl_gui_alv_grid,
                   print_tol FOR EVENT print_top_of_list
                             OF cl_gui_alv_grid.

ENDCLASS.                    "lcl_handler DEFINITION

*---------------------------------------------------------------------*
CLASS: lcl_handler IMPLEMENTATION.
  METHOD on_doubleclick.
...
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
ENDMETHOD.                    "on_doubleclick

METHOD print_top.
  DATA lv_pos TYPE i.
  FORMAT COLOR COL_HEADING.
  WRITE: / sy-datum.
  lv_pos = sy-linsz / 2 - 3. " length of pagno: 6 chars
  WRITE AT lv_pos sy-pagno.
  lv_pos = sy-linsz - 11. " length of username: 12 chars
  WRITE: AT lv_pos sy-uname.
  ULINE.
ENDMETHOD.                    "print_top

METHOD print_tol.
  DATA: ls_so_car LIKE LINE OF so_car,
        ls_so_con LIKE LINE OF so_con.
  CONSTANTS: lc_end TYPE i VALUE 20.

  FORMAT COLOR COL_HEADING.
  WRITE: / 'Select options'(000), AT lc_end space.
  SKIP.
  WRITE: / 'Airlines'(001), AT lc_end space.
  ULINE AT /(lc_end).
  FORMAT COLOR COL_NORMAL.
  LOOP AT so_car INTO ls_so_car.
    WRITE: / ls_so_car-sign,
             ls_so_car-option,
             ls_so_car-low,
             ls_so_car-high.
  ENDLOOP.

  SKIP.

  FORMAT COLOR COL_HEADING.
  WRITE: / 'Connections'(002), AT lc_end space .
  ULINE AT /(lc_end).
  FORMAT COLOR COL_NORMAL.
  LOOP AT so_con INTO ls_so_con.
    WRITE: / ls_so_con-sign,
             ls_so_con-option,
             ls_so_con-low NO-ZERO,
             ls_so_con-high NO-ZERO.
  ENDLOOP.

  SKIP.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
  ENDMETHOD.                    "print_tolENDCLASS.

ENDCLASS.                    "lcl_handler IMPLEMENTATION


***********************************************************************
*ABAP events
***********************************************************************
*---------------------------------------------------------------------*
START-OF-SELECTION.
...
  CALL SCREEN 100.

***********************************************************************
*PBO modules
***********************************************************************
...
*---------------------------------------------------------------------*
MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

* register event handlers
    SET HANDLER:
      lcl_handler=>on_doubleclick  FOR go_alv_grid,
      lcl_handler=>print_top       FOR go_alv_grid,
      lcl_handler=>print_tol       FOR go_alv_grid.

    gs_variant-report = sy-cprog.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
    IF NOT pa_lv IS INITIAL.
      gs_variant-variant = pa_lv.
    ENDIF.

    gs_print-prntlstinf = 'X'.
    gs_print-grpchgedit = 'X'.

* define layout
    gs_layout-grid_title = 'Flights'(h01).
    gs_layout-no_hgridln = 'X'.
    gs_layout-no_vgridln = 'X'.
...

    go_alv_grid->set_table_for_first_display(
      EXPORTING
        i_structure_name = 'SFLIGHT'
        is_variant       = gs_variant
        i_save           = 'A'
        is_layout        = gs_layout
        is_print         = gs_print
      CHANGING
        it_fieldcatalog  = gt_field_cats
        it_outtab        = gt_flights
      EXCEPTIONS
        OTHERS           = 1 ).
    IF sy-subrc <> 0.
      MESSAGE a012(bc405_408).
    ENDIF.

  ENDIF.
ENDMODULE.                 " create_and_transfer OUTPUT


*********************************************************************
*PAI modules
*********************************************************************
...
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Describe how events are triggered and how they can be handled
- List suitable events for a particular task
- Create and register an event handler method
- Add customer functions to the toolbar
- Define a handler method for print list events such as TOP-OF-PAGE

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV Grid Control: Further Methods

Lesson Overview

ALV Grid Control: Methods

Lesson Objectives

After completing this lesson, you will be able to:

- Find suitable methods for a task
- Implement a method call in an ALV program

Business Example

You are a programmer at ABC Travel. Your manager wants to know which methods are available for the ALV programming.

Finding Suitable Methods for a Task

Class Builder: Class `cl_gui_alv_grid`

- *Methods* tab page:
  - Parameters
  - Exceptions
  - Source code
- Class documentation: Link to SAP Library, *ALV Grid Control* section

SAP Library: *ALV Grid Control* section

ALV Grid Control → Methods of the Class `cl_gui_alv_grid`:

- Documentation
- Use
- Parameters

Figure 117: Information about methods

Several Selected Methods

Several Selected Methods

Method	Description
get_current_cell	Get current cell
get_selected_cells	Get selected cells

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Method	Description
get_selected_columns	Get selected columns
get_selected_rows	Get selected rows
get_subtotals	Get current subtotals of the ALV Grid Control
set_toolbar_interactive	Set toolbar status
set_user_command	Set user command

The **constructor, set_table_for_first_display** and **refresh_table_display** methods have already been discussed.

The **set_toolbar_interactive** method triggers the **toolbar** event. If you want to expand the toolbar to include your own functions, use this method to construct the toolbar again.

The **set_user_command** method gives you the opportunity to replace standard functions of the ALV Grid Control with functions of your own. To do this, you must query the current function code during the **before_user_command** event and change it using **set_user_command**.

Basic Information for Working with Methods

- Determine for yourself which methods are available for solving a problem.
- Become familiar with the interface of the selected method.
- Add the method call using drag & drop or using “ABAP Objects Pattern”.

Implementing Method Calls

Problem:

To implement a self-defined function, it can be necessary to determine the current cell (cursor).

Solution: Method `get_current_cell`

```
METHOD on_user_command.
   DATA: lv_row TYPE i,
         lv_col TYPE i.

   CASE e_ucomm.
     WHEN 'FC1'.
*      implementing user-defined function

        go_alv_grid->get_current_cell(
          IMPORTING
            e_row = lv_row
            e_col = lv_col ).
...
```

Figure 118: Programming example: Current cell

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Problem:

A standard function of the ALV Grid is to be replaced by a user-defined function.

Solution: Method `set_user_command`

```
* class implementation
  METHOD on_before_user_command.
    CASE e_ucomm.
      WHEN cl_gui_alv_grid=>mc_fc_detail.
        go_alv_grid->set_user_command(
          EXPORTING
            i_ucomm = 'DIS_BOOK').

    ENDCASE.
  ENDMETHOD.          "on_before_user_command

  METHOD on_user_command.
    CASE e_ucomm.
      WHEN 'DIS_BOOK'.
        ...
```

Figure 119: Programming example: Replacing standard function

The **on_before_user_command** handler method handles the **before_user_command** event. This is triggered before a user command (standard command or custom command) is processed.

Here it is possible to redefine a standard command, for example, if the detailed view is used to display dependent data.

In the handler method, check whether the function code for the detailed view was entered (WHEN cl_gui_alv_grid->mc_fc_detail). If this is the case, set the function code of your self-defined command (in this case, DIS_BOOK). Essentially, the function code is replaced.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 12: ALV Grid Control: Events III (Expanding the Toolbar and Evaluating Your Own Function Codes)

Exercise Objectives

After completing this exercise, you will be able to:

- Expand the toolbar of the ALV Grid Control to include your own buttons
- Evaluate additional function codes

Business Example

Your users want you to expand the toolbar to include special additional functions.

Task 1:

In your ALV program, expand the toolbar of the ALV to include customer-specific additional functions.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_EVENT_S3*

1. Expand your program from the last exercise or copy the model solution of the last task *(BC405_EVENT_S2)*.
2. Expand the application toolbar of the ALV Grid Control to include two self-defined functions, *Economy Utilization (Percent)* and *Economy Utilization for Marked Flights (Percent)*.

   Follow the instructions below:

   Define a static method to handle the *toolbar* event (suggested name: *on_toolbar*). Become familiar with the export parameters of this event.

   You need a work area for the function attributes. Define this locally in the method (we suggest the name: *ls_button*).

   Valuate the fields with your own function codes, texts, quick info texts, and key types.

   Default values (for both functions)

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Function	Economy Utilization for All Flights	Economy Utilization for Marked Flights
function	PERCENTAGE	PERCENT-AGE_MARKED
text	% total	% marked
quickinfo	Total Economy Utilization	Economy Util. (marked flights)
butn_type	0	0

Insert the filled work area for the new pushbutton in the instance attribute for the application toolbar (*alv->mt_toolbar*).

In order to set the new buttons apart from the rest in the application toolbar, you can insert a separator. To do this, you have to add an instance attribute before the actual button in the *ls_button* work area. Fill the *butn_type* field (value: 3), all other fields remain initial.

Register the methods in the ALV Grid Control.

Activate and test your program.

Task 2:

Handler for the First Pushbutton

1. Implement the handler method for the customer pushbuttons.

   To do this, proceed as follows:

   Define a static method to handle the *user_command* event (suggested name: *on_user_command*). Import the function code of the pushbutton using the *e_ucomm* parameter.

   Implement the method if the economy class utilization is to be displayed (function code PERCENTAGE): Determine the percentage of economy seat utilization via the internal table of the flight data using a loop. Observe the *SEATSOCC* and *SEATSMAX* fields. Display the utilization percentage in an information message.

   Register the methods in the ALV Grid Control.

2. Activate and test your program.

Task 3:

Handler for the Second Pushbutton

1. Expand the handler method for the customer pushbuttons.

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

In the *on_user_command* method, call the *GET_SELECTED_ROWS* method to determine which rows are selected. If no rows are selected, display a corresponding information message.

Proceed as with the calculation of the total utilization, but take only the selected flights into account.

2. Activate and test your program.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 12: ALV Grid Control: Events III (Expanding the Toolbar and Evaluating Your Own Function Codes)

Task 1:

In your ALV program, expand the toolbar of the ALV to include customer-specific additional functions.

Program: *ZBC405_##_ALV*

Model Solution: *BC405_EVENT_S3*

1. Expand your program from the last exercise or copy the model solution of the last task *(BC405_EVENT_S2)*.

   a) Proceed in the usual manner.

2. Expand the application toolbar of the ALV Grid Control to include two self-defined functions, *Economy Utilization (Percent)* and *Economy Utilization for Marked Flights (Percent)*.

   Follow the instructions below:

   Define a static method to handle the *toolbar* event (suggested name: *on_toolbar*). Become familiar with the export parameters of this event.

   You need a work area for the function attributes. Define this locally in the method (we suggest the name: *ls_button*).

   Valuate the fields with your own function codes, texts, quick info texts, and key types.

   Default values (for both functions)

Function	**Economy Utilization for All Flights**	**Economy Utilization for Marked Flights**
function	PERCENTAGE	PERCENT-AGE_MARKED
text	% total	% marked
quickinfo	Total Economy Utilization	Economy Util. (marked flights)
butn_type	0	0

   Insert the filled work area for the new pushbutton in the instance attribute for the application toolbar (*alv->mt_toolbar*).

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

In order to set the new buttons apart from the rest in the application toolbar, you can insert a separator. To do this, you have to add an instance attribute before the actual button in the *ls_button* work area. Fill the *butn_type* field (value: 3), all other fields remain initial.

Register the methods in the ALV Grid Control.

Activate and test your program.

a) See result at the end.

Task 2:

Handler for the First Pushbutton

1. Implement the handler method for the customer pushbuttons.

   To do this, proceed as follows:

   Define a static method to handle the *user_command* event (suggested name: *on_user_command*). Import the function code of the pushbutton using the *e_ucomm* parameter.

   Implement the method if the economy class utilization is to be displayed (function code PERCENTAGE): Determine the percentage of economy seat utilization via the internal table of the flight data using a loop. Observe the *SEATSOCC* and *SEATSMAX* fields. Display the utilization percentage in an information message.

   Register the methods in the ALV Grid Control.

   a) See result at the end.

2. Activate and test your program.

   a) Press **`Ctrl + F3`** and **`F8`**.

Task 3:

Handler for the Second Pushbutton

1. Expand the handler method for the customer pushbuttons.

   In the *on_user_command* method, call the *GET_SELECTED_ROWS* method to determine which rows are selected. If no rows are selected, display a corresponding information message.

   Proceed as with the calculation of the total utilization, but take only the selected flights into account.

   a) See result at the end.

2. Activate and test your program.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

a) Press **Ctrl + F3** and **F8**.

Result

Main program

```
REPORT bc405_event_s3.

...

**********************************************************************
*Local class for ALV event handling
**********************************************************************
CLASS: lcl_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: on_doubleclick FOR EVENT double_click
                                  OF cl_gui_alv_grid
                                  IMPORTING es_row_no,
                   print_top FOR EVENT print_top_of_page
                             OF cl_gui_alv_grid,
                   print_tol FOR EVENT print_top_of_list
                             OF cl_gui_alv_grid,
                   on_toolbar FOR EVENT toolbar
                              OF cl_gui_alv_grid
                              IMPORTING e_object,
                   on_user_command FOR EVENT user_command
                                   OF cl_gui_alv_grid
                                   IMPORTING e_ucomm.
ENDCLASS.                    "lcl_handler DEFINITION

*---------------------------------------------------------------------*
CLASS: lcl_handler IMPLEMENTATION.

...

**********************************************************************
*       Expands toolbar                                              *
**********************************************************************
  METHOD on_toolbar.
    DATA ls_button TYPE stb_button.

*   Separator
    ls_button-butn_type = 3.
    INSERT ls_button INTO TABLE e_object->mt_toolbar.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
*   Create button "total percentage of occupied seats"
    CLEAR ls_button.
    ls_button-function  = 'PERCENTAGE'.
    ls_button-text      = '% (total)'(tot).
    ls_button-quickinfo = 'Occupation (total)'(z01).
    ls_button-butn_type = 0.
    INSERT ls_button INTO TABLE e_object->mt_toolbar.

*   Create button "percentage of occupied seats for marked flights"
    CLEAR ls_button.
    ls_button-function  = 'PERCENTAGE_MARKED'.
    ls_button-text      = '% (marked)'(mar).
    ls_button-quickinfo = 'Occupation (marked flights)'(z02).
    ls_button-butn_type = 0.
    INSERT ls_button INTO TABLE e_object->mt_toolbar.

  ENDMETHOD.                    "on_toolbar

***********************************************************************
*         Handles self-defined user-commands                          *
***********************************************************************
  METHOD on_user_command.
    DATA: lv_occupied   TYPE i,
          lv_capacity   TYPE i,
          lv_percentage TYPE p LENGTH 8 DECIMALS 1,
          lv_text       TYPE string,

*   needed for method get_selected_rows
          lt_row_no     TYPE lvc_t_roid,
          ls_row_no     TYPE lvc_s_roid.

    CASE e_ucomm.
*     total percentage of occupied seats
      WHEN 'PERCENTAGE'.
        LOOP AT gt_flights
          INTO gs_flight.
          lv_occupied = lv_occupied + gs_flight-seatsocc.
          lv_capacity = lv_capacity + gs_flight-seatsmax.
        ENDLOOP.
        lv_percentage = lv_occupied / lv_capacity * 100.
        MOVE lv_percentage TO lv_text.
        lv_text = 'Percentage of occupied seats:'(occ) &&
                  ` ` &&
                  lv_text.
```

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
        MESSAGE lv_text TYPE 'I'.

*     percentage of occupied seats for marked flights
      WHEN 'PERCENTAGE_MARKED'.
*       providing IDs of selected rows
        go_alv_grid->get_selected_rows(
          IMPORTING
            et_row_no = lt_row_no ).
        IF lines( lt_row_no ) > 0.
          LOOP AT lt_row_no INTO ls_row_no.
            READ TABLE gt_flights INTO gs_flight INDEX ls_row_no-row_id.
            lv_occupied = lv_occupied + gs_flight-seatsocc.
            lv_capacity = lv_capacity + gs_flight-seatsmax.
          ENDLOOP.
          lv_percentage = lv_occupied / lv_capacity * 100.
          MOVE lv_percentage TO lv_text.
          lv_text = 'Percentage of occupied seats:'(occ) &&
                    ` ` &&
                    lv_text.
          MESSAGE lv_text TYPE 'I'.
        ELSE.
          MESSAGE 'Please mark at least one row'(one) TYPE 'I'.
        ENDIF.
    ENDCASE.
  ENDMETHOD.                    "on_user_command
ENDCLASS.                    "lcl_handler IMPLEMENTATION

...

*********************************************************************
*PBO modules
*********************************************************************

...

MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

* register event handlers
    SET HANDLER:
      lcl_handler=>on_toolbar      FOR go_alv_grid,
      lcl_handler=>on_user_command FOR go_alv_grid,
      lcl_handler=>on_doubleclick  FOR go_alv_grid,
      lcl_handler=>print_top       FOR go_alv_grid,
      lcl_handler=>print_tol       FOR go_alv_grid.

    gs_variant-report = sy-cprog.
    IF NOT pa_lv IS INITIAL.
      gs_variant-variant = pa_lv.
    ENDIF.

...

ENDMODULE.                 " create_and_transfer OUTPUT


***********************************************************************
*PAI modules
***********************************************************************
...
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Find suitable methods for a task
- Implement a method call in an ALV program

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV Grid Control: Context Menus

Lesson Overview

In this lesson, you learn how to display various context menus on the ALV.

Lesson Objectives

After completing this lesson, you will be able to:

- Use context menus
- Create context menus for ALV Grid Controls
- Process functions from context menus

Business Example

You are working as a programmer for ABC Travel. Various context menus are to be provided on the ALV containing the bookings.

Using Context Menus

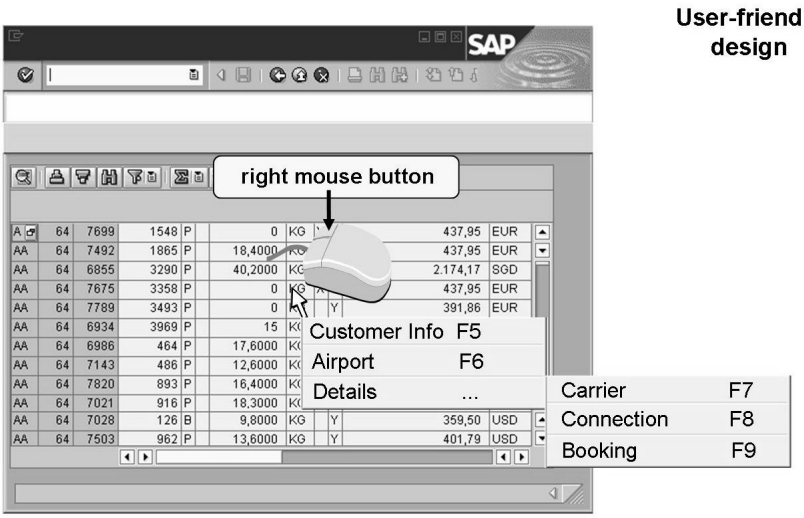


Figure 120: Using Context Menus

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

You can use a context menu to provide functions to the user that depend on the respective program context. Use context menus if the functions concerned are to be assigned to the control (or even part of it) and not to the entire screen. Note the following software ergonomics rules for designing context menus:

- Restrict the context menu to a maximum of two hierarchy levels.
- Restrict the context menu to a maximum of ten entries, but map all available buttons.
- Use separator lines for visual structuring.
- Object-specific functions should be placed at the top of the menu.

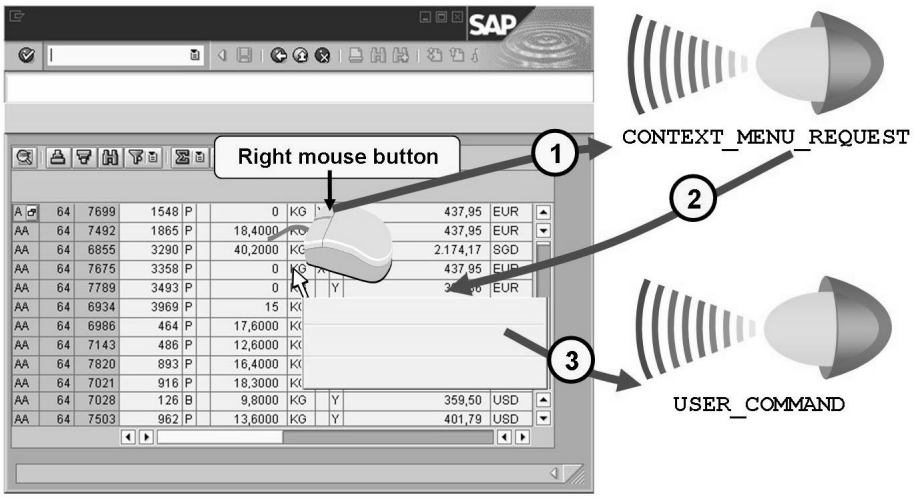


Figure 121: Context Menus: Process Flow

A context menu is displayed if the user chooses the right mouse button (or *Shift + F10*), while the mouse pointer is positioned over a GUI Control. The pressing of the right mouse button causes the ALV Grid Control to trigger the event *context_menu_request*. The context is determined by the position of the mouse pointer, that is, by the corresponding control that triggers the alert. If the user selects a function from the context menu, the ALV Grid Control triggers the event *user_command*. By default, an ALV Grid Control has a context menu that you can replace or extend.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: on_context_menu_request
      FOR EVENT context_menu_request
        OF cl_gui_alv_grid
      IMPORTING e_object.
    ...
ENDCLASS.

CLASS lcl_event_handler IMPLEMENTATION.
  METHOD on_context_menu_request.
    ...
  ENDMETHOD.
 ...
ENDCLASS.
 ...
SET HANDLER
  lcl_event_handler=>on_context_menu_request
  FOR go_alv_grid.
```

Static method can react to event `context_menu_request` and imports a reference to an object of class `cl_ctmenu`.

Implementation of the static method

Registration of the handler method

Figure 122: Event CONTEXT_MENU_REQUEST

Pressing the right mouse button leads to the event *context_menu_request* being triggered by the Control over which the mouse pointer is positioned. To display a specific context menu, you need to implement a handler method for the event and register this handler method at the application server (`SET HANDLER ... FOR go_alv_grid.`).

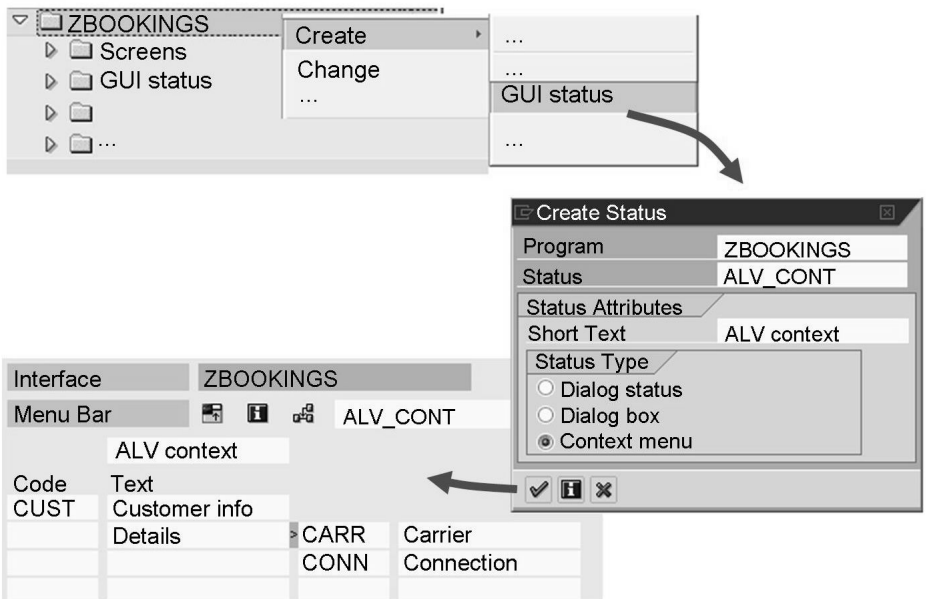


Figure 123: Static Context Menu

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

The simplest way to create a context menu is to load a GUI status of the type *context menu*. You can load a context menu in the object list of the Object Navigator, for example. Choose *context menu* as the status type and enter the required function codes and their texts in the Menu Painter. You create a submenu by entering text but no code for a line in the Menu Painter. A small triangle at the right indicates that there is a submenu. A maximum of three levels is possible.

```
* load predefined context menu
METHOD on_context_menu_request
  cl_ctmenu=>load_gui_status(
  cl_ctmenu=>load_gui_status(
      program  = sy-cprog
      status   = 'ALV_CONT'
*     disable  =
      menu     = e_object ).
      ...
* add new function
  e_object->add_function(
      fcode    = 'DETAILS'
      text     = 'Display more details'(t01) ).
ENDMETHOD.
```

Figure 124: Loading and Extending a Context Menu

Use the static method *load_gui_status* of the class *cl_ctmenu* to load the GUI status in the context menu. The parameters are assigned the following values:

program

Program name of the ABAP program in which the GUI status was defined (this is the same as `sy-cprog`, if the status of the interface belongs to the interface of the current program)

status

Name of the predefined GUI status

disable

Table of functions that are to be inactive in the context menu (Table type: *ui_functions*)

menu

Reference to the object of the class *cl_ctmenu* into which the GUI status is to be loaded.

You can extend the loaded context menu using the method *add_function* of the class *cl_ctmenu*.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Methods of the Class CL_CTMENU:

Method	*Description*
ADD_FUNCTION	Add a function
ADD_SEPARATOR	Add a separator
HIDE_FUNCTIONS	Hide functions
SHOW_FUNCTIONS	Show functions
DISABLE_FUNCTIONS	Deactivate functions
ENABLE_FUNCTIONS	Activate functions

Figure 125: Dynamic Context Menus

As well as the static method *load_gui_status*, the class *cl_ctmenu* provides a range of other methods. You can use these to create or adjust the context menu at runtime (usually depending on values in certain data objects). The corresponding methods are then called within the handler method. For more information, see the documentation for class *cl_ctmenu*.

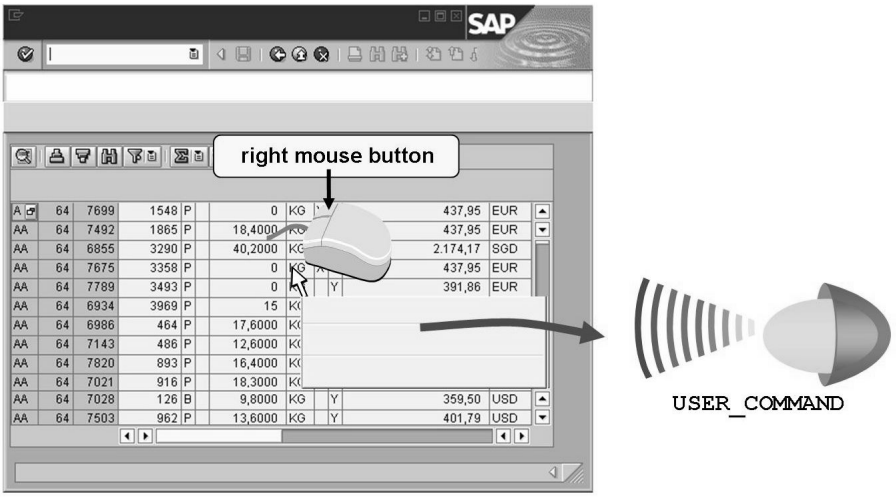


Figure 126: Processing Functions of Context Menus

If the user chooses a function of the context menu, the event *user_command* is triggered. The event exports the code of the function that the user selected using the parameter *e_ucomm*.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: on_user_command
      FOR EVENT user_command
      OF cl_gui_alv_grid
      IMPORTING e_ucomm.
ENDCLASS.

CLASS lcl_event_handler IMPLEMENTATION.
  METHOD on_user_command.
    CASE e_ucomm.
      WHEN 'FC1'.
        ...
      WHEN 'FC2'.
    ENDCASE.
  ENDMETHOD.
ENDCLASS.
```

static method `on_user_command` imports the function code of the event

Case distinction for the function chosen by the user

Figure 127: Event user_command

A function code triggered via the context menu is processed in the same was as a function code that is triggered by a pushbutton that you have defined. You must define, implement, and register a corresponding handler method for the *user_command* event of your ALV instance on the application server. You then use the function code (*e_ucomm*) within the handler method to decide which function is to be executed.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Exercise 13: ALV Grid Control: Context Menus

Exercise Objectives

After completing this exercise, you will be able to:

- Create context menus
- Process functions of context menus

Business Example

You are working as a programmer for ABC Travel. The users want individual context menus to be available on the ALV.

Task:

Extend your program so that if the user right-clicks the field for the carrier, the context menu of the ALV Grid Control is extended with an additional function. The normal context menu should appear for all of the other fields in the ALV.

Program *ZBC408_##_ALV*.

Copy from: *BC405_EVENT_S3*.

Model Solution: *BC405_EVENT_S4*

1. Define a variable of type I in the class containing the event handler methods. This variable will be used to interchange row information between the handler methods.

2. Define a handler method for the context menu call

   Create a static handler method called *on_context_menu_request* that can be registered for the event *context_menu* of the class *cl_gui_alv_grid*. Check in the Class Builder to see what name the parameter needs to have.

   Implement the following method: Using the *get_current_cell* method (of the *cl_gui_alv_grid* class) query the field when the mouse was when the right-click was performed. To do this, evaluate the parameter *es_col_id*. You should also store the value of parameter *e_row* in the global variable defined in the last step. *e_row* is the row of the internal table on which the mouse is placed. The normal context menu of the ALV Grid Control is only expanded to include a further function if the mouse was on the field with the airline: '*CARRIER_INFO*' function code with the text "Airline name". Use the method *add_function* of the class *cl_ctmenu* to do this.

   Register the new method for your ALV instance.

3. Define a handler method for the triggering of a context menu function

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Expand the *on_user_command* method for the new function: If *CARRIER_INFO* is triggered, the complete name of the airline should be displayed in an information message (I message). Read the appropriate name for the row that was clicked from the table *SCARR* and store it in an auxiliary variable (the row id is stored in the variable defined in the first step). You can use the following syntax for the report (as of SAP Web Application Sever 6.10): `MESSAGE <var> TYPE 'I'.`

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Solution 13: ALV Grid Control: Context Menus

Task:

Extend your program so that if the user right-clicks the field for the carrier, the context menu of the ALV Grid Control is extended with an additional function. The normal context menu should appear for all of the other fields in the ALV.

Program *ZBC408_##_ALV*.

Copy from: *BC405_EVENT_S3*.

Model Solution: *BC405_EVENT_S4*

1. Define a variable of type I in the class containing the event handler methods. This variable will be used to interchange row information between the handler methods.

   a) See result at the end.

2. Define a handler method for the context menu call

   Create a static handler method called *on_context_menu_request* that can be registered for the event *context_menu* of the class *cl_gui_alv_grid*. Check in the Class Builder to see what name the parameter needs to have.

   Implement the following method: Using the *get_current_cell* method (of the *cl_gui_alv_grid* class) query the field when the mouse was when the right-click was performed. To do this, evaluate the parameter *es_col_id*. You should also store the value of parameter *e_row* in the global variable defined in the last step. *e_row* is the row of the internal table on which the mouse is placed. The normal context menu of the ALV Grid Control is only expanded to include a further function if the mouse was on the field with the airline: '*CARRIER_INFO*' function code with the text "Airline name". Use the method *add_function* of the class *cl_ctmenu* to do this.

   Register the new method for your ALV instance.

   a) See result at the end.

3. Define a handler method for the triggering of a context menu function

   Expand the *on_user_command* method for the new function: If *CARRIER_INFO* is triggered, the complete name of the airline should be displayed in an information message (I message). Read the appropriate name for the row that was clicked from the table *SCARR* and store it in an auxiliary variable (the row id is stored in the variable defined in the first step). You can use the following syntax for the report (as of SAP Web Application Sever 6.10): `MESSAGE <var> TYPE 'I'.`

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

a) See result at the end.

Result

Main program

```
REPORT bc405_event_s4.

...

*********************************************************************
*Local class for ALV event handling
*********************************************************************
CLASS: lcl_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-DATA: gv_rowid type i.
    CLASS-METHODS: on_doubleclick FOR EVENT double_click
                                  OF cl_gui_alv_grid
                                  IMPORTING es_row_no,
                   print_top FOR EVENT print_top_of_page
                             OF cl_gui_alv_grid,
                   print_tol FOR EVENT print_top_of_list
                             OF cl_gui_alv_grid,
                   on_toolbar FOR EVENT toolbar
                              OF cl_gui_alv_grid
                              IMPORTING e_object,
                   on_user_command FOR EVENT user_command
                                   OF cl_gui_alv_grid
                                   IMPORTING e_ucomm,
                   on_context_menu_request FOR EVENT context_menu_request
                                           OF cl_gui_alv_grid
                                           IMPORTING e_object.
ENDCLASS.                    "lcl_handler DEFINITION

*---------------------------------------------------------------------*
CLASS: lcl_handler IMPLEMENTATION.

...

*********************************************************************
*         Handles self-defined user-commands                        *
*********************************************************************
  METHOD on_user_command.
    DATA: lv_occupied   TYPE i,
          lv_capacity   TYPE i,
```

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
          lv_percentage TYPE p LENGTH 8 DECIMALS 1,
          lv_text       TYPE string,
          lv_carrname   TYPE scarr-carrname,

*   needed for method get_selected_rows
          lt_row_no     TYPE lvc_t_roid,
          ls_row_no     LIKE LINE OF lt_row_no.

    CASE e_ucomm.
      WHEN 'PERCENTAGE'.
*       total percentage of occupied seats
        LOOP AT gt_flights
        INTO gs_flight.
          lv_occupied = lv_occupied + gs_flight-seatsocc.
          lv_capacity = lv_capacity + gs_flight-seatsmax.
        ENDLOOP.
        lv_percentage = lv_occupied / lv_capacity * 100.
        MOVE lv_percentage TO lv_text.
        CONCATENATE 'Percentage of occupied seats:'(occ)
        lv_text INTO lv_text SEPARATED BY space.
        MESSAGE lv_text TYPE 'I'.

*     percentage of occupied seats for marked flights
      WHEN 'PERCENTAGE_MARKED'.
*       providing IDs of selected rows
        go_alv_grid->get_selected_rows(
          IMPORTING
            et_row_no = lt_row_no ).
        IF lines( lt_row_no ) > 0.
          LOOP AT lt_row_no INTO ls_row_no.
            READ TABLE gt_flights INTO gs_flight INDEX ls_row_no-row_id.
            lv_occupied = lv_occupied + gs_flight-seatsocc.
            lv_capacity = lv_capacity + gs_flight-seatsmax.
          ENDLOOP.
          lv_percentage = lv_occupied / lv_capacity * 100.
          MOVE lv_percentage TO lv_text.
          CONCATENATE 'Percentage of occupied seats:'(occ)
          lv_text INTO lv_text SEPARATED BY space.
          MESSAGE lv_text TYPE 'I'.
        ELSE.
          MESSAGE 'Please mark at least one row'(one) TYPE 'I'.
        ENDIF.

      WHEN 'CARRIER_INFO'.
        READ TABLE gt_flights INTO gs_flight INDEX gv_rowid.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
        SELECT SINGLE carrname
          FROM scarr
          INTO lv_carrname
          WHERE carrid = gs_flight-carrid.
        MESSAGE lv_carrname TYPE 'I'.

    ENDCASE.
  ENDMETHOD.                    "on_user_command

  METHOD on_context_menu_request.
    DATA: ls_column_info TYPE lvc_s_col.

    go_alv_grid->get_current_cell(
      IMPORTING
        e_row     = gv_rowid
        es_col_id = ls_column_info ).
    CASE ls_column_info-fieldname.
      WHEN 'CARRID'.
        e_object->add_function(
          EXPORTING
            fcode = 'CARRIER_INFO'
            text  = 'Carrier info'(me1) ).
      WHEN OTHERS.
    ENDCASE.
  ENDMETHOD.                    "on_context_menu_request

ENDCLASS.                    "lcl_handler IMPLEMENTATION

...

*********************************************************************
*PBO modules
*********************************************************************

...

MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
    ENDIF.

    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

* register event handlers
    SET HANDLER:
      lcl_handler=>on_toolbar      FOR go_alv_grid,
      lcl_handler=>on_user_command FOR go_alv_grid,
      lcl_handler=>on_doubleclick  FOR go_alv_grid,
      lcl_handler=>print_top       FOR go_alv_grid,
      lcl_handler=>print_tol       FOR go_alv_grid,
      lcl_handler=>on_context_menu_request
                                   FOR go_alv_grid.

    gs_variant-report = sy-cprog.
    IF NOT pa_lv IS INITIAL.
      gs_variant-variant = pa_lv.
    ENDIF.

...

ENDMODULE.                 " create_and_transfer OUTPUT


***********************************************************************
*PAI modules
***********************************************************************
...
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Use context menus
- Create context menus for ALV Grid Controls
- Process functions from context menus

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Describe how events are triggered and how they can be handled
- List suitable events for a particular task
- Create and register an event handler method
- Add customer functions to the toolbar
- Define a handler method for print list events such as TOP-OF-PAGE
- Find suitable methods for a task
- Implement a method call in an ALV program
- Use context menus
- Create context menus for ALV Grid Controls
- Process functions from context menus

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Unit 7

Logical Databases

Internal Use SAP Partner Only

Unit Overview

This unit explains what logical databases are and how logical databases are connected to programs.

Unit Objectives

After completing this unit, you will be able to:

- List the advantages of logical databases
- List important performance characteristics of a logical database
- List the subobjects of a logical database
- Identify selections in the logical database
- Retrieve data using a logical database

Unit Contents

Lesson: Logical Databases: Introduction.....250
Lesson: Logical Databases: Subobjects and Data Retrieval 255
Exercise 14: GET Events.....267
Exercise 15: GET LATE Events and Checks from Internal Program Selections.....273

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson: Logical Databases: Introduction

Lesson Overview

In this lesson, you learn about the most important characteristics and advantages of logical databases.

Lesson Objectives

After completing this lesson, you will be able to:

- List the advantages of logical databases
- List important performance characteristics of a logical database

Business Example

You are working as a programmer for ABC Travel. You need to explain the relevance of logical databases, since these are to be used in the future for data retrieval in reports.

Logical Databases: Overview

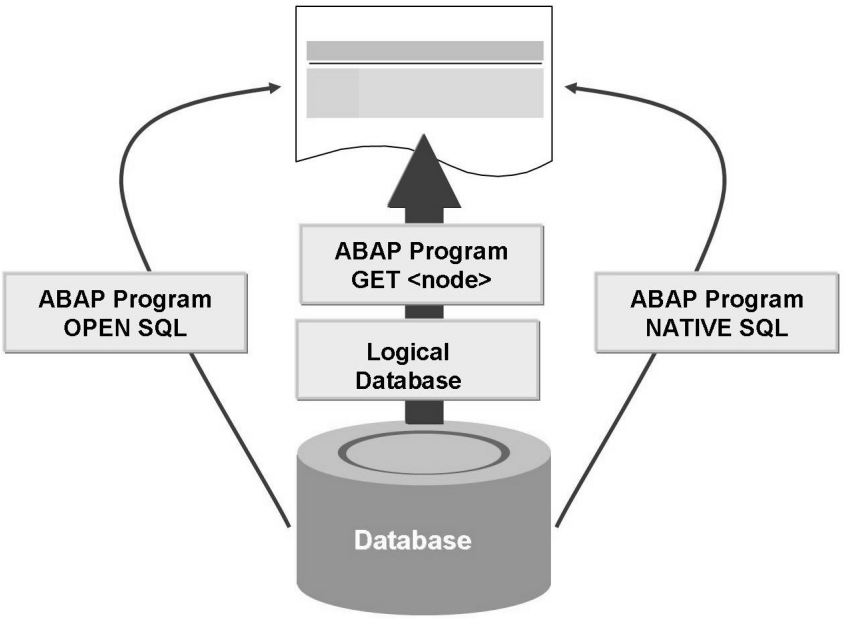


Figure 128: Generating Lists

In general, the system reads data that will appear in a list from the database.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

You can either use OPEN SQL or NATIVE SQL statements to read data from the database. However, you should avoid Native SQL to ensure portability from one Relational Database Management System (RDBMS) to another. You also need to recode Native SQL every time you upgrade your RDBMS. No restrictions or requirements of this type apply to OPEN SQL. OPEN SQL is a platform-independent SQL variant of ABAP. OPEN SQL statements are implemented in the relevant NATIVE SQL by the database interface.

The use of a logical database provides you with an alternative to having to program database accesses individually. Logical databases retrieve data records and make them available to the ABAP program. They also provide a security advantage, since the authorization checks in the logical database's program instead of in the report.

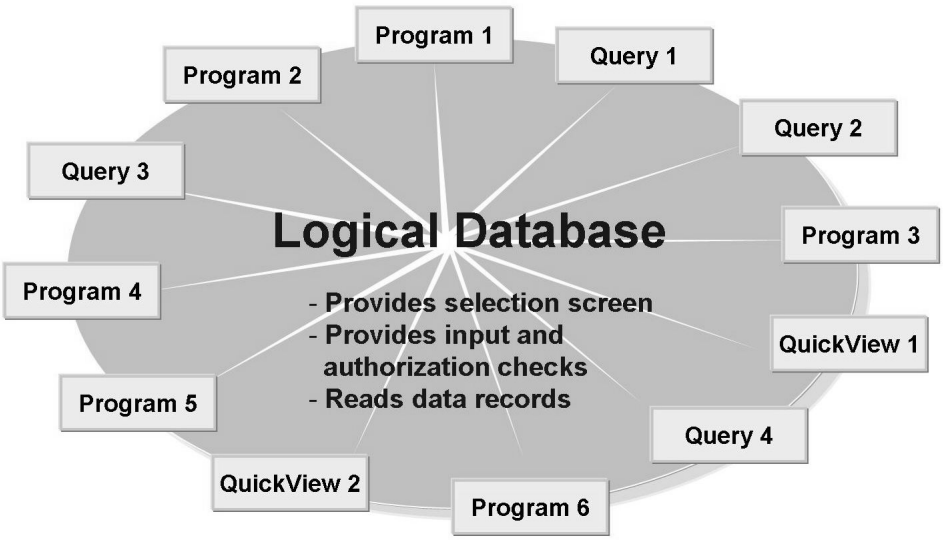


Figure 129: Advantages of a Logical Database

The same logical database can be the data source for a large number of reports. In the case of executable programs, the LDB is entered in the program attributes or called using function module *LDB_PROCESS*.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Logical databases offer various advantages over other methods of data retrieval:

- The selection screen is provided. The use of selection screen versions or variants or separate program extensions provides the required flexibility.
- The programmers do not need to know the exact structure or relationship of the involved tables; the data is provided to the application program in the primary key sequence. Performance improvements within logical databases directly affect all programs linked to the logical database, without having to change the programs themselves.
- Maintenance and enhancements can be performed at a central location.
- Authorization checks are usually performed in the logical database's program.

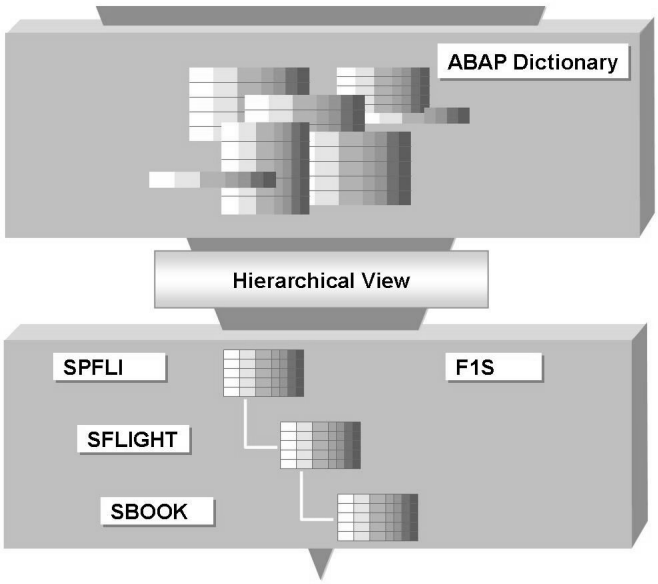


Figure 130: Logical Database: Overview

A logical database is an ABAP program that reads data from the database and makes it available to other programs.

A hierarchical structure determines the order in which the data is supplied to the programs. A logical database also provides a selection screen that checks user entries and performs error dialogs. You can add your own parameters and selection options.

There are approximately 240 logical databases for *SAP NetWeaver 7.0*.

If the name of a logical database consists of three characters, the first two are the name of the logical database, and the third represents the work area. You can find a list of the possible values for the third character in the program attributes (choose F4 on the *Application* field). For example, DDF is the name for “customer data”, and F stands for “Finances”.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

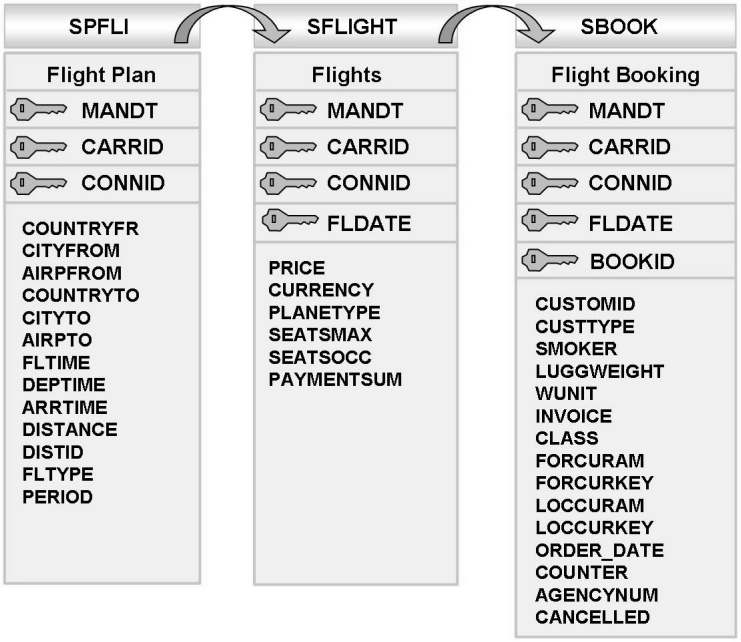


Figure 131: Logical Database: F1S Nodes

The demo programs and exercises for ABAP courses and ABAP documentation refer to the SAP system's *BC_TRAVEL* flight data model, which is found in the *BC_DATAMODEL* package.

The following tables make up logical database *F1S*:

- *SPFLI*: Flight connections
- *SFLIGHT*: Flights
- *SBOOK*: Bookings

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- List the advantages of logical databases
- List important performance characteristics of a logical database

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: Logical Databases: Subobjects and Data Retrieval

Lesson Overview

In this lesson, you learn how to use logical databases for data retrieval.

Lesson Objectives

After completing this lesson, you will be able to:

- List the subobjects of a logical database
- Identify selections in the logical database
- Retrieve data using a logical database

Business Example

You are working as a programmer for ABC Travel. Your manager wants to have a hierarchical list of the current flight connections and flights. It is your task to implement this as small as possible a development effort.

LDB Subobjects and Data Retrieval

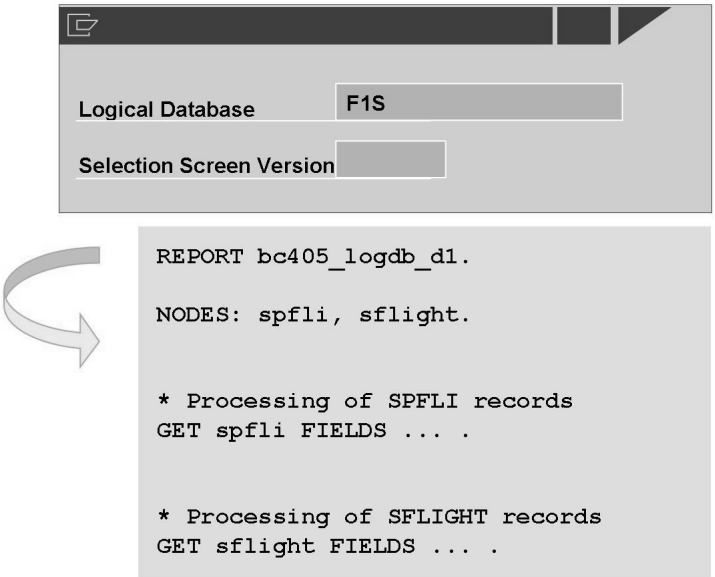


Figure 132: Example Program for a Logical Database

You need to enter the name of the logical database in the program attributes.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Use the `NODES <knoten>` statement to specify the nodes of the logical database that you want to use in the program. `NODES` allocates the appropriate storage space for the node, that is, a work area or table area, depending on the node type. Note that you can still use the keyword `TABLES`.

When the logical database reads the data records, it issues the keyword `PUT <node>` which triggers the `GET <node>` event in the report program.

Name of the Node		Node Type	Short Text
SPFLI		Table	Flight plan
SFLIGHT		Table	Flight table
SBOOK		Table	Flight bookings

Defines the hierarchy (read sequence) of the data

Possible node types:

Database table	Table or structure from the ABAP Dictionary. Name must be identical to the node name.
DDIC type	DDIC type table or structure. Name can be different from the node name; deep structures are possible.
Data type	Type that has been defined in a type group
Dynamically typed	Type is specified in program

Figure 133: LDB Subobjects: Structure

Logical databases are made up of several subobjects. The structure determines the hierarchy, and therefore the read sequence of the data records.

Node names can contain up to 14 characters. There are four different node types:

- Table (type T): The node name is the name of a transparent table (this type corresponds to the concept prior to *Release 4.0A*). The table name must be identical to the node name. Deep types are not allowed.
- DDIC type (type S): You can use any node name. The node is assigned a structure or a table type from the Dictionary. The node name can differ from the type name. Deep structures are possible.
- Type groups (type C): The node type is defined in a type group. The name of the type group must be maintained in the *Type group* field. You should generally prefer DDIC types, since other applications that use the logical database (such as SAP Query) can access them (for short texts, and so on).
- Dynamic nodes (type A): These nodes do not have a fixed type; they are not classified until the program runtime. Which types are generally allowed is determined when the structure is created.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

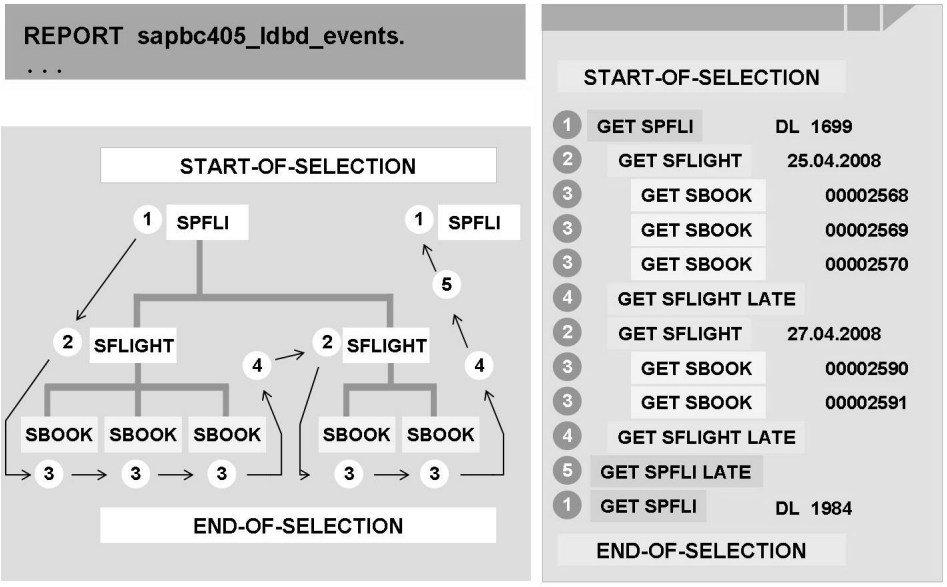


Figure 134: Events in Logical Databases

The `START-OF-SELECTION` event is triggered before control is given to the read routine of the logical database. The `END-OF-SELECTION` event is triggered after all `GET` events have been processed - that is, all data records have been read and processed.

The `GET <node>` event triggers whenever the logical database supplies data for this node. This means that `GET` events are processed several times, and that data has already been read from the database for these events. The sequence in which the `GET` events are processed is determined by the structure of the logical database.

The `GET <node> LATE` event triggers when all subordinate nodes of `<node>` nodes have been processed, before the data is read for the next `<node>` (whenever a hierarchy level has been completed).

If you use the GET event for outputting a classic list, the system automatically creates a line feed at the beginning of an event and sets the standard formats (for example, font color and brightness).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

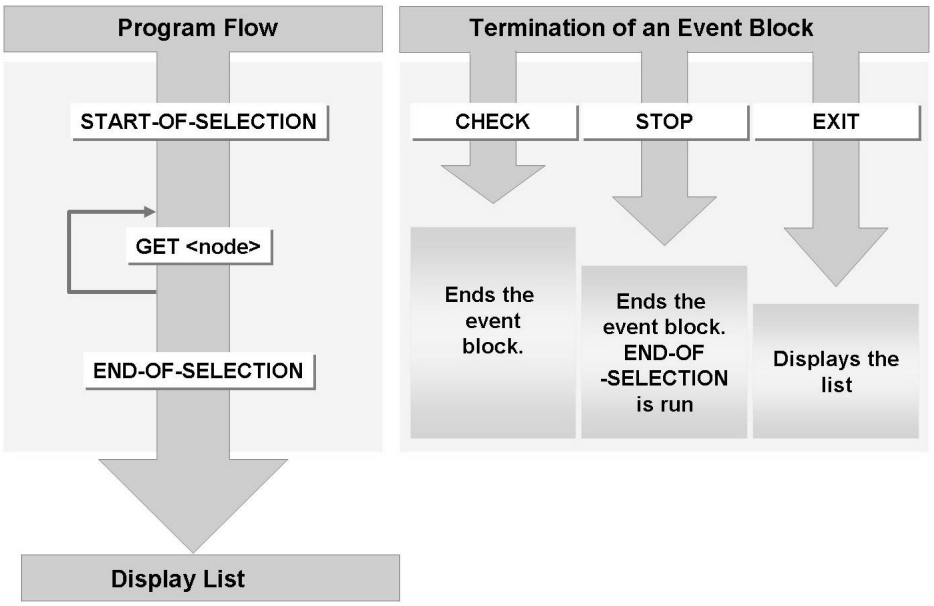


Figure 135: Program Flow and Termination Alternatives

The keyword `CHECK` terminates the current processing block, as long as the condition is false.

The keyword `STOP` terminates the program processing. However, in contrast to the `EXIT` statement, the processing block `END-OF-SELECTION` is processed first (if it exists).

If the `STOP` statement is within the `END-OF-SELECTION` processing block, program processing ends immediately and a list is displayed.

The `EXIT` statement exits the program without first executing the `END-OF-SELECTION` event. The list that was generated before the generation of the `EXIT` keyword is displayed.

You can also use the `REJECT` statement. The data record is not processed further. When reading the next data record, processing continues on the same hierarchy level. `REJECT` can also be used in a subroutine.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Selection include db<ldbname>sel:
definition of the selection screen

SELECT-OPTIONS: ... .
PARAMETERS: ... FOR NODE <node>.

Standard Selection Screen
Screen: 1000

- Allow field selection
- Define selection screen versions
- Allow dynamic selections

  The fields in the dynamic selections are determined using the selection view:
  CUS or SAP are possible as the origin of the view.

Figure 136: LDB Subobjects: Selections

Logical database programs have an include called *DB<name>SEL*, in which the selection screen of the logical database is defined. The addition `FOR NODE` assigns selections to individual nodes. The actual appearance of a selection screen therefore depends on the `NODES` statement in your program.

You can enable a **field selection** for the individual nodes in the logical database program. This occurs with the keyword `FIELD SELECTION FOR <node>` in the `SELECTION-SCREEN` statement. This code in the logical database program allows the application program to use the `GET <node> FIELD <field list>` statement to restrict the amount of data retrieved.

Some logical databases even note dynamic selections for individual nodes. This can be selected via *Dynamic Selections* with the `DYNAMIC SELECTIONS FOR <node>` addition. The *Dynamic Selections* button then appears on the selection screen of your report. You can determine which selection fields can be set by choosing a particular selection view yourself (type *CUS*) or the selection view (type *SAP*) provided by the SAP system.

Some large logical databases have multiple selection screen versions. Each selection screen version contains a subset of selection criteria or parameters (language element: `EXCLUDE`) or both. You specify which selection screen is to be displayed by entering the version number of the selection screen in the program attributes.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

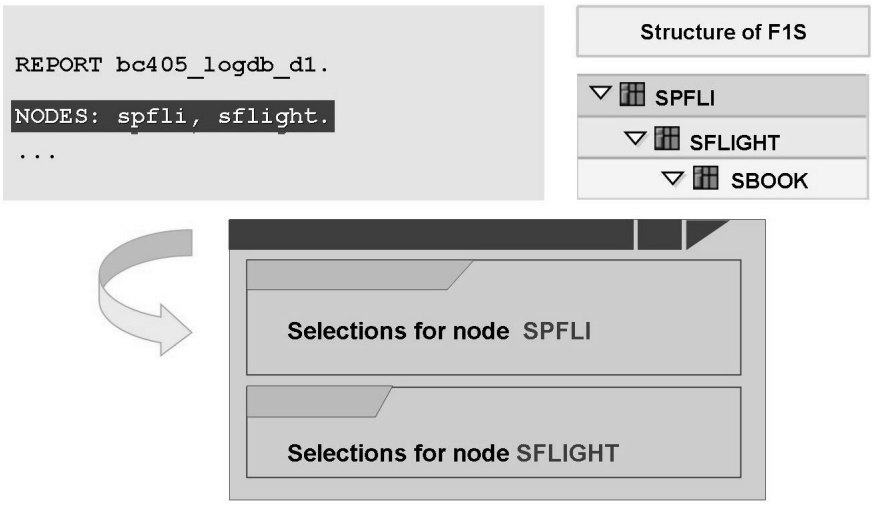


Figure 137: Selection Screen for the Logical Database

When you enter a logical database in the attributes of your type -1 program, the system processes the selection screen of the logical database. The concrete characteristics of the selection screen depend upon the node specified in the `NODES` statement. If you specify a node of type T (table), you can also declare the table work area with the `TABLES` statement.

If you address only subordinate nodes (in the hierarchy) of the logical database in the program (for example *sflight*), the selection criteria for the superior node in the hierarchy (*spfli*) also appear. You can therefore restrict the dataset to be read so that it meets your specific requirements.

Hint: A logical database **always reads in accordance with its structure**. Therefore, if you only require data from a node deep in the hierarchy, you should either find another logical database in which the desired node is higher in the structure, or program the data retrieval yourself.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

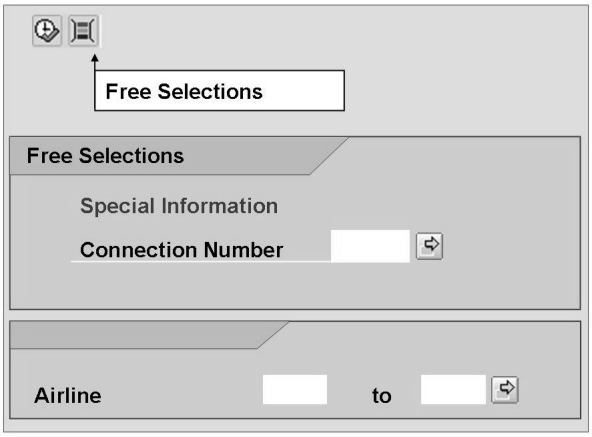


Figure 138: Logical Database: Dynamic Selections

If the logical database supports dynamic selections, the button for *Dynamic Selections* appears on the selection screen. If the user presses this button, additional input fields appear on the selection screen. The user can select additional database fields here. The system transfers the selections directly to the logical database program and to the database (dynamic selections).

The selection view determines which fields are displayed on the selection screen. In order to create your own selection view for a logical database, select the following menu path (starting from the ABAP editor): *Other Object... → "More" tab page → Selection view → Continue*. In the following dialog box, enter the following values: Origin of view: *CUS*. Name of view: *STANDARD*. A selection view with the origin *CUS* covers a selection view of the origin *SAP*.

In the definition of the selection view, first define field groups. Then assign the individual fields of a node to a field group by numbering the field groups with the fields. When you do this, you can also define whether these fields are to be immediately available if the user wants to work with *Dynamic Selections*.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

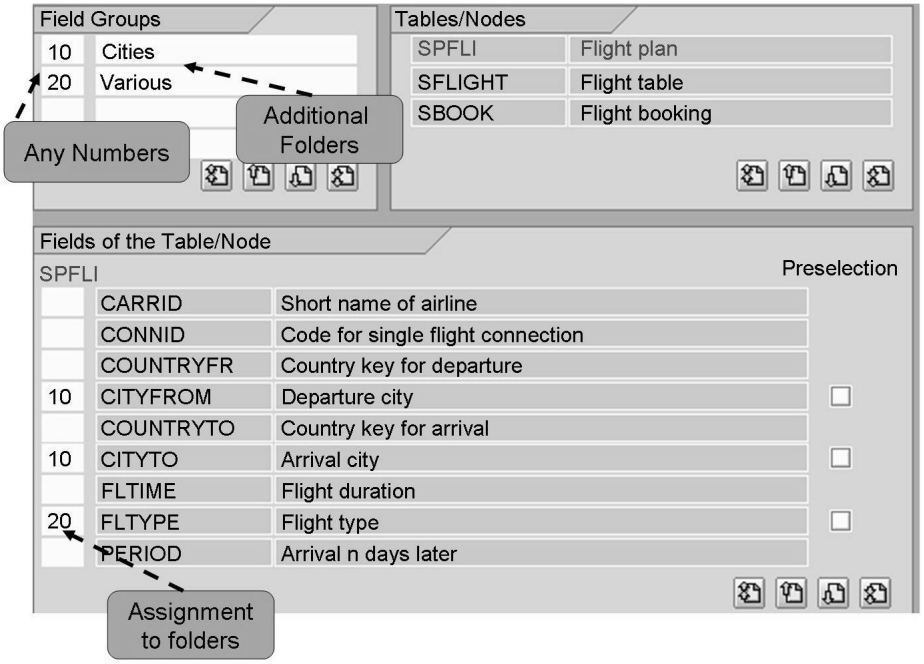


Figure 139: Selection Views

Naming convention sapdb<ldbname>

Database accesses, input and authorization chekcs

```
sapdb<ldbname>
FORM init.
...
ENDFORM init.

FORM put_spfli.
...
ENDFORM put_spfli.

FORM put_sflight.
...
ENDFORM put_sflight.
```

Figure 140: LDB Subobjects: Database Programs

The logical database programs are called *SAPDB<ldbname>* for the logical database `<ldbname>`. The programs consist of a collection of subroutines, each of which is executed at a particular event. For example, subroutine `init` is processed once at the start of the database program. This subroutine can be used to define default values for the selection screen of the LDB.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

There are other subroutines that are processed at the PBO (**P**rocess **B**efore **O**utput) and PAI (**P**rocess **A**fter **I**nput) events of the selection screen. Checks, such as authorization checks (`AUTHORITY-CHECK`), are usually performed during the PAI event.

The database accesses (`SELECT` statements) are programmed in the `put_<node>` subroutines. These subroutines may be processed several times, depending on which selection criteria the user specifies. The sequence in which these subroutines are processed is determined by the structure of the logical database.

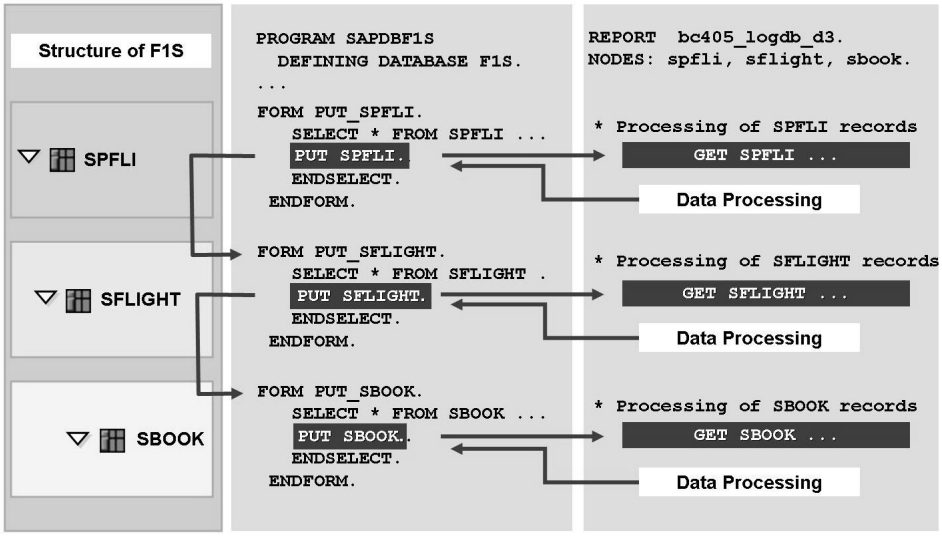


Figure 141: Interdependency: LDB and Program

When a program that uses a logical database is started, control is initially passed to the database program of the logical database. Each event has a corresponding subroutine in the database program (for example, subroutine `init` for event `INITIALIZATION`). During the interaction between the LDB and the associated report, the subroutine is always processed first, followed by the event (if there is one in the report).

Logical database programs read data from a database according to the structure declared for the logical database. They begin with the root node and then process the individual branches consecutively from top to bottom.

The logical database reads the data in the `put_<node>` subroutines. During the `PUT` event, control is passed from the database program to the `GET` event of the associated report. The data is made available in the corresponding work areas in the report. The processing block defined for the `GET` event is executed. Control then returns to the logical database. `PUT` triggers the following subroutine in accordance with the structure. This flow is continued until the report has collected all the available data.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The **read depth** in the structure depends on the `GET` events specified in the program. A logical database reads to the lowest `GET` event contained within the structure attributes. Only those `GET` events for which processing is supposed to take place are written into the report program. Logical databases read all data records found on the direct access path.

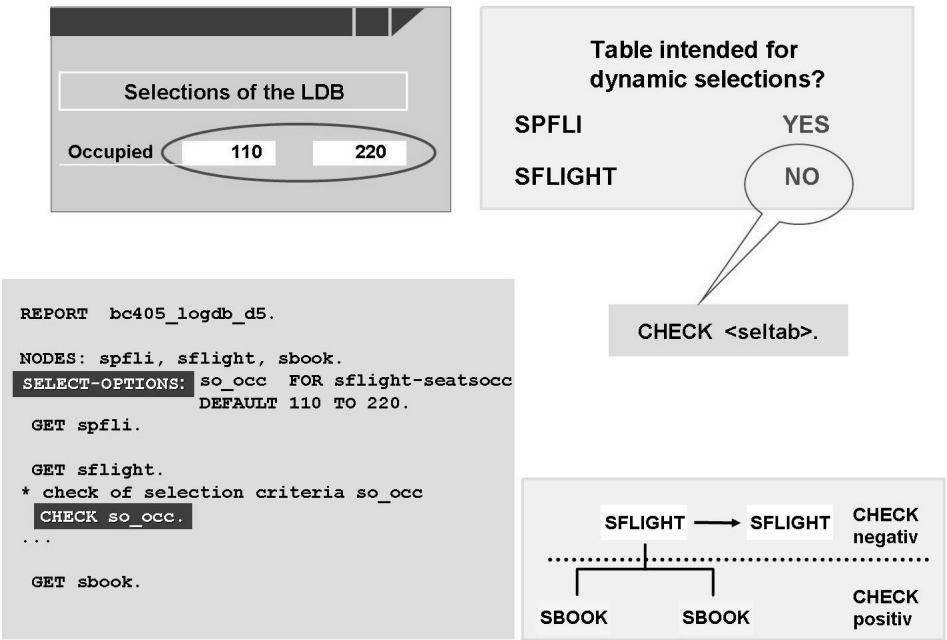


Figure 142: Checking Internal Program Selections

If you specify a logical database and declare additional `SELECT-OPTIONS` in the program attributes that refer to the fields of a node **not** designated for dynamic selection, you must use the `CHECK <seltab>` statement to see if the current data record fulfills the selection criteria.

If the result of the check is negative, processing of the current processing block ends.

If you want to extend your selection screen with `PARAMETERS` (single values), you need to use `CHECK` in every case to check, even if dynamic selections are possible for the node. In this case, you must enter a complete logical condition (the abridged syntax form `CHECK <seltab>` is only available for selection options).

Extending the Selection Screen with *PARAMETERS*

```
NODES:
  spfli,
  sflight.
```


© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
 ...
PARAMETERS:
  pa_ftp TYPE spfli-fltype.
 ...
GET spfli.
  CHECK spfli-fltype = pa_ftp.
 ...
GET SFLIGHT.
 ...
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 14: GET Events

Exercise Objectives

After completing this exercise, you will be able to:

- Generate a list whose data is read from a logical database

Business Example

You are working as a programmer for ABC Travel. You are to produce a hierarchical list containing flight connections, flights, and bookings.

Task:

Create the program *ZBC405_##_LOGDB1* with a TOP include. Ensure you specify ***Executable program*** as the program type. Enter the logical database *F1S* in the program attributes.

Model solution for the task: *BC405_LOGDB_S1*.

1. The logical database should supply the program with data for nodes *SPFLI*, *SFLIGHT*, and *SBOOK*.
2. Create a list that displays the following data:

Table	Fields
SPFLI	• *CARRID* • *CONNID* • *CITYFROM* • *AIRPFROM* • *CITYTO* • *AIRPTO*
SFLIGHT	• *FLDATE* • *PRICE* • *CURRENCY* • *PLANETYPE* • *SEATSMAX* • *SEATSOCC* • *FREE_SEATS*
SBOOK	• *BOOKID* • *CUSTOMID*

Continued on next page

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Table	Fields
	• *SMOKER* • *WUNITLUGGWEIGHT*

The *FREE_SEATS* field is not a table field – it has to be calculated in the program.

3. Use the following addition for the output of the *SFLIGHT-PRICE* field: `CURRENCY SFLIGHT-CURRENCY` `CURRENCY SFLIGHT-CURRENCY`. This way, the sum of money is output appropriately.

   In the same way, use the following addition for the output of the *SBOOK-LUGGWEIGHT* field: `UNIT SBOOK-WUNIT`. This formats the amount.

4. Generate a list in which each line outputs the information for one read record (see above).

   Hint: To output the fields, use the pattern functions available in the ABAP Editor.

5. **Optional:** Set the selection screen of the logical database so that when you start your program, the airlines from AA to LH are entered.

   Hint: To determine the field names, start the program, press **`F1`** on the input fields for the airline and go to *Technical Information*.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 14: GET Events

Task:

Create the program *ZBC405_##_LOGDB1* with a TOP include. Ensure you specify ***Executable program*** as the program type. Enter the logical database *F1S* in the program attributes.

Model solution for the task: *BC405_LOGDB_S1*.

1. The logical database should supply the program with data for nodes *SPFLI*, *SFLIGHT*, and *SBOOK*.

   a) See result.

2. Create a list that displays the following data:

Table	Fields
SPFLI	• *CARRID* • *CONNID* • *CITYFROM* • *AIRPFROM* • *CITYTO* • *AIRPTO*
SFLIGHT	• *FLDATE* • *PRICE* • *CURRENCY* • *PLANETYPE* • *SEATSMAX* • *SEATSOCC* • *FREE_SEATS*
SBOOK	• *BOOKID* • *CUSTOMID* • *SMOKER* • *WUNITLUGGWEIGHT*

The *FREE_SEATS* field is not a table field – it has to be calculated in the program.

a) See result.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

3. Use the following addition for the output of the *SFLIGHT-PRICE* field: `CURRENCY SFLIGHT-CURRENCY` `CURRENCY SFLIGHT-CURRENCY.` This way, the sum of money is output appropriately.

   In the same way, use the following addition for the output of the *SBOOK-LUGGWEIGHT* field: `UNIT SBOOK-WUNIT`. This formats the amount.

   a) See result.

4. Generate a list in which each line outputs the information for one read record (see above).

   Hint: To output the fields, use the pattern functions available in the ABAP Editor.

   a) See result.

5. **Optional:** Set the selection screen of the logical database so that when you start your program, the airlines from AA to LH are entered.

   Hint: To determine the field names, start the program, press **F1** on the input fields for the airline and go to *Technical Information*.

   a) See result.

Result

TOP include

```
*&---------------------------------------------*
*& Report  BC405_LOGDB_S2
*&---------------------------------------------*

* Used nodes of logical database F1S
NODES: spfli, sflight, sbook.

* Variables
DATA: gv_free_seats TYPE sflight-seatsocc.
```

Main program

```
*&---------------------------------------------*
*& Include BC405_LOGDB_S2_TOP
*&---------------------------------------------*
REPORT   bc405_logdb_s1.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
INCLUDE bc405_logdb_s1_top.

*Begin of optional part
*&---------------------------------------------------------------------*
*&   Event INITIALIZATION
*&---------------------------------------------------------------------*
INITIALIZATION.
  carrid-sign   = 'I'.
  carrid-option = 'BT'.
  carrid-low    = 'AA'.
  carrid-high   = 'LH'.
  APPEND carrid.
*End of optional part

*&-----------------------------------------------
*&   Event GET SPFLI
*&-----------------------------------------------
GET spfli.
* Data output SPFLI
  WRITE:    spfli-carrid,
            spfli-connid,
            spfli-cityfrom,
            spfli-airpfrom,
            spfli-cityto,
            spfli-airpto.

*&-----------------------------------------------
*&   Event GET SFLIGHT
*&-----------------------------------------------
GET sflight.
* Calculate free seats
  gv_free_seats = sflight-seatsmax - sflight-seatsocc.

* Data output SFLIGHT
  WRITE:   sflight-fldate,
           sflight-price CURRENCY sflight-currency,
           sflight-currency,
           sflight-planetype,
           sflight-seatsmax,
           sflight-seatsocc,
           gv_free_seats.

*&-----------------------------------------------
*&   Event GET SBOOK
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
*&---------------------------------------------
GET sbook.
  WRITE:   sbook-bookid,
           sbook-customid,
           sbook-smoker,
           sbook-luggweight UNIT sbook-wunit,
           sbook-wunit.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 15: GET LATE Events and Checks from Internal Program Selections

Exercise Objectives

After completing this exercise, you will be able to:

- Generate a list whose data is read using a logical database
- Check internal selections for their validity

Business Example

Task:

Extend the program *ZBC405_##_LOGDB1* or copy the *BC405_LOGDB_S1* model solution to the program *ZBC405_##_LOGDB2*.

Model solution for the task: *BC405_LOGDB_S2*.

1. Add another `SELECT-OPTIONS` statement for the customer number (*SBOOK-CUSTOMID*) to the selection screen. Frame the selection and maintain the selection text.

   Ensure that only bookings that meet the specified selection criteria are output in the list.

2. In the list, output a solid line when all the bookings for a date have been output, and when a flight connection has been completely output. Output each flight connection on a new page. To do this, use the statements `ULINE.` (for the solid line) and `NEW-PAGE.` (for the page break) in the appropriate places.

3. **Optional:** Within your program, hide the selection field for the destination airport, which is displayed by default (without changing the logical database).

4. **Optional:** Create/expand the CUS selection view for the F1S logical database. (Please take into account that this task can only be completed by one participant at a time, as locks are set.) Then test the your unchanged program again.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 15: GET LATE Events and Checks from Internal Program Selections

Task:

Extend the program *ZBC405_##_LOGDB1* or copy the *BC405_LOGDB_S1* model solution to the program *ZBC405_##_LOGDB2*.

Model solution for the task: *BC405_LOGDB_S2*.

1. Add another `SELECT-OPTIONS` statement for the customer number (*SBOOK-CUSTOMID*) to the selection screen. Frame the selection and maintain the selection text.

   Ensure that only bookings that meet the specified selection criteria are output in the list.

   a) See the solution at the end of the task.

2. In the list, output a solid line when all the bookings for a date have been output, and when a flight connection has been completely output. Output each flight connection on a new page. To do this, use the statements `ULINE.` (for the solid line) and `NEW-PAGE.` (for the page break) in the appropriate places.

   a) See the solution at the end of the task.

3. **Optional:** Within your program, hide the selection field for the destination airport, which is displayed by default (without changing the logical database).

   a) See the solution at the end of the task.

4. **Optional:** Create/expand the CUS selection view for the F1S logical database. (Please take into account that this task can only be completed by one participant at a time, as locks are set.) Then test the your unchanged program again.

   a) See the description in the course materials.

Result

TOP include

```
*&---------------------------------------------*
*& Report  BC405_LOGDB_S2
*&---------------------------------------------*

* Required nodes of logical database F1S
NODES: spfli, sflight, sbook.
```

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
* Additional selections
SELECTION-SCREEN BEGIN OF BLOCK order WITH FRAME.
SELECT-OPTIONS: so_cust FOR sbook-customid.
SELECTION-SCREEN END OF BLOCK order.

* Variables
DATA: gv_free_seats LIKE sflight-seatsocc.
```

Main program

```
*&---------------------------------------------------*
*& Include BC405_LOGDB_S2_TOP
*&---------------------------------------------------*
REPORT bc405_logdb_s2 LINE-SIZE 83.

INCLUDE bc405_logdb_s2_top.


*&---------------------------------------------------*
*&   Event INITIALIZATION
*&---------------------------------------------------*
INITIALIZATION.
  carrid-sign = 'I'.
  carrid-option = 'BT'.
  carrid-low = 'AA'.
  carrid-high = 'LH'.
  APPEND carrid.

*&---------------------------------------------------*
*&   Event AT SELECTION-SCREEN OUTPUT
*&---------------------------------------------------*
AT SELECTION-SCREEN OUTPUT.
  LOOP AT SCREEN.
    IF screen-name CS 'AIRP_TO'.
      screen-active = 0.
      MODIFY SCREEN.
    ENDIF.
  ENDLOOP.

&-----------------------------------------------
*&   Event GET SPFLI
*&-----------------------------------------------
GET spfli.
* Data output SPFLI
  WRITE:    spfli-carrid,
```

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
            spfli-connid,
            spfli-cityfrom,
            spfli-airpfrom,
            spfli-cityto,
            spfli-airpto.

*&---------------------------------------------------
*&   Event GET SFLIGHT
*&---------------------------------------------------
GET sflight.
* Calculate free seats
  gv_free_seats = sflight-seatsmax - sflight-seatsocc.

* Data output SFLIGHT
  WRITE:   sflight-fldate,
           sflight-price CURRENCY sflight-currency,
           sflight-currency,
           sflight-planetype,
           sflight-seatsmax,
           sflight-seatsocc,
           gv_free_seats.

*&---------------------------------------------------
*&   Event GET SBOOK
*&---------------------------------------------------
GET sbook.

* Check select-option
  CHECK so_cust.

  WRITE:   sbook-bookid,
           sbook-customid,
           sbook-smoker,
           sbook-luggweight UNIT sbook-wunit,
           sbook-wunit,
           sbook-order_date.

*&---------------------------------------------------
*&   Event GET SPFLI LATE
*&---------------------------------------------------
GET spfli LATE.
  ULINE.
  NEW-PAGE.

*&---------------------------------------------------
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
*&   Event GET SFLIGHT LATE
*&-----------------------------------------------
GET sflight LATE.
  ULINE.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- List the subobjects of a logical database
- Identify selections in the logical database
- Retrieve data using a logical database

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- List the advantages of logical databases
- List important performance characteristics of a logical database
- List the subobjects of a logical database
- Identify selections in the logical database
- Retrieve data using a logical database

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Unit 8

Data Retrieval Without Logical Databases

Internal Use SAP Partner Only

Unit Overview

In a relational database model, as it is used in an SAP system, information is usually distributed across multiple database tables. In this unit, different possibilities to read information from multiple database tables are discussed.

Unit Objectives

After completing this unit, you will be able to:

- Implement database accesses across multiple tables
- Rate the different techniques with regard to optimized performance
- Use other additions of the SELECT statement

Unit Contents

Lesson: Evaluating Multiple Database Tables 282
Exercise 16: Data Retrieval with the Inner Join 289
Lesson: Further Additions to the SELECT Statement 295
Exercise 17: Further Additions to the SELECT Statement: Aggregation and GROUP BY 303
Exercise 18: Further Additions to the SELECT Statement: GROUP BY with COUNT (DISTINCT) 307

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson: Evaluating Multiple Database Tables

Lesson Overview

In this lesson, you will learn to retrieve data from the database using the `SELECT` statement. You will learn to read data from multiple tables using the join and other techniques. You will learn to rate different techniques to achieve an optimized runtime requirement.

Lesson Objectives

After completing this lesson, you will be able to:

- Implement database accesses across multiple tables
- Rate the different techniques with regard to optimized performance

Business Example

You are a programmer for ABC Travel. The manager needs a list with flight data which must be retrieved from multiple database tables. You must implement the necessary program.

Data Retrieval Overview

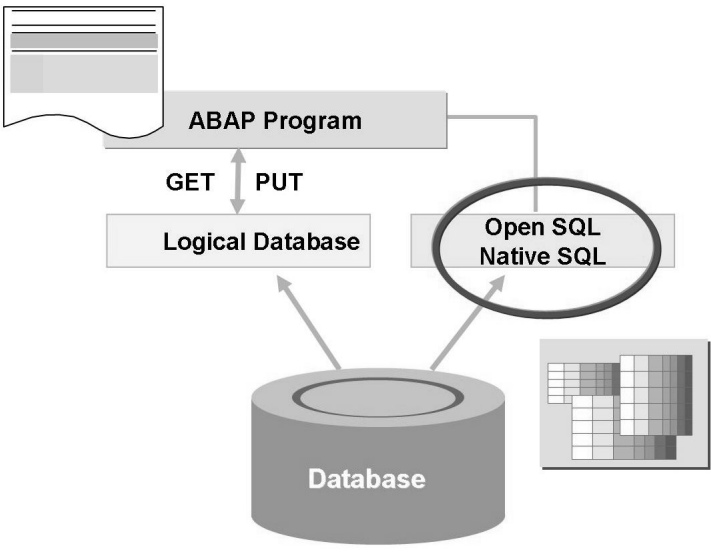


Figure 143: Data Retrieval Overview

Programming database access directly into the program itself can be done using either Open SQL or Native SQL statements.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Open SQL is preferable to Native SQL since it is not database-specific and guarantees that your program is independent of the database engine. Use of Open SQL allows for portability between different RDMBS systems and allows your company to change to a different database system. ABAP Open SQL is completely integrated in to the Workbench which provides syntax checking of SQL statements. ABAP Open SQL also allows the SAP system to make use of data buffers where appropriate.

Native SQL statements are included into a program using:

```
EXEC SQL [PERFORMING <form>].
```

<Native SQL statements>.

```
ENDEXEC.
```

Do **NOT** use Native SQL to change the SAP database system. The referential integrity of the SAP system data is maintained by the SAP system programs. Using Native SQL can quite easily corrupt your data. If you must use Native SQL, consider putting the code in an include so that it can be modified more easily if your RDBMS changes. Restrict your Native SQL to Standard SQL (ISO9075:1992).

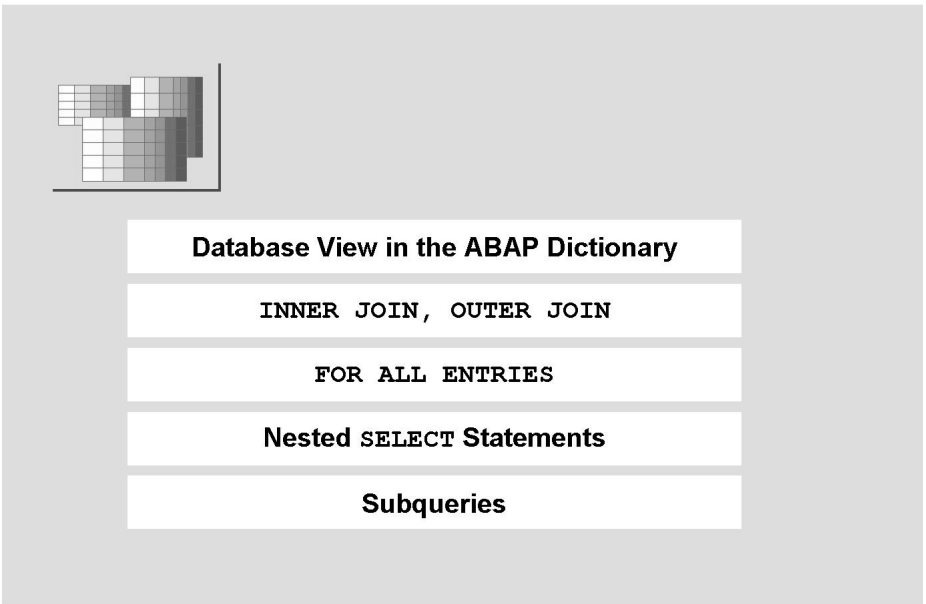


Figure 144: Reading Multiple Database Tables

In order to ensure optimized performance in a client/server environment, reading multiple database tables requires strong consideration for the SQL statements used.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

To ensure optimal database performance, follow these general rules:

- Keep the number of hits as low as possible (use WHERE conditions, for example).
- Keep data transfer between the application server and the database server to a minimum (use field lists, for example).
- Reduce the number of database queries (use table joins instead of nested SELECT statements, for example).
- Reduce search size (this optimizes access to the database).
- Minimize database server load (use SAP buffers, for example).

You should always try to maintain good performance for your programs. You should always carry out a runtime analysis (transaction SE30), or an SQL trace (transaction ST05) if the program includes SQL statements.

Reading Multiple Database Tables

Database View in the ABAP Dictionary

```
REPORT bc405_select_d1.
...
SELECT carrid carrname connid
       cityfrom cityto fldate
       seatsmax seatsocc
  INTO TABLE gt_flights
  FROM sv_flights
  WHERE cityfrom IN so_cityf
    AND cityto IN so_cityt
    AND seatsocc < sv_flights~seatsmax
  ORDER BY carrid connid fldate.
...
```

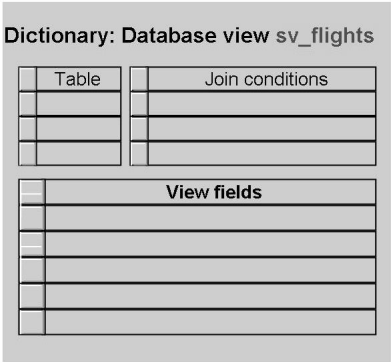


Figure 145: Reading Multiple Database Tables I

You can create database views in the ABAP Dictionary. Views are typically application-specific and allow you to work with multiple database tables. Database views are realized on the database as an INNER JOIN.

Database views can be buffered. You can then read from views using the SAP system buffer on the relevant application server. The same rules apply when buffering views as when buffering tables (mainly reads).

Database view advantages:

- Central maintenance
- Accessible to all users
- Only one `SELECT` statement is required in the program

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

One disadvantage of the view is its low flexibility.

To create database view, go to the ABAP Dictionary. Here, create a view of the *Database view* type. On the *Table/Join Conditions* tab page, enter the appropriate tables. After you have entered the first table, you can usually select further tables using the *Relationships* button and the system will propose the join conditions. On the *View Flds* tab page, enter the fields used. Make the work easier by using the *Table fields* button. Save and activate the database view.

INNER JOIN

```
REPORT bc405_select_d2.
...
SELECT spfli~carrid spfli~connid
       spfli~cityfrom spfli~cityto
       sflight~fldate sflight~seatsmax
       sflight~seatsocc
  INTO TABLE gt_flights
  FROM spfli INNER JOIN sflight
    ON spfli~carrid = sflight~carrid
   AND spfli~connid = sflight~connid
     WHERE spfli~carrid IN so_carr
       AND spfli~connid IN so_conn.
...
```

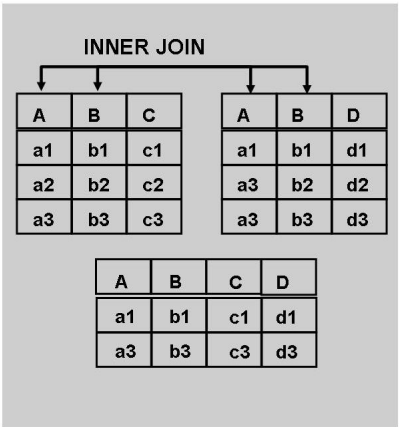


Figure 146: Reading Multiple Database Tables IIa

The transparent database tables are combined to form one results table to represent the resulting set of a join. The join conditions are applied to this results table. The resulting set for an inner join logic contains only those records that have matching records in each transparent table.

Join conditions are not limited to key fields. They can be declared using any fields for which a foreign key relationship is defined or, in some cases, between fields that have a common domain.

If columns from two tables have the same name, you have to ensure that the field labels are unique by prefixing the table name or alias name.

A table join is an efficient way of reading from the database. The database decides which table is read first and which index is selected.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

OUTER JOIN

```
REPORT bc405_select_d4.
...
SELECT scarr~carrid scarr~carrname
       spfli~connid spfli~cityfrom
       spfli~cityto
 INTO TABLE gt_flights
 FROM  scarr LEFT OUTER JOIN spfli
   ON  scarr~carrid = spfli~carrid
   ORDER BY scarr~carrid spfli~connid.
...
```

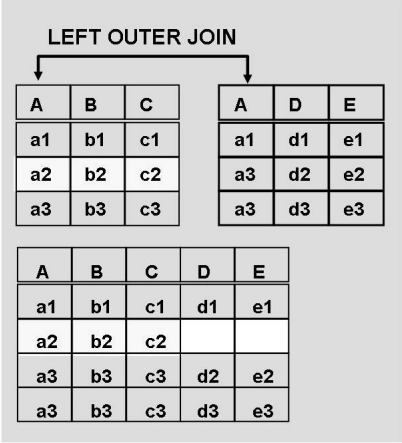


Figure 147: Reading Multiple Database Tables IIb

For a `LEFT OUTER JOIN`, results tables can also contain entries from the designated left-hand table without the presence of corresponding data records (join conditions) from the table on the right. These table fields are filled by the database with null values. The null values corresponding to the ABAP type are initialized before being transferred to your program.

It makes sense to use a `LEFT OUTER JOIN` when data from the table on the left is needed and there are no corresponding entries in the table on the right. For example: not all airlines (table *SCARR*) have flights listed (table *SPFLI*), but all airline names are supposed to be displayed in the list.

The following limitations apply to `LEFT OUTER JOIN`:

- You can only have a table or a view to the right of the `JOIN` operator; you cannot have another join statement.
- Only `AND` can be used as a logical operator in the `ON` condition.
- Every comparison in the `ON` condition must contain a field from the table on the right.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

FOR ALL ENTRIES

```
REPORT  bc405_select_d5.
... .
SELECT carrid connid ...
 INTO TABLE gt_spfli FROM spfli
  WHERE ... .

* Check whether at least one record was found
IF sy-subrc ne 0. EXIT. ENDIF.


* Delete duplicates
SORT gt_spfli.
DELETE ADJACENT DUPLICATES FROM gt_spfli.

SELECT  carrid connid fldate ...
  INTO TABLE gt_flights FROM sflight
   FOR ALL ENTRIES IN gt_spfli
    WHERE carrid = gt_spfli-carrid
    AND   connid = gt_spfli-connid.
.... .
```

itab_splfi

LH	0400	...
LH	0402	...

Implementation

```
where (     carrid = 'LH'
        and connid = '0400')

or    (     carrid = 'LH'
        and connid = '0402')
or    ( . . .
```

Figure 148: Reading Multiple Database Tables III

`FOR ALL ENTRIES` works with the database in a set-oriented way. Initially, all data is collected in an internal table. You must evaluate the value of `SY-SUBRC` after each `SELECT` statement. If the result of the first array fetch returns no data, the following `SELECT` statements will retrieve all entries from the database.

The `SELECT...FOR ALL ENTRIES IN <itab>` is generally mapped in a `SELECT` statement with an external `OR` link (depending on optimizer and profile parameters). The system only selects those table entries that meet the logical condition (`WHERE carrid = itab_sflight-carrid`), replacing the placeholders (`itab_spfli-carrid`) with values from each entry in the internal table `itab_spfli`. Note that `itab_spfli-carrid` is a placeholder, and not a component of the internal table. Duplicates are prevented by `DELETE ADJACENT DUPLICATES`. The internal table can, in principle, be as large as you want it to be.

Using `FOR ALL ENTRIES` is recommended when data is not being read from the database, that is, it is already available in the program, for example, if the user has input the data. Otherwise a join is recommended.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Nested SELECT Statements

```
REPORT  bc405_select_d6.
... .
 SELECT carrid connid cityfrom ...
  INTO gs_spfli FROM spfli
    WHERE cityfrom IN so_cityf
    AND   cityto   IN so_cityt.
      APPEND gs_spfli TO gt_spfli.
   SELECT  carrid connid fldate ...
    INTO gs_flight FROM sflight
      WHERE carrid = gs_spfli-carrid
      AND   connid = gs_spfli-connid.
       APPEND gs_flight TO gt_flights.
     SELECT carrid connid fldate bookid ...
      INTO gs_booking FROM sbook
        WHERE carrid = gs_flight-carrid
        AND   connid = gs_flight-connid
        AND   fldate = gs_flight-fldate.
         APPEND gs_booking TO gt_bookings.
     ENDSELECT.
   ENDSELECT.
 ENDSELECT.
... .
```

Figure 149: Reading Multiple Database Tables IV

Another method for reading related tables is to use nested `SELECT` statements. This technique should be avoided and used only as a last resort when no other data retrieval method is possible. The biggest disadvantage of the nested `SELECT` statements is that the inner `SELECT` is executed for each record retrieved by the outer `SELECT` statements. Under these circumstances, good performance is only possible if you make sure that the table with the fewest expected records is located furthest to the outside.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Exercise 16: Data Retrieval with the Inner Join

Exercise Objectives

After completing this exercise, you will be able to:

- Use an ABAP join to read data from several different DB tables

Business Example

You are a programmer for ABC Travel. The manager needs a list that shows which plane types are used for which flights.

Task 1:

Inner join with two tables

1. Create the *ZBC405_##_INNER_JOIN* program.

   Model Solution: *BC405_SELECT_S1*.

2. Create a work area and a corresponding internal table in which the following fields of the *SPFLI* (flight connections) and *SFLIGHT* (flights) tables can be included:

Technical name	Description
SPFLI~CARRID	Airline
SPFLI~CONNID	Flight connection number
SFLIGHT~FLDATE	Flight date
SPFLI~CITYFROM	Departure city
SPFLI~CITYTO	Arrival city
SPFLI~FLTIME	Flight duration
SFLIGHT~PLANETYPE	Plane type
SFLIGHT~SEATSOCC	Occupied economy class seats
SFLIGHT~SEATSMAX	Economy class capacity

   Link the tables using an `INNER JOIN` and read the data in the internal table.

   Create an ALV using the FACTORY method of the *CL_SALV_TABLE* class and display this by calling the DISPLAY method of the same class.

   Add a selection option for airlines and evaluate this in your `SELECT` statement.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Activate and test your program.

Task 2:

Optional: Inner join with three tables

1. Enhance your *ZBC405_##_INNER_JOIN* program or copy the sample solution *BC405_SELECT_S1* from the previous exercise.

   Model solution for the task: *BC405_SELECT_S1_OPT*

2. Expand your work area and your table to include a column for the *PLANETYPE, OP_SPEED, SPEED_UNIT* fields of the SAPLANE table.

   Expand your join accordingly.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Solution 16: Data Retrieval with the Inner Join

Task 1:

Inner join with two tables

1. Create the *ZBC405_##_INNER_JOIN* program.

   Model Solution: *BC405_SELECT_S1*.

   a) Self-explanatory

2. Create a work area and a corresponding internal table in which the following fields of the *SPFLI* (flight connections) and *SFLIGHT* (flights) tables can be included:

Technical name	Description
SPFLI~CARRID	Airline
SPFLI~CONNID	Flight connection number
SFLIGHT~FLDATE	Flight date
SPFLI~CITYFROM	Departure city
SPFLI~CITYTO	Arrival city
SPFLI~FLTIME	Flight duration
SFLIGHT~PLANETYPE	Plane type
SFLIGHT~SEATSOCC	Occupied economy class seats
SFLIGHT~SEATSMAX	Economy class capacity

   Link the tables using an `INNER JOIN` and read the data in the internal table.

   Create an ALV using the FACTORY method of the *CL_SALV_TABLE* class and display this by calling the DISPLAY method of the same class.

   Add a selection option for airlines and evaluate this in your `SELECT` statement.

   Activate and test your program.

   a) See result.

Task 2:

Optional: Inner join with three tables

1. Enhance your *ZBC405_##_INNER_JOIN* program or copy the sample solution *BC405_SELECT_S1* from the previous exercise.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Model solution for the task: *BC405_SELECT_S1_OPT*

a) Proceed in the usual manner.

2. Expand your work area and your table to include a column for the *PLANETYPE, OP_SPEED, SPEED_UNIT* fields of the SAPLANE table.

   Expand your join accordingly.

   a) See the items indicated as optional in the result.

Result

Main program

```
*&---------------------------------------------
*&   Solution: Data retrieval with inner join
*&   and three tables
*&---------------------------------------------
REPORT bc405_select_s1_opt.

* work area type for join
TYPES: BEGIN OF gty_join,
        carrid     TYPE spfli-carrid,
        connid     TYPE spfli-connid,
        fldate     TYPE sflight-fldate,
        cityfrom   TYPE spfli-cityfrom,
        cityto     TYPE spfli-cityto,
        fltime     TYPE spfli-fltime,
        seatsmax   TYPE sflight-seatsmax,
        seatsocc   TYPE sflight-seatsocc,
****************Begin of optional part****************
        planetype  TYPE saplane-planetype,
        op_speed   TYPE saplane-op_speed,
        speed_unit TYPE saplane-speed_unit,
****************End of optional part*****************
      END OF gty_join.

DATA: gt_join TYPE TABLE OF gty_join,
      gs_join TYPE          gty_join.

DATA: go_alv_grid TYPE REF TO cl_salv_table.

SELECT-OPTIONS: so_car FOR gs_join-carrid MEMORY ID car.


****************************************************
```

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
START-OF-SELECTION.
* retrieve data
  SELECT
      spfli~carrid spfli~connid
      fldate
      cityfrom cityto fltime
      sflight~seatsmax seatsocc

****************Begin of optional part****************
      sflight~planetype op_speed speed_unit
****************End of optional part*****************

    INTO CORRESPONDING FIELDS OF TABLE gt_join
    FROM spfli INNER JOIN sflight
    ON spfli~carrid = sflight~carrid AND
       spfli~connid = sflight~connid

****************Begin of optional part****************
    INNER JOIN saplane
    ON sflight~planetype = saplane~planetype
****************End of optional part*****************

    WHERE spfli~carrid IN so_car.


* create ALV
  cl_salv_table=>factory(
    IMPORTING
      r_salv_table   = alv
    CHANGING
      t_table        = gt_join ).

* display data
  alv->display( ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Implement database accesses across multiple tables
- Rate the different techniques with regard to optimized performance

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: Further Additions to the SELECT Statement

Lesson Overview

In this lesson, you learn how you can perform groupings of several data records with *SELECT*, and how you can sort the data records at *SELECT*.

Lesson Objectives

After completing this lesson, you will be able to:

- Use other additions of the SELECT statement

Business Example

You are working as a programmer for ABC Travel. The directors want an overview of the occupied and free seats for every flight connection. To do this, you need to read the individual flights that belong to the corresponding flight connection, and add up the desired values. It is your task to do this with a minimum of data transfer between the database server and the application server.

Other Additions for the SELECT Statement

The additions to the `SELECT` statement introduced in this lesson make it possible get data from the database efficiently and in a consolidated manner.

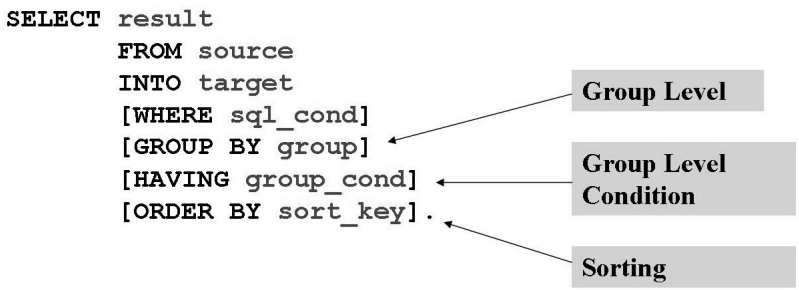


Figure 150: Further Additions to SELECT Statement

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Aggregate Functions

`MAX`	Returns the maximum value of the column
`MIN`	Returns the minimum value of the column
`AVG`	Returns the average value of the column
`SUM`	Returns the sum of the column
`COUNT`	Counts values or rows as follows:
`COUNT( DISTINCT s )`	Returns the number of different values in column `s`.
`COUNT( * )`	Returns the total number of rows in the resulting set

Figure 151: Overview of the Aggregate Functions

In the `SELECT` statement, you can choose how many entered columns are to be listed as arguments of the aggregate expressions above. In aggregate expressions values from multiple rows are calculated to produce a single value (the `DISTINCT` addition prevents values from occurring more than once):

1. *MAX([DISTINCT] col)* determines the maximum value of the values of column *col* in the resulting set or the current group.
2. *MIN([DISTINCT] col)* determines the minimum value of the values of column *col* in the resulting set or the current group.
3. *AVG([DISTINCT] col)* determines the average value of the values of column *col* in the resulting set or the current group. The data type of the column must be numerical.
4. *SUM([DISTINCT] col)* determines the total of the values of column *col* in the resulting set or the current group. The data type of the column must be numerical.
5. *COUNT([DISTINCT] col)* determines the number of different values of column *col* in the resulting set or the current group.
6. *COUNT(*)* (or *count(*)*) determines the number of rows of the resulting set or the current group. In this case, no column name is specified.

SELECT – GROUP BY: Control level

Syntax: ... ... *GROUP BY {col1 col2 ...}*

The `GROUP BY` addition calculates groups of rows in the given columns *col1 col2 ...* which have the same contents and the resulting set is a single row. When you use `GROUP BY`, this means that only individual columns are entered behind `SELECT`, not all columns, using *. If you are using `GROUP BY`, all columns which are directly entered behind `SELECT` and not entered as an argument of an aggregate function must be listed. On the other hand, if you are using `GROUP BY`, each column listed behind `SELECT` which is not listed behind `GROUP BY` must

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

be entered as the argument of an aggregate function. The aggregate functions (such as number of rows, total) determine how the contents of this column in the combined row are calculated from the contents of all rows of a group. The same column indicators must be entered behind `GROUP BY` as are entered behind `SELECT`. The columns entered behind `GROUP BY` cannot be of the *STRING* or *RAWSTRING* type and if you are using `GROUP BY`, they cannot be accessed on pooled or cluster tables.

If you are using aggregate expressions, all column indicators which are not listed as the argument of an aggregate function must be listed behind the `GROUP BY` addition. The aggregate functions evaluate the contents of groups in the database system defined using `GROUP BY` and transfer the result into the calculated rows of the resulting set.

The data type of aggregate expressions with the functions *MAX*, *MIN*, *SUM* is the data type of the corresponding column in the ABAP Dictionary. Aggregate expressions with the function *AVG* have the data type *FLTP* and those with *COUNT* have the data type *INT4*. You need to specify an appropriate corresponding data object after *INTO* or *APPENDING*.

GROUP BY and Aggregate Expressions

```
TYPES: BEGIN OF gty_flight,
        carrid      TYPE sflight-carrid,
        connid      TYPE sflight-connid,
        seatsmax    TYPE sflight-seatsmax,
        seatsocc    TYPE sflight-seatsocc,
        n_o_flights TYPE i,
       END OF gty_flight,

DATA: gt_flights TYPE TABLE OF gty_flight,
      gs_flight  TYPE          gty_flight.

SELECT-OPTIONS: so_car FOR gs_flight-carrid.

START-OF-SELECTION.

  SELECT
    carrid connid
    SUM( seatsmax )
    SUM( seatsocc )
    COUNT( * )
    INTO TABLE gt_flights
    FROM sflight
    WHERE carrid IN so_car
    GROUP BY carrid connid.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

GROUP BY with COUNT(DISTINCT <col>) and Column Aliases

```
TYPES: BEGIN OF gty_flight,
        carrid      TYPE sflight-carrid,
        num_connid  TYPE sflight-connid,
        sum_max     TYPE sflight-seatsmax,
        sum_occ     TYPE sflight-seatsocc,
        num_flights TYPE i,
       END OF gty_flight,

DATA: gt_flights TYPE TABLE OF gty_flight,
      gs_flight  TYPE          gty_flight.

SELECT-OPTIONS: so_car FOR gs_flight-carrid.

  SELECT
    carrid
    COUNT( DISTINCT connid ) AS num_connid
    SUM( seatsmax ) AS sum_max
    SUM( seatsocc ) AS sum_occ
    COUNT( * ) AS num_flights
    INTO CORRESPONDING FIELDS OF TABLE gt_flights
    FROM sflight
    WHERE carrid IN so_car
    GROUP BY carrid.
```

At field level assignment with `INTO CORRESPONDING FIELDS` (and later for sorting) you must assign aggregated columns a unique name. You can access this using the `AS` addition to a column in the `SELECT` clause.

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Airline	Flight	Departure	Arrival	Capacity	Occupied	
UA	3517	FRANKFURT	NEW YORK	6.545	4.467	17
LH	402	FRANKFURT	NEW YORK	6.460	4.333	17
LH	2402	FRANKFURT	BERLIN	6.460	4.366	17
UA	3504	SAN FRANCISCO	FRANKFURT	6.460	4.620	17
UA	3516	NEW YORK	FRANKFURT	6.460	4.536	17
AA	17	NEW YORK	SAN FRANCISCO	5.775	3.917	16
AZ	790	ROME	OSAKA	5.775	3.778	15
JL	408	FRANKFURT	TOKYO	5.775	3.700	15
AZ	788	ROME	TOKYO	5.700	3.704	15
AZ	789	TOKYO	ROME	5.700	3.761	15
DL	1984	SAN FRANCISCO	NEW YORK	5.700	3.982	15
JL	407	TOKYO	FRANKFURT	5.700	3.874	15
SQ	2	SINGAPORE	SAN FRANCISCO	5.700	3.873	15
SQ	15	SAN FRANCISCO	SINGAPORE	5.700	3.927	15
LH	400	FRANKFURT	NEW YORK	4.760	3.261	17
AA	64	SAN FRANCISCO	NEW YORK	4.200	2.737	15
DL	106	NEW YORK	FRANKFURT	4.200	2.626	15
QF	5	SINGAPORE	FRANKFURT	4.200	2.845	15
SQ	988	SINGAPORE	TOKYO	4.200	2.866	15
LH	401	NEW YORK	FRANKFURT	3.740	2.426	17
UA	941	FRANKFURT	SAN FRANCISCO	3.740	2.664	17
AZ	555	ROME	FRANKFURT	3.300	2.111	15
DL	1699	NEW YORK	SAN FRANCISCO	3.300	2.273	15

Figure 152: Result of a GROUP BY Database Query

SELECT – HAVING: Control Level Conditions

Syntax: `... HAVING <sql_cond> ...`

The `HAVING` addition limits the number of rows in the resulting set which can be calculated for groups using the logical expression *sql_cond* for these rows. The syntax of the logical expression *sql_cond* relates to the syntax of the logical expression of the `WHERE` condition. The comparisons of the logical expression evaluate the contents of row groups.

If a grouping occurs using the `GROUP BY` addition, all columns directly entered in the *sql_cond* condition using their names *col* must be listed behind `GROUP BY`. For any column of the database tables or views listed behind `FROM`, any aggregate expressions can be entered in the comparisons of *sql_cond*. This type of aggregate expression is evaluated for each row group defined in `GROUP BY` and the result is used as the operand of the comparison. If this type of column is listed simultaneously behind `SELECT` the aggregate expressions behind `SELECT` and behind `HAVING` can also differ.

HAVING Clause

```
TYPES: BEGIN OF gty_flight,
         carrid       TYPE sflight-carrid,
         seatsmax_avg TYPE sflight-seatsmax,
         seatsocc_avg TYPE sflight-seatsocc,
       END OF gty_flight,

DATA: gt_flights TYPE TABLE OF gty_flight,
      gs_flight  TYPE          gty_flight.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
SELECT-OPTIONS: so_car   FOR gs_flight-carrid.
SELECT-OPTIONS: so_avgmx FOR gs_flight-seatsmax,
                so_avgoc FOR gs_flight-seatsocc.

  SELECT
    carrid
    AVG( seatsmax ) AS seatsmax_avg
    AVG( seatsocc ) AS seatsocc_avg
    INTO CORRESPONDING FIELDS OF TABLE gt_flights
    FROM sflight
    WHERE carrid IN so_car
    GROUP BY carrid
    HAVING AVG( seatsmax ) IN so_avgmx
       AND AVG( seatsocc ) IN so_avgoc.
```

SELECT – ORDER BY: Sorts

Syntax:

ORDER BY { {PRIMARY KEY}

| { {col1|a1} [ASCENDING|DESCENDING]

{col2|a2} [ASCENDING|DESCENDING] ...}

The `ORDER BY` addition sorts the resulting set based on the contents of the entered columns. The sequence of the rows in the resulting set with regards to all columns that are not listed behind `ORDER BY` is undefined and can be different if there are multiple listings in the same `SELECT` statement.

PRIMARY KEY

If all columns are entered using * behind `SELECT`, and if a single database table (not a view or join expression) is entered behind `FROM`, the resulting set with the `PRIMARY KEY` addition can be sorted in ascending order based on the content of the primary key of this database table.

ORDER BY {col1|a1} [ASCENDING|DESCENDING]

A list of columns for any column specifications behind `SELECT` can be entered behind `ODER BY` according to which sorting should take place. Only columns which are listed behind `SELECT` can be entered. A column can be entered directly using the column indicators *col1 col2* or using the alternative column names *alias1 alias2* (`SELECT col1 AS alias1 ...`). The latter is necessary, if you want to sort by columns that are specified as aggregate expressions.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The ASCENDING and DESCENDING additions determine whether sorting takes place in ascending or descending order. If neither of these additions is specified, the system sorts in ascending order. The priority of the sorting follows the order in which the components *col1 col2 ...* or *alias1 alias2* are specified.

When you enter a single column in the ORDER BY addition, the SELECT statement handles the SAP buffering.

For performance reasons, you should only perform a sort in the database system if the sorting is supported by an index.

ORDER-BY Clause

```
SELECT
   carrid
   AVG( seatsmax ) AS seatsmax_avg
   AVG( seatsocc ) AS seatsocc_avg
   INTO CORRESPONDING FIELDS OF TABLE gt_flights
   FROM sflight
   WHERE carrid IN so_car
   GROUP BY carrid
   HAVING AVG( seatsmax ) IN so_avgmx
      AND AVG( seatsocc ) IN so_avgoc
   ORDER BY seatsmax_avg DESCENDING
            seatsocc_avg DESCENDING.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 17: Further Additions to the SELECT Statement: Aggregation and GROUP BY

Exercise Objectives

After completing this exercise, you will be able to:

- Control level processing at SELECT

Business Example

You are working as a programmer for ABC Travel. The manager needs a list of the flight connections and the capacity and occupancy of the economy class for each. The individual flights should not be displayed.

Task:

Output consolidated flight data using a program.

Program *ZBC405_##_GROUP_BY*.

Model Solution: *BC405_SELECT_S2*

1. Create the program *ZBC405_##_GROUP_BY*.
2. Read aggregated data from the database table *SFLIGHT* (flights) to an internal table. Display the following fields in each row:

Field name	Description
CARRID	Carrier name
CONNID	Connection number
SEATSMAX	Capacity of Economy Class for each connection
SEATSOCC	Utilization of Economy Class for each connection
NUM_FLIGHTS	Number of flights for each connection

   First define a suitable internal table and an associated work area.

   Now retrieve the data from the database and pass it to the internal table. Note when accessing the database that the values for some fields need to be combined across multiple rows.

3. Output the internal table with a full screen ALV.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 17: Further Additions to the SELECT Statement: Aggregation and GROUP BY

Task:

Output consolidated flight data using a program.

Program *ZBC405_##_GROUP_BY*.

Model Solution: *BC405_SELECT_S2*

1. Create the program *ZBC405_##_GROUP_BY*.

   a) Self-explanatory

2. Read aggregated data from the database table *SFLIGHT* (flights) to an internal table. Display the following fields in each row:

Field name	Description
CARRID	Carrier name
CONNID	Connection number
SEATSMAX	Capacity of Economy Class for each connection
SEATSOCC	Utilization of Economy Class for each connection
NUM_FLIGHTS	Number of flights for each connection

First define a suitable internal table and an associated work area.

Now retrieve the data from the database and pass it to the internal table. Note when accessing the database that the values for some fields need to be combined across multiple rows.

a) See result.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

3. Output the internal table with a full screen ALV.
   a) See result.

Result

Main program

```
REPORT bc405_select_s2.

TYPES: BEGIN OF gty_flight,
         carrid      TYPE sflight-carrid,
         connid      TYPE sflight-connid,
         seatsmax    TYPE sflight-seatsmax,
         seatsocc    TYPE sflight-seatsocc,
         num_flights TYPE i,
       END OF gty_flight.

DATA: gt_flights TYPE TABLE OF gty_flight,
      gs_flight  TYPE          gty_flight.

DATA: go_alv_grid TYPE REF TO cl_salv_table.

SELECT-OPTIONS: so_car FOR gs_flight-carrid.


START-OF-SELECTION.

  SELECT
      carrid
      connid
      SUM( seatsmax )
      SUM( seatsocc )
      COUNT( * )
    INTO TABLE gt_flights
    FROM sflight
    WHERE carrid IN so_car
    GROUP BY carrid connid.

  cl_salv_table=>factory(
    IMPORTING
      r_salv_table   = go_alv_grid
    CHANGING
      t_table        = gt_flights ).

  go_alv_grid->display( ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 18: Further Additions to the SELECT Statement: GROUP BY with COUNT (DISTINCT)

Exercise Objectives

After completing this exercise, you will be able to:

- Use COUNT(DISTINCT <column>)
- Use the HAVING clause

Business Example

You are working as a programmer for ABC Travel. The directors require a list of the flight connections and their respective capacity and utilization in Economy Class, and the respective number of associated connections and flights. The individual connections and flights are not to be displayed.

Task:

Write a program that displays consolidated flight data and connections.

Program *ZBC405_##_GROUP_BY2*.

Model Solution: *BC405_SELECT_S3*

1. Create the program *ZBC405_##_GROUP_BY2*.
2. Read aggregated data from the database table *SFLIGHT* (flights) to an internal table. The internal table should store the following information:

Field name	Description
CARRID	Carrier
SEATSMAX	Total capacity of Economy Class for each airline
SEATSOCC	Total utilization of Economy Class for each airline
SEATSMAX_AVG	Average capacity of Economy Class for each airline
SEATSOCC_AVG	Average utilization of Economy Class for each airline
NUM_CONNID	Number of connections for each airline
NUM_FLIGHTS	Number of flights for each airline

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

First define a suitable internal table and an associated work area.

Now retrieve the data from the database and pass it to the internal table. Note when accessing the database that the values for some fields need to be combined across multiple rows.

Tip: The *COUNT(*)* option is not suited to determining the number of connections per airline.

3. Allow the user to restrict the retrieved data records using the selection screen. To do this, create three select options:

No.	Meaning
1	Airline
2	Average economy class capacity by airline
3	Average economy class occupancy by airline

Ensure that the entries in the select option fields are taken into account at *SELECT*.

Hint: The *WHERE* condition is not suited to checking aggregated values against restrictions.

4. Sort the records by the following criteria during the data retrieval:

No.	Field	Sort direction
1	Average capacity of Economy Class for each airline	Descending
2	Average economy class occupancy by airline	Descending

5. Output the internal table with a full screen ALV.

Optional: For ALV, format the column headers for the aggregated values.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 18: Further Additions to the SELECT Statement: GROUP BY with COUNT (DISTINCT)

Task:

Write a program that displays consolidated flight data and connections.

Program *ZBC405_##_GROUP_BY2*.

Model Solution: *BC405_SELECT_S3*

1. Create the program *ZBC405_##_GROUP_BY2*.

   a) Self-explanatory

2. Read aggregated data from the database table *SFLIGHT* (flights) to an internal table. The internal table should store the following information:

Field name	Description
CARRID	Carrier
SEATSMAX	Total capacity of Economy Class for each airline
SEATSOCC	Total utilization of Economy Class for each airline
SEATSMAX_AVG	Average capacity of Economy Class for each airline
SEATSOCC_AVG	Average utilization of Economy Class for each airline
NUM_CONNID	Number of connections for each airline
NUM_FLIGHTS	Number of flights for each airline

   First define a suitable internal table and an associated work area.

   Now retrieve the data from the database and pass it to the internal table. Note when accessing the database that the values for some fields need to be combined across multiple rows.

   Tip: The *COUNT(*)* option is not suited to determining the number of connections per airline.

   a) See result at the end.

3. Allow the user to restrict the retrieved data records using the selection screen. To do this, create three select options:

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

No.	Meaning
1	Airline
2	Average economy class capacity by airline
3	Average economy class occupancy by airline

Ensure that the entries in the select option fields are taken into account at *SELECT*.

Hint: The *WHERE* condition is not suited to checking aggregated values against restrictions.

a) See result at the end.

4. Sort the records by the following criteria during the data retrieval:

No.	Field	Sort direction
1	Average capacity of Economy Class for each airline	Descending
2	Average economy class occupancy by airline	Descending

a) See result at the end.

5. Output the internal table with a full screen ALV.

Optional: For ALV, format the column headers for the aggregated values.

a) See result at the end.

Result

Main program

```
REPORT bc405_select_s3.

TYPES: BEGIN OF gty_flight,
         carrid       TYPE sflight-carrid,
         seatsmax     TYPE sflight-seatsmax,
         seatsocc     TYPE sflight-seatsocc,
         seatsmax_avg TYPE sflight-seatsmax,
         seatsocc_avg TYPE sflight-seatsocc,
         num_conn     TYPE i,
         num_flights  TYPE i,
       END OF gty_flight.
```

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
DATA: gt_flights TYPE TABLE OF gty_flight,
      gs_flight  TYPE          gty_flight.

DATA: go_alv_grid TYPE REF TO cl_salv_table.

SELECT-OPTIONS: so_car FOR gs_flight-carrid.
SELECTION-SCREEN
  BEGIN OF BLOCK bl1 WITH FRAME TITLE text-bl1.
SELECT-OPTIONS:
  so_avgmx FOR gs_flight-seatsmax,
  so_avgoc FOR gs_flight-seatsocc.
SELECTION-SCREEN END OF BLOCK bl1.


*---------------------------------------------------------------------------*
START-OF-SELECTION.
  SELECT
      carrid
      COUNT( DISTINCT connid ) AS num_conn
      SUM( seatsmax ) AS seatsmax
      SUM( seatsocc ) AS seatsocc
      AVG( seatsmax ) AS seatsmax_avg
      AVG( seatsocc ) AS seatsocc_avg
      COUNT( * ) AS num_flights
    INTO CORRESPONDING FIELDS OF TABLE gt_flights
    FROM sflight
    WHERE carrid IN so_car
    GROUP BY carrid
    HAVING AVG( seatsmax ) IN so_avgmx
       AND AVG( seatsocc ) IN so_avgoc
    ORDER BY
      seatsmax_avg DESCENDING
      seatsocc_avg DESCENDING.

* create ALV
  cl_salv_table=>factory(
    IMPORTING
      r_salv_table   = go_alv_grid
    CHANGING
      t_table        = gt_flights ).


*******************************************************
* To define headers (ALV OM), the following coding has
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* to be added (optional part)
*******************************************************
  DATA go_columns TYPE REF TO cl_salv_columns_table.
  DATA go_column  TYPE REF TO cl_salv_column.

* defining headers for ALV columns
  go_columns = go_alv_grid->get_columns( ).

* number of connections
  go_column ?= go_columns->get_column(
               columnname = 'NUM_CONN' ).
  go_column->set_short_text(
    value = 'Connect.'(con) ).

* number of flights
  go_column ?= go_columns->get_column(
               columnname = 'NUM_FLIGHTS' ).
  go_column->set_short_text(
    value = 'Flights'(flg) ).

* average of seatsmax
  go_column ?= go_columns->get_column(
               columnname = 'SEATSMAX_AVG' ).
  go_column->set_short_text(
    value = 'Avg.Cap.'(avm) ).

* average of seatsocc
  go_column ?= go_columns->get_column(
               columnname = 'SEATSOCC_AVG' ).
  go_column->set_short_text(
    value = 'Avg.occ.'(avo) ).

*******************************************************

* display data
  go_alv_grid->display( ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Use other additions of the SELECT statement

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Implement database accesses across multiple tables
- Rate the different techniques with regard to optimized performance
- Use other additions of the SELECT statement

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Unit 9

Call Programs with Data Transfer

Internal Use SAP Partner Only

Unit Overview

This unit explains how executable programs and transactions can be called from other programs. In addition, the data transfer between the programs is examined.

Unit Objectives

After completing this unit, you will be able to:

- Describe different memory areas
- Call executable programs
- Call transactions
- Pass data using memory areas

Unit Contents

Lesson: Calling Programs and Passing Data 316
Exercise 19: Calling Programs and Passing Data 333

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: Calling Programs and Passing Data

Lesson Overview

This lesson introduces various techniques for calling programs and techniques for passing data from program to program.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe different memory areas
- Call executable programs
- Call transactions
- Pass data using memory areas

Business Example

You are working as a programmer for ABC Travel. Your manager wants to display flight data. He wants to be able to start a detailed report that displays the bookings for a selected row by double-clicking the row.

Overview: Calling Programs and Passing Data

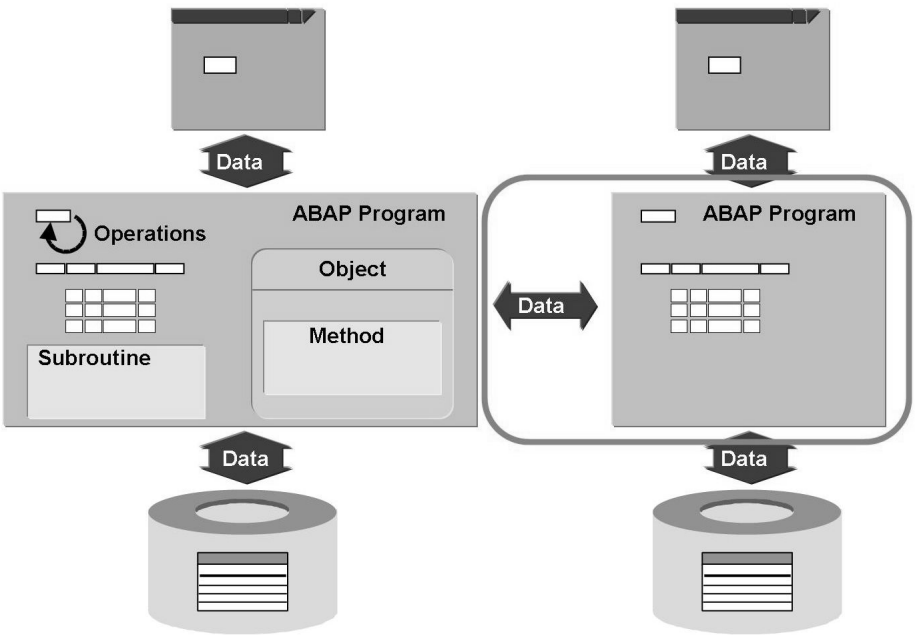


Figure 153: Overview: Calling Programs and Passing Data

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

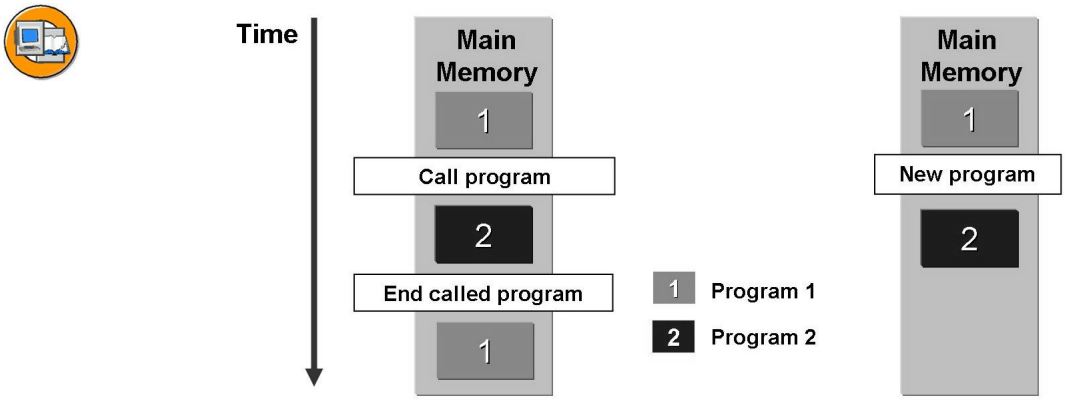


Figure 154: Synchronous Program Calls

There are two ways to call an ABAP program call from another ABAP program that is already running:

- Interrupt the current program to run the new program. The called program is executed, and processing then returns to the calling program.
- End the current program and start the new program.

Complete ABAP programs in a single user session can only run sequentially. This technique is known as synchronous calls.

If you want to run functions in parallel, you need to use function modules. For more information about this technique, see the documentation for `CALL FUNCTION... STARTING NEW TASK ... .`

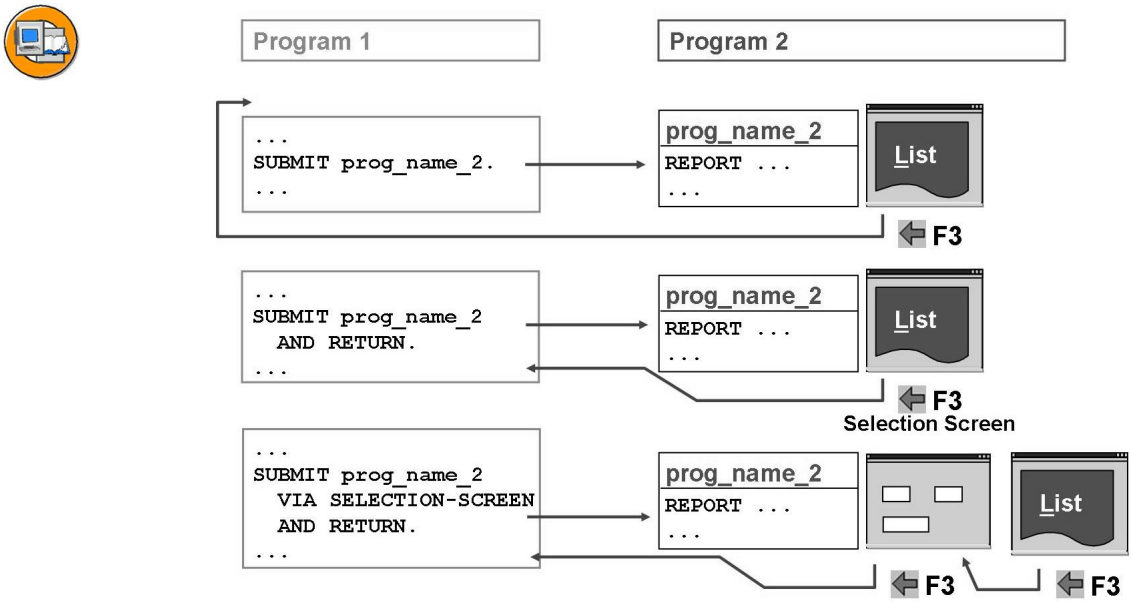


Figure 155: Running Executable Programs

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

To start an executable program (type 1), use the statement `SUBMIT`.

If you use the addition `VIA SELECTION-SCREEN`, the system displays the standard selection screen of the program (if one has been defined).

If you use the addition `AND RETURN`, the system resumes processing at the first statement after the `SUBMIT` statement, as soon as the called program is complete.

For more information, see the documentation for the `SUBMIT` statement.

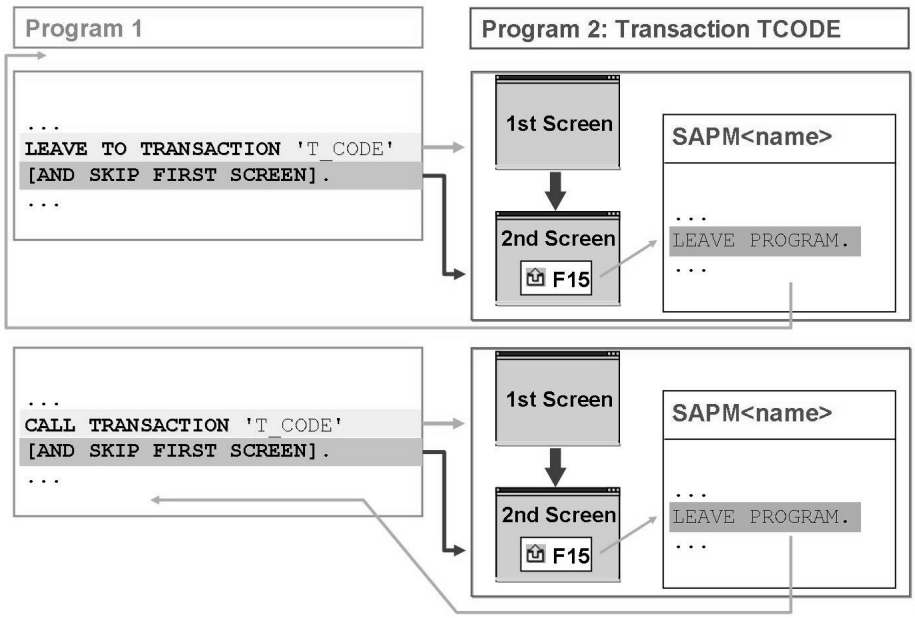


Figure 156: Calling a Transaction

When you use the `LEAVE TO TRANSACTION <t_code>` statement, the system terminates the current program and starts the transaction with transaction code `<t_code>`. The statement has the same meaning as entering **`/n<t_code>`** in the command field.

`CALL TRANSACTION <t_code>` allows you to insert an ABAP program with a transaction code into the call chain.

You can use the `LEAVE PROGRAM` statement to force the termination of an ABAP program. If the statement occurs in a program that you called using `CALL TRANSACTION <t_code>` or `SUBMIT <prog_name> AND RETURN`, the system resumes processing at the next statement after the call in the calling program.

In all other cases, the user returns to the application menu from which the original program was started.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

You can use the addition `AND SKIP FIRST SCREEN` to prevent the system displaying the contents of the first transaction screen. However, it still processes the flow logic.

If you started a transaction with `CALL TRANSACTION` that uses update techniques, you can use the addition `UPDATE` to specify the update technique that the program should use [(asynchronous (default), synchronous, or local)]. For more information, see the online documentation.

Memory Management

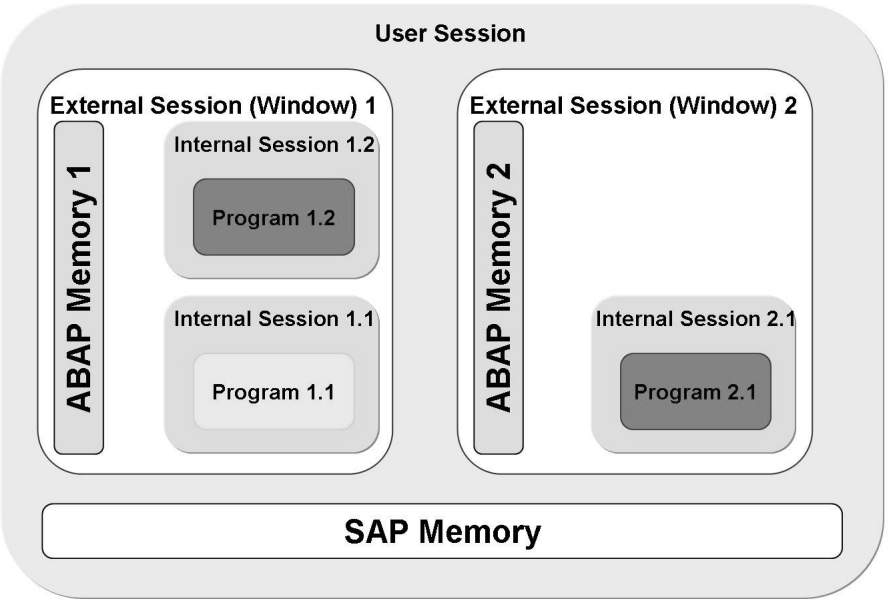


Figure 157: Logical Memory Model

The way the main memory is organized from the program's point of view can be easily illustrated with the above model. The model differentiates between internal and external sessions:

- An **external session** is usually an SAP GUI window. You create a new session by choosing the menu path *System → Create Session*, or by entering **`/o<T_CODE>`** in the command field. By default, you may have a maximum of six external session open simultaneously in one user session.
- External sessions are divided into **internal sessions** (which have been created on a **stack**). Each program that you run occupies its own internal session. Each external session can contain a maximum of nine internal sessions.

Each program that you start runs in an internal session. Data for the program is **only visible within** an internal session; data visibility is usually restricted to the relevant program.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The following pages show how the stack changes in an external session with various program calls.

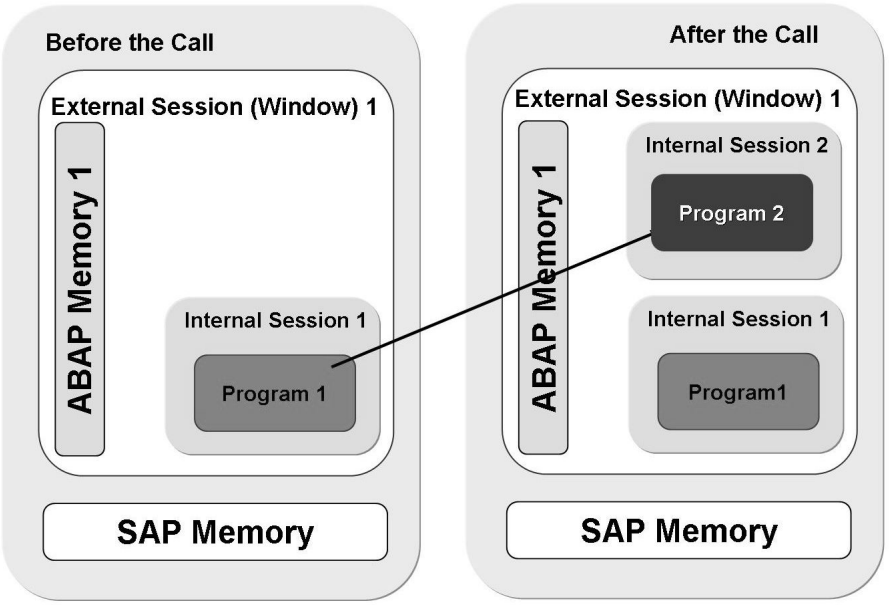


Figure 158: CALL TRANSACTION, SUBMIT ... AND RETURN

`CALL TRANSACTION <t_code>` or `SUBMIT <programm> AND RETURN` opens a new internal session that creates a new program context. The internal sessions of an external session create a memory stack. The new session is placed at the top of the stack.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

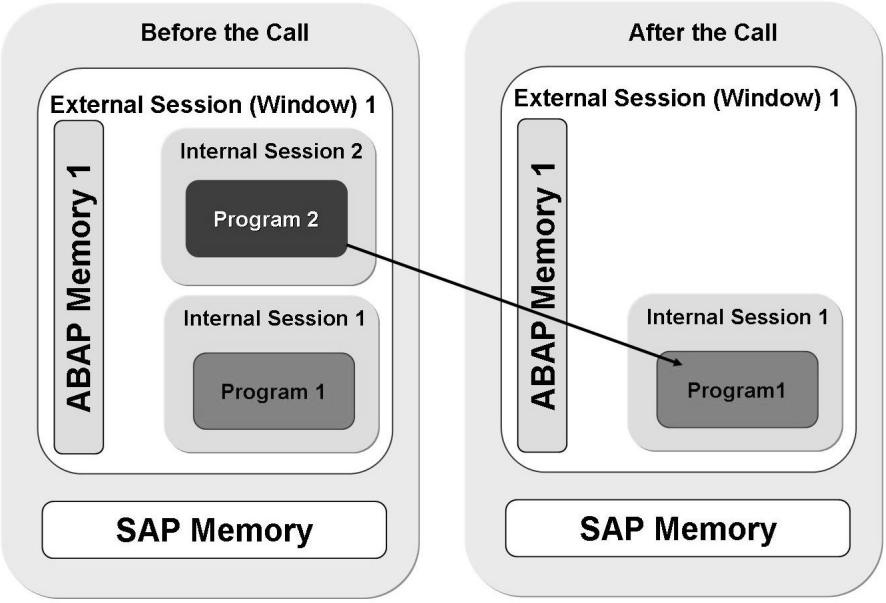


Figure 159: Implicit Exit from a Program / LEAVE PROGRAM

At the end of the execution of a program, the internal session that is uppermost in the stack is removed and the system continues processing the calling program.

The same behavior occurs if the statement `LEAVE PROGRAM` is explicitly processed.

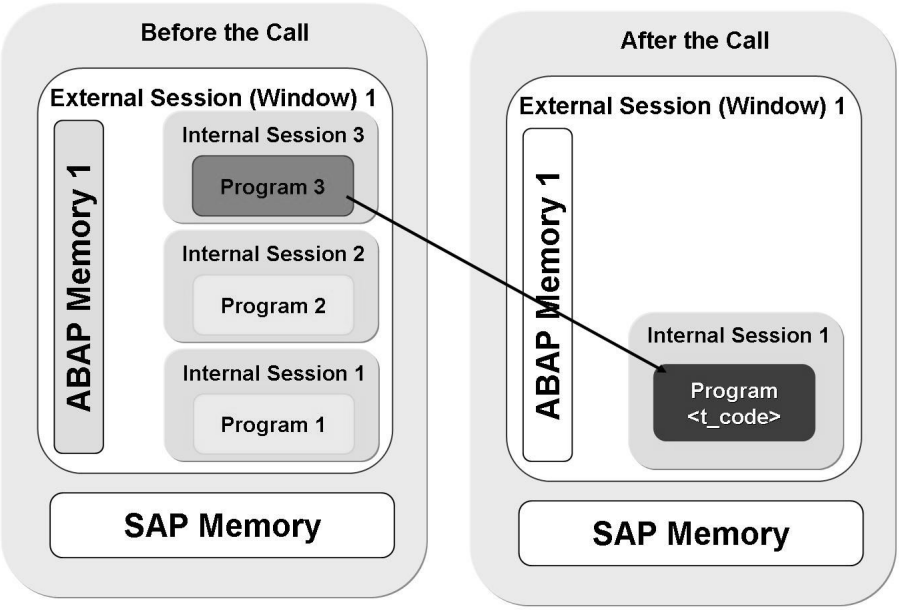


Figure 160: LEAVE TO TRANSACTION <t_code>

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

`LEAVE TO TRANSACTION` removes all internal sessions from the stack and opens a new internal session that contains the program context of the calling program.

After the program call, the ABAP memory is reset. All data in the ABAP memory is removed after the call.

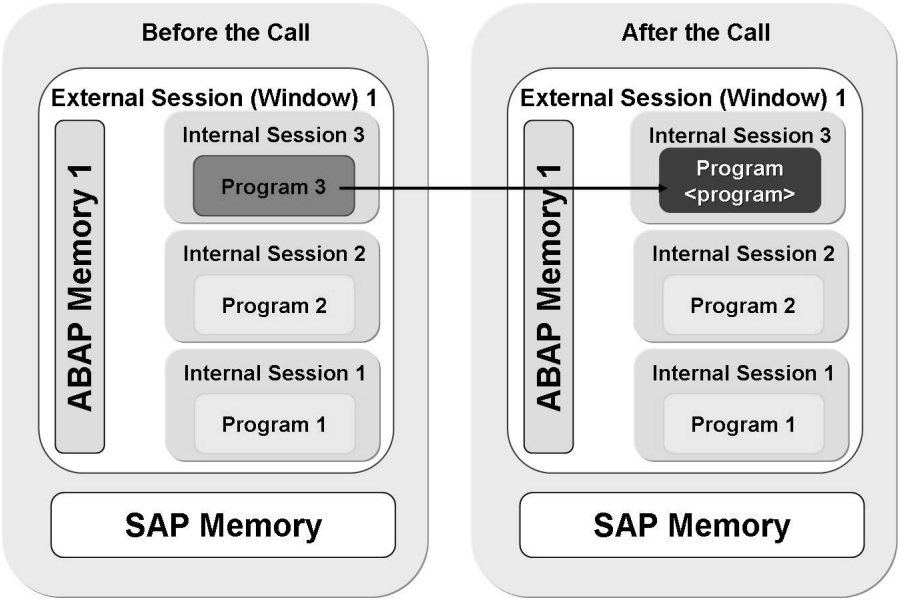


Figure 161: SUBMIT <program>

`SUBMIT <program>` replaces the internal session of the program performing the call with the internal session of the program that has been called. The new internal session contains the program context of the called program.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

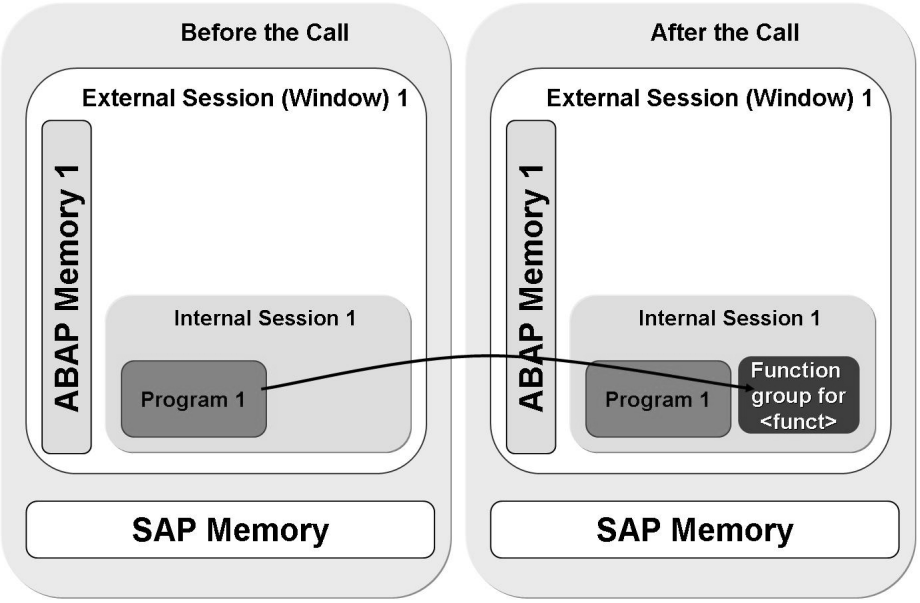


Figure 162: Calling Function Modules

When you call a function module, the following steps are performed:

- A check determines whether your program has already called a function module from the same function group.
- If not, the system loads the associated function group into the internal session of the calling program as an additional program group. This initializes the global data of the function group.
- If your program has already used a function module from the same function group, the function module that has now been called can access the global data of the function group. The system does not need to load the function group again.

All called function modules of the same function group access the global data of the function group within the internal session.

The function group needs to be loaded in every internal session used. If a new internal session calls a function, the called function group needs to be loaded in the new internal session.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Passing Data Between Programs

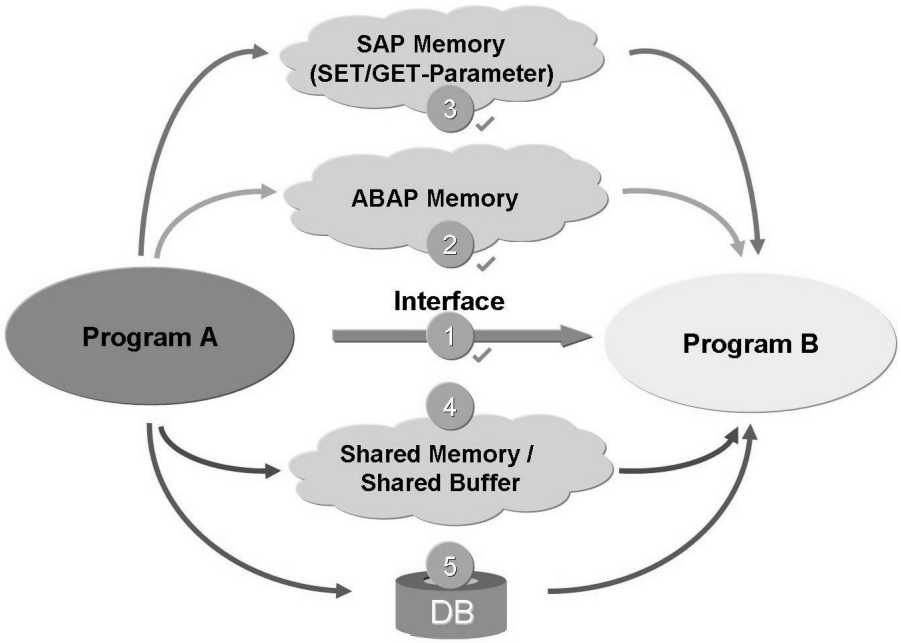


Figure 163: Overview: Passing Data Between Programs

There are various ways of passing data to programs running in separate internal sessions:

You can use the following:

- The interface of the called program (usually a standard selection screen)
- ABAP memory
- SAP memory
- Shared memory / Shared buffer
- Database tables
- Local files on your presentation server

The following pages illustrate the first three methods listed.

For more information about passing data using database tables and using the *shared memory* or the *shared buffer*, see the documentation for the statements `EXPORT` and `IMPORT`.

For more information about transferring data between an ABAP program and its presentation server, see the documentation for the function modules `GUI_UPLOAD` and `GUI_DOWNLOAD` or the documentation for the class `CL_GUI_FRONTEND_SERVICES`.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

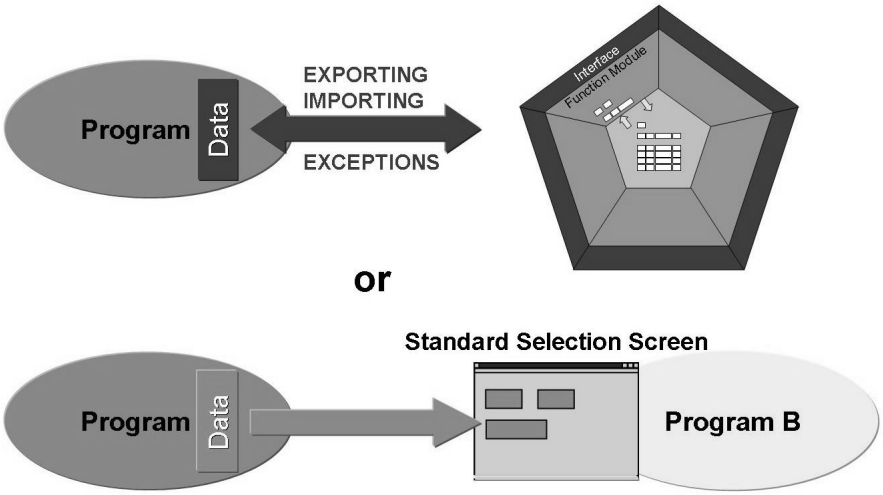


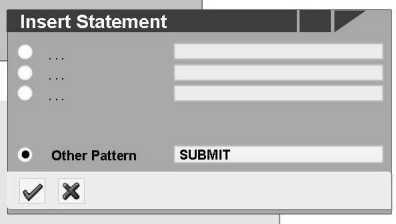
Figure 164: Passing Data Using the Program Interface

Function modules have an interface that the calling program and the function module use to exchange data. Subroutines use a similar technique. There are restrictions when using the interface of a remote-enabled function module. If you call an ABAP program with a standard selection screen, you can pass data for the input fields in the call. There are two ways to do this:

- By specifying a variant for the selection screen in the program call
- By specifying values for the input fields in the program call

```
DATA lt_set TYPE|LIKE RANGE OF type|dataobject}.

SUBMIT prog_name AND RETURN [VIA SELECTION-SCREEN]
WITH parameter   EQ|NE|... val
WITH sel_opt     EQ|NE|... val        SIGN 'I'|'E'}
WITH sel_opt     BETWEEN val1 AND val2  SIGN 'I'|'E'}
WITH sel_opt NOT BETWEEN val1 AND val2  SIGN 'I'|'E'}
WITH sel_opt IN lt_set
... .
```




```
MODULE user_command_0200 Input.
  ...
  CASE sy-ucomm.
    WHEN 'CTRYFR'.
          SUBMIT sapbc406_tabd_hashed
                 WITH pa_ctry = sdyn_conn-countryfr
                 AND RETURN.
  ...
  ENDCASE.
ENDMODULE.                        " USER_COMMAND_0200  INPUT
```

Figure 165: Passing Values for Input Fields

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

You can use the addition `WITH` of the `SUBMIT` statement to assign values to a field on a standard selection screen. The keywords `EQ`, `NE`, ...; `I`, and `E` have the same meanings as for the selection criteria.

If you want to pass several selections to a selection option, you can use the `<data_object> TYPE|LIKE RANGE OF` statement to define a range table and add the selections to this internal table. You then pass the whole table to the executable program using `WITH <sel_opt> IN <data_object>`.

If you want to display the standard selection screen when the program is called, use the addition `VIA SELECTION-SCREEN`.

If you are working with `SUBMIT`, use the pattern function in the ABAP Editor to insert a suitable template for the desired program call. It automatically provides the names of the parameters and selection criteria that are available on the standard selection screen.

For more information about working with variants and other syntax variants of the `WITH` addition, see the documentation for the `SUBMIT` statement.

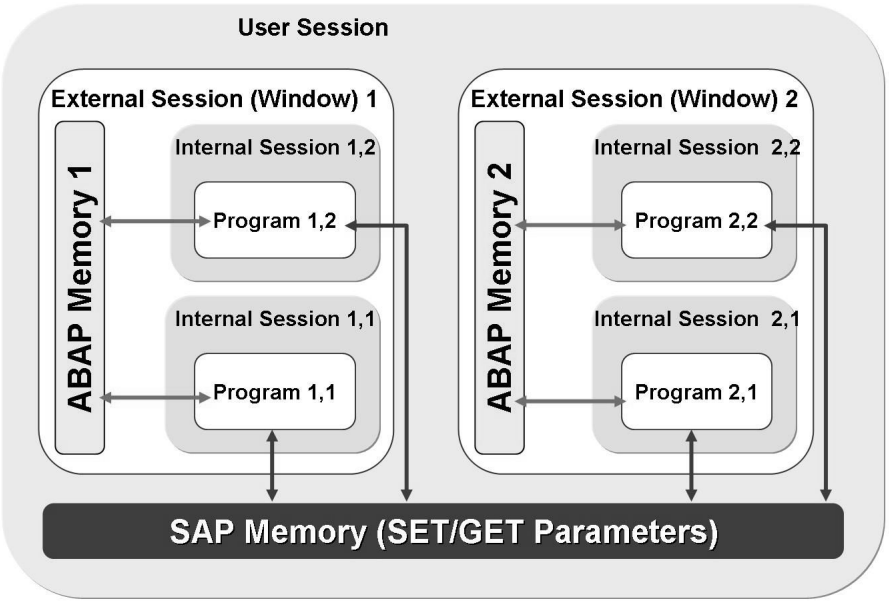


Figure 166: ABAP Memory and SAP Memory

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

You can use SAP memory and ABAP memory to pass data between programs.

- SAP memory is a user-specific memory area for storing field values that relate to parameter IDs. It is only suitable for passing data between internal sessions to a limited extent. Values in SAP memory are retained for the duration of a user session. The memory can be used between different sessions of the same user session. You can use the contents of the SAP memory as default values for screen fields. All external sessions can work with the SAP memory.
- The ABAP memory is also user-specific. There is a local ABAP memory for each external session. You can use it to exchange any ABAP variables (fields, structures, internal tables, complex objects) between the internal sessions of an external session. It cannot be shared with other external sessions.
- If the user ends an external session (by entering /i in the command field), the associated ABAP memory is automatically released.

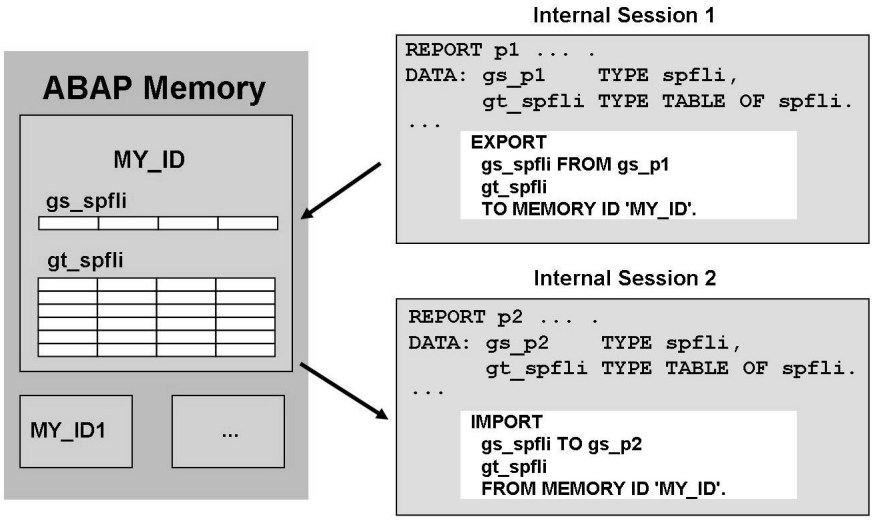


Figure 167: Passing Parameters Using ABAP Memory

Use `EXPORT ... TO MEMORY ID <id>` to create a data cluster in the ABAP memory. The data cluster stores any number of ABAP variables with their current values in ABAP memory. The `ID <id>` addition can be up to 60 characters long and enables you to identify different data clusters.

Another `EXPORT TO MEMORY` statement for the same data cluster overwrites the old cluster.

Using the `IMPORT ... FROM MEMORY ID <id>` statement, you can copy data from ABAP memory into the corresponding fields of your ABAP program. You can also restrict the selection to a part of the data cluster in the `IMPORT` statement.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The variables into which you want to read data from the cluster in ABAP memory must have the same type in the exporting and importing programs. You need to use the same name for the data object in both the `IMPORT` and in the `EXPORT` statements. You can use the `TO` clause to copy data to a data object with a different name; however, it must have the same type. Use the `FREE MEMORY ID <id>` to release a data cluster.

If you run another program with `CALL TRANSACTION` or `SUBMIT`, you can use the ABAP memory to pass data. However, you cannot use the statement `LEAVE TO TRANSACTION`, since the ABAP memory is reset after this call.

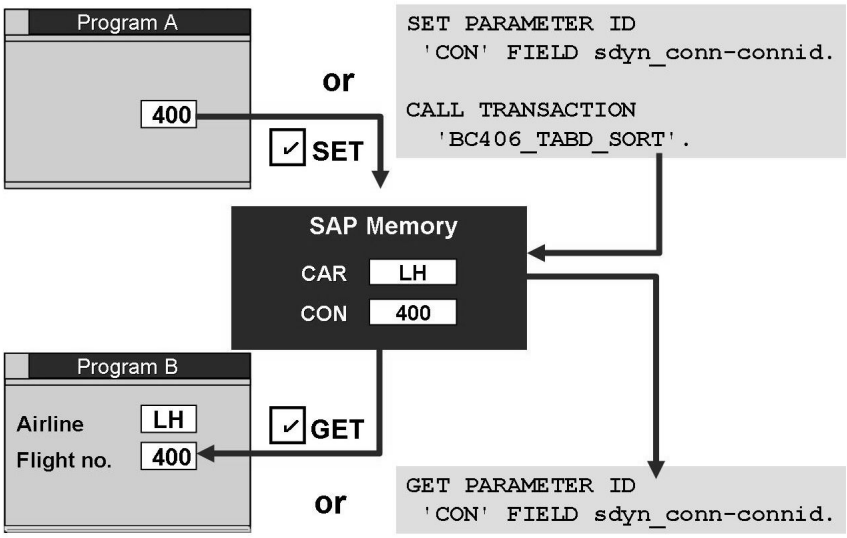


Figure 168: Passing Parameters Using SAP Memory

You can define memory areas (parameters) in the SAP memory in a number of ways:

- Add the `MEMORY ID <par_id>` or `PARAMETERS` or `SELECT-OPTIONS` option to the statement.
- Create input/output fields with reference to the ABAP Dictionary.

  These pass the parameter names of the data element to which you are referring.

  To find out the names of the parameters that are assigned input fields, display the F1 help for a field and then choose *Technical Info.*.
- You can also directly fill the memory area using the `SET PARAMETER ID '<par_id>' FIELD <var>` statement and read from it directly using the `GET PARAMETER ID '<par_id>' FIELD <var>` statement.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

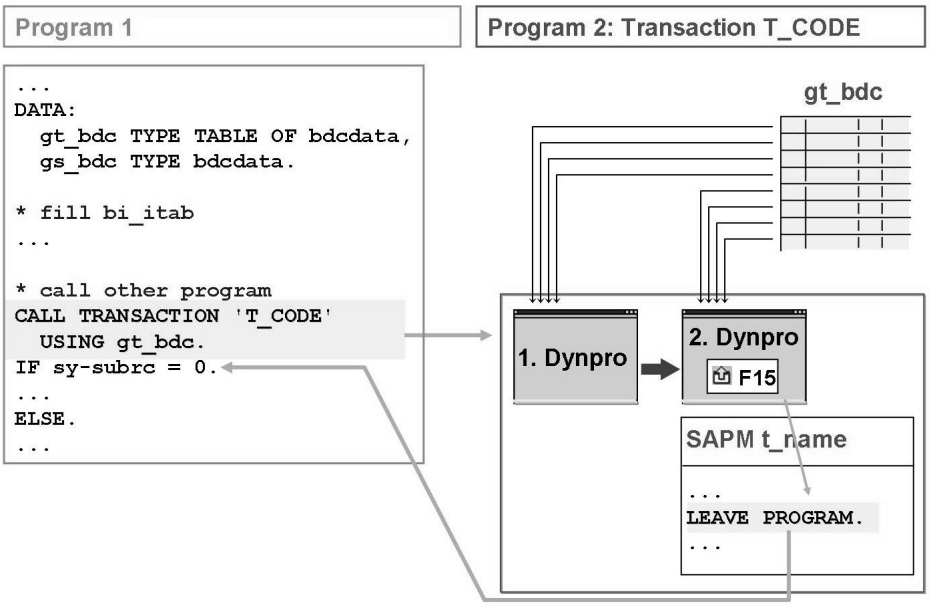


Figure 169: Passing Data Using Internal Tables

When you call a transaction using the `CALL TRANSACTION <t_code> USING <gt_btc>` statement, you can execute the <T_CODE> transaction using values from the internal table <*gt_btc*> in the screen fields. The internal table must have the structure `BDCDATA` and be filled accordingly.

You can use the addition `MODE` to specify whether the screen contents are always to be displayed (**A** - default setting), are only to be displayed if an error occurs (**E**), or are never to be displayed (**N**). You can use the `MESSAGES INTO <gt_messages>` statement to specify an internal table into which all system messages should be written. The internal table must have the structure `BDCMSGCOLL`.

You can use the field *sy_subrc* to find out whether the transaction was successfully executed.

You can use this technique if:

- You are performing the processing in the foreground but have not filled the input fields with `GET` parameters.
- You want to execute the transaction invisibly. In this case, you also need to pass the function codes in the table.

These technique is also one of the options for passing data to from external systems. To do this, however, you need to completely fill the internal table with the structure *BDCDATA*.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Field Name	program	dynpro	dynbegin	fnam	fval
Length	40	4	1	132	132
Description	Program name	Screen number	First record	Field name	Field val.
Note when Filling	Only in first record on screen	Only in first record on screen	'X' for first record on screen, otherwise ' '		Case-sensitive

Figure 170: Fields in the Global Type BDCDATA

Fill the internal table in batch input format: Each screen that you want to process automatically in the transaction must be identified by a line in which only the *program*, *dynpro* and *dynbegin* fields are filled. After the record that identifies the screen, use a separate *bdcdata* record for each field you want to fill. These records use the fields *fnam* and *fval*. You can fill the following fields:

- Input/Output Fields: With Application Data
- The command field (*bdc_okcode*): with a function code
- The cursor position field (*bdc_cursor*): with a field name

You can also use the `CALL TRANSACTION` technique to pass data from external systems. For more information about this technique, see the online documentation.

program	dynpro	dynbegin	fnam	fval
SAPBC405_CALD_CREATE_CUSTOMER	0100	X		
			SCUSTOM-NAME	<current_name>
			SCUSTOM-CITY	<current_city>
			BDC_OKCODE	SAVE

Entry in command field

Save

```
DATA:
  gs_bdcdata TYPE bdcdata,
  gt_bdcdata TYPE TABLE OF bdcdata.

*** fill the bdcdata-table ...

CALL TRANSACTION 'BC405_CALD_CRE_CUST'
                 USING gt_bdcdata
                 MODE 'N'.
IF sy-subrc <> 0.
  MESSAGE ... WITH sy-subrc.
ENDIF.
```

Figure 171: Example: Passing Data Using Internal Tables

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

In the above example, a transaction is called that does not implement an import from ABAP memory and whose fields are not set as `GET` parameters. The application data is passed in an internal table and processed invisibly. You can see the corresponding internal table with the format *bdcdata* above. At runtime, `current_name` stands for the customer name from the input field and `current_city` for the city. You can use the field *bdc_okcode* to access the command field, in which you enter the function code that would have been triggered if the user had chosen a function key, button, or a menu entry (or had entered a code directly in the command field).

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 19: Calling Programs and Passing Data

Exercise Objectives

After completing this exercise, you will be able to:

- Call an executable program from another program
- Preassign its selection criteria with values
- Use the ABAP memory to pass complex data structures from one program to another

Business Example

You are working as a programmer for ABC Travel. Your manager wants to display a list of flight connections in the ALV. Various existing programs are to be called from this list:

- Double-clicking a flight connection is to call the program that you have already created that displays a list of flights for the selected flight connection.
- A customer function is to call a transaction to maintain a flight connection.
- A customer function is to call an existing program that displays flights from any number of flight connections (multiple selection).

Task:

Expand a program to include calls from other included programs.

Template: *BC405_CALL_T1*.

Model Solution: *BC405_CALL_S1*

1. Copy the program *BC405_CALL_T1* to the program to be created, *ZBC405_##_CALL*. Familiarize yourself with the program.

   Hint: If you have handled the implementation of the ALV using the Object Model, use the program *BC405_CALL_T1_OM* as a template and edit it correspondingly.

2. Now implement a handling of the event *DOUBLE_CLICK*. To do this, create a handler method (definition and implementation) and register it.

   This method is to determine in which data row the user double-clicked. The report that you developed in previous exercises, *ZBC405_##_ALV* is then to be called in such a way that only flights for the selected flight connection

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

are displayed. To do this, you need to pass the correct data to the selection screen of the called report. When the user exits the called report, control is to return to the calling report.

Hint: Instead of your solution from previous exercises, you can also call *BC405_EVENT_S4*, or for the object model *BC405_OM_OTHER_S1*.

3. There is already a customer function (function code: *MAINTAIN*) available in the GUI status of your program. However, there is no handler method for it yet.

   Create an appropriate handler method for the event *USER_COMMAND*. Process the function code *MAINTAIN*:

   Hint: For the ALV Object Model the event is called *ADDED_FUNCTION*.

   Call the transaction BC405CALL. Pass the key data for the selected flight connection to this transaction using the SAP memory (SET/GET parameters). The selected flight connection is the one on which the cursor is placed. Use a suitable method related to the ALV object to find out on which cell the cursor is placed. You can determine the correct data record from the internal data table on the basis of the row number.

   Do not forget to register the method.

4. There is a further customer function available in the GUI status of your program (function code: *MULTI_ROWS*). However, no function is implemented for this either.

   Extend the handler method for the event *USER_COMMAND*. Process the function code *MULTI_ROWS*:

   Hint: For the ALV Object Model the event is called *ADDED_FUNCTION*.

   Call the report *BC405_CALL_FLIGHTS*. Data is passed to this report using the ABAP memory. The called report expects the data in the ABAP memory with the ID *BC405*. The data must contain a data object called *MEM_IT_SPLFI*. This must be an internal table with the row type *SPFLI*.

   Proceed as follows:

   Familiarize yourself with the report to be called, *BC405_CALL_FLIGHTS*.

Continued on next page

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Retrieve all selected rows of the ALV with a suitable method of the ALV object. The correct method returns an internal table with the indexes of the selected rows.

Determine the associated data set for each row of this internal table. Copy this into an internal table of row type *SPFLI* (suggested name for the internal table: *LT_SPFLI*).

Export the table that you have just filled, *LT_SPFLI*, to the ABAP memory, memory ID *BC405*. When doing so, ensure that you assign the correct name to the table in the memory area. (What is this name? Find this out from the report *BC405_CALL_FLIGHTS*.)

Now call report *BC405_CALL_FLIGHTS* in such a way that control returns to the calling report after the called report is ended.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 19: Calling Programs and Passing Data

Task:

Expand a program to include calls from other included programs.

Template: *BC405_CALL_T1*.

Model Solution: *BC405_CALL_S1*

1. Copy the program *BC405_CALL_T1* to the program to be created, *ZBC405_##_CALL*. Familiarize yourself with the program.

   Hint: If you have handled the implementation of the ALV using the Object Model, use the program *BC405_CALL_T1_OM* as a template and edit it correspondingly.

   a) Self-explanatory

2. Now implement a handling of the event *DOUBLE_CLICK*. To do this, create a handler method (definition and implementation) and register it.

   This method is to determine in which data row the user double-clicked. The report that you developed in previous exercises, *ZBC405_##_ALV* is then to be called in such a way that only flights for the selected flight connection are displayed. To do this, you need to pass the correct data to the selection screen of the called report. When the user exits the called report, control is to return to the calling report.

   Hint: Instead of your solution from previous exercises, you can also call *BC405_EVENT_S4*, or for the object model *BC405_OM_OTHER_S1*.

   a) See result at the end.

3. There is already a customer function (function code: *MAINTAIN*) available in the GUI status of your program. However, there is no handler method for it yet.

   Create an appropriate handler method for the event *USER_COMMAND*. Process the function code *MAINTAIN*:

   Hint: For the ALV Object Model the event is called *ADDED_FUNCTION*.

Continued on next page

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Call the transaction BC405CALL. Pass the key data for the selected flight connection to this transaction using the SAP memory (SET/GET parameters). The selected flight connection is the one on which the cursor is placed. Use a suitable method related to the ALV object to find out on which cell the cursor is placed. You can determine the correct data record from the internal data table on the basis of the row number.

Do not forget to register the method.

a) See result at the end.

4. There is a further customer function available in the GUI status of your program (function code: *MULTI_ROWS*). However, no function is implemented for this either.

   Extend the handler method for the event *USER_COMMAND*. Process the function code *MULTI_ROWS*:

   Hint: For the ALV Object Model the event is called *ADDED_FUNCTION*.

   Call the report *BC405_CALL_FLIGHTS*. Data is passed to this report using the ABAP memory. The called report expects the data in the ABAP memory with the ID *BC405*. The data must contain a data object called *MEM_IT_SPLFI*. This must be an internal table with the row type *SPFLI*.

   Proceed as follows:

   Familiarize yourself with the report to be called, *BC405_CALL_FLIGHTS*.

   Retrieve all selected rows of the ALV with a suitable method of the ALV object. The correct method returns an internal table with the indexes of the selected rows.

   Determine the associated data set for each row of this internal table. Copy this into an internal table of row type *SPFLI* (suggested name for the internal table: *LT_SPFLI*).

   Export the table that you have just filled, *LT_SPFLI*, to the ABAP memory, memory ID *BC405*. When doing so, ensure that you assign the correct name to the table in the memory area. (What is this name? Find this out from the report *BC405_CALL_FLIGHTS*.)

   Now call report *BC405_CALL_FLIGHTS* in such a way that control returns to the calling report after the called report is ended.

   a) See result at the end.

Continued on next page

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Result

Main program

```
REPORT bc405_call_s1.

INCLUDE bc405_call_s1_top.
INCLUDE bc405_call_s1_k01.


START-OF-SELECTION.
* retrieve data
  SELECT *
    FROM spfli
    INTO TABLE gt_spfli
    WHERE carrid IN so_car
      AND connid IN so_con.

  CHECK sy-subrc EQ 0.

  CALL SCREEN 100.


***********************************************************************
*PBO modules
***********************************************************************
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.                 " clear_ok_code  OUTPUT

*----------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'DYN'.
  SET TITLEBAR 'T1'.
ENDMODULE.                 " STATUS_0100  OUTPUT

*----------------------------------------------------------------------*
MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
      MESSAGE a010(bc405_408).
    ENDIF.

    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <> 0.
      MESSAGE a010(bc405_408).
    ENDIF.

    SET HANDLER:
      lcl_handler=>on_double_click FOR go_alv_grid,
      lcl_handler=>on_toolbar FOR go_alv_grid,
      lcl_handler=>on_user_command FOR go_alv_grid.

    gs_layout-sel_mode = 'A'.

    go_alv_grid->set_table_for_first_display(
      EXPORTING
        i_structure_name = 'SPFLI'
        is_layout        = gs_layout
      CHANGING
        it_outtab        = gt_spfli
      EXCEPTIONS
        OTHERS           = 1 ).
    IF sy-subrc <> 0.
      MESSAGE a012(bc405_408).
    ENDIF.

  ENDIF.
ENDMODULE.                 " create_and_transfer  OUTPUT


*********************************************************************
*PAI modules
*********************************************************************
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK' OR 'EXIT' OR 'CANCEL'.
      SET SCREEN 0.
  ENDCASE.
ENDMODULE.                 " USER_COMMAND_0100  INPUT
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

TOP include

```
DATA: gt_spfli TYPE TABLE OF spfli,
      gs_spfli TYPE          spfli,
      ok_code  LIKE          sy-ucomm.

DATA: go_alv_grid  TYPE REF TO cl_gui_alv_grid,
      go_container TYPE REF TO cl_gui_custom_container.

DATA: gs_layout TYPE lvc_s_layo.

SELECT-OPTIONS: so_car FOR gs_spfli-carrid,
                so_con FOR gs_spfli-connid.
```

K01 include (classes)

```
*---------------------------------------------------------------------*
*       CLASS lcl_handler DEFINITION
*---------------------------------------------------------------------*
*       Local class for ALV event handling
*---------------------------------------------------------------------*
CLASS: lcl_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: on_toolbar FOR EVENT toolbar
                              OF cl_gui_alv_grid
                              IMPORTING e_object,
                   on_user_command FOR EVENT user_command
                                   OF cl_gui_alv_grid
                                   IMPORTING e_ucomm,
                   on_double_click FOR EVENT double_click
                                   OF cl_gui_alv_grid
                                   IMPORTING es_row_no.
ENDCLASS.                     "lcl_handler DEFINITION


*---------------------------------------------------------------------*
*       CLASS lcl_handler IMPLEMENTATION
*---------------------------------------------------------------------*
CLASS: lcl_handler IMPLEMENTATION.

**********************************************************************
*       Expands toolbar                                              *
**********************************************************************
  METHOD on_toolbar.
    DATA ls_button TYPE stb_button.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
* Separator
    ls_button-butn_type = 3.
    INSERT ls_button INTO TABLE e_object->mt_toolbar.

* Create button "Modify bookings"
    CLEAR ls_button.
    ls_button-function  = 'MAINTAIN'.
    ls_button-icon      = icon_change.
    ls_button-quickinfo = 'Maintain connection'(z01).
    ls_button-butn_type = 0.
    INSERT ls_button INTO TABLE e_object->mt_toolbar.

* Create button "Display bookings"
    CLEAR ls_button.
    ls_button-function  = 'MULTI_ROWS'.
    ls_button-icon      = icon_flight.
    ls_button-quickinfo = 'Flights of several connections'(z02).
    ls_button-butn_type = 0.
    INSERT ls_button INTO TABLE e_object->mt_toolbar.
  ENDMETHOD.                    "on_toolbar

***********************************************************************
*         Handles self-defined user-commands                          *
***********************************************************************
  METHOD on_user_command.
    DATA: lt_row_no TYPE          lvc_t_roid,
          ls_row_no TYPE          lvc_s_roid,
          lt_spfli  TYPE TABLE OF spfli,
          ls_spfli  TYPE          spfli.

    CASE e_ucomm.
      WHEN 'MAINTAIN'.
* detect selected row
        go_alv_grid->get_current_cell(
          IMPORTING
            es_row_no = ls_row_no ).

        READ TABLE gt_spfli INTO gs_spfli INDEX ls_row_no-row_id.

        IF sy-subrc = 0.
* set SAP memory parameters
          SET PARAMETER:
            ID 'CAR' FIELD gs_spfli-carrid,
            ID 'CON' FIELD gs_spfli-connid.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* call transaction
          CALL TRANSACTION 'SAPBC405CAL'.
        ENDIF.

      WHEN 'MULTI_ROWS'.
        go_alv_grid->get_selected_rows(
          IMPORTING
            et_row_no     = lt_row_no ).

        LOOP AT lt_row_no INTO ls_row_no.
          READ TABLE gt_spfli INTO gs_spfli INDEX ls_row_no-row_id.
          CHECK sy-subrc EQ 0.
          ls_spfli = gs_spfli.
          APPEND ls_spfli TO lt_spfli.
        ENDLOOP.

        EXPORT mem_it_spfli FROM lt_spfli TO MEMORY ID 'BC405'.
        SUBMIT bc405_call_flights AND RETURN.

    ENDCASE.
  ENDMETHOD.                    "on_user_command

***********************************************************************
*         Handles double-click                                        *
***********************************************************************
  METHOD on_double_click.
    READ TABLE gt_spfli INTO gs_spfli INDEX es_row_no-row_id.
    CHECK sy-subrc EQ 0.

    SUBMIT bc405_event_4 AND RETURN
            WITH so_car EQ gs_spfli-carrid
            WITH so_con EQ gs_spfli-connid.
  ENDMETHOD.                    "on_double_click

ENDCLASS.                    "lcl_handler IMPLEMENTATION
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Describe different memory areas
- Call executable programs
- Call transactions
- Pass data using memory areas

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Describe different memory areas
- Call executable programs
- Call transactions
- Pass data using memory areas

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Unit 10

Background Processing

Internal Use SAP Partner Only

Unit Overview

In this unit, the definition of background jobs is discussed.

Unit Objectives

After completing this unit, you will be able to:

- List the various phases of background processing

Unit Contents

Lesson: Background Processing 346
Exercise 20: Background Processing.....351

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson: Background Processing

Lesson Overview

In this lesson, you learn how background processing can be used for reporting.

Lesson Objectives

After completing this lesson, you will be able to:

- List the various phases of background processing

Business Example

You are working as a programmer for ABC Travel. Your manager wants you to set up a background job that is run every day at 8:00 PM. The background job is to compile data about all bookings made during that day. To be able to implement this, you need to know the different phases of background processing.

The Phases of Background Processing

There are several reasons to run a report in the background, for example:

- When a program converts large datasets and therefore requires a long runtime, it makes sense to start it in a background process.

  Background execution take place without user dialogs, and can take place in parallel to online operations. The dialog work processes are available for online processing. Background jobs are executed in special work processes (background processes), which allows distributed processing.
- An evaluation must be executed periodically.
- Some evaluations must be carried out based on events, for example immediately after a particular system status has been reached.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

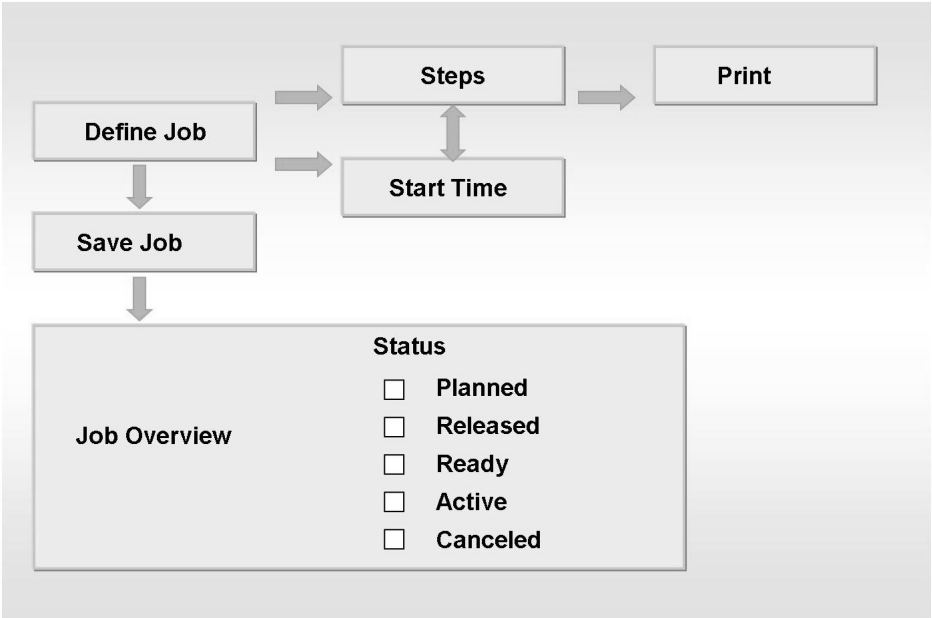


Figure 172: The Phases of Background Processing

To start a program in the background, you must first include it in a job.

Use the job definition to determine which programs (steps) will run during this job. You can specify print parameters and set the start time for the execution of the job.

The job overview (transaction SM37) tells you the current status of the job.

The Job “Wizard” simplifies the creation of background jobs.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

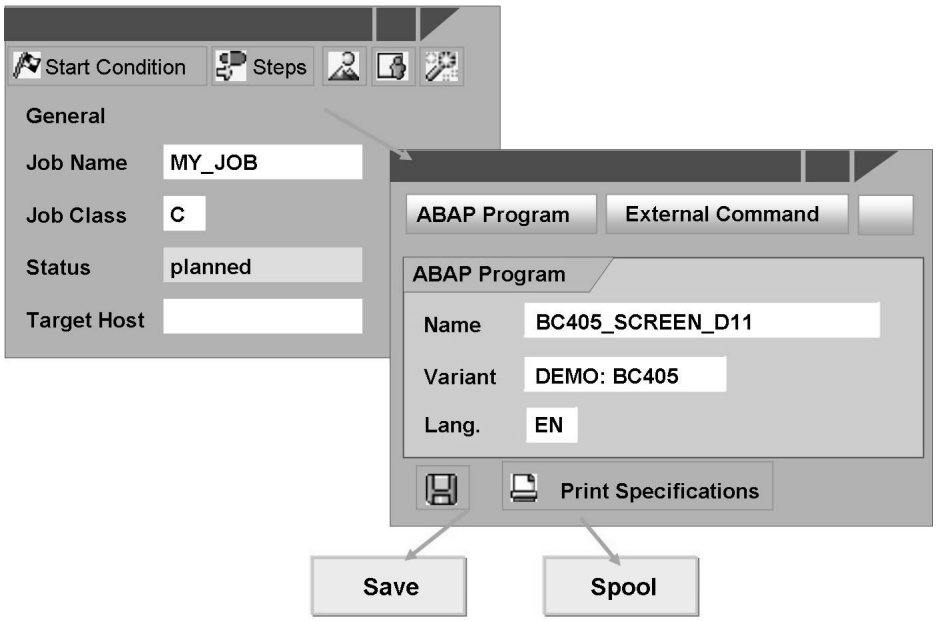


Figure 173: Defining Steps

Define Job is located under *System → Services → Jobs → Define Job* (transaction SM36). First assign a job name (of your choice) and define the priority (job class) and the target host (F4 help).

Then define the individual steps of the job. If you want the program to run with a selection screen, you also have to specify a variant. The list can be stored in the spool or printed immediately. This depends on the specified print parameters. When you have defined all the steps, save them and return to the initial screen of the job definition.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

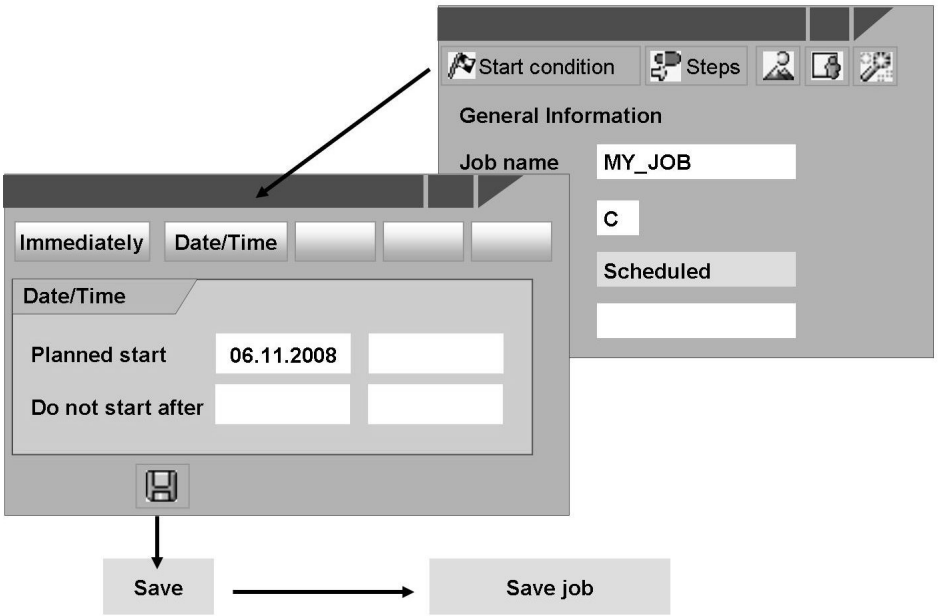


Figure 174: Determining the Start Date and Releasing the Job

Once you have defined the steps, you can determine the start date for the job. For example, you can start the job on a certain day at a certain time.

Once you have defined the start date, save your entries and return to the initial screen of the job definition. Now save the job, which releases it to run at the specified time.

You can also use the automated job scheduling with the function modules in function groups *BTCH* and *BTC2*.

ALV in Background Processing

The ALV OM can only be controlled in the dialog in the “fullscreen” and “in the container” presentation forms. If you call the ALV in background operation, the print list (ALV Classic) is generated instead. You can either make the layout settings explicitly (using initial sortings, aggregations, and so on), or by specifying a layout variant.

If you attempt to instantiate the control in the display format “in the container”, an exception occurs in background operation. Therefore, always choose the display formats “full screen” or “classic ABAP list” for background processing. For the “fullscreen” presentation, the output is implicitly converted to the classic list form.

In order to determine whether the program is running in the background, you can query the system variable *sy-batch*.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The ALV Grid Control is automatically converted to the classic list. Note that during the background processing, you have to ignore the fact that the creation of the container by `CREATE OBJECT` fails.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 20: Background Processing

Exercise Objectives

After completing this exercise, you will be able to:

- Allow the user to run the program in the background

Business Example

You are working as a programmer for ABC Travel. The users would like to run the ALV Grid program in the background.

Task:

Execute an ALV Grid program in the background.

Template: *BC405_CREATE_S2*

Model Solution: *BC405_BACKGR_S1*

1. Copy the template or your solution from chapter "ALV: Creation" to the new name *ZBC405_##_BACKGR*.
2. Save a selection screen variant for your report that also contains a value for the field *Layout Variant*. Extend the checks of *sy-subrc* after the creation of the objects (`CREATE OBJECT`). The program is only to terminate if *sy-subrc* does not equal 0 and if *sy-batch* has an initial value.

   Then execute the report in the background. Check the result in the job overview (transaction SM37). Has the layout been adopted for print output?

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 20: Background Processing

Task:

Execute an ALV Grid program in the background.

Template: *BC405_CREATE_S2*

Model Solution: *BC405_BACKGR_S1*

1. Copy the template or your solution from chapter "ALV: Creation" to the new name *ZBC405_##_BACKGR*.
   a) Perform as usual.
2. Save a selection screen variant for your report that also contains a value for the field *Layout Variant*. Extend the checks of *sy-subrc* after the creation of the objects (`CREATE OBJECT`). The program is only to terminate if *sy-subrc* does not equal 0 and if *sy-batch* has an initial value.

   Then execute the report in the background. Check the result in the job overview (transaction SM37). Has the layout been adopted for print output?

   a) To define a variant click the *SAVE* button on the selection screen.
   b) To define a job, start transaction SM37. Click **`CTRL + F1`** to start the job wizard.
   c) Source code changes see result at the end.

Result

Main program

```
*&---------------------------------------------------------------------*
*& Report  BC405_BACKGR_S1                                            *
*&---------------------------------------------------------------------*
REPORT  bc405_backgr_s1.

* work area and internal table for ALV data
DATA: gt_flights TYPE TABLE OF sflight,
      gs_flight  TYPE          sflight.

DATA  ok_code    LIKE sy-ucomm.

DATA: go_alv_grid  TYPE REF TO cl_gui_alv_grid,
      go_container TYPE REF TO cl_gui_custom_container.

SELECT-OPTIONS: so_car FOR gs_flight-carrid,
                so_con FOR gs_flight-connid.
```

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
***********************************************************************
*ABAP events
***********************************************************************
START-OF-SELECTION.
  SELECT * FROM sflight
    INTO TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

  CALL SCREEN 100.


***********************************************************************
*PBO modules
***********************************************************************
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.                 " clear_ok_code  OUTPUT

*----------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'DYN'.
  SET TITLEBAR 'T1'.
ENDMODULE.                 " STATUS_0100  OUTPUT

*----------------------------------------------------------------------*
MODULE create_and_transfer OUTPUT.
  IF go_container IS INITIAL.
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0 AND sy-batch IS INITIAL.
      MESSAGE a010(bc405_408).
    ENDIF.

    CREATE OBJECT go_alv_grid
      EXPORTING
        i_parent = go_container
      EXCEPTIONS
        others   = 1.
    IF sy-subrc <> 0 AND sy-batch IS INITIAL.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
      MESSAGE a010(bc405_408).
    ENDIF.

    go_alv_grid->set_table_for_first_display(
      EXPORTING
        i_structure_name = 'SFLIGHT'
        i_save           = 'A'
      CHANGING
        it_outtab        = gt_flights
      EXCEPTIONS
        OTHERS           = 1 ).
    IF sy-subrc <> 0.
      MESSAGE a012(bc405_408).
    ENDIF.

  ENDIF.
ENDMODULE.                 " create_and_transfer  OUTPUT


*********************************************************************
*PAI Modules
*********************************************************************
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK' OR 'EXIT' OR 'CANCEL'.
      SET SCREEN 0.
  ENDCASE.
ENDMODULE.                 " USER_COMMAND_0100  INPUT
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- List the various phases of background processing

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- List the various phases of background processing

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Unit 11

ALV Object Model (Optional)

Internal Use SAP Partner Only

Unit Overview

This unit shows how to create the ALV using the ALV Object Model. The ALV Object Model (ALV OM) is available as of *SAP NetWeaver 2004*. Although it is the newer technique, the classic ALV is more common in development. Therefore, this topic is optional.

This unit explains how to:

- Create and display an ALV (full screen, in container, as classical ABAP list)
- Adapt the layout
- Adapt column properties
- Adapt cell properties
- Adapt the toolbar
- Handle events
- Influence the list output

Unit Objectives

After completing this unit, you will be able to:

- Create an ALV with the output formats “full screen” and “classic ABAP list”
- Display data in the ALV
- Display an ALV as a subarea of a dynpro
- Explain the class hierarchy of the ALV OM
- Describe the basic procedure to change properties of an ALV
- Set the title of an ALV
- Set other general display properties of the ALV
- Set display properties that apply to all output columns
- Define properties of individual columns (principle)
- Add an additional column to the ALV and format it (principle)
- Define a column title
- Define or change key columns
- Define or change a DDIC reference

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

- Trigger sorting, filtering, and aggregation in the ALV from the application
- Activate selection mode and make selections
- Work with layout variants
- Explain the concept of columns with special technical meanings
- Define the color of columns, rows, and cells
- Define text properties
- Control the display of numerical values
- Make generic ALV functions available with the desired scope
- Add customer functions to the ALV
- Handle ALV events such as the double-click or the call of a customer function
- Display interactive elements in the ALV output
- Make settings for printing from the ALV
- Handle the events *TOP_OF_PAGE* and *END_OF_PAGE*
- Compile design objects and assign a header or footer area

Unit Contents

Lesson: Creating and Displaying the ALV OM 359
Exercise 21: ALV OM: Create for Fullscreen 369
Exercise 22: ALV OM: Create for Classic List 373
Lesson: ALV OM: Creating in the Container 376
Exercise 23: Creating a Container Screen for an ALV 389
Exercise 24: ALV OM: Creating in the Container 397
Lesson: ALV OM: Fetching and Modifying Subobjects of a Main Class .405
Lesson: ALV OM: Adjusting the Display 413
Exercise 25: ALV OM: Setting General Display Properties 419
Lesson: ALV OM: Defining Column Properties 428
Exercise 26: ALV OM: Determining Column Properties.....439
Lesson: ALV OM: Basic Functions and Layout Variants.....446
Exercise 27: ALV OM: Basic Functions and Layout Variants.....457
Lesson: ALV OM: Output Appearance 465
Exercise 28: ALV OM: Output Appearance 487
Lesson: ALV OM: Functions, Events, and Interactive Elements.....497
Exercise 29: ALV OM: Making Generic Functions Available 517
Exercise 30: ALV OM: Your Own Functions and Event Handling....521
Exercise 31: ALV OM: Handling Single- and Double-Clicks.....531
Lesson: ALV OM: Print Design and the Header and Footer 538
Exercise 32: ALV OM: Print Design and the Header and Footer.....551

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson: Creating and Displaying the ALV OM

Lesson Overview

This lesson shows you how to create an ALV Object Model for the “full screen” and “classic ABAP list” output formats. The procedure is very similar for both output formats: You create a data object that represents the ALV. This object requires the following information:

1. Display as a list (yes/no)
2. Internal table with data to be displayed

On the basis of this information, an ALV is created in one of the two output formats.

Lesson Objectives

After completing this lesson, you will be able to:

- Create an ALV with the output formats “full screen” and “classic ABAP list”
- Display data in the ALV

Business Example

You are working as a programmer for ABC Travel. Your manager wants to display flight data with an ALV, and to be able to choose between full screen mode and a list. It is your task to implement this.

Basic Principle of the ALV OM (Simplest Call)

At least the following two methods in the ALV main class *CL_SALV_TABLE* must be called to obtain the desired ALV output:

- The *FACTORY* method to instantiate an object of the ALV main class. This specifies the internal data table from which data is displayed. The display type (classic list, full screen, in container) is also defined.
- The *DISPLAY* method for ALV output on the screen.

The *FACTORY* method is a class method. It returns an ALV instance and is used instead of the otherwise usual `CREATE OBJECT` This instance is accessed during all other method calls to individually adjust the ALV.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

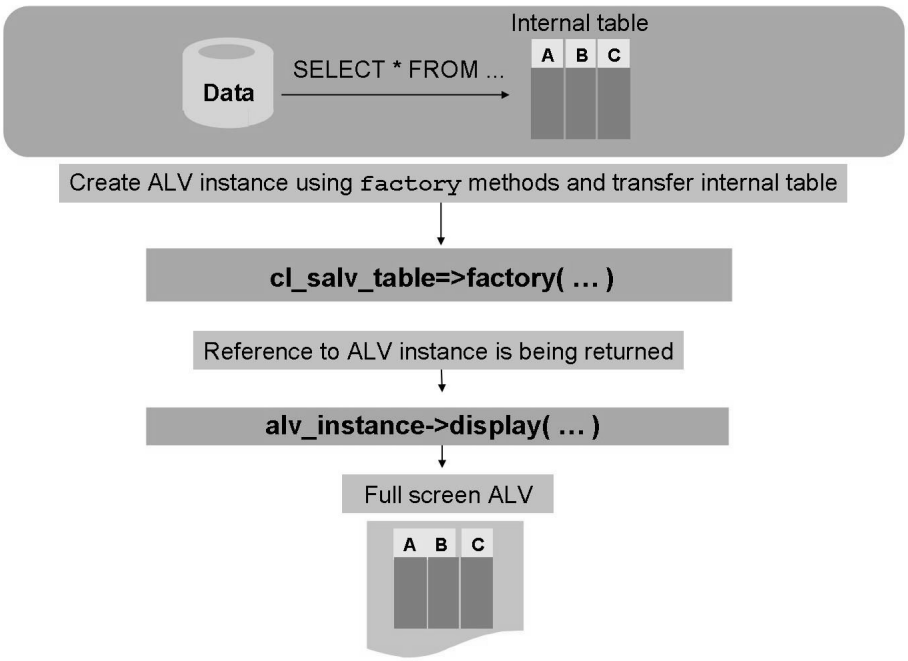


Figure 175: Basic Principle of the ALV OM (Simplest Call)

For the call of the *FACTORY* method, you first need to create a reference and type it with the class *CL_SALV_TABLE*. An internal table is also required. This must be filled with the data to be displayed. The ALV instance is then created by the *FACTORY* method. The parameters have the following meanings for this:

r_salv_table

Reference to the created ALV

t_table

Name of the internal table that contains the display data

To display the data, you only need to call the *DISPLAY* method as the instance method for the ALV reference. The figure below shows the corresponding method call in the functional writing style, which means that instead of writing the keywords **`CALL METHOD instance->method <parameters>`**, you write: **`instance->method( <parameters> )`**. Both writing styles are equivalent. However, we will see later that functional writing style can be preferable.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* Isn't it REALLY simple?

DATA: gt_bookings  TYPE TABLE OF sbook.
DATA: go_alv_table TYPE REF TO   cl_salv_table.

*... Select data
  SELECT * FROM sbook
    INTO TABLE gt_bookings.

*... Create instance
  CALL METHOD cl_salv_table=>factory
    IMPORTING
      r_salv_table = go_alv_table
    CHANGING
      t_table      = gt_bookings.

*... Display ALV
  go_alv_table->display( ).
```

Figure 176: Simplest Call - Source Code

This program displays the data of the internal table in an ALV in full screen.

The following describes how you can easily implement method calls and what you need to consider when defining reference variables.

Generating Method Calls

To implement a method call of a global class, use either the ABAP Objects pattern or the drag and drop function of the *Object Navigator* (transaction SE80).

The creation of the statement `CALL METHOD` with the ABAP Objects pattern is described in the following:

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

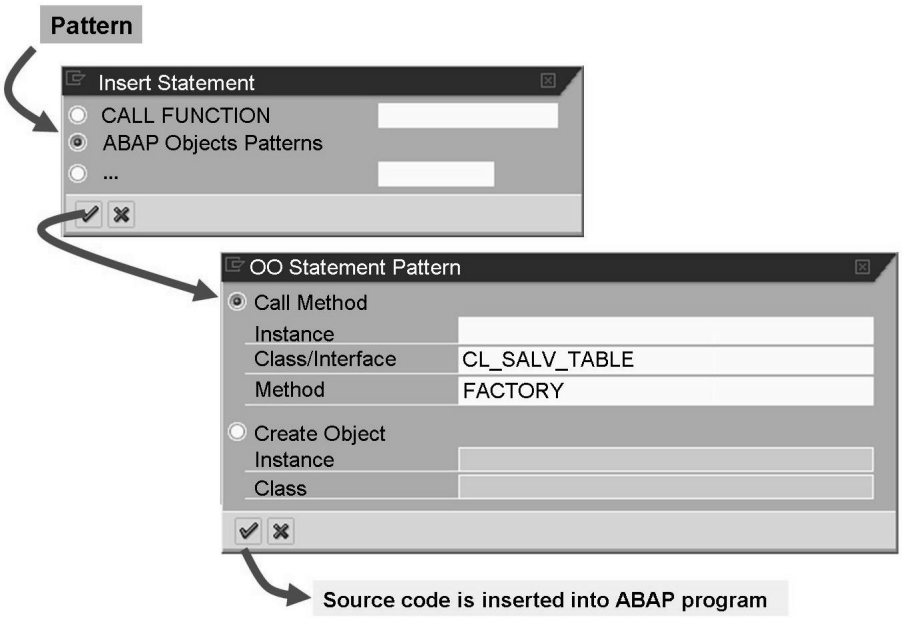


Figure 177: Inserting `CALL METHOD` by Pattern

On the "Insert Statement" initial screen, choose the *ABAP Objects Patterns* option.

On the next screen, choose the *Call Method* option. Specify

- For *Instance*: the reference variable for the object for which the method is to be used (if you want to generate the call of a class method – as in this case – leave the field empty)
- For *Class/Interface*: the class of the object (in this example, *CL_SALV_TABLE*),
- For *Method*: the method names (in this example, *FACTORY*).

The statement `CALL METHOD`, including all parameters and exceptions, is then inserted at the cursor position in the ABAP source code. Optional parameters and exceptions are inserted as comments.

In this way, you can avoid typing errors and omissions.

You then need to complete the actual parameters and possibly program the exception handling.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

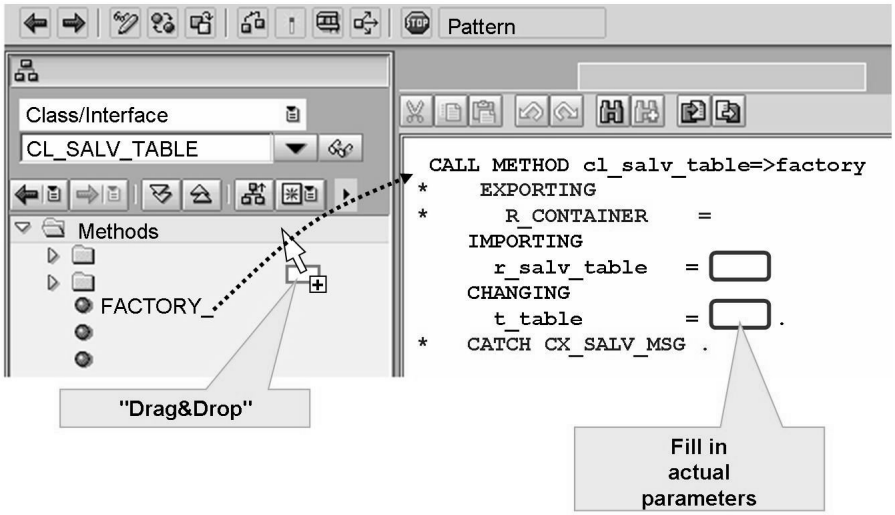


Figure 178: Inserting `CALL METHOD` by Drag and Drop

Using drag and drop, you can display the global class (in this example, *CL_SALV_TABLE*) in the navigation area of the ABAP Workbench and drag the method to the editing area by holding down the left mouse button.

The ABAP statement `CALL METHOD`, including the interface parameters and exceptions, is inserted at the current cursor position in the source code of the program. Optional parameters and exceptions are inserted as comments.

You then need to complete the actual parameters and possibly program the exception handling. For the example above, this is sufficient, since it is a call for a class method. However, if you want to generate the call for an instance method (for example, for the *GET_COLUMNS* method from the *CL_SALV_TABLE* class), you generate a method call in the `CALL METHOD xxxxxxxx->get_columns` format. In this case, you then need to insert the name of the correct references variables instead of xxxxxxx.

Beginning with *Web Application Server 6.20*, you can generate the `CALL-METHOD` statement in either the functional or classic writing style. You can set this in the ABAP Editor by choosing *Utilities → Settings*, *ABAP Editor → Pattrn*, and checking *Functional Writing Style for CALL METHOD*.

Functional Writing Style for CALL METHOD

```
cl_salv_table=>factory(
  IMPORTING
    r_salv_table    = go_alv_table
  CHANGING
    t_table         = gt_flights
  ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Reference Variables

```
DATA: <variable_name> "     Name of the reference variable
      TYPE REF TO     "     Specification of the variable as an object reference
      <class_name>.   "     Name of a class
```

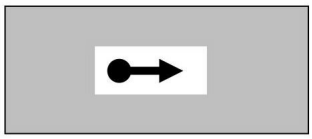


Figure 179: Syntax: Defining a Reference Variable

If, as for the use of the ALV, your program is to use instances of classes (object-oriented programming), you require pointers to the objects (instances) generated and administered in the main memory of the SAP Web AS. These pointers are administered in your ABAP program using a reference variable.

You generate the reference variable in your program using the statement `DATA` and assign a name for your reference variable. The field is typed by means of `TYPE REF TO <class_name>`. The addition `TYPE REF TO` means that the field can include a pointer to an instance. You need to specify the name of a class after `TYPE REF TO`. The reference variable can then contain pointers to instances of this class.

A reference variable is typed to a class. This class can be locally or globally defined. The classes for the ALV are globally defined, that is, they were created with the Class Builder and are therefore globally visible.

Exception Handling for Method Calls with Class-Based Exceptions

Beginning with *Web Application Server 6.10*, you can trigger class-based exceptions for method calls. Previously, there was only exception handling with the addition `EXCEPTIONS` for the `CALL METHOD` statement. If the exception is implemented using the new procedure, when you generate the `CALL METHOD` statement using a pattern, the system creates a (commented out)

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

CATCH statement under the actual method call for all exception classes that are assigned to the method. To activate exception handling, you still need to add the following programming:

1. A TRY-ENDTRY block that encloses the method call
2. Remove the commenting for the CATCH statement
3. The INTO <referenz_variable> addition for the CATCH statement (type the reference variable with the exception class)
4. The actual exception handling in the CATCH block

Example of Exception Handling with Exception Classes

```
DATA: go_alv_table TYPE REF TO cl_salv_table,
      go_alv_error TYPE REF TO cx_salv_msg.
TRY.
  cl_salv_table=>factory(
    IMPORTING
      r_salv_table   = go_alv_table
    CHANGING
      t_table        = gt_bookings ).

CATCH cx_salv_msg INTO go_alv_error.

  cl_sapbc405_exc_handler=>process_alv_error_msg(
    EXPORTING
      r_error = go_alv_error ).
ENDTRY.
```

The enclosure by a TRY...ENDTRY block means that an exception that occurs within it at runtime is caught in the associated CATCH block (as long as the exception is specified in the CATCH statement). The enclosure with TRY...ENDTRY can be created at the same time that you generate the method call with the pattern function or the drag and drop function if you set it that way in *Utilities → Settings*, *ABAP Editor → Pattrn*, field *For Except. Classes, CALL METHOD, Generate W/O TRY...ENDTRY.*

For the correct typing for the variable that refers to the exception object, see the specification after the keyword CATCH (for example, CATCH cx_salv_msg: So here you would have to type the reference variable using TYPE REF TO cx_salv_msg).

The called method for exception handling can then react to the exception situation, for example, by displaying a meaningful termination message. Other solutions would also be possible here, however, such as calling a subroutine.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The source code of the exception handling method called in the last example is shown below. Now you call the *GET_MESSAGE* method from the *CX_SALV_MSG* class. This returns a data object of type *BAL_S_MSG*, from which the actual error message can be taken.

Example of an Exception Handling Method

```
METHOD process_alv_error_msg.

* Signature: ix_error TYPE REF TO CX_SALV_ERROR
*            iv_type  TYPE        SYMSGTY  DEFAULT 'A'

  DATA ls_result_message TYPE bal_s_msg .

  ls_result_message = ix_error->get_message(  ).
  MESSAGE ID     ls_result_message-msgid
          TYPE   iv_type
          NUMBER ls_result_message-msgno
          WITH   ls_result_message-msgv1
                 ls_result_message-msgv2
                 ls_result_message-msgv3
                 ls_result_message-msgv4.
ENDMETHOD.
```

Caution: Note that exceptions that are not handled always lead to a runtime error when the related error occurs.

Output as a Full Screen ALV or Classic List

Of the three output options, the full screen ALV is the standard. To display the ALV as a classic list, you need to assign the value **X** to the parameter *list_display* when calling the *FACTORY* method. If the parameter has a space as its value, the ALV is displayed in full screen mode.

Display as a Classic List

```
cl_salv_table=>factory(
  EXPORTING
    list_display   = 'X'
  IMPORTING
    r_salv_table   = go_alv_table
  CHANGING
    t_table        = gt_bookings
  ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Changing Contents and Structure

You can change the internal data table in various ways. You need to proceed differently, depending on which type of change you are using, to make the change visible at the front end in the ALV output.

You can make the following changes:

- Change the content or sequence of individual data records. You then need to refresh the ALV output.
- Replace the entire data table.

Refreshing the Display of the ALV Output

If you have changed single values, added lines or deleted lines in the data table, you must also reproduce those changes in the ALV output: You refresh the ALV display.

You do not need to refresh the ALV output for the following changes:

- Changes to the display of columns
- Changes to the sequence of columns in the ALV display
- Changing the ABAP dictionary reference for columns (see lesson “ALV OM: Determining Column Properties”)

By default, the ALV output is automatically scrolled to the start when you refresh it. However, you can specify that the current display (position on the screen) is not changed.

There are two ways you can refresh the ALV display:

- All settings for filters and sorting are retained. (This setting can lead to newly-inserted data records not being displayed, because precisely these data records are hidden by the filter setting.)
- All settings for filters and sorting are deleted.

Refreshing the ALV Output

Function	Class	Method
Refresh ALV output without deleting filter and sorting settings	*CL_SALV_TABLE*	*REFRESH* (*REFRESH_MODE* parameter: ***SOFT***)
Refresh ALV output, deleting filter and sorting settings	*CL_SALV_TABLE*	*REFRESH* (*REFRESH_MODE* parameter: ***FULL***)

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Changing the Structure of the Data Table

You can assign a new structure and new content to the ALV with the method *SET_DATA* of the class *CL_SALV_TABLE* (parameter *t_table*).

If you assign a new structure with *SET_DATA*, all objects that are related to the previous structure are deleted, such as all sorting objects.

New Structure and New Contents

Function	Class	Method
Assign a new structure and new content	*CL_SALV_TABLE*	*SET_DATA* (Parameter *T_TABLE*)

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 21: ALV OM: Create for Fullscreen

Exercise Objectives

After completing this exercise, you will be able to:

- Generate an ALV
- Display an ALV in full screen mode

Business Example

You are a programmer for ABC Travel. The users of your program for flight data reporting want to display the data in an ALV in full screen mode.

Task:

Perform the following task:

1. Create the program *ZBC405_##_OM*.

   Model Solution: *BC405_OM_CREATE_S1*

2. Create an internal table `gt_flights` with the row type `sflight`. Create a work area for the internal table.

3. Create two select-options, one for airlines, and one for connection numbers. Then retrieve the flight data taking account of the selection options and store the data in the internal table `gt_flights`.

4. Create a reference variable `go_alv_table` that is typed with `cl_salv_table`.

   Implement the method call for creating the ALV instance with the class method *FACTORY* of the class `CL_SALV_TABLE`. Generate the statement using the ABAP pattern or by drag and drop, and assign the values to the parameters of the method call.

5. Implement handling for the exception `cx_salv_msg` of the method *FACTORY*. Follow these steps:

   Create a reference variable `go_alv_error` of type `cx_salv_error`.

   Activate the generated `CATCH` statement after the method call and extend the statement with the addition `INTO go_alv_error`.

   Call the global method *PROCESS_ALV_ERROR_MSG* of the class *CL_SAPBC405_EXC_HANDLER* and pass the object `go_alv_error` for the *r_error* parameter.

6. Call the instance method *DISPLAY* of the class *CL_SALV_TABLE* for your ALV instance.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 21: ALV OM: Create for Fullscreen

Task:

Perform the following task:

1. Create the program *ZBC405_##_OM*.

   Model Solution: *BC405_OM_CREATE_S1*

   a) Proceed in the usual manner.

2. Create an internal table `gt_flights` with the row type `sflight`. Create a work area for the internal table.

   a)
```
* Internal table
DATA: gt_flights TYPE TABLE OF sflight.
* Workarea for data fetch
DATA: gs_flight TYPE           sflight.
```

3. Create two select-options, one for airlines, and one for connection numbers. Then retrieve the flight data taking account of the selection options and store the data in the internal table `gt_flights`.

   a)
```
SELECT-OPTIONS:
  so_car FOR gs_flight-carrid MEMORY ID car,
  so_con FOR gs_flight-connid.
...
* retrieve data into internal table
  SELECT * FROM sflight INTO TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.
```

4. Create a reference variable `go_alv_table` that is typed with `cl_salv_table`.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Implement the method call for creating the ALV instance with the class method *FACTORY* of the class `CL_SALV_TABLE`. Generate the statement using the ABAP pattern or by drag and drop, and assign the values to the parameters of the method call.

a)

```
* create ALV instance with class-method cl_salv_table=>factory
      cl_salv_table=>factory(
*       EXPORTING
*         LIST_DISPLAY   = IF_SALV_C_BOOL_SAP=>FALSE
*         R_CONTAINER    =
*         CONTAINER_NAME =
        IMPORTING
          r_salv_table   = go_alv_table
        CHANGING
          t_table        = gt_flights ).
```

5. Implement handling for the exception `cx_salv_msg` of the method *FACTORY*. Follow these steps:

   Create a reference variable `go_alv_error` of type `cx_salv_error`.

   Activate the generated `CATCH` statement after the method call and extend the statement with the addition `INTO go_alv_error`.

   Call the global method *PROCESS_ALV_ERROR_MSG* of the class *CL_SAPBC405_EXC_HANDLER* and pass the object `go_alv_error` for the *r_error* parameter.

   a)

```
DATA: go_alv_error TYPE REF TO cx_salv_error.
...
TRY.
    cl_salv_table=>factory( ... ).
  CATCH cx_salv_msg INTO go_alv_error.

    cl_sapbc405_exc_handler=>process_alv_error_msg(
      EXPORTING
        r_error = go_alv_error ).
ENDTRY.
```

6. Call the instance method *DISPLAY* of the class *CL_SALV_TABLE* for your ALV instance.

   a)

```
* display data
  go_alv_table->display( ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 22: ALV OM: Create for Classic List

Exercise Objectives

After completing this exercise, you will be able to:

- Display an ALV as a classic list or in full screen mode

Business Example

You are a programmer for ABC Travel. The users of your program for flight data reporting want to display the data in an ALV, choosing between full screen mode and a classic list.

Task:

Perform the following task:

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_CREATE_S1*.

   Model Solution: *BC405_OM_CREATE_S2*

2. On the selection screen, create a radio button group with the values **Fullscreen** and **Classic List**.

3. Extend the call of the class method *FACTORY* of the class *CL_SALV_TABLE* so that the data can be displayed as a full screen ALV or as a classic list, as desired. It is best if you specify the passed value for the `EXPORT` parameter *list_display* in a variable beforehand.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 22: ALV OM: Create for Classic List

Task:

Perform the following task:

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_CREATE_S1*.

   Model Solution: *BC405_OM_CREATE_S2*

   a) Proceed in the usual manner.

2. On the selection screen, create a radio button group with the values **Fullscreen** and **Classic List**.

   a)
```
SELECTION-SCREEN BEGIN OF BLOCK alv WITH FRAME TITLE text-alv.
PARAMETERS: pa_full RADIOBUTTON GROUP alvd DEFAULT 'X',
            pa_list RADIOBUTTON GROUP alvd.
SELECTION-SCREEN END OF BLOCK alv.
```

3. Extend the call of the class method *FACTORY* of the class *CL_SALV_TABLE* so that the data can be displayed as a full screen ALV or as a classic list, as desired. It is best if you specify the passed value for the `EXPORT` parameter *list_display* in a variable beforehand.

   a)
```
DATA: gv_list_display TYPE sap_bool.
...
* decide about list display
  IF pa_full IS NOT INITIAL.
    gv_list_display = if_salv_c_bool_sap=>false.
  ELSE.
    gv_list_display = if_salv_c_bool_sap=>true.
  ENDIF.
...
  cl_salv_table=>factory(
    EXPORTING
      list_display = gv_list_display
    IMPORTING
      r_salv_table = go_alv_table
    CHANGING
      t_table      = gt_flights ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Create an ALV with the output formats “full screen” and “classic ABAP list”
- Display data in the ALV

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV OM: Creating in the Container

Lesson Overview

In this lesson, you learn how to display the ALV in a container area of a screen. This is useful if you want to display other screen elements, such as input and output fields on the screen at the same time as the ALV.

Lesson Objectives

After completing this lesson, you will be able to:

- Display an ALV as a subarea of a dynpro

Business Example

You are working as a programmer for ABC Travel. Your manager wants the flight data to be displayed in an ALV that only includes part of the screen.

Displaying the ALV as a Grid in a Custom Container

This procedure allows you to display the ALV in only part of a screen, while the other part of the screen can be used for other screen elements. While a programmer does not need to think about screen control and screen layout at all with full screen and the classic list, the following steps are essential here:

- Create a screen
- Assign at least a minimum status at PBO to be able to leave the dynpro
- Implement at least a minimum flow logic for PAI, to be able to leave the dynpro
- Create a custom control area on the dynpro
- For PBO: Create a container control instance and assigning it to the custom control area of the screen
- Create an ALV instance with the class method *FACTORY* of class *CL_SALV_TABLE* using the parameter *r_container*. The ALV is linked with the container control using this parameter.

These steps are described in more detail in the following.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

ALV, Container Control and Screen

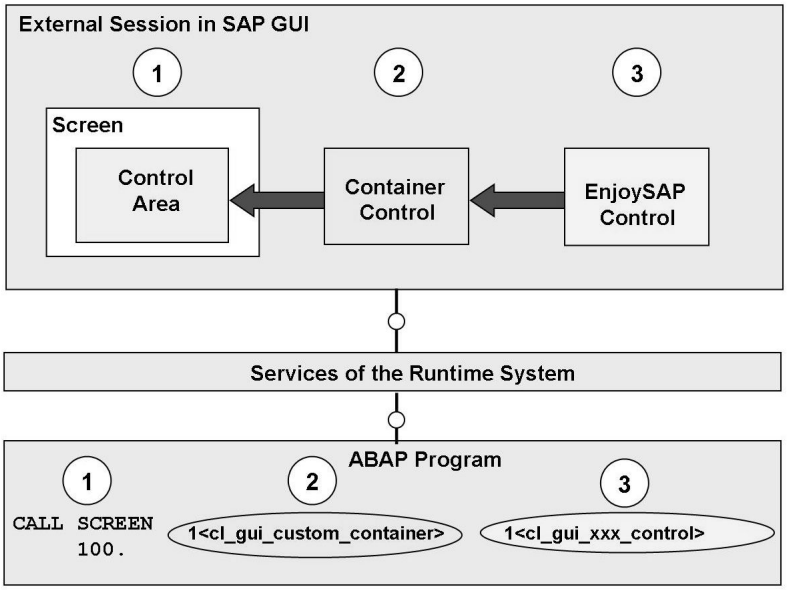


Figure 180: Controls and Dynpros

An ALV **in a screen** cannot be generated on the presentation server in isolation, it always needs two partners: a screen and a so-called Container Control.

The container control attaches an ALV to a screen:

- The dynpro contains an area reserved for the container (similar to a subscreen area that is reserved for a subscreen dynpro).
- You assign an instance of the container control class (here: cl_gui_custom_container) to the reserved area on the screen.
- You assign the actual ALV instance to the container control instance.

All GUI controls (including the ALV and Container Control) are controlled on the presentation server. On the other hand, all ABAP programs run on the application server. So that a program can influence the controls at the presentation server, representative instances are created. Changes to these representative instances are automatically transferred to the presentation server. A program for generating an ALV in a screen area therefore communicates with two objects and a dynpro.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

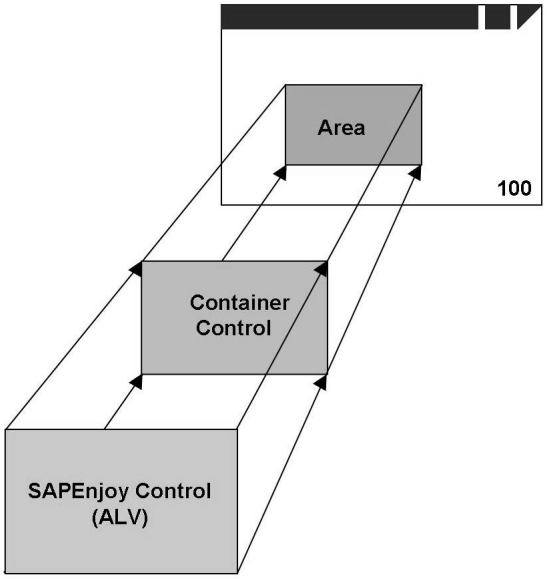


Figure 181: Graphic Illustration of the Objects

If we look at it graphically, attaching an ALV to a screen is just like “nesting” two areas, one inside the other:

- An area is reserved on the screen to display a control.
- A container control is visualized in this area.
- The area of the container control itself acts as the basis for the visualization of the ALV Control.

The following now shows what you need to do to be able to perform the three specified steps in an ABAP program.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

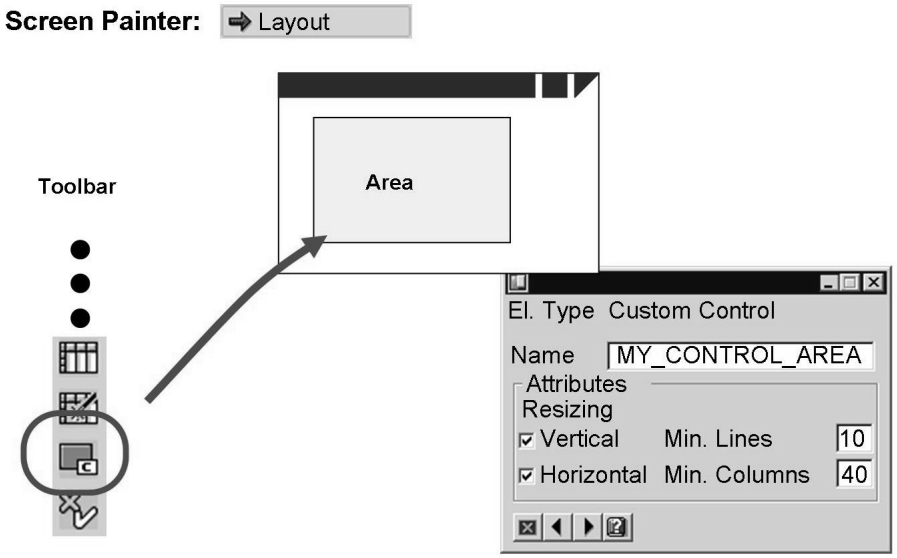


Figure 182: Creating the Custom Control Screen Element

To reserve a screen area for an ALV, start the graphical Fullscreen Editor of the Screen Painter (choose the *Layout* pushbutton).

In the toolbar on the left next to the editing area, there is an icon with the text *Custom Control*. Operation:

- Choose the *Custom Control* button (usually with the left mouse button). Go to the editing area of the screen and define the size and position of the area on the screen. To do this, proceed as follows:

  You determine the upper left corner of the area by pressing the select button of your mouse. Hold down the mouse button and drag the mouse diagonally to the right and down, to the desired lower right corner of the area to be created. As soon as you let go of the mouse button, the lower right corner is fixed.

- You can change the size and position of the area by drag-and-drop at any time using the drag points that appear at the edge of the area.

Enter a name for the newly created screen element (above: `my_control_area`).

You can use the two attributes *Resizing vertical* and *Resizing horizontal* to define whether the area is also to be rescaled, if necessary, if the size of the screen changes. If you set the attributes, you can use the two attributes *Min. Lines* and *Min. Columns* to define the minimum size to which the area can be minimized. The upper limit for the size of the area is defined by the created size.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Generating the Container Control

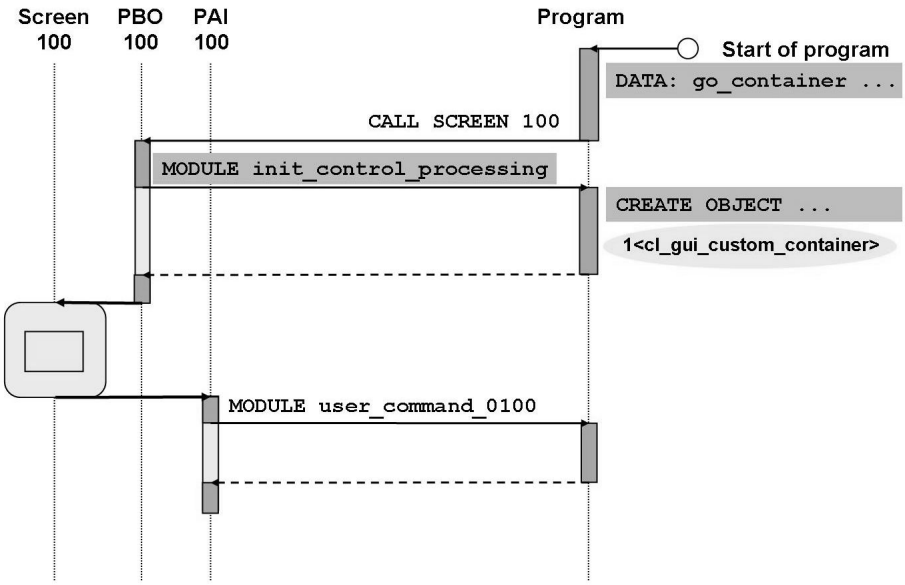


Figure 183: Creating a Container Control Instance: Process Flow

You then need to generate an instance of the container. You should do this in your program before the screen on which the ALV is to appear is displayed at the presentation server. The processing time PBO of the dynpro that contains the reserved area is suitable for this.

To make your program easier to follow, place all control programming in a separate module (in this example: Module `init_control_processing`).

To be able to create a container control instance, you also require a reference variable for your instance.

The individual steps are explained in more detail in the following.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
DATA: <object_name> TYPE REF TO <class_name>.
CREATE OBJECT <object_name>
  EXPORTING
    <parameter_name_1> = <program_value_1>
      ...
    <parameter_name_n> = <program_value_n>
  EXCEPTIONS
    <exception_name_1> = <return_code_1>
      ...
    <exception_name_m> = return_code_m>.
```

Interface of the constructor (Method CONSTRUCTOR)

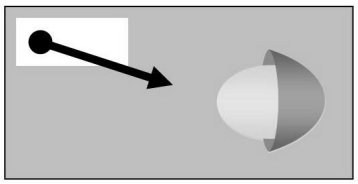


Figure 184: Syntax: Creating Instances

Using the `CREATE OBJECT <object_name>` ABAP statement, create an instance of the class for which the `<object_name>` reference variable has been assigned a type.

If the class for which you want to generate an instance has a method with the name *CONSTRUCTOR*, this method is automatically executed when the instance is generated. As an interface, the *CONSTRUCTOR* method can only have `IMPORTING` parameters and exceptions (`EXCEPTIONS`). The interface must also be specified in the `CREATE OBJECT` statement. When you do so, you need to assign values to all of the mandatory `IMPORTING` parameters, which do not have any default values.

The syntax of the interface for the call is the same as for function modules or normal method calls:

- The names of the interface parameters of the method are in the `EXPORTING` part on the left, and the literals or program variables (fields) are on the right of the equal sign.
- You can assign whole numbers as desired to the exceptions listed in the interface These are then in the field `sy-subrc` after the method has been executed.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

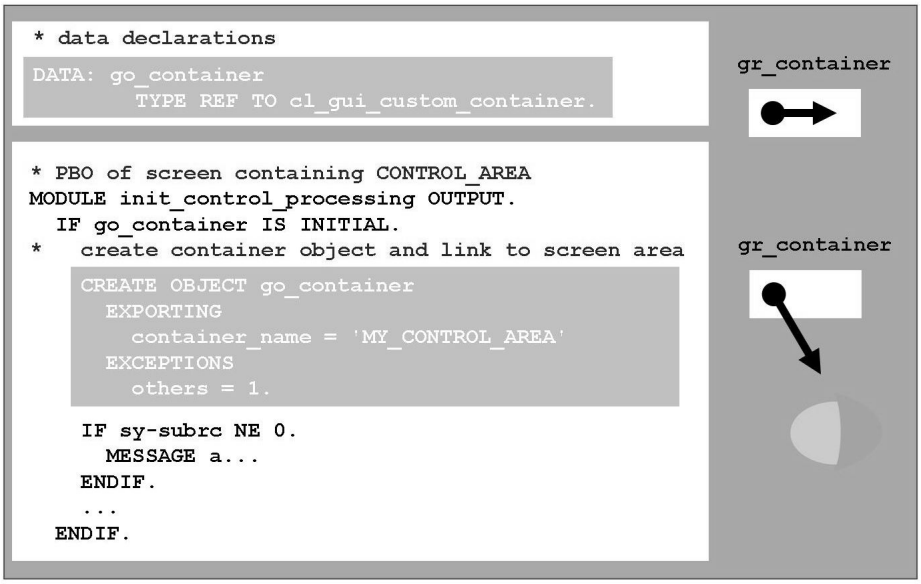


Figure 185: Example: Creating a Container Control Instance

To create a reference variable for your container control instance, use `DATA` to create a variable (above: `go_container`), which is assigned a type using `TYPE REF TO cl_gui_custom_container`.

You generate a container control instance at the presentation server by using the ABAP statement `CREATE OBJECT` to generate an instance of the class *CL_GUI_CUSTOM_CONTAINER* as a representative object.

Using the `IMPORTING` parameter *container_name*, you transfer the name of the area reserved for the control on the screen (above: `MY_CONTROL_AREA`).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

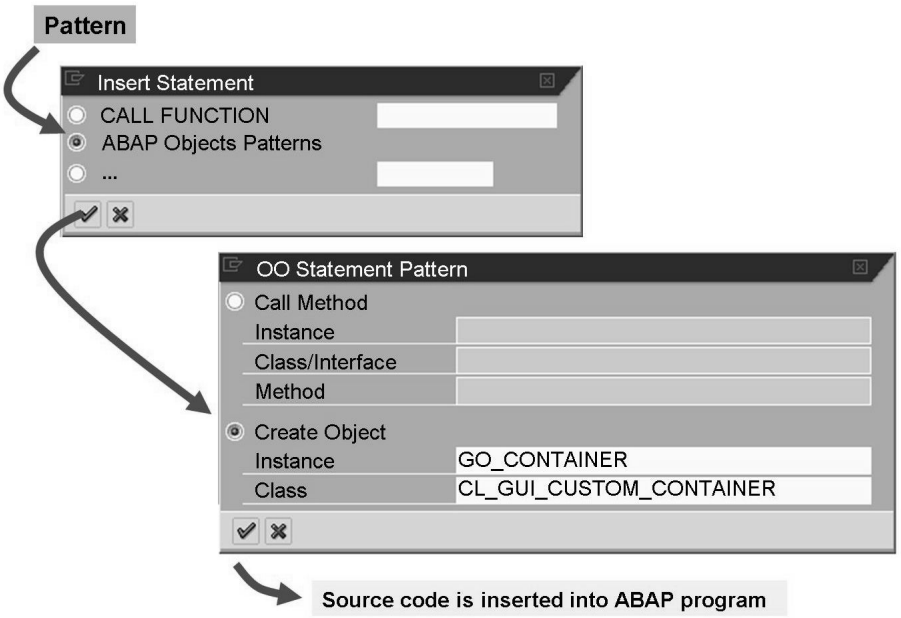


Figure 186: Inserting `CREATE OBJECT` by Pattern

Use the ABAP Objects pattern to insert the `CREATE OBJECT` statement.

On the *Insert Statement* initial screen, choose the *ABAP Objects Patterns* option.

On the next screen, choose the *Create Object* option. Specify the reference variable for the object to be created under *Instance*, and the class of the object to be created under *Class*.

The statement `CREATE OBJECT`, including all parameters and exceptions of the constructor method, is then inserted in the ABAP source code Optional parameters and exceptions are inserted as comments.

In this way, you can avoid typing errors and omissions.

You then need to complete the actual parameters and possibly program the exception handling.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

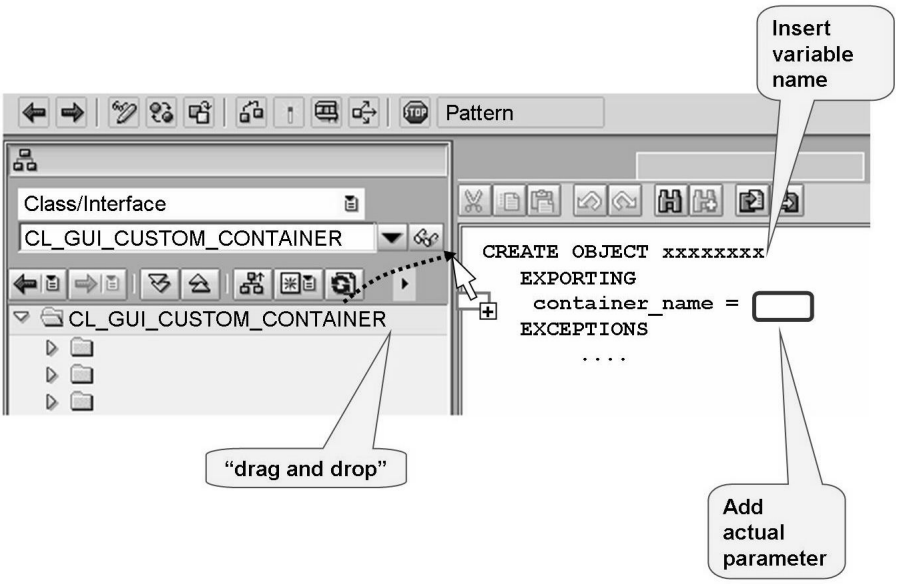


Figure 187: Inserting `CREATE OBJECT` by Drag and Drop

You can also use the drag-and-drop function of the ABAP Workbench to insert the statement `CREATE OBJECT` for a global class.

You can display the global class (in this example: *CL_GUI_CUSTOM_CONTAINER*) in the navigation area and transfer it to the editing area using drag-and-drop.

The ABAP statement `CREATE OBJECT`, including the interface parameters and exceptions, is inserted at the current cursor position in the source code of the program. Optional parameters and exceptions are inserted as comments.

You then need to complete the actual parameters and possibly program the exception handling.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

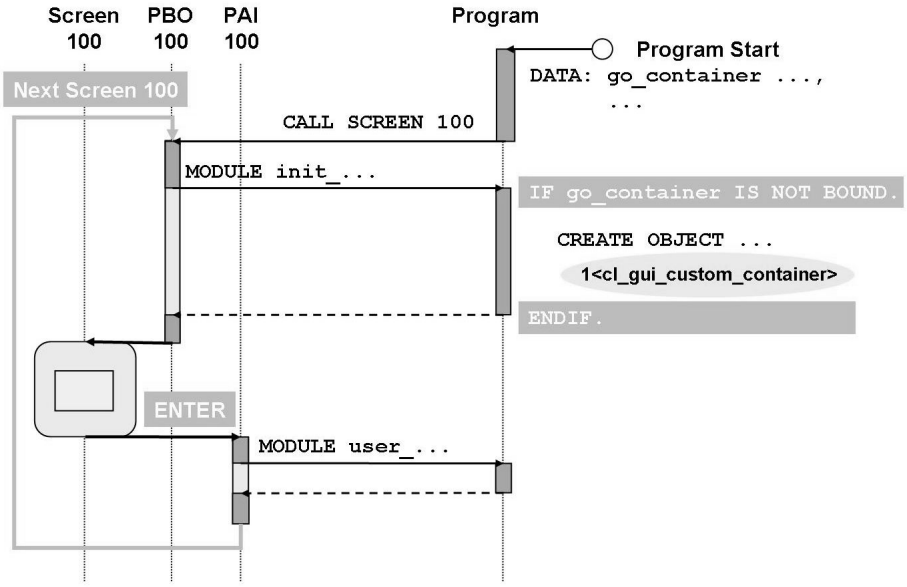


Figure 188: Avoiding Multiple Generation of Controls

If you package the control-specific processing in a PBO module, you need to ensure that an instance is only generated the first time the dynpro is called. Otherwise there is a danger that a new object will be generated every time the dynpro is processed:

- The container screen usually has itself statically as the next screen.
- Each action on the screen that triggers PAI (such as <Enter>) means that the dynpro is processed again after PAI has been processed.
- If method calls were performed for PAI operations at the presentation server control, their results would not be visible in the SAP GUI window after the dialog step, since a new instance would be linked with the container screen at every PBO.

You can query the container reference for initial content (above: `go_container`) to make sure that no unwanted instances are being generated.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

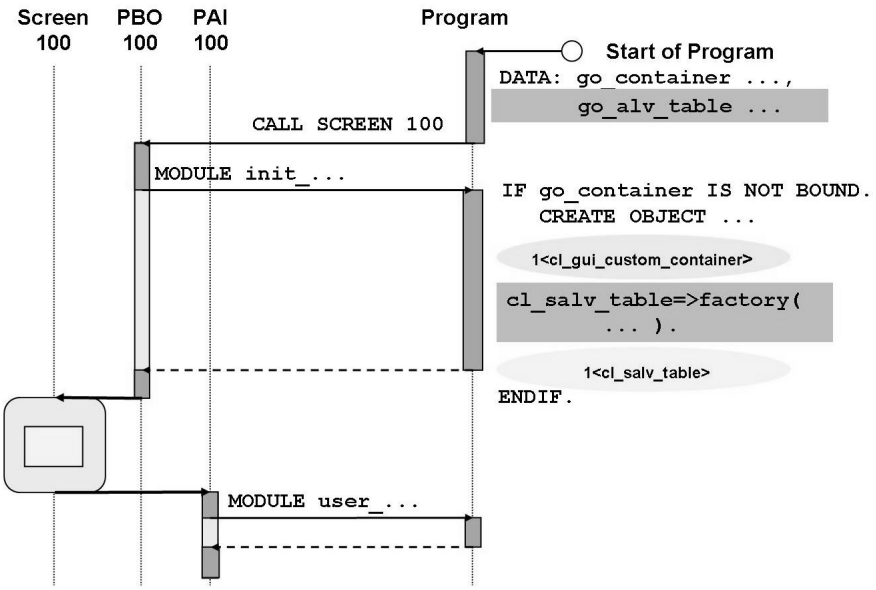


Figure 189: Creating an ALV instance: Process Flow

You create an ALV instance in the same way that you created the instance for the container control.

The ALV instance must be created after the container control instance, but before the SAP GUI window is sent to the presentation server. However, the instance is generated with the class method *FACTORY* of class *CL_SALV_TABLE*, not using a `CREATE OBJECT` command. When calling the method, pay particular attention to the following parameters:

r_container

Here, you have to specify the name of the container control you previously created (in the example `go_container`).

list_display

This parameter does not need to be used here. If you only want to decide the format the ALV is to be displayed at runtime, this parameter must always have the value “space”, otherwise runtime errors occur.

You should evaluate the parameters ***t_table*** and ***r_salv_table*** as described under “Creating and Displaying the ALV OM” above.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* data declarations
DATA: go_container     TYPE REF TO cl_gui_custom_container,
      go_alv_table     TYPE REF TO cl_salv_table.
```

```
MODULE init_control_processing OUTPUT.

  IF go_container IS INITIAL.
    ...
*   create ALV object and link to container
    cl_salv_table=>factory(
         EXPORTING
           r_container    = go_container
         IMPORTING
           r_salv_table   = go_alv_table
         CHANGING
           t_table        = gt_bookings ).

ENDIF.
```

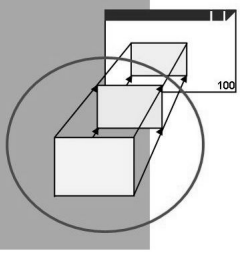


Figure 190: Creating an ALV instance

Finally, you now need to call the method *DISPLAY* of the class *CL_SALV_TABLE* for the ALV instance that you have just created. Note that this must also take place in the PBO module.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 23: Creating a Container Screen for an ALV

Exercise Objectives

After completing this exercise, you will be able to:

- Create a screen
- Prepare the screen for an ALV in the container
- Program a simple command processing

Business Example

You are a programmer at ABC Travel. The users of your program for flight data reporting would like the flight data to be displayed in an ALV on their screen and to be able to access entry fields for their own flight bookings at the same time. Your first task is to program a container screen and flow logic for the ALV.

Task:

Solve the following problem:

1. Enhance your program *ZBC405_##_OM* or copy the program *BC405_OM_CREATE_S2*.

   Model Solution: *BC405_OM_CREATE_S3*

2. Add a parameter for “Output in the container” to the “Fullscreen/List” radio button group.

3. Implement a `CASE` structure: If the user has chosen *Fullscreen* or *List*, the system should run the original processing; if he has chosen “Output in the container”, a different processing should be run.

4. Create a screen 100 to be called at `START-OF-SELECTION` time. Tip: This is best achieved by forward navigation from your coding. Make sure you have chosen the screen type *Normal* in the dynpro attributes and that the next screen is set to 100.

5. In the screen 100, use the graphical Screen Painter to create a custom control area with the following attributes:

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Name	**MY_CONTROL_AREA**
Resizing vertical	On
Resizing horizontal	On
Visible length	100 (suggestion)
Height	25 (suggestion)

6. Using forward navigation in the flow logic of screen 100, create a PBO module with the name *STATUS_0100*.

   Create the module in a new Include. Starting from this module, create a status *DYN* of the *Dialog Status* type. In the Menu Painter, provide the functions *Back, Cancel* and *Exit* with function codes (such as *BACK*, *CANCEL* and *EXIT*).

   To be able to query the OK code, you have to name the corresponding entry in the screen's element list (as a rule, it is *ok_code*). Besides, you need a field of the same name in the ABAP coding, to which you have to assign a type with reference to *sy-ucomm*.

   Then create a PAI module *USER_COMMAND_0100* (either in the main program or in a new Include) and program the command processing there. The selection screen should be displayed in the case of all three commands (*Back*, *Cancel* and *Exit*).

   Test whether the screen call and return to the selection screen are functioning correctly.

   Optional: Make it possible for the user to also end the program by selecting a menu entry.

7. Create and set a title

   Starting from the PBO module *STATUS_0100*, use forward navigation to create a title bar named *T100*. As the title, you can choose e.g. “Flights”.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 23: Creating a Container Screen for an ALV

Task:

Solve the following problem:

1. Enhance your program *ZBC405_##_OM* or copy the program *BC405_OM_CREATE_S2*.

   Model Solution: *BC405_OM_CREATE_S3*

   a) self-explanatory

2. Add a parameter for "Output in the container" to the "Fullscreen/List" radio button group.

   a) See the complete solution after the last step of the task

3. Implement a `CASE` structure: If the user has chosen *Fullscreen* or *List*, the system should run the original processing; if he has chosen "Output in the container", a different processing should be run.

   a) See the complete solution after the last step of the task

4. Create a screen 100 to be called at `START-OF-SELECTION` time. Tip: This is best achieved by forward navigation from your coding. Make sure you have chosen the screen type *Normal* in the dynpro attributes and that the next screen is set to 100.

   a) See the complete solution after the last step of the task

5. In the screen 100, use the graphical Screen Painter to create a custom control area with the following attributes:

Name	**MY_CONTROL_AREA**
Resizing vertical	On
Resizing horizontal	On
Visible length	100 (suggestion)
Height	25 (suggestion)

   a) See the complete solution after the last step of the task

6. Using forward navigation in the flow logic of screen 100, create a PBO module with the name *STATUS_0100*.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Create the module in a new Include. Starting from this module, create a status *DYN* of the *Dialog Status* type. In the Menu Painter, provide the functions *Back, Cancel* and *Exit* with function codes (such as *BACK*, *CANCEL* and *EXIT*).

To be able to query the OK code, you have to name the corresponding entry in the screen's element list (as a rule, it is *ok_code*). Besides, you need a field of the same name in the ABAP coding, to which you have to assign a type with reference to *sy-ucomm*.

Then create a PAI module *USER_COMMAND_0100* (either in the main program or in a new Include) and program the command processing there. The selection screen should be displayed in the case of all three commands (*Back*, *Cancel* and *Exit*).

Test whether the screen call and return to the selection screen are functioning correctly.

Optional: Make it possible for the user to also end the program by selecting a menu entry.

a) See the complete solution after the last step of the task

7. Create and set a title

Starting from the PBO module *STATUS_0100*, use forward navigation to create a title bar named *T100*. As the title, you can choose e.g. “Flights”.

a) See the following for the source code of the model solution.

Main program

```
REPORT bc405_om_create_s3.

INCLUDE bc405_om_create_s3_top.
INCLUDE bc405_om_create_s3_e01.
INCLUDE bc405_om_create_s3_o01.
INCLUDE bc405_om_create_s3_i01.
```

TOP Include

```
DATA: gt_flights TYPE TABLE OF sflight.
DATA: gs_flight  TYPE          sflight.

DATA: gv_list_display TYPE sap_bool.

DATA: go_alv_table TYPE REF TO cl_salv_table.
DATA: go_alv_error TYPE REF TO cx_salv_error.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
* user-command from dynpro
DATA: ok_code LIKE sy-ucomm.

SELECT-OPTIONS:
  so_car FOR gs_flight-carrid MEMORY ID car,
  so_con FOR gs_flight-connid.

SELECTION-SCREEN SKIP 1.
SELECTION-SCREEN BEGIN OF BLOCK alv WITH FRAME TITLE text-alv.
PARAMETERS: pa_full RADIOBUTTON GROUP alvd,
            pa_cont RADIOBUTTON GROUP alvd DEFAULT 'X',
            pa_list RADIOBUTTON GROUP alvd.
SELECTION-SCREEN END OF BLOCK alv.
```

E01 Include

```
*&---------------------------------------------------------------------*
*&   Event START-OF-SELECTION
*&---------------------------------------------------------------------*
START-OF-SELECTION.
* retrieve data into internal table
  SELECT * FROM  sflight
    INTO TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

  CASE 'X'.
    WHEN pa_full OR pa_list.

* decide about list display
      IF pa_list IS NOT INITIAL.
        gv_list_display = if_salv_c_bool_sap=>true.
      ELSE.
        gv_list_display = if_salv_c_bool_sap=>false.
      ENDIF.

* create ALV instance
      TRY.
          cl_salv_table=>factory(
            EXPORTING
              list_display   = gv_list_display
*             R_CONTAINER    =
*             CONTAINER_NAME =
            IMPORTING
```

Continued on next page

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
              r_salv_table   = go_alv_table
            CHANGING
              t_table        = gt_flights ).
        CATCH cx_salv_msg INTO go_alv_error.
          cl_sapbc405_exc_handler=>process_alv_error_msg(
              r_error = go_alv_error
*             TYPE    = 'A'
                 ).
      ENDTRY.

* display ALV
      go_alv_table->display( ).

    WHEN pa_cont.
      CALL SCREEN 100.
  ENDCASE.
```

Flow Logic of Screen 100

```
PROCESS BEFORE OUTPUT.
  MODULE status_0100.
*
PROCESS AFTER INPUT.
  MODULE user_command_0100.
```

O01 Include

```
*&---------------------------------
*&      Module  STATUS_0100  OUTPUT
*----------------------------------
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'S100'.
  SET TITLEBAR 'T100'.
ENDMODULE.   " STATUS_0100  OUTPUT
```

I01 Include

```
*&--------------------------------------
*&      Module  USER_COMMAND_0100  INPUT
*--------------------------------------
MODULE user_command_0100 INPUT.
  CASE sy-ucomm.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
    WHEN 'BACK' OR 'EXIT' OR 'CANCEL'.
      SET SCREEN 0.
  ENDCASE.
ENDMODULE. " USER_COMMAND_0100  INPUT
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 24: ALV OM: Creating in the Container

Exercise Objectives

After completing this exercise, you will be able to:

- Create a dynpro
- Prepare the dynpro for an ALV Grid Control
- Program simple command processing
- Link a container control to a dynpro
- Display an ALV in a container control

Business Example

You are a programmer for ABC Travel. The users of your program for flight data reporting want, on the one hand, to have the flight data displayed in an ALV, and, on the other hand, to have input fields available for their own flight bookings. It is now your task to display the ALV on part of the screen. It is not your task to program the input fields.

Task:

Perform the following task:

1. Enhance your *ZBC405_##_OM* program or copy the program *BC405_OM_CREATE_S3*.

   Model Solution: *BC405_OM_CREATE_S4*

2. Define a reference variable (suggested name: *go_container*) for an object of the class *CL_GUI_CUSTOM_CONTAINER*.

3. Implement a module in the PBO event block of screen 100 in which the objects are to be created (suggested name: *create_and_display*). Use the pattern button or drag and drop to code the `CREATE OBJECT` command. Valuate the obligatory parameter: *container_name* for the Custom Container Control (enter the name of the custom control area here).

   Evaluate the exceptions by combining them with *OTHERS*. If an error occurs, call message 015 from the message class BC405 as a termination message. Generate the objects only if the reference variable *go_container* for the container has initial status.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

4. Use the class method *FACTORY* of the class *CL_SALV_TABLE* to generate an ALV object. When you do so, ensure that you assign the previously created container object to the parameter *r_container* parameter. Implement a *TRY...ENDTRY* block around the method call.
5. Call the instance method *DISPLAY* of the class *CL_SALV_TABLE* for your ALV instance.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 24: ALV OM: Creating in the Container

Task:

Perform the following task:

1. Enhance your *ZBC405_##_OM* program or copy the program *BC405_OM_CREATE_S3*.

   Model Solution: *BC405_OM_CREATE_S4*

   a) Proceed in the usual manner.

2. Define a reference variable (suggested name: *go_container*) for an object of the class *CL_GUI_CUSTOM_CONTAINER*.

   a) See complete solution after last exercise step

3. Implement a module in the PBO event block of screen 100 in which the objects are to be created (suggested name: *create_and_display*). Use the pattern button or drag and drop to code the `CREATE OBJECT` command. Valuate the obligatory parameter: *container_name* for the Custom Container Control (enter the name of the custom control area here).

   Evaluate the exceptions by combining them with *OTHERS*. If an error occurs, call message 015 from the message class BC405 as a termination message. Generate the objects only if the reference variable *go_container* for the container has initial status.

   a) See the complete solution after the last step of the task

4. Use the class method *FACTORY* of the class *CL_SALV_TABLE* to generate an ALV object. When you do so, ensure that you assign the previously created container object to the parameter *r_container* parameter. Implement a *TRY...ENDTRY* block around the method call.

   a) See the complete solution after the last step of the task

5. Call the instance method *DISPLAY* of the class *CL_SALV_TABLE* for your ALV instance.

   a) The complete source code for the model solution is shown below.

Main program

```
REPORT bc405_om_create_s4.

INCLUDE bc405_om_create_s4_top.
INCLUDE bc405_om_create_s4_e01.
INCLUDE bc405_om_create_s4_o01.
```

Continued on next page

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
INCLUDE bc405_om_create_s4_i01.
```

TOP Include

```
DATA: gt_flights TYPE TABLE OF sflight.
DATA: gs_flight  TYPE          sflight.

DATA: gv_list_display TYPE sap_bool.

* Reference for container control
DATA: go_container TYPE REF TO cl_gui_custom_container.
DATA: go_alv_table TYPE REF TO cl_salv_table.
DATA: go_alv_error TYPE REF TO cx_salv_error.

* user-command from dynpro
DATA: ok_code LIKE sy-ucomm.

SELECT-OPTIONS:
  so_car FOR gs_flight-carrid MEMORY ID car,
  so_con FOR gs_flight-connid.

SELECTION-SCREEN SKIP 1.
SELECTION-SCREEN BEGIN OF BLOCK alv WITH FRAME TITLE text-alv.
PARAMETERS: pa_full RADIOBUTTON GROUP alvd,
            pa_cont RADIOBUTTON GROUP alvd DEFAULT 'X',
            pa_list RADIOBUTTON GROUP alvd.
SELECTION-SCREEN END OF BLOCK alv.
```

E01 Include

```
*&---------------------------------------------------------------------*
*&   Event START-OF-SELECTION
*&---------------------------------------------------------------------*
START-OF-SELECTION.
* retrieve data into internal table
  SELECT * FROM sflight
    INTO TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

  CASE 'X'.
    WHEN pa_full OR pa_list.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* decide about list display
      IF pa_list IS NOT INITIAL.
        gv_list_display = if_salv_c_bool_sap=>true.
      ELSE.
        gv_list_display = if_salv_c_bool_sap=>false.
      ENDIF.

* create ALV instance
      TRY.
          cl_salv_table=>factory(
            EXPORTING
              list_display   = gv_list_display
*             R_CONTAINER    =
*             CONTAINER_NAME =
            IMPORTING
              r_salv_table   = go_alv_table
            CHANGING
              t_table        = gt_flights ).
        CATCH cx_salv_msg INTO go_alv_error.
          cl_sapbc405_exc_handler=>process_alv_error_msg(
              r_error = go_alv_error
*             TYPE    = 'A'
                ).
      ENDTRY.

* display ALV
      go_alv_table->display( ).

    WHEN pa_cont.
      CALL SCREEN 100.
  ENDCASE.
```

Screen 100 Flow Logic

```
PROCESS BEFORE OUTPUT.
  MODULE status_0100.
  MODULE create_control.

PROCESS AFTER INPUT.
  MODULE user_command_0100.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

O01 Include

```
*&---------------------------------------------------------------------*
*&      Module  STATUS_0100  OUTPUT
*&---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'S100'.
  SET TITLEBAR 'T100'.
ENDMODULE.   " STATUS_0100  OUTPUT

*&---------------------------------------------------------------------*
*&      Module  create_control  OUTPUT
*&---------------------------------------------------------------------*
MODULE create_control OUTPUT.
  IF go_container IS NOT BOUND.

* create container control and link it to the dynpro
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTAINER'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a015(bc405).
    ENDIF.

* create ALV and link it to the container control
    TRY.
        cl_salv_table=>factory(
          EXPORTING
*           LIST_DISPLAY   = IF_SALV_C_BOOL_SAP=>FALSE
            r_container    = go_container
*           CONTAINER_NAME =
          IMPORTING
            r_salv_table   = go_alv_table
          CHANGING
            t_table        = gt_flights ).
      CATCH cx_salv_msg INTO go_alv_error.
    ENDTRY.

* display ALV
    go_alv_table->display( ).
  ENDIF.
ENDMODULE.    " create_control  OUTPUT
```

Continued on next page

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

I01 Include

```
*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.
  CASE sy-ucomm.
    WHEN 'BACK' OR 'EXIT' OR 'CANCEL'.
      SET SCREEN 0.
  ENDCASE.
ENDMODULE. " USER_COMMAND_0100  INPUT
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Display an ALV as a subarea of a dynpro

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV OM: Fetching and Modifying Subobjects of a Main Class

Lesson Overview

This lesson will acquaint you with the basic principle for all changes to the ALV: subobjects of the main class of the ALV are fetched and modified. A subobject represents a particular aspect of the ALV, such as the layout or a column. First, the class hierarchy of the ALV OM is considered, and examples are then used to show what a change to an ALV property looks like in practice.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the class hierarchy of the ALV OM
- Describe the basic procedure to change properties of an ALV

Business Example

You are working as a programmer for ABC Travel. Over time, the users of the flight data report have repeatedly come to you with changes that they would like in the appearance of the ALV, for example, with regard to the first sorting, the initial column order, and so on. Before you deal with the individual tasks, you first want to obtain a conceptual understanding of the basic procedure in a case of this sort.

Superclasses and Subclasses in the ALV OM

To change an ALV using the ALV OM approach, the programmer needs to have an object-oriented view of the ALV. To put it in a simple way: the ALV itself is an object, and its components, such as columns, functions, sorting etc. are subobjects of the ALV. If a programmer wants to change something about the ALV, he or she simply needs to

1. First fetch the relevant subobject that is responsible for the aspect to be change, that is, make it available for a changing access
2. Call the correct method to change the status of the subobject

The developer does not need to do anything else; the changes take effect immediately.

In the case of some elements of the ALV, such as columns and functions, there are other levels of superobjects and subobjects. The superobject “columns” contains properties that apply for all columns or that affect the interaction of all columns (such as the column order). The subobject “column xyz” (for example, the column

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

"Airline") contains properties that are only valid for one column, such as the column title. For changes here, follow the same procedure as for objects directly subordinated to the ALV: fetch the subobject and change it.

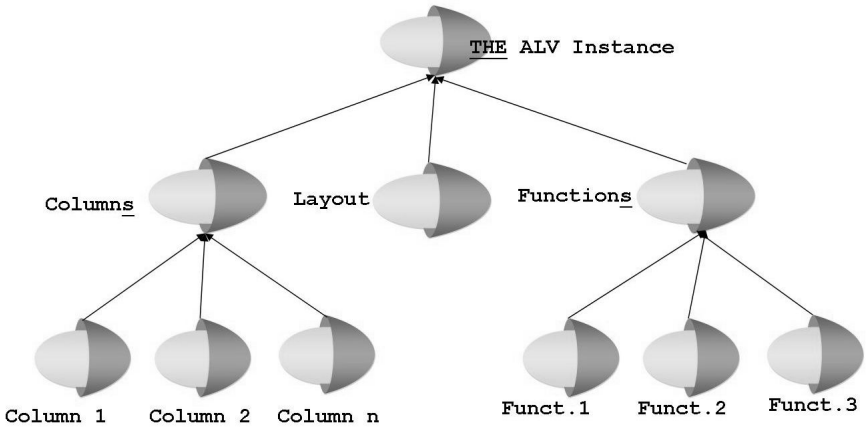


Figure 191: Principle of the ALV OM Object Model

The ALV OM Object Model

The uppermost class of the ALV OM class hierarchy is the class ***CL_SALV_TABLE***. ("Table" because this is the ALV to display simple two-dimensional tables. There is also an ALV to display trees and for hierarchical-sequential lists.) In subordinate classes, name conventions can help you find your way: If the name is set in the plural (for example, *CL_SALV_COLUMN**S**_TABLE*), it refers to the properties of a group of ALV elements. If the name is in the singular (such as *CL_SALV_COLUMN_TABLE*), you are dealing with the properties of a single element.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

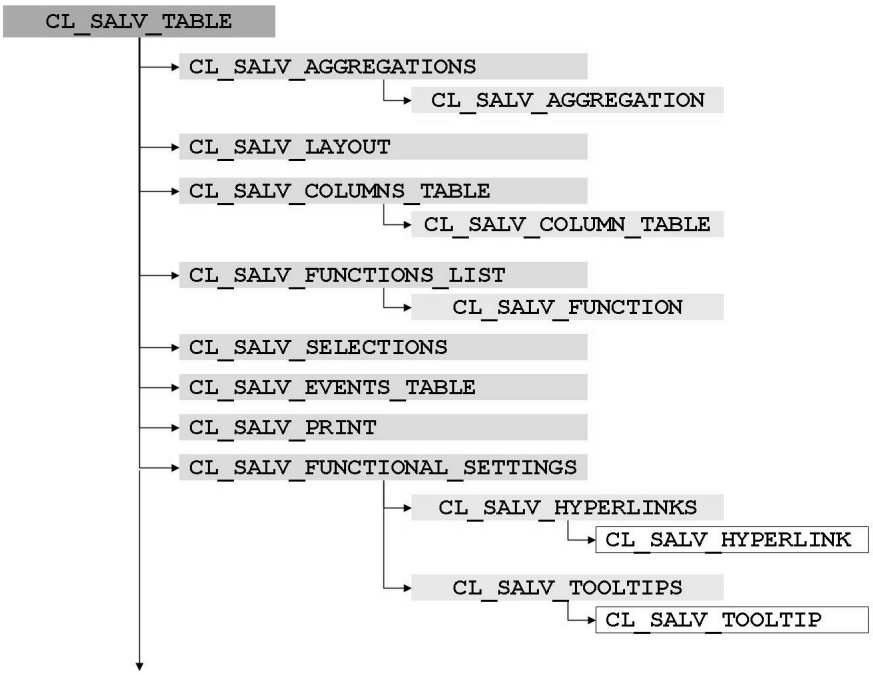


Figure 192: ALV OM Object Model Screen 1

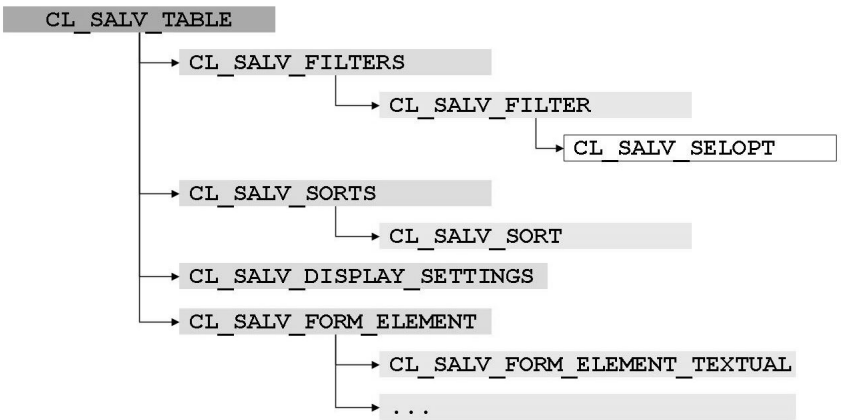


Figure 193: ALV OM Object Model Screen 2

Caution: The classes displayed here are not superclasses and subclasses in the sense of object orientation, that is, the classes lower in the hierarchy here do not inherit, for example, the properties of the classes higher in the hierarchy. The terms “hierarchy”, “superobject”, and “subobject” should rather illustrate how all objects together represent the ALV.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Changing the Properties of the ALV

You basically need to perform the two work steps below to change the properties of an ALV:

1. **Fetching a subobject:** The class *CL_SALV_TABLE* has *GET* methods for each subclass which you can use to fetch the appropriate object for the ALV. The subobject is, therefore, not newly created with a *CREATE OBJECT*, but rather is made available using a *CALL METHOD*.
2. **Applying the method to the subobject:** Each subclass provides methods suitable for the object you are dealing with (thus, the class *CL_SALV_COLUMNS* contains the method *SET_OPTIMIZE*, which ensures that all columns are displayed in their optimal width). This methods can be applied to the subobject that you have already fetched.

The same procedure essentially applies for working with subobjects of the second or third level (sub-subobjects). In this way, the class *CL_SALV_COLUMNS_TABLE* has a *GET* method with which you can fetch an object of the type *CL_SALV_COLUMN_TABLE* (that is, a single column).

The following examples should clarify this principle. These examples are intended only to illustrate the principle. You will learn all of the important details about working with the ALV elements used here, such as columns, aggregations, and functions, in later lessons.

```
* reference for columns object
DATA: lo_columns TYPE REF TO cl_salv_columns_table.
```

```
* get columns object
go_alv_table->get_columns(
    RECEIVING
      value = lo_columns ).

* optimize column widths
lo_columns->set_optimize( ).

* set position of a certain column
lo_columns->set_column_position(
    columnname = 'PLANETYPE'
    position   = 5 ).
```

Figure 194: Fetch Columns – Optimize Column Width

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

First, the object for all columns is fetched. Then two method calls are performed for the object, to optimize the column width and to assign a column position to a specific column (in this case: `planetype`).

From this example, you can learn the following:

- If the object for an ALV element is retrieved once, you can then make any number of changes to the object (using method calls).
- The programmer must, above all, know which method is the correct one and how he or she can easily find it. The entry points for this are the meaningful class names. There are a large number of methods for every class. Classes and methods are documented in detail in the Class Builder and in the SAP Library.
- The functional writing style is often significantly shorter for the *CALL-METHOD* command than the classic writing style.

```
* reference for aggregations object
  DATA lo_aggrs TYPE REF TO cl_salv_aggregations.
* reference for one aggregation object
  DATA lo_aggr  TYPE REF TO cl_salv_aggregation.
```

```
* get aggregations object
  lo_aggrs = go_alv_table->get_aggregations( ).

* get one specific aggregation object
  lo_aggr = lr_aggrs->get_aggregation(
              columnname = 'FREE_SEATS' ).

* set kind of aggregation for that aggr.
  lo_aggr->set(
    aggregation = if_salv_c_aggregation=>maximum ).
```

Figure 195: Fetch Aggregations – Change an Aggregation

This example shows how you can access subobjects of the second level. This two-level structure exists, for example, for columns - column, functions - function or, as in this case, for aggregations - aggregation. (Aggregation is a superordinate term for combinations across multiple rows, such as totals, averages, and so on.)

First, the object that represents all aggregations of the ALV is retrieved (in this case, `lo_aggrs`). To now be able to access an individual aggregation (for example, to change the type to “Maximum”), a *GET* method is performed on the “Aggregations” object again, which retrieves the object “Individual Aggregation for Column xyz” (in this case: `lo_aggr`) (a column can have a maximum of one aggregation). You can then change this individual aggregation as desired.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

This example deals with an especially strong feature of the functional method call: If a method has a `RETURNING` parameter, it can be called in the form `x = y->f( ... )`. In the example above, the method *GET_AGGREGATIONS* of the class *CL_SALV_TABLE* returns precisely one value, that being a reference to an object of the type *CL_SALV_AGGREGATIONS*. In this case, the functional writing style is especially useful. The classic writing style would be:

```
CALL METHOD alv->get_aggregations
  RECEIVING
    value = lo_aggrs.
```

Overview of the GET Methods for the ALV OM

In the list below, each of the *GET* methods listed is shown opposite the object that it delivers, together with its short description.

GET Methods of the CL_SALV_TABLE Class

GET Method	Delivered Object Is of the Type
GET_COLUMNS	*CL_SALV_COLUMNS_TABLE*: Column superclass
GET_FUNCTIONS	*CL_SALV_FUNCTIONS_ LIST*: Functions such as sorting, aggregating, etc.
GET_FUNCTIONAL_SETTINGS	*CL_SALV_FUNCTIONAL_ SETTINGS*: Hyperlink and tooltip superclass
GET_DISPLAY_SETTINGS	*CL_SALV_DISPLAY_SETTINGS*: Defining the display, i.e. title, stripe pattern, lines, etc.
GET_EVENT	*CL_SALV_EVENTS_ TABLE*: Events
GET_LAYOUT	*CL_SALV_LAYOUT*: Layout variant storage, management, etc.
GET_PRINT	*CL_SALV_PRINT*: Printer settings
GET_SELECTIONS	*CL_SALV_SELECTIONS*: Selection mode and selections

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

GET Method	Delivered Object Is of the Type
GET_AGGREGATIONS	*CL_SALV_AGGREGATIONS*: Aggregated objects
GET_SORTS	*CL_SALV_SORTS*: Sorting criteria
GET_FILTERS	*CL_SALV_FILTERS*: Filter criteria

There are also *GET* methods for subobjects of the second level (with regard to the ALV object). These do not apply to the ALV object, but rather to the affected subobjects of the first level, such as the object "All Columns of the ALV" (class *SALV_COLUMNS_TABLE*).

The overview below shows the *GET* methods used from these subclasses and the object classes that they deliver.

GET Methods for Subclasses of the *CL_SALV_TABLE* Class

Class CL_SALV_...	GET method GET_...	Returns Object of Type
AGGREGATIONS	*AGGREGATION*	*CL_SALV_AGGREGATION*: an individual aggregation
COLUMNS_TABLE	*COLUMN*	*CL_SALV_COLUMN*: one individual column
FUNCTIONAL_ SETTINGS	*HYPERLINKS*	*CL_SALV_HYPERLINKS*: Superclass for all hyperlinks
FUNCTIONAL_ SETTINGS	*TOOLTIPS*	*CL_SALV_TOOLTIPS*: Superclass for all tooltips
FUNCTIONS_LIST	*FUNCTIONS*	*CL_SALV_FUNCTION*: one individual function
FILTERS	*FILTER*	*CL_SALV_FILTER*: one individual filter criterion
SORTS	*SORTS*	*CL_SALV_SORT*: one individual sorting criterion

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Explain the class hierarchy of the ALV OM
- Describe the basic procedure to change properties of an ALV

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV OM: Adjusting the Display

Lesson Overview

In this lesson, you learn how you can use the ALV Object Model to set general display properties of the ALV. These properties include the contents and size of the title and the indicators “Striped pattern for output lines” and “Separators displayed”. You also learn how display properties that apply for all of the columns are defined, such as the “Optimize column width” and “Freeze key columns at horizontal scrolling” indicators.

Lesson Objectives

After completing this lesson, you will be able to:

- Set the title of an ALV
- Set other general display properties of the ALV
- Set display properties that apply to all output columns

Business Example

You are working as a programmer for ABC Travel. You are to change the title of the ALV for a program that displays flight data and to preconfigure the display of the ALV so that the key columns remain fixed if the user scrolls horizontally.

Title

The settings for the title are made with methods from the *CL_SALV_DISPLAY_SETTINGS* class. To do this, you first need to fetch (make available) the object in which the attributes of the title and other display properties of the ALV are stored. This is done using the method *GET_DISPLAY_SETTINGS* of the class *CL_SALV_TABLE*, which you must call as the instance method for the previously created ALV and which returns the display object (to be precise, one would have to say: the display properties object). To change the title, you then call the instance method *SET_LIST_HEADER* for the display object.

Fetching a Display Object and Setting the Title

```
 DATA: go_display TYPE REF TO cl_salv_display_settings,
       gv_title   TYPE        lvc_title.

* get object for display attributes
  go_display = go_alv_table->get_display_settings( ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* set title
  gv_title = text-ttl.
  go_display->set_list_header( value = gv_title ).
* header size: not suitable for classical list
  IF pa_cont = 'X' OR pa_full = 'X'.
    go_display->set_list_header_size(
      value = cl_salv_display_settings=>c_header_size_medium  ).
  ENDIF.
```

The code example shows another method call, the method *SET_LIST_HEADER_SIZE*. There are three possible values available here (small, medium, large), with which you can set the size of the title above the ALV display in full screen display. The size is set by means of class constant which is called `cl_salv_display_settings=>c_header_size_medium` in this example (Observe the correct notation **=>**!). To determine the name of a class constant, proceed as follows:

1. Switch to the Class Builder and select the relevant class (in the example, *CL_SALV_DISPLAY_SETTINGS*).
2. Go to the *Methods* tab page, place the cursor (in the example, *SET_LIST_HEADER_SIZE*) and choose the *Parameters* function. Note the associated type of the parameter used (in this case, the type is *salv_de_header_size* for the parameter *value*).
3. Now switch to the *Attributes* tab page and search for attributes that are typed with the type you have just noted (column *Associated Type*) and that have the attribute type *Constant*. Note the name of the class constant that you want to use.

Using class constants in the method call means that the values used in the program code are more meaningful, that is, the program is easier to read, and you do not need to deal with internal implementation of the method as much.

The example also shows that not every method is suitable for every output format. In this way, the method *SET_LIST_HEADER_SIZE* is not suitable for output with the classic ABAP list. To avoid runtime errors here, you should thoroughly test all output formats for programs that are to implement multiple output formats of the ALV. The documentation (see below) tells you which method is suitable for which output format. However, as a basic principle, only methods of less importance cannot be used with all three output formats.

Finally, the example shows that you always need to assign compatible data objects to method parameters. A program text element like the text `ttl` used here can have a maximum length of 132 characters (this would correspond to an assignment of the type `TYPE C(132)`). However, the parameter *value* of the method *SET_LIST_HEADER* has been assigned a type of `C(60)`. If the text element was

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

assigned directly to the parameter, this might trigger a runtime error. However, in this case a help field with exactly the same type as the parameter is used instead. This help field is first assigned to the text element and then to the parameter.

Overview of the Class CL_SALV_DISPLAY_SETTINGS

***GET* method**	**Class *CL_SALV_TABLE*** **Method *GET_DISPLAY_SETTINGS*** **returns an object of the type *CL_SALV_DISPLAY_SETTINGS***
Important Methods:	
SET_LIST_HEADER	Set header: This is displayed as the title of the screen and, in the case of full screen, also above the ALV area.
SET_LIST_HEADER_SIZE	Setting the size of a header
SET_STRIPED_PATTERN	Set the striped pattern That is, the lines are alternately colored intensively and not intensively
IS_STRIPED_PATTERN	Check whether the striped pattern is activated
SET_HORIZONTAL_LINES	Activate or deactivate horizontal lines
SET_VERTICAL_LINES	Switching vertical lines on or off
SET_SUPPRESS_EMPTY_DATA	Determine whether an empty data table is displayed (only for the classic list)
GET_SUPPRESS_EMPTY_DATA	Check if an empty data table is displayed

The Class Builder and the SAP Library provide information about all of the methods available. In the Class Builder, all methods are listed on the *Methods* tab page. There is a short text next to every method. You can use the *Class Documentation* function to obtain detailed documentation for all methods. Among other things, this documentation explains all method parameters and lists valid input values.

The method names are also descriptive. You should differentiate between *SET* methods, which change a property, and *GET* or *IS* methods, which query and return a property (or its status).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

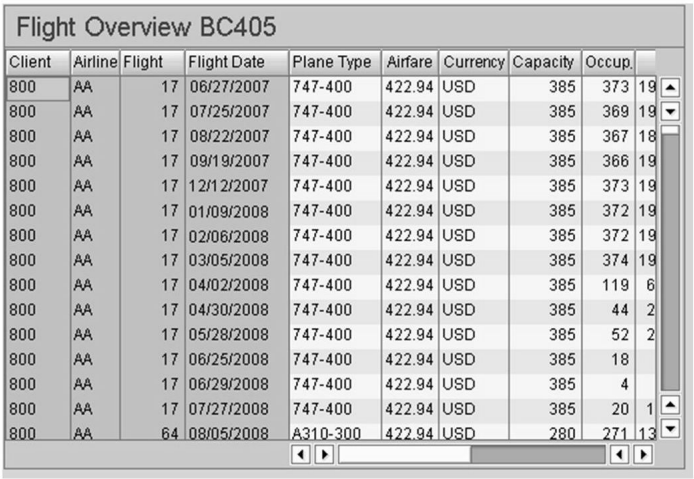
 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Naming Conventions for Methods

Name Begins	Meaning
SET	Define or change property
GET	Query (retrieve) property
IS	

Maximizing Column Width



Zebra stripes

Fixed columns

Figure 196: General Display Properties

Setting General Column Properties

With column properties, you need to differentiate between properties that can be defined for all columns, and properties that apply only to a single column (such as the column header). In this section, we consider the former type of properties.

Overview of the Class CL_SALV_COLUMNS_TABLE

***GET* method**	**Class *CL_SALV_TABLE*** **Method *GET_COLUMNS*** **returns an object of the class *CL_SALV_COLUMNS_TABLE***
Important methods:	
SET_OPTIMIZE	Optimize columns All columns have the optimal width.
SET_COLUMN_POSITION	Set column position

© 2011 SAP AG. All rights reserved.

***GET* method**	**Class *CL_SALV_TABLE*** **Method *GET_COLUMNS*** **returns an object of the class *CL_SALV_COLUMNS_TABLE***
Important methods:	
	A specific column is assigned a specific column position in the ALV.
SET_KEY_FIXATION	Freeze key columns If the user scrolls horizontally, the key columns do not move.

The overview above provides the most important information about using the class *CL_SALV_COLUMNS_TABLE*. There are a number of other important methods in this class, each of which has a special function that requires explanation. These will be introduced later.

The following programming example shows again how an object that represents the properties of the ALV aspect in which you are interested (in this case, the general column properties) is first retrieved, and individual properties are then changed or set using method calls.

Changing General Column Attributes

```
 DATA: go_columns TYPE REF TO cl_salv_columns_table.

* get columns' object
  go_columns = go_alv_table->get_columns( ).
* optimize column widths
  go_columns->set_optimize( ).
* fix key columns
  go_columns->set_key_fixation( ).
* display column PASSNAME next to column CUSTOMID
* (you'll need to count it out)
  go_columns->set_column_position(
      columnname = 'PASSNAME'
      position   = 6 ).
```

Note also that you need to obtain information about the types of the method parameters. In this way, the parameter *position* of the method *SET_COLUMN_POSITION* requires a value of the type “integer”. If you need to specify a column or structure name for a method call, you always need to do so

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

with upper case letters. Note also that you do not need to specify parameters at all for some method calls. This is always the case if there is a default value for an obligatory parameter, or if there are only optional parameters. In a case of this type, you should always check, however, what default value is set.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 25: ALV OM: Setting General Display Properties

Exercise Objectives

After completing this exercise, you will be able to:

- Set the general display properties of the ALV OM

Business Example

You are working as a programmer for ABC Travel. Your manager wants the ALV in your flight display program to have a particular title and to be initially set so that the lines are striped and that the cells are not separated by lines.

Task:

Set the ALV title and determine some further display properties of the ALV.

1. Enhance your *ZBC405_##_OM* program or copy the program *BC405_OM_CREATE_S4*.

   Model Solution: *BC405_OM_DESIGN_S1*

2. The ALV settings you program in this and the following exercises should be usable for all output formats (fullscreen, container, classic list). To achieve this, you should create a subroutine that can then be called both from the `START-OF-SELECTION` block and from the PBO module once the ALV instance has been generated.

   Now create a subroutine (suggested name: *define_settings*). When you call the subroutine, pass the reference to the ALV object, so that it is not necessary to access global data objects in the subroutine. Implement the call after the *FACTORY* method (class *CL_SALV_TABLE*) is called in each case.

3. Now set a number of display properties of the ALV.

   First use the relevant method to retrieve the object that controls the display properties of the ALV. To do this, first define a reference with the correct class type. Then retrieve the object.

   Then assign a title of your choice. Use a program text element to do this.

   Suppress the output of the horizontal separators.

   Create a striped pattern (horizontal).

4. To retain the clarity of your program (with respect to the following exercises), the handling of each aspect of the ALV should be performed in a separate subroutine, each of which is called by the central subroutine *define_settings*.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

To achieve this, now create a subroutine for setting the general display properties of the ALV (suggested name: *set_display*). Package all statements for setting the output that you programmed in the previous exercise step into this subroutine. (Note: You then have to also transfer the reference to the ALV to avoid using global data objects in the subprogram.)

5. Now read the information about the *CL_SALV_COLUMNS_TABLE* class. To do this, call the Class Builder and search the information available there about the available methods. What are the parameters of the *SET_OPTIMIZE* method? What are their types and default values, and are they obligatory or not? Analyze the same questions for the parameters of the method *SET_COLUMN_POSITION*.

6. Now change some general column properties. To do this, create another subroutine (suggested name *set_columns*), to which you pass the reference to the ALV object and which is called from the central subroutine *define_settings*.

   First retrieve the object for the general column properties (class *CL_SALV_COLUMNS_TABLE*). Then set the following:

   - Optimal column width for all columns
   - Freeze the key columns
   - Display the “Aircraft Type” column directly after the key fields

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 25: ALV OM: Setting General Display Properties

Task:

Set the ALV title and determine some further display properties of the ALV.

1. Enhance your *ZBC405_##_OM* program or copy the program *BC405_OM_CREATE_S4*.

   Model Solution: *BC405_OM_DESIGN_S1*

   a) Proceed in the usual manner.

2. The ALV settings you program in this and the following exercises should be usable for all output formats (fullscreen, container, classic list). To achieve this, you should create a subroutine that can then be called both from the `START-OF-SELECTION` block and from the PBO module once the ALV instance has been generated.

   Now create a subroutine (suggested name: *define_settings*). When you call the subroutine, pass the reference to the ALV object, so that it is not necessary to access global data objects in the subroutine. Implement the call after the *FACTORY* method (class *CL_SALV_TABLE*) is called in each case.

   a) See complete solution after the last exercise step

3. Now set a number of display properties of the ALV.

   First use the relevant method to retrieve the object that controls the display properties of the ALV. To do this, first define a reference with the correct class type. Then retrieve the object.

   Then assign a title of your choice. Use a program text element to do this.

   Suppress the output of the horizontal separators.

   Create a striped pattern (horizontal).

   a) See the complete solution after the last task step

4. To retain the clarity of your program (with respect to the following exercises), the handling of each aspect of the ALV should be performed in a separate subroutine, each of which is called by the central subroutine *define_settings*.

   To achieve this, now create a subroutine for setting the general display properties of the ALV (suggested name: *set_display*). Package all statements for setting the output that you programmed in the previous exercise step into this subroutine. (Note: You then have to also transfer the reference to the ALV to avoid using global data objects in the subprogram.)

   a) See the complete solution after the last task step

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

5. Now read the information about the *CL_SALV_COLUMNS_TABLE* class. To do this, call the Class Builder and search the information available there about the available methods. What are the parameters of the *SET_OPTIMIZE* method? What are their types and default values, and are they obligatory or not? Analyze the same questions for the parameters of the method *SET_COLUMN_POSITION*.

   a) Call the Class Builder and select the class *CL_SALV_COLUMNS_TABLE*. Switch to the *Methods* tab page. Place the cursor on the *SET_OPTIMIZE* method. Choose the *Parameters* function. Now read the information about the parameters.

      Now choose the *Class Documentation* function. Also follow the links for the individual methods.

6. Now change some general column properties. To do this, create another subroutine (suggested name *set_columns*), to which you pass the reference to the ALV object and which is called from the central subroutine *define_settings*.

   First retrieve the object for the general column properties (class *CL_SALV_COLUMNS_TABLE*). Then set the following:

   - Optimal column width for all columns
   - Freeze the key columns
   - Display the “Aircraft Type” column directly after the key fields

   a) See the complete solution after the last task step

Result

Main program

```
REPORT bc405_om_create_s1.

INCLUDE bc405_om_design_s1_top.
INCLUDE bc405_om_design_s1_e01.
INCLUDE bc405_om_design_s1_o01.
INCLUDE bc405_om_design_s1_i01.
INCLUDE bc405_om_design_s1_f01.
```

E01 Include

```
START-OF-SELECTION.
* retrieve data into internal table
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
  SELECT * FROM  sflight
    INTO TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

  CASE 'X'.
    WHEN pa_full OR pa_list.

* decide about list display
      IF pa_list IS NOT INITIAL.
        gv_list_display = if_salv_c_bool_sap=>true.
      ELSE.
        gv_list_display = if_salv_c_bool_sap=>false.
      ENDIF.

* create ALV instance
      TRY.
          cl_salv_table=>factory(
           EXPORTING
             list_display   = gv_list_display
            IMPORTING
              r_salv_table   = go_alv_table
            CHANGING
              t_table        = gt_flights ).
        CATCH cx_salv_msg INTO go_alv_error.
          cl_sapbc405_exc_handler=>process_alv_error_msg(
              r_error = go_alv_error ).
      ENDTRY.

* define settings
      PERFORM define_settings USING go_alv_table.
* display ALV
      go_alv_table->display( ).

    WHEN pa_cont.
      CALL SCREEN 100.
  ENDCASE.
```

O01 Include

```
...

MODULE create_control OUTPUT.
  IF go_container IS NOT BOUND.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* create container control and link it to the dynpro
    CREATE OBJECT go_container
      EXPORTING
        container_name = 'MY_CONTAINER'
      EXCEPTIONS
        others         = 1.
    IF sy-subrc <> 0.
      MESSAGE a015(bc405).
    ENDIF.

* create ALV and link it to the container control
    TRY.
        cl_salv_table=>factory(
          EXPORTING
*           LIST_DISPLAY   = IF_SALV_C_BOOL_SAP=>FALSE
            r_container    = go_container
*           CONTAINER_NAME =
          IMPORTING
            r_salv_table   = go_alv_table
          CHANGING
            t_table        = gt_flights ).
      CATCH cx_salv_msg INTO go_alv_error.
    ENDTRY.

* define settings (reuse form)
    PERFORM define_settings USING go_alv_table.

* display ALV
    go_alv_table->display( ).
  ENDIF.
ENDMODULE.    " create_control  OUTPUT
```

F01 Include

```
*&---------------------------------------------------------------------*
*&      Form  define_settings
*&---------------------------------------------------------------------*
FORM define_settings
  USING po_alv_table TYPE REF TO cl_salv_table.

  PERFORM: set_display USING po_alv_table,
           set_columns USING po_alv_table.
ENDFORM.                    " define_settings
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
*&---------------------------------------------------------------------*
*&      Form  set_display
*&---------------------------------------------------------------------*
FORM set_display
  USING po_alv_table  TYPE REF TO cl_salv_table.

  DATA: lo_display TYPE REF TO cl_salv_display_settings,
        lv_title   TYPE        lvc_title.

* get display settings object
  lo_display = po_alv_table->get_display_settings( ).
* set header
  lv_title = text-ttl.
  lo_display->set_list_header( value = text-ttl ).
* set horizontal lines off
  lo_display->set_horizontal_lines( value = ' ' ).
* set striped pattern
  lo_display->set_striped_pattern( value = 'X' ).

ENDFORM.                    " set_display

*&---------------------------------------------------------------------*
*&      Form  set_columns
*&---------------------------------------------------------------------*
FORM set_columns
  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA lo_columns TYPE REF TO cl_salv_columns_table.

* get columns object
  lo_columns = po_alv_table->get_columns( ).

* fix key fields fix
  lo_columns->set_key_fixation(
*    VALUE  = IF_SALV_C_BOOL_SAP~TRUE
         ).

* optimize columns' width
  lo_columns->set_optimize(
*    VALUE  = IF_SALV_C_BOOL_SAP~TRUE
         ).

* bring column PLANETYPE to position 5
  lo_columns->set_column_position(
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
        columnname = 'PLANETYPE'
        position   = 5
           ).
ENDFORM.                    " set_columns
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Set the title of an ALV
- Set other general display properties of the ALV
- Set display properties that apply to all output columns

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV OM: Defining Column Properties

Lesson Overview

In this lesson, you learn to set display properties for an individual column, such as a “Column Title” or “the Column is lead column” indicator. You also learn how to add an additional column to the display and how you can format this column.

Lesson Objectives

After completing this lesson, you will be able to:

- Define properties of individual columns (principle)
- Add an additional column to the ALV and format it (principle)
- Define a column title
- Define or change key columns
- Define or change a DDIC reference

Business Example

You are working as a programmer for ABC Travel. Your manager wants the booking list to display for every booking whether the customer is leaving his or her native country, entering his or her native country, or neither. It is your task to retrieve these values and to display them in the ALV in an appropriate format (that is, with a suitable heading, correct field width, and so on).

A Single Column - A Second Level Subobject of the ALV

We have seen that an object of the class *CL_SALV_COLUMN**S**_TABLE* is used to represent all columns. To make the properties of an individual column available (and therefore make it possible to query and change them), we now retrieve a subobject of that object of the type *CL_SALV_COLUMN_TABLE*.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

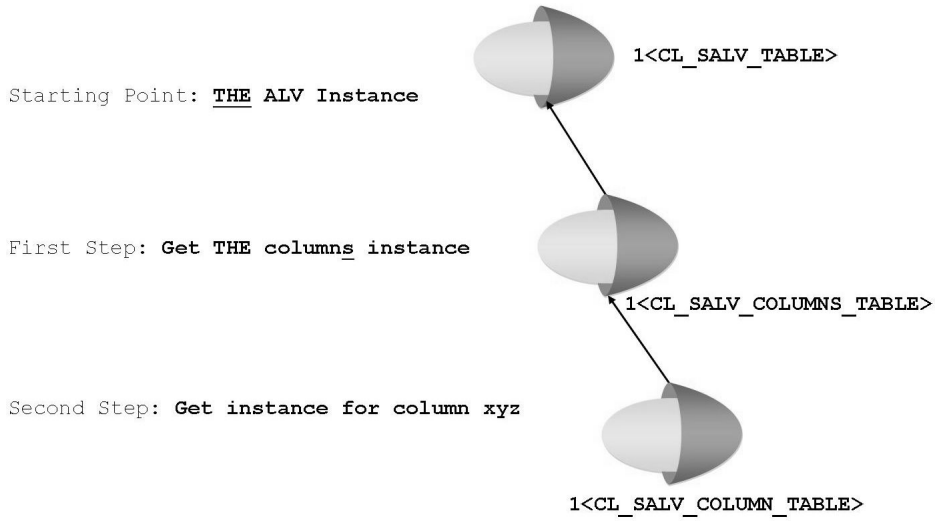


Figure 197: Procedure for Retrieving a Single Column

It is not possible to retrieve a single column if you have not already retrieved the instance for all columns.

To retrieve the instance of a single column, use the *GET* method *GET_COLUMN* of the class *CL_SALV_COLUMNS_TABLE*. This is used as an instance method for the instance "All Columns". You are already familiar with the principle of this procedure from the previously used *GET* methods. The only difference is that in this case, the ALV main object is not the instance used, but rather the "All Columns" object.

Example: Create and change an object for the MANDT column

```
* go_alv_table TPYE REF TO cl_salv_table.
DATA: go_columns_table TYPE REF TO cl_salv_columns_table,
      go_column_table  TYPE REF TO cl_salv_column_table,
      go_column        TYPE REF TO cl_salv_column.

* get columns' object
  go_columns_table = go_alv_table->get_columns( ).
**********************************************
* settings valid for one column                                        *
**********************************************
* get MANDT column object
  CALL METHOD go_columns_table->get_column
    EXPORTING
      columnname = 'MANDT'
    RECEIVING
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
      value       = go_column_table.
  go_column ?= go_column_table.

* column is technical - should never be seen
  go_column->set_technical( ).
```

In the source code example shown here, the object for *CL_SALV_COLUMNS_TABLE* is created first, and then the object for *CL_SALV_COLUMN_TABLE*. You can see that the operator `?=`, the casting operator, is used to do so. You require the casting operator (and therefore casting) if the static type assigned to the assignment target is a subclass (here in the sense of object orientation) of the static type of the source. (Static type means: the type specified in the source code under `TYPE REF TO`. (Dynamic type means: the type the object really has during runtime. This can be the same class specified for the static type, or a subclass of that class.)

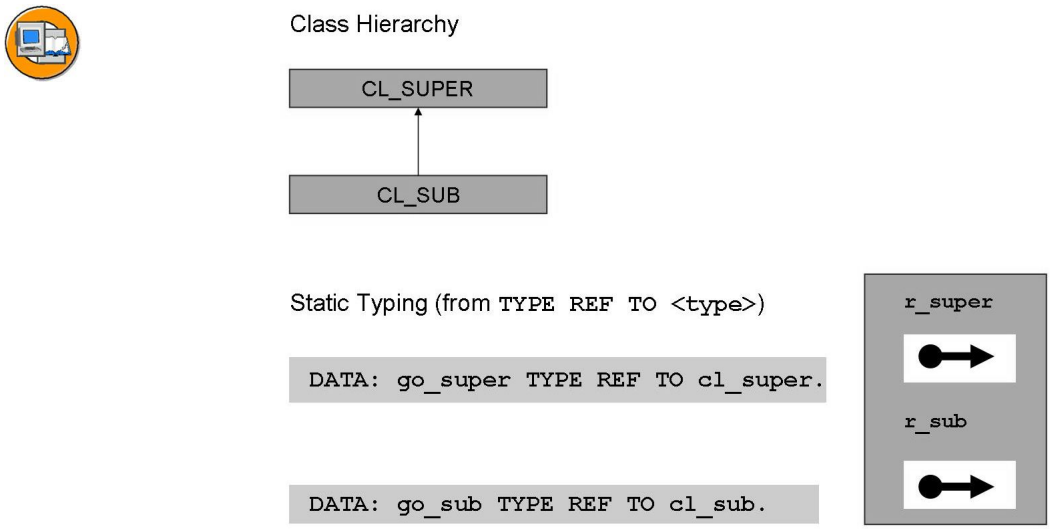


Figure 198: Static Type

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

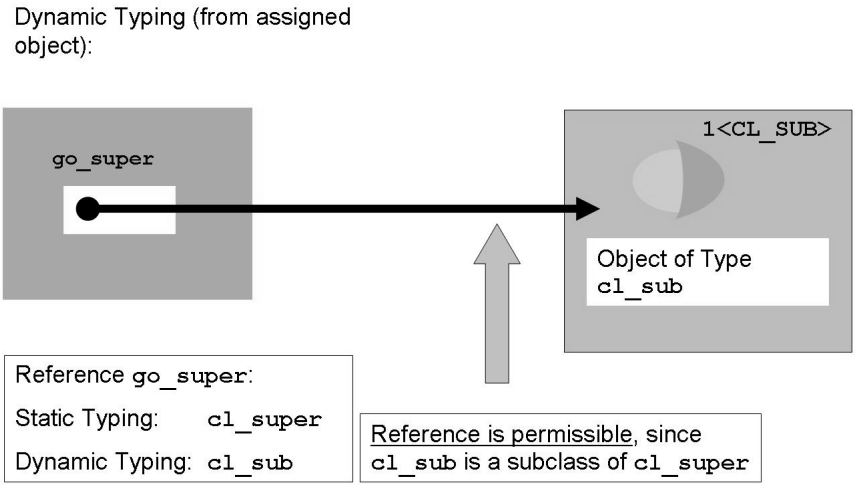


Figure 199: Dynamic Type

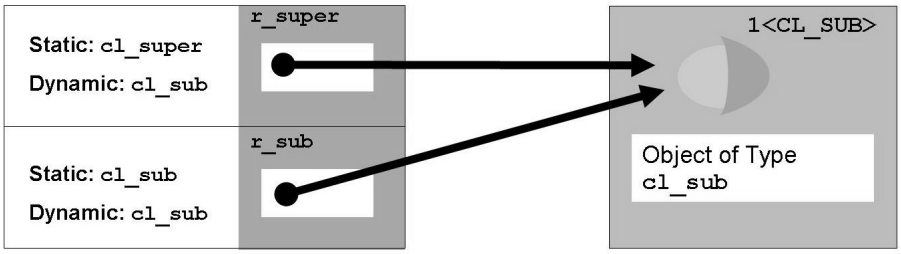
The parameter *value* of the method *GET_COLUMN* is assigned a static type with a reference of the class *CL_SALV_COLUMN*. The reference `go_column_table` used in the source code section shown above is assigned a type in exactly the same way. However, the class that represents the properties of a column of an ALV is called *CL_SALV_COLUMN_TABLE* and is a subclass of *CL_SALV_COLUMN*. However, when the instance is created (during the program execution), it is not an object of the superclass *CL_SALV_COLUMN* that is created, but rather (correctly) a subobject of the class *CL_SALV_COLUMN_TABLE*. The dynamic type assigned to the object that is passed by the parameter *value* is therefore the subclass.

We are now attempting to use the statement `go_column ?= go_column_table.` to assign a reference that is assigned a higher static type to a reference that is assigned a lower static type. This is only successful if the dynamic type of the assigned object is compatible with the static type of the assignment target, that is the same class or a subclass. Since this can only be checked at runtime, it is not possible to rule out runtime errors at this point during the syntax check, and this is signified in the source code by the casting operator `?=`. This is also referred to as a downcast or narrowing cast: from the general to the specific, i.e. the reference to an object becomes more specific (narrowing) or is assigned a type from a lower-level class (downcast).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Downcast (Narrowing Cast): **go_sub ?= go_super.**



The address from **go_super** is copied to **go_sub**. The permissibility of the dynamic typing can only be tested at runtime. This is highlighted by **?=**.

Figure 200: Downcast (Narrowing Cast)

In the example above, the classic writing style with `CALL METHOD` was used for the method call. Since the method returns precisely one `RECEIVING` parameter, you can also call it as a functional method. This has the great advantage that you can leave out the auxiliary variable and can combine the casting with the method call.

CALL METHOD Combined with Type Casting

```
DATA: go_columns_table TYPE REF TO cl_salv_columns_table,
      go_column_table  TYPE REF TO cl_salv_column_table.
*     go_column        TYPE REF TO cl_salv_column.

...
************************************************
* settings valid for one column
************************************************
* get MANDT column object - functional writing
  go_column ?= go_columns_table->get_column( columnname = 'MANDT' ).

* get MANDT column object - classical writing
*  go_columns_table->get_column(
*    EXPORTING
*      columnname = 'MANDT'
*    RECEIVING
*      value      = go_column_table ).
*
* go_column ?= go_column_table.
```


© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
* column is technical - can never be seen
  lo_column->set_technical( ).
```

Overview: Methods of Class CL_SALV_COLUMN_TABLE

The *CL_SALV_COLUMN_TABLE* class is one of the most powerful classes of the ALV OM, since it provides a wide range of methods. There are methods for the following settings:

Set the visibility of the column

visible, hidden, never visible (technical column)

Change the column title

Content and size

Key columns

Define a column as a key column, determine the color of individual cells in the key columns, prevent scrolling of the key columns, prevent the key columns being hidden

Define the properties of the text in the column

Determine the alignment of the text in the column, specify lower- and upper-case, determine the display of empty cells, change the cell color

Determine the appearance

Influence the display of numerical values, arrange a data record in multiple lines

Define the width of the column

Optimize or fix width

Columns with particular meanings

Color column (technical column), counter column to determine the number of lines, cell type column, hyperlink administration column, exception column

In the following lessons, we will learn more about the aspects of column settings listed above, step-by-step. In the next section, we create an additional column for the ALV and format it.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Adding and Formatting an Additional Column in the ALV

As a basic principle, data that is to be displayed by the ALV must be in the internal display table. The internal table is usually first filled with data from the database. However, you often also want to display additional derived data, such as the difference between two numbers from the database.

To do this, you need to extend the definition of the internal table appropriately to then be able to store the additional calculated values.

Extending the Internal Table

```
* Type for row of internal table
TYPES: BEGIN OF gty_booking.
         INCLUDE     TYPE sbook.
TYPES:   country     TYPE scustom-country,
         leaves_home TYPE c LENGTH 1,
         goes_home   TYPE c LENGTH 1,
       END OF gty_booking.
* internal table and work area
DATA: gt_bookings TYPE TABLE OF gty_booking,
      gs_booking  TYPE          gty_booking.
...
```

Extending and Filling the Internal Table

```
...
* retrieve data
  SELECT *
    INTO CORRESPONDING FIELDS OF TABLE gt_bookings
...
* calculate additional data
  LOOP AT gt_bookings INTO gs_booking.
...
* does the customer leave his or her country?
    IF gs_booking-country = gs_spfli-countryfr.
      gs_booking-leaves_home = 'X'.
    ENDIF.

* does the customer go to his or her country?
    IF gs_booking-country = gs_spfli-countryto.
      gs_booking-goes_home = 'X'.
    ENDIF.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
    MODIFY gt_bookings
      FROM gs_booking TRANSPORTING leaves_home goes_home.
  ENDLOOP.
```

The new columns are immediately displayed in the ALV. The formatting at this point depends on the type assigned to the new column in the internal table. If the new column is assigned a type using the Dictionary (for example, with `TYPE sflight-seatsocc`), then the formatting corresponds to this Dictionary object. However, if the new column is assigned a type with predefined ABAP types, there is only minimal formatting (for example, no column heading). You can use method calls to change the formatting in both cases. In one case to change the column heading, for example, in the other case, to create a column heading in the first place, for example.

Defining the Title of a Column

```
  DATA: go_columns   TYPE REF TO cl_salv_columns_table,
        go_column    TYPE REF TO cl_salv_column_table.
  DATA: gv_shorttext TYPE        scrtext_s.
...
* get columns' object
  go_columns = go_alv_table->get_columns( ).
* column LEAVES_HOME
  go_column ?= go_columns->get_column( columnname = 'LEAVES_HOME' ).

* use compatible data object to avoid runtime error
  gv_shorttext = text-lvs. " leaving home
* set short text
  go_column->set_short_text( value = gv_shorttext ).

* column GOES_HOME
  go_column ?= go_columns->get_column( columnname = 'GOES_HOME' ).

  gv_shorttext = text-cms. " coming home
  go_column->set_short_text( value = gv_shorttext ).
```

What stands out in the above example is that the reference variable `go_column` can first be used for the `leaves_home` column and then for the `goes_home` column. The respective column object is retained irrespective of whether the reference variable points to it or not. This means that you do not have to create a new reference variable for each column.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Other methods for defining a column title are listed below. All methods belong to the class *CL_SALV_COLUMN_TABLE*.

Methods for Defining a Column Title (Class CL_SALV_COLUMN_TABLE)

Method	Functions
SET_LONG_TEXT	Sets the long column title
SET_MEDIUM_TEXT	Sets the medium column title
SET_SHORT_TEXT	Sets the short column title

You can store three column titles of different lengths for each column: a short, an intermediate and a long one. The column title that fits best into the column is displayed, depending on how wide the user makes the column. For every *SET* method, there is also a corresponding *GET* method, which can be used to query a value (such as the short column title with *GET_SHORT_TEXT*).

If you have not defined your own column title, it is taken from the data element associated with the column (default behavior).

Defining or Changing an ABAP Reference for a Column

The ABAP Dictionary reference (DDIC reference) for a column is automatically created at the start of the ALV generation based on the type assigned to the corresponding data column in the internal table. It includes the data element used, the data type, the number of decimal places, and the domain. You can query these values with the following methods:

Methods to Evaluate the ABAP Dictionary Reference (Class CL_SALV_COLUMN_TABLE)

Method	Functions
GET_DDIC_REFERENCE	Retrieves the DDIC reference for the column. The method returns either table and column, such as table *SBOOK* and column *BOOKID*, if the ALV column is assigned the type `TYPE sbook-bookid`, or the data element, such as *S_PHONENO*, if the ALV column is assigned the type with `TYPE s_phoneno`.
GET_DDIC_ROLLNAME	Retrieves the name of the DDIC data element
GET_DDIC_DATATYPE	Retrieves the DDIC data type

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Method	Functions
GET_DDIC_DECIMALS	Retrieves the DDIC decimal places
GET_DDIC_DOMAIN	Retrieves the DDIC domain
GET_DDIC_OUT-PUTLEN	Fetch output length acc. to DDIC

If the DDIC reference is now to be changed (for example, to obtain different column headings or different F1 help), there are two methods available:

Methods for Changing the ABAP Dictionary Reference (Class CL_SALV_COLUMN_TABLE)

Method	Functions
SET_DDIC_REFERENCE	Change the DDIC reference for a single column
APPLY_DDIC_STRUCTURE	Change DDIC reference of several columns: assigns the descriptions of identically named fields from an ABAP dictionary structure to all columns

Defining or Changing Key Columns

As far as possible, the key columns are automatically determined at the start of the ALV generated based on the types assigned to the data columns in the internal table.

You can make the following settings for key columns:

Methods for Setting Key Columns (Class CL_SALV_COLUMN_TABLE)

Method	Functions
SET_KEY	Define key columns
SET_KEY_PRESENCE_ REQUIRED	Prevent the hiding of the key columns (It is then not possible for the user to hide the key columns.)

For all of the methods listed above, there are corresponding methods that check a status (such as *GET_KEY*, also class *CL_SALV_COLUMN_TABLE*). These methods are always in the same class as the associated SET method. A complete list is not provided here and elsewhere for reasons of clarity. For more information, see the system documentation.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Defining the Visibility of a Column

The visibility of a column can have three statuses:

- Visible
- Not visible, but can be shown if desired
- Never visible (technical column)

By default, business data columns are set to “visible” and control columns (such as the column containing the color information) are set to “technical”.

If necessary, you can adjust this using the following methods:

Methods for Setting the Visibility of a Column (Class CL_SALV_COLUMN_TABLE)

Method	Functions
SET_VISIBLE	Set the column to visible (with parameter *VALUE*, possible values 'X' or ' ')
SET_TECHNICAL	Define the column as a technical column (with parameter *VALUE*, possible values 'X' or ' ')

Other Methods for Individual Columns (Class CL_SALV_COLUMN_TABLE)

Method	Functions
SET_TOOLTIP	Sets the tooltip for a column
SET_F1_ROLLNAME	Defines the data element from which the F1 help takes its text

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 26: ALV OM: Determining Column Properties

Exercise Objectives

After completing this exercise, you will be able to:

- Define properties of individual columns
- Add an additional column to the ALV
- Define a column title
- Define the visibility of a column

Business Example

You are working as a programmer for ABC Travel. Your manager wants you to display the number of free seats on a flight in an additional column. You are also to temporarily or permanently hide a number of columns.

Task:

Create an additional output column for the ALV and define a number of details for individual columns.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_DESIGN_S1*.

   Model Solution: *BC405_OM_DESIGN_S2*

2. Expand the internal table with the flights: Add a column for the free seats (suggested name: `seatsfree`). To do this, define a local structure type and then assign this type to the internal table.

   Calculate the content of the `seatsfree` column in a loop. Change the contents of the internal table using `MODIFY ... TRANSPORTING ...`.

3. Assign appropriate text for the headings (short, medium, and long) and the tooltip of the column `seatsfree`.

4. Change the properties of the “Client” column so that this column can never be displayed by the user.

5. The “Price” column should not be displayed, but should be available in the column set.

6. The column `seatsfree` should be displayed in position 6 after the “Aircraft Type” column.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 26: ALV OM: Determining Column Properties

Task:

Create an additional output column for the ALV and define a number of details for individual columns.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_DESIGN_S1*.

   Model Solution: *BC405_OM_DESIGN_S2*

   a) Self-explanatory

2. Expand the internal table with the flights: Add a column for the free seats (suggested name: `seatsfree`). To do this, define a local structure type and then assign this type to the internal table.

   Calculate the content of the `seatsfree` column in a loop. Change the contents of the internal table using `MODIFY ... TRANSPORTING ...`.

   a) See complete solution after the last exercise step

3. Assign appropriate text for the headings (short, medium, and long) and the tooltip of the column `seatsfree`.

   a) See the complete solution after the last task step

4. Change the properties of the "Client" column so that this column can never be displayed by the user.

   a) See the complete solution after the last task step

5. The "Price" column should not be displayed, but should be available in the column set.

   a) See the complete solution after the last task step

6. The column `seatsfree` should be displayed in position 6 after the "Aircraft Type" column.

   a) See the complete solution after the last task step.

Result

Main program

```
REPORT bc405_om_create_s2.

INCLUDE bc405_om_design_s2_top.
INCLUDE bc405_om_design_s2_e01.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
INCLUDE bc405_om_design_s2_o01.
INCLUDE bc405_om_design_s2_i01.
INCLUDE bc405_om_design_s2_f01.
```

TOP Include

```
* type for internal table
TYPES: BEGIN OF gty_flight.
         INCLUDE   TYPE sflight.
TYPES:   seatsfree TYPE sflight-seatsocc,
       END OF gty_flight.
* Internal table / workarea
DATA: gt_flights TYPE TABLE OF gty_flight
DATA: gs_flight  TYPE          gty_flight.


...
```

E01 Include

```
START-OF-SELECTION.
* retrieve data into internal table
  SELECT * FROM  sflight
    INTO CORRESPONDING FIELDS OF TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

* fill additional data fields
  LOOP AT gt_flights INTO gs_flight.
    gs_flight-seatsfree =
      gs_flight-seatsmax + gs_flight-seatsmax_b
      + gs_flight-seatsmax_f
      - gs_flight-seatsocc - gs_flight-seatsocc_b
      - gs_flight-seatsocc_f.
    MODIFY gt_flights FROM gs_flight TRANSPORTING seatsfree.
  ENDLOOP.

* adjustments for ALV type
  CASE 'X'.
    WHEN pa_full OR pa_list.

...
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

F01 Include

```
...

*&---------------------------------------------------------------------*
*&      Form  set_columns
*&---------------------------------------------------------------------*
*      -->po_alv_table  text
*----------------------------------------------------------------------*
FORM set_columns
  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA: lo_columns TYPE REF TO cl_salv_columns_table,
        lo_column  TYPE REF TO cl_salv_column_table.
* help fields for title text
  DATA: lv_scrtext_s TYPE scrtext_s,
        lv_scrtext_m TYPE scrtext_m,
        lv_scrtext_l TYPE scrtext_l.
* help field for tooltip
  DATA: lv_lvc_tip TYPE lvc_tip.
* help field for column position
  DATA: lv_pos TYPE i.

************************************************************************
* general columns' settings                                            *
************************************************************************

* get columns object
  lo_columns = po_alv_table->get_columns( ).

* fix key fields
  lo_columns->set_key_fixation(
*    VALUE  = IF_SALV_C_BOOL_SAP~TRUE
         ).

* optimize columns' width
  lo_columns->set_optimize(
*    VALUE  = IF_SALV_C_BOOL_SAP~TRUE
         ).

* bring column PLANETYPE to position 5
  lo_columns->set_column_position(
      columnname = 'PLANETYPE'
      position   = 5
         ).
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* show column SEATSFREE in position 6
  lo_columns->set_column_position(
      columnname = 'SEATSFREE'
      position   = 6
         ).

**********************************************************************
* single columns' settings                                           *
**********************************************************************
* Column SEATSFREE
* 1. get object
  TRY.
      lo_column ?= lo_columns->get_column( columnname = 'SEATSFREE' ).
    CATCH cx_salv_not_found INTO go_alv_error.
      cl_sapbc405_exc_handler=>process_alv_error_msg(
          r_error = go_alv_error
          type    =  'A'
             ).
  ENDTRY.
* 2. change titles (might as well be done via new data element)
* short
  lv_scrtext_s = text-sfr.
  lo_column->set_short_text( value = lv_scrtext_s ).
* medium
  lv_scrtext_m = text-sfm.
  lo_column->set_medium_text( value = lv_scrtext_m ).
* long
  lv_scrtext_l = text-sfl.
  lo_column->set_long_text( value = lv_scrtext_l ).

* 3. set tooltip
  lv_lvc_tip = text-sft.
  lo_column->set_tooltip( value = lv_lvc_tip ).

* column MANDT
* 1. get object
  TRY.
      lo_column ?= lo_columns->get_column( columnname = 'MANDT' ).
    CATCH cx_salv_not_found INTO go_alv_error.
      cl_sapbc405_exc_handler=>process_alv_error_msg(
          r_error = go_alv_error
          type    =  'A'
             ).
  ENDTRY.
* 2. set as technical
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
  lo_column->set_technical(
*    VALUE  = IF_SALV_C_BOOL_SAP=>TRUE ).

* column PRICE
  lo_column ?= lo_columns->get_column( columnname = 'PRICE' ).
  lo_column->set_visible(
      value  = if_salv_c_bool_sap=>false ).

ENDFORM.                    " set_columns
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Define properties of individual columns (principle)
- Add an additional column to the ALV and format it (principle)
- Define a column title
- Define or change key columns
- Define or change a DDIC reference

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV OM: Basic Functions and Layout Variants

Lesson Overview

In this lesson, you learn how you can trigger basic ALV OM functions such as sort, filter, aggregate, and so on from the application. You also learn how you can set selection mode, and how to create and maintain layout variants.

Lesson Objectives

After completing this lesson, you will be able to:

- Trigger sorting, filtering, and aggregation in the ALV from the application
- Activate selection mode and make selections
- Work with layout variants

Business Example

You are working as a programmer for ABC Travel. Your manager wants the data in your booking list to be sorted initially by the “Baggage Weight” field.

Sortings

When setting sortings from the application, the basic principle is:

1. Retrieve the main object for all sortings (in the same way as the main object for all columns)
2. Starting from this main object, either create a new sorting or retrieve and, if necessary change, a single existing sorting (that is, for one column)

A sorting here always means a sort setting for exactly one column. Therefore, if the ALV display is, for example, sorted by three fields, there are three sortings. Another suitable word would be sort criterion.

An essential prerequisite for any sorting setting is to retrieve the main sorting object of the type *CL_SALV_SORTS*. You can retrieve this object using the method call *GET_SORTS* for the main ALV object.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Overview of the Class CL_SALV_SORTS

GET Method	Class *CL_SALV_COLUMNS_TABLE* Method *GET_SORTS*: Returns the object of the type *CL_SALV_SORTS*
Important Methods:	
ADD_SORT	Add a sorting object to a column For this, you can set: the position, the sorting direction, whether subtotals should be issued or not, how a control level change should be indicated and whether the sorting is to be mandatory, that is not changeable by the user.
CLEAR	delete all sort objects
REMOVE_SORT	remove sort object of a column
SET_POSITION	Set the sort order of the columns (weighting of the sort criteria)
SET_COMPRESSED_SUBTOTAL	Hide partial values for subtotals, that is, only the subtotals are visible, but not the associated partial values
SET_GROUP_ACTIVE	Activate grouping Without this activation, group level change handling is not possible.

Create Sorting

```
  DATA: go_sorts TYPE REF TO cl_salv_sorts.

* get SORTS object
  go_sorts = go_alv_table->get_sorts( ).

* enable grouping
  go_sorts->set_group_active(
*    VALUE  = IF_SALV_C_BOOL_SAP=>TRUE
         ).

* add SORT object
  go_sorts->add_sort(
    columnname = 'CARRID'
```

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
    position   = 1
*   SEQUENCE   = IF_SALV_C_SORT=>SORT_UP
*   SUBTOTAL   = IF_SALV_C_BOOL_SAP=>FALSE
*   GROUP      = IF_SALV_C_SORT=>group_with_newpage
*   OBLIGATORY = IF_SALV_C_BOOL_SAP=>FALSE
         ).
```

You can change individual sortings by retrieving the relevant sorting, that is making it available using a representative object. The procedure is similar to retrieving a single column using all columns – in this case, a single sorting is retrieved (an object of the class *CL_SALV_SORT*) using all sorts (that is, using the object described above of the class *CL_SALV_SORTS*).

The following methods exist for the class *CL_SALV_SORT*:

Overview of the Class CL_SALV_SORT

***GET* method**	**Class *CL_SALV_SORTS* Method *GET_SORT*: Returns the object of type *CL_SALV_SORT***
Important methods:	
SET_GROUP	Set the representation of a control level change: no representation, underscore or page break (only visible in the classic ABAP list and on the printout)
SET_OBLIGATORY	Determine whether the sorting can be changed
SET_SEQUENCE	Set sort sequence: ascending or descending
SET_SUBTOTAL	Calculate subtotal

Some properties of a sorting, such as sort order, subtotals, and so on, which are set when a sorting is created with the method *ADD_SORT* (class *CL_SALV_SORTS*) can be changed individually later.

Changing a Sorting

```
  DATA: go_sorts TYPE REF TO cl_salv_sorts,
        go_sort  TYPE REF TO cl_salv_sort.

* get SORTS object
  go_sorts = go_alv_table->get_sorts( ).

* add SORT object
  go_sorts->add_sort(
    columnname = 'FLDATE'
```


© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
    position   = 1
    group      = if_salv_c_sort=>group_with_newpage
         ).

* change sort of FLDATE: Grouping shall be with underline
* 1.get sort object for column FLDATE
  go_sort = go_sorts->get_sort( columnname = 'FLDATE' ).

* 2.set grouping new
  go_sort->set_group(
    value = if_salv_c_sort=>group_with_underline
      ).
```

You have the following options at a group level change:

Types of Display for Group Level Change

Interface Constant	Attribute
IF_SALV_C_SORT=>GROUP_NONE	No display
IF_SALV_C_SORT=>GROUP_WITH_UNDERLINE	Underlined
IF_SALV_C_SORT=>GROUP_WITH_NEWPAGE	Page change

The group level change is only performed if it was first activated in principle by the execution of the method *SET_GROUP_ACTIVE* (class *CL_SALV_SORTS*) and if the classic ABAP list is chosen as the output format.

Filterings

Filterings of the records to be displayed can either be made by the user or can be triggered from the application. In the latter case, the procedure is very similar to that for the sortings.

Defining Filterings from the Application – Procedure

1. Retrieve the object for all filters (class *CL_SALV_FILTERS*) with the method *GET_FILTERS* (class *CL_SALV_TABLE*).
2. Create a filter or retrieve the object for a specific filter (class *CL_SALV_FILTER*) with the methods *ADD_FILTER* or *GET_FILTER* (class *CL_SALV_FILTERS*).
3. If necessary, change properties of a specific filter.

However, there is also another level of objects here, the filter conditions. There can be multiple filter conditions for a particular filter (such as for the “Customer Number” column), for example, “smaller than 20” and “larger than 1000”. A filter condition corresponds in your structure to a line of a selection option, as it is used in a selection screen.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Creating Filters and Filter Conditions

```
  DATA: go_filters TYPE REF TO cl_salv_filters,
        go_filter  TYPE REF TO cl_salv_filter.

* 1. get FILTERS object
  go_filters = go_alv_table->get_filters( ).

* 2. add filter for column LUGGWEIGHT
  go_filters->add_filter(
    columnname = 'CUSTOMID'
    sign       = 'I'
    option     = 'LT'
    low        = '20'
*   HIGH       =
      ).

* 3. get certain FILTER object
  go_filter = go_filters->get_filter(
                columnname = 'CUSTOMID' ).
* 4. add another selection to this filter object
  go_filter->add_selopt(
    sign   = 'I'
    option = 'GT'
    low    = '1000'
*   HIGH   =
      ).
```

Aggregations

Aggregations are combinations of values across all rows of the ALV. This includes totaling, maximum, minimum, and average.

In a very similar way to the sortings, aggregations can be created from the application as well as by the user. To do this, you first need to retrieve **the** aggregation object (class *CL_SALV_AGGREGATIONS*) (with the method *GET_AGGREGATIONS* of the class *CL_SALV_TABLE*). You can then create and change aggregations. In the latter case, you do this by retrieving the individual aggregation and then changing it (for example, by changing the aggregation type from “Total” to “Average”). There can be exactly one aggregation for a column. This means that an aggregation can be identified by the assigned column. The following section of source code shows the important steps.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Creating and Changing an Aggregation

```
  DATA: go_aggregs TYPE REF TO cl_salv_aggregations,
        go_aggreg  TYPE REF TO cl_salv_aggregation.

* 1. get AGGREGATIONS object
  go_aggregs = go_alv_table->get_aggregations( ).

* 2. add an aggregation
  go_aggregs->add_aggregation(
    columnname  = 'LUGGWEIGHT'
    aggregation = if_salv_c_aggregation=>total
         ).
...
* change an aggregation
* 1. get AGGREGATION object
  go_aggreg = go_aggregs->get_aggregation(
                columnname = 'LUGGWEIGHT' ).

* 2. change aggregation type to average
  go_aggreg->set(
    aggregation = if_salv_c_aggregation=>average
         ).
```

Interface constants are used to determine the type of the aggregation. This have descriptive names. You have the following options for aggregation:

Possible Forms of Aggregation and Associated Interface Constants

Instance	Interface constant
No aggregation	*IF_SALV_C_AGGREGATION=>NONE*
Totaling	*IF_SALV_C_AGGREGATION=>TOTAL*
Maximum	*IF_SALV_C_AGGREGATION=>MAXIMUM*
Minimum	*IF_SALV_C_AGGREGATION=>MINIMUM*
Average	*IF_SALV_C_AGGREGATION=>AVERAGE*

Selections

You can use your application to determine whether and in which context the user can select areas of the ALV output. It is also possible to select areas of the ALV output, or to query which areas are selected, from the application.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The basic procedure for working with selections is again:

1. Retrieve the object (of the class *CL_SALV_SELECTIONS*) for the selections with the method *GET_SELECTIONS* (class *CL_SALV_TABLE*).
2. Activate selection mode and work with the selections.

Activating Selection Mode

```
  DATA: go_selections TYPE REF TO cl_salv_selections.

* precondition: go_alv_table is the ALV object
* (TYPE REF TO cl_salv_table)
  go_selections = go_alv_table->get_selections( ).

* multiple: several columns, several rows, no row button
  go_selections->set_selection_mode(
    value  = if_salv_c_selection_mode=>multiple ).
```

The selection modes for the classic ALV list are fundamentally different from those for the other two display formats. All selection modes are presented in the following:

Selection Modes

Display Format	Mode	Description
Full Screen / Container	*MULTIPLE*	Multiple columns and multiple rows
	CELL	Multiple columns, multiple rows, multiple cells, row selection button
	ROW_COLUMN	Multiple columns, multiple rows, row selection button
	NONE	Multiple columns or a single row
Classic ABAP List	*SINGLE*	Multiple columns, several rows, row selection button
	NONE	Multiple columns

The modes listed are in each case specified as identically named constants of the *IF_SALV_C_SELECTION_MODE* interface (for example, `IF_SALV_C_SELECTION_MODE=>ROW_COLUMN`).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The following methods also exist for the class *CL_SALV_SELECTIONS*:

Overview of the Class CL_SALV_SELECTIONS

***GET* method**	**Class *CL_SALV_TABLE* Method *GET_SELECTIONS* Returns an object of the type *CL_SALV_SELECTIONS***
Important methods:	
SET_SELECTION_MODE	Set selection mode
SET_SELECTED_ROWS	Select rows
SET_SELECTED_COLUMNS	Select columns
SET_SELECTED_CELLS	Select cells
SET_CURRENT_CELL	Set current cell

There are also corresponding *GET* methods for all of the *SET* methods.

Layouts

In a layout (also known as a layout variant), the user can store information about column structure, sort criteria, filter conditions, and different display options, and so on. In this way, he or she can repeatedly display the ALV output with the desired properties. Using your application, you can influence the possibilities for the user to use layouts. You can neither generate nor delete layouts using the ALV object model.

Layouts can be available for only one user or for all users of your application. The layout is known as a user-specific layout or as a cross-user layout, depending on which of these is the case. You can determine whether a user can store only user-specific layouts, or whether he or she is also permitted to store cross-user layouts.

Mandatory Naming Conventions for Layouts

Layout Type	**First Character of the Layout Name**
User-specific	A - Z
Cross-user	/
Standard layout (delivered by SAP)	0 - 9

All layouts for all applications are stored centrally. To allow the assignment of a layout to an application, a layout key is defined. All layouts that users store for your application are then identified by this layout key and a name.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The following actions are possible for layouts:

- Retrieve layouts
- Define the layout
- Determine for whom the user is permitted to store a layout
- Influence the layout at the start of the ALV output
- Display a dialog box for the layout

Using Layouts

```
  DATA: go_layout TYPE REF TO cl_salv_layout,
        gs_key    TYPE        salv_s_layout_key.

* precondition: po_alv_table is the ALV object
* (TYPE REF TO cl_salv_table)

* 1. get the LAYOUT object
  go_layout = go_alv_table->get_layout( ).

* 2. set the layout key
  gs_key-report = sy-cprog.
  go_layout->set_key( value = gs_key  ).

* 3. set save restriction:
* none / only user-dependent / only user-independent
  go_layout->set_save_restriction(
    value  = if_salv_c_layout=>restrict_none
      ).

* 4. set: setting of default layout is allowed / isn't allowed
  go_layout->set_default( value = 'X' ).

* 5. set initial layout (best from selection screen)
* (initial layout "wins" against default layout)
  IF pa_layou IS NOT INITIAL.
    go_layout->set_initial_layout( value = pa_layou ).
  ENDIF.
```

The code example can help to explain the procedure for working with layouts: First get the object for the layout. Then you can use the *SET_KEY* method (class *CL_SALV_LAYOUT*) to determine the identifying key for all layouts of this application. To this purpose, you transfer a structure of the type *SALV_S_LAYOUT_KEY* in which at least the `REPORT` field must have been

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

maintained. You can restrict the creation of layouts by the users to certain layout types (user-specific, cross-user). You use the method *SET_SAVE_RESTRICTION* (class *CL_SALV_LAYOUT*) to do this. The `VALUE` parameter can have the following values:

Possible Restrictions when Creating Layouts

Value for `VALUE` (class constant *IF_SALV_C_LAYOUT=>...*)	Meaning
RESTRICT_NONE	No restriction That is, user-specific and cross-user layouts can be created
RESTRICT_USER_DEPENDENT	Only user-specific layouts can be created
RESTRICT_USER_INDEPENDENT	Only cross-user layouts can be created
No value	No authorization to create layouts

When creating layouts, it is possible to declare the layout as the default layout, so that the initial display of the ALV is set in accordance with this layout. (In this case, a user-specific default layout is used in preference to a cross-user default layout.) Authorization to declare a layout as the default is assigned using the method *SET_DEFAULT* (class *CL_SALV_LAYOUT*).

You can also use the method *SET_INITIAL_LAYOUT* (class *CL_SALV_LAYOUT*) to set a start layout dynamically. (This dynamically-set layout is used in any case, even if there is a default layout.) The name of the start layout is transferred using the `VALUE` parameter.

Selecting Layouts in the Selection Screen

```
PARAMETERS: pa_layou TYPE disvariant-variant.
...
AT SELECTION-SCREEN ON VALUE-REQUEST FOR pa_layou.
  PERFORM f4layouts
    CHANGING pa_layou.

FORM f4layouts  CHANGING p_layout TYPE disvariant-variant.
  DATA: ls_layout TYPE salv_s_layout_info,
        ls_key    TYPE salv_s_layout_key.

  ls_key-report = sy-cprog.

  ls_layout = cl_salv_layout_service=>f4_layouts(
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
    s_key    = ls_key
*   LAYOUT   =
*   RESTRICT = IF_SALV_C_LAYOUT=>RESTRICT_NONE
         ).

  pa_layou = ls_layout-layout.

ENDFORM.
```

So that the user can select a start layout in the selection screen, it is best to implement the coding shown above logically. The ABAP event `AT SELECTION-SCREEN ON VALUE-REQUEST FOR <field>` is created for the layout entry field. At the core of the handling of this ABAP event, the class method *CL_SALV_LAYOUT_SERVICES=>F4_LAYOUTS* is called. This method contains the call of a selection dialog box with which the user can choose the desired layout. You can then find the chosen layout in the `LAYOUT` component of the method return parameter of the structure type *SALV_S_LAYOUT_INFO*. The last step is then to set this layout as the start layout using the method *SET_INITIAL_LAYOUT* (class *CL_SALV_LAYOUT*) described above.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Exercise 27: ALV OM: Basic Functions and Layout Variants

Exercise Objectives

After completing this exercise, you will be able to:

- Create a sorting from the application
- Create an aggregation from the application
- Create the selection mode
- Allow the user to create layout variants
- Allow the user to select a start layout on the selection screen

Business Example

You are working as a programmer for ABC Travel. Your manager wants the data in your flight data report to be sorted initially by the *Airline*, *Connection Number*, and *Flight Date* field. The total of the *Free Seats* column is also to be displayed.

Task:

Create an initial sorting and an initial totaling. Activate selection mode. Allow the user to work with layout variants. Implement these as far as possible using packaged subroutines to continue to structure the source code clearly.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_DESIGN_S2*.

   Model Solution: *BC405_OM_DESIGN_S3*

2. The flights are to be initially sorted by the fields *Airline*, *Connection Number*, and *Flight Date*.

3. The total is to be initially displayed for the *Free Seats* field.

   A subtotal is also to be created for each flight connection. (To do this, change the corresponding property of the associated sorting.)

4. Set the selection mode so that the user can select multiple rows, multiple columns, and multiple cells. Test the execution in full screen and in the classic ABAP list.

5. Allow the user to create layouts (cross-user and user-specific). The user should also be able to make a layout the default layout.

   Create two layouts and make one of these the default layout.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

6. **Optional:** Now make it possible for the user to already choose a layout on the selection screen. To achieve this, create a suitable parameter on the selection screen and handle the ABAP event `AT SELECTION-SCREEN ON VALUE-REQUEST FOR <parameter>`. Call the method that executes the selection dialog for layouts in this handling. Then pass the selected layout to the layout object with a suitable method.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 27: ALV OM: Basic Functions and Layout Variants

Task:

Create an initial sorting and an initial totaling. Activate selection mode. Allow the user to work with layout variants. Implement these as far as possible using packaged subroutines to continue to structure the source code clearly.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_DESIGN_S2*.

   Model Solution: *BC405_OM_DESIGN_S3*

   a) Self-explanatory

2. The flights are to be initially sorted by the fields *Airline*, *Connection Number*, and *Flight Date*.

   a) See the complete solution after the last task step.

3. The total is to be initially displayed for the *Free Seats* field.

   A subtotal is also to be created for each flight connection. (To do this, change the corresponding property of the associated sorting.)

   a) See the complete solution after the last task step.

4. Set the selection mode so that the user can select multiple rows, multiple columns, and multiple cells. Test the execution in full screen and in the classic ABAP list.

   a) See the complete solution after the last task step.

5. Allow the user to create layouts (cross-user and user-specific). The user should also be able to make a layout the default layout.

   Create two layouts and make one of these the default layout.

   a) See the complete solution after the last task step.

6. **Optional:** Now make it possible for the user to already choose a layout on the selection screen. To achieve this, create a suitable parameter on the selection screen and handle the ABAP event `AT SELECTION-SCREEN ON VALUE-REQUEST FOR <parameter>`. Call the method that executes the selection dialog for layouts in this handling. Then pass the selected layout to the layout object with a suitable method.

   a) See the complete solution after the last task step.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Result

Main program

```
REPORT bc405_om_create_s3.

INCLUDE bc405_om_design_s3_top.
INCLUDE bc405_om_design_s3_e01.
INCLUDE bc405_om_design_s3_o01.
INCLUDE bc405_om_design_s3_i01.
INCLUDE bc405_om_design_s3_f01.
```

TOP Include

```
...
DATA: gs_layout TYPE salv_s_layout_info.
DATA: gs_key    TYPE salv_s_layout_key.

SELECTION-SCREEN BEGIN OF BLOCK alv WITH FRAME TITLE text-alv.
PARAMETERS: pa_full RADIOBUTTON GROUP alvd,
            pa_cont RADIOBUTTON GROUP alvd DEFAULT 'X',
            pa_list RADIOBUTTON GROUP alvd.
SELECTION-SCREEN END OF BLOCK alv.
PARAMETERS: pa_layou TYPE slis_vari.
```

E01 Include

```
...

    WHEN pa_cont.
      CALL SCREEN 100.
  ENDCASE.

*****************************************************************
AT SELECTION-SCREEN ON VALUE-REQUEST FOR pa_layou.
  gs_key-report = sy-cprog.

  gs_layout =
    cl_salv_layout_service=>f4_layouts(
       s_key    = gs_key
*      restrict = if_salv_c_layout=>restrict_none
         ).

  pa_layou = gs_layout-layout.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

F01 Include

```
...

*&---------------------------------------------------------------------*
*&      Form  define_settings
*&---------------------------------------------------------------------*
FORM define_settings
  USING po_alv_table TYPE REF TO cl_salv_table.

  PERFORM: set_display    USING po_alv_table,
           set_columns    USING po_alv_table,
           set_sorts      USING po_alv_table,
           set_aggregs    USING po_alv_table,
           set_selections USING po_alv_table,
           set_layout     USING po_alv_table.
ENDFORM.                    " define_settings

...

*&---------------------------------------------------------------------*
*&      Form  set_sorts
*&---------------------------------------------------------------------*
FORM set_sorts  USING  po_alv_table TYPE REF TO cl_salv_table.

  DATA lo_sorts TYPE REF TO cl_salv_sorts.

* get the SORTS object
  lo_sorts = po_alv_table->get_sorts( ).

* add sorts
  lo_sorts->add_sort(
      columnname = 'CARRID'
      position   = 1
*     SEQUENCE   = IF_SALV_C_SORT=>SORT_UP
*     SUBTOTAL   = IF_SALV_C_BOOL_SAP=>FALSE
*     GROUP      = IF_SALV_C_SORT=>GROUP_NONE
*     OBLIGATORY = IF_SALV_C_BOOL_SAP=>FALSE
        ).

  lo_sorts->add_sort(
      columnname = 'CONNID'
      position   = 2
*     SEQUENCE   = IF_SALV_C_SORT=>SORT_UP
      subtotal   = if_salv_c_bool_sap=>true
*     GROUP      = IF_SALV_C_SORT=>GROUP_NONE
```

Continued on next page

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
*     OBLIGATORY = IF_SALV_C_BOOL_SAP=>FALSE
         ).

  lo_sorts->add_sort(
      columnname = 'FLDATE'
      position   = 3
*     SEQUENCE   = IF_SALV_C_SORT=>SORT_UP
*     SUBTOTAL   = IF_SALV_C_BOOL_SAP=>FALSE
*     GROUP      = IF_SALV_C_SORT=>GROUP_NONE
*     OBLIGATORY = IF_SALV_C_BOOL_SAP=>FALSE
         ).

* compress subtotal's rows
*  lo_sorts->set_compressed_subtotal(
*      value  = 'CONNID'
*         ).
ENDFORM.                    " set_sorts

*&---------------------------------------------------------------------*
*&      Form  set_aggregs
*&---------------------------------------------------------------------*
FORM set_aggregs  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA lo_aggregs TYPE REF TO cl_salv_aggregations.

  lo_aggregs = po_alv_table->get_aggregations( ).

  lo_aggregs->add_aggregation(
      columnname  = 'SEATSFREE'
      aggregation = if_salv_c_aggregation=>total
         ).

ENDFORM.                    " set_aggregs

*&---------------------------------------------------------------------*
*&      Form  set_selections
*&---------------------------------------------------------------------*
FORM set_selections  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA lo_selections TYPE REF TO cl_salv_selections.

* get the SELECTIONS object
  lo_selections = po_alv_table->get_selections( ).
* set the selection mode
  lo_selections->set_selection_mode(
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
      value  = if_salv_c_selection_mode=>cell
         ).

ENDFORM.                    " set_selections

*&---------------------------------------------------------------------*
*&      Form  set_layout
*&---------------------------------------------------------------------*
FORM set_layout USING po_alv_table TYPE REF TO cl_salv_table.

  DATA: lo_layout TYPE REF TO cl_salv_layout,
        ls_key    TYPE        salv_s_layout_key.

* get the LAYOUT object
  lo_layout = po_alv_table->get_layout( ).
* set the layout key
  ls_key-report = sy-cprog.
  lo_layout->set_key( value = ls_key ).

* set save restriction
* (none restriction is intended, so we may use the default)
  lo_layout->set_save_restriction(
*    VALUE  = IF_SALV_C_LAYOUT=>RESTRICT_NONE
         ).

* allow setting a default layout
  lo_layout->set_default( value = 'X' ).

* set initial layout
  lo_layout->set_initial_layout( value = pa_layou ).

ENDFORM.                    " set_layout
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Trigger sorting, filtering, and aggregation in the ALV from the application
- Activate selection mode and make selections
- Work with layout variants

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV OM: Output Appearance

Lesson Overview

This lesson is about being able to control the appearance of the ALV output. You learn how to set the colors of columns, rows, and cells, how to control text properties (such as right-aligned/left-aligned), and how to control the display of numerical values (for example, the correct formatting of a monetary amount depending on the associated currency key).

The concept of columns with special technical meanings is also introduced.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the concept of columns with special technical meanings
- Define the color of columns, rows, and cells
- Define text properties
- Control the display of numerical values

Business Example

You are working as a programmer for ABC Travel. In your booking report, you are to highlight the row in color for all bookings for which there is a luggage weight greater than 50kg. The “Smoker” column is also to be highlighted for all smokers.

Columns with Special Technical Meanings

Columns from the internal data table are usually directly displayed in the ALV output. However, you can define columns that have a special technical function. Columns are then either not displayed at all, or display their content in a particular way. You can use the following types of special columns:

Control columns

These columns are usually technical columns and are therefore not displayed themselves. They contain information about how the contents of another column are displayed. Examples of control columns:

- Color column (technical column)
- Currency column (visible column)

You can also assign many of the properties that you assign to specific cells using control columns to whole columns (for example, assigning a whole column a currency).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Graphics columns

You use a special writing style to display icons and symbols (such as @AB@). So that these character strings are interpreted as identifiers of graphics and not as text, you correspondingly define the column as an icon or symbol column.

Columns with special functions

These columns contain functions that are only meaningful in very specific contexts. Examples of these columns:

- Text column for captions in result rows for intermediate results
- Counter column for determining the number of rows

Control Columns

You use control columns to assign certain properties to one or more cells. An entry in the control column always refers to the cells of the row that contains the entry.

There are two types of control columns:

- Control columns that have a fixed assignment to a particular column
- Control columns containing information for any number of columns

Control Columns that Are Assigned to a Column

Specifications in the control column refer exclusively to the one assigned column. The cells of this control column can each contain only a single value, which controls the contents of the linked cell.

…	Reference Column	…	Assigned Control Column	…
			Attribute 1 (pink)	
			Attribute 2 (blue)	
			Attribute 3 (green)	

Figure 201: Control Columns that Are Assigned to a Column

This type of control column is used to determine the following properties:

- Currency column
- Column for decimal places
- Column for units of measure
- Rounding column

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Control columns containing information for any number of columns

The cells of this control column contain complete tables:

- Each of these tables refers exclusively to cells in the same row of the ALV output.
- If the table contains multiple rows, each of these rows controls a particular column of the ALV output.
- Each row then contains the name of the column of the ALV output to which the settings refer (in the graphic below, rows 1 and 2).
- If no column name is specified, all columns of the ALV output receive the settings (in the graphic below, row 3).

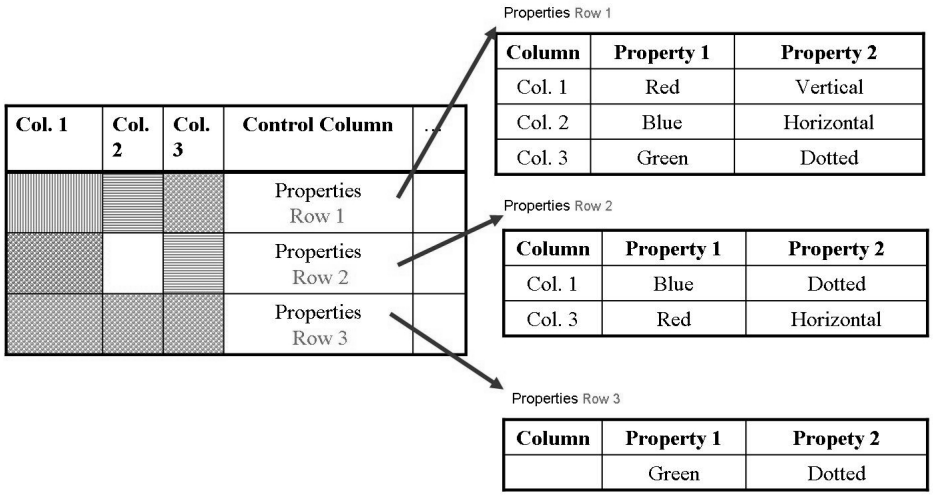


Figure 202: Control columns containing information for any number of columns

This type of control column is used to determine the following properties:

- **Cell types**
  1. Normal text
  2. Button
  3. Checkbox (*changeable* and not *changeable*)
  4. Link
- **Cell colors**
- **Hyperlinks**

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Using Graphics Columns

You can display icons or symbols only in columns that are defined only for this purpose. They can only contain entries that uniquely identify an icon or a symbol in your system. By default, the procedure for creating a graphics column is as follows:

1. Create an additional column in the internal data table (assign a type with *ICON_D* or *SYMB*) and fill it with the desired icon / the desired symbol (technical key, such as `@01@`).

   When filling with the correct technical key, you can simplify your work by using logical names (such as `icon_positive`). For these logical names to be available, the type group *ICON* must be included with the command `TYPE-POOLS: icon.`.

2. Define the additional column as an icon column by calling the method *SET_ICON* of the class *CL_SALV_COLUMN_LIST* (and in the same way for symbols with the method *SET_SYMBOL* of the class *CL_SALV_COLUMN_LIST*).

Caution: You may be wondering about the fact that the class *CL_SALV_COLUMN_LIST* is suddenly being mentioned, although it has not previously been mentioned. This class is a superclass of the familiar class *CL_SALV_COLUMN_TABLE*. Subclasses inherit all components of the superclass, therefore including the *SET_ICON* and *SET_SYMBOL* methods addressed here. However, the superclass has been called because the system documentation for a component is always available at the initial class.

Using Graphics Columns - Filling the Internal Data Table with Additional Data

```
* icons constants
TYPE-POOLS: icon.
* Type for row of internal table
TYPES: BEGIN OF gty_booking.
         INCLUDE      TYPE sbook.
* additional fields
TYPES:   invoice_icon TYPE icon-id,
       END OF gty_booking.

* internal table / work area
DATA: gt_bookings TYPE TABLE OF gty_booking,
      gs_booking  TYPE          gty_booking,
...
LOOP AT gt_bookings INTO gs_booking.
```


© 2011 SAP AG. All rights reserved.

```
* icon: invoice yes / no
  IF gs_booking-invoice = 'X'.
    gs_booking-invoice_icon = icon_positive.
  ELSE.
    gs_booking-invoice_icon = icon_negative.
  ENDIF.
  MODIFY gt_bookings FROM gs_booking
         TRANSPORTING invoice_icon.
ENDLOOP.
```

Using Graphics Columns - Defining an Icon Column

```
*precondition: COLUMN object has been retrieved
  DATA: go_columns TYPE REF TO cl_salv_columns_table,
        go_column  TYPE REF TO cl_salv_column_table.
...
* column INVOICE_ICON
  go_column ?= go_columns->get_column( columnname = 'INVOICE_ICON' ).
* set it as icon column
  go_column->set_icon( ).
```

Exception Column

The exception column is a special form of the graphics column: In it, you use certain icons to represent a status of the current data record.

The most frequent way to represent exception icons is to use traffic light symbols: For example, you can use the icons , and to indicate the release status.

Unlike common graphics columns, the exception column has additional properties:

- The values in the exception column must be between 1 and 5.
- You can only display a certain set of icons. You can choose which icons you want to display from a number of exception groups. Only the values permitted in this exception group are then displayed in the exception column itself. The standard exception group is the group “1” (*LI GHTS*). The other possible exception groups are described in the system documentation of the method *SET_EXCEPTION_COLUMN* of the class *CL_SALV_COLUMNS_LIST*.
- The user cannot hide the exception column.
- The exception column is always displayed as the first column.

The following values apply for the standard exception group “1”:

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Possible Values for Standard Exception Group “1”

Icon	Value	Name
	1	*ICON_RED_LIGHT*
	2	*ICON_YELLOW_LIGHT*
	3	*ICON_GREEN_LIGHT*
	0	*ICON_LIGHT_OUT*

The basic procedure for creating an exception column is as follows:

Procedure for Creating an Exception Column

1. Create an additional column in the internal data table (assign it the type *N*).
2. Fill the additional column with the desired value.
3. Define the additional column as an exception column by calling the method *SET_EXCEPTION_COLUMN* of class *CL_SALV_COLUMNS_LIST*.

Caution: You may be wondering about the fact that the class *CL_SALV_COLUMNS_LIST* is also suddenly being discussed, although it has not previously been mentioned. Exactly as above, this is also a superclass of a class that is already familiar.

Defining Exception Columns

```
* Type for row of internal table
TYPES: BEGIN OF gty_booking.
         INCLUDE     TYPE sbook.
* additional fields
TYPES:   class_light TYPE n,
       END OF gty_booking.
...
* calculate additional data
  LOOP AT gt_bookings INTO gs_booking.
* exception value
    CASE gs_booking-class.
      WHEN 'F'.
        gs_booking-class_light = '1'.
      WHEN 'C'.
        gs_booking-class_light = '2'.
      WHEN 'Y'.
        gs_booking-class_light = '3'.
    ENDCASE.
    MODIFY gt_bookings FROM gs_booking
           TRANSPORTING class_light.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
  ENDLOOP.
...
* set exception column
  go_columns->set_exception_column(
      value     = 'CLASS_LIGHT'
      group     = '1' ).
```

Counter Column to Determine the Number of Rows

You can determine and display the number of data records in the ALV output. To do this, you require a special column, the counter column. The rows of this counter column never display data. However, the result row contains the number of data records.

Procedure for Creating a Counter Column:

1. Create an additional column in the internal data table (assign it the type *TYPE i*).
2. Define the additional column as a counter column by calling the method *SET_COUNT_COLUMN* of class *CL_SALV_COLUMNS_LIST*.

Define the Color of Columns, Rows, and Cells

You can assign different colors to the columns, rows, and cells of your ALV output. You can control the combination of text and background color to a certain degree.

- You select a color from a set of possible colors.
- You define whether the color is to be light or dark (intensity)
- You determine whether the color is to apply for the background or for the text (inverse display)

The following colors are available to you:

No.	Color	INTENSIFIED	INTENSIFIED OFF	INVERSE
0	COL_BACKGROUND	0123456789	0123456789	0123456789
1	COL_HEADING	0123456789	0123456789	0123456789
2	COL_NORMAL	0123456789	0123456789	0123456789
3	COL_TOTAL	0123456789	0123456789	0123456789
4	COL_KEY	0123456789	0123456789	0123456789
5	COL_POSITIVE	0123456789	0123456789	0123456789
6	COL_NEGATIVE	0123456789	0123456789	0123456789
7	COL_GROUP	0123456789	0123456789	0123456789

Figure 203: Possible Colors for the ALV Output

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

You can make the following color settings:

- Key columns have a color assigned by the system. In key columns, you can only change the color in individual cells. However, you cannot simply change the color for the entire key column from the start.
- You can color rows or cells using the color column. You can color entire columns using the color property of the column.

Changing the Column Color

The color is a property of the column. This means that you can change it directly. To do this use, the method *SET_COLOR* of the class *CL_SALV_COLUMN_LIST*.

To be able to use the logical color names in the program (such as `col_group`), you need to include the type group *COL* with the command `TYPE-POOLS: col`.

Changing the Column Color

```
TYPE-POOLS: col.
  DATA: gs_lvc_s_colo TYPE        lvc_s_colo.
  DATA: go_columns    TYPE REF TO cl_salv_columns_table,
        go_column     TYPE REF TO cl_salv_column_table.

* Precondition: COLUMN object has been retrieved!
* column CUSTOMID
  go_column ?= go_columns->get_column( columnname = 'CUSTOMID' ).

* set column's color
  gs_lvc_s_colo-col = col_group.
  gs_lvc_s_colo-int = 0.
  gs_lvc_s_colo-inv = 1.
  go_column->set_color( value = gs_lvc_s_colo ).
```

Changing the Colors of Cells

To highlight individual cells in color, you need to extend the row type of the output table (data table) with an internal table. The contents of this internal table are not displayed, but are used only for the layout. For each data record of the data table, you list the fields that are to be highlighted in color and the color code in its internal table. Every field to be highlighted requires a separate entry in this internal table.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

- **Adding an internal field color table to the data table**
- **This internal table contains an entry for every highlighted field of the data row.**

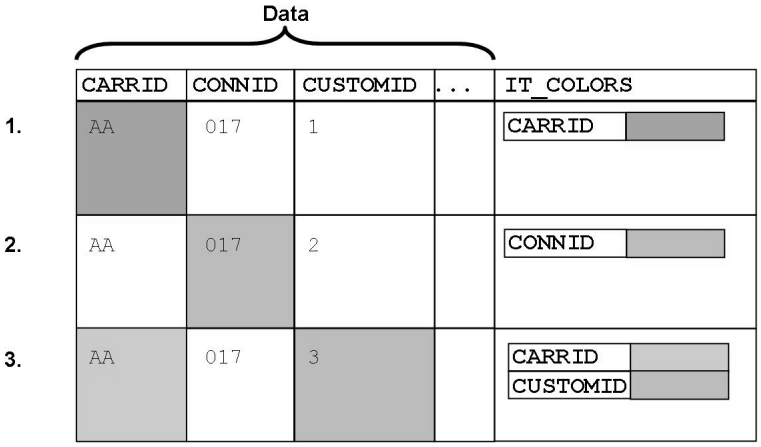


Figure 204: Changing the Cell Color (1)

The basic procedure to change the color of an individual cell:

- Extend the row structure of the data table with an internal field color table of type *lvc_t_scol*.
- Within a loop, use the database table to define for every record which cells are to be highlighted with which colors.
- Specify the name of the field color table with the method *SET_COLOR_COLUMN* of the class *CL_SALV_COLUMNS_LIST*. (The method is performed on the object that represents all columns.)

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

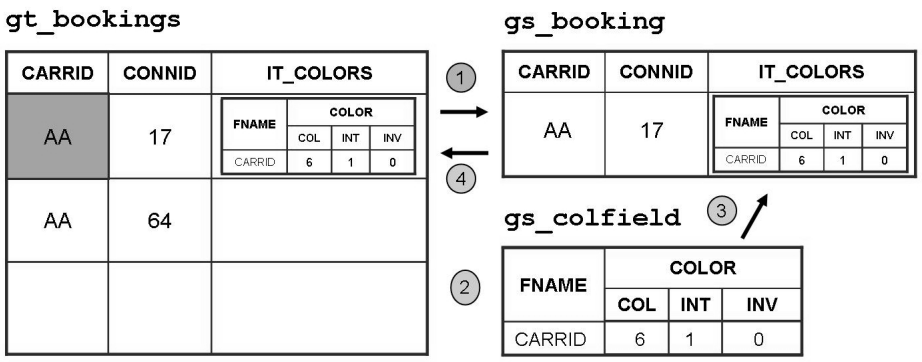


Figure 205: Changing the Cell Color (2)

1. Read the internal data table *gt_bookings* into the associated work area *gs_booking* record-by-record (with `LOOP`). When you do so, you can color different cells for each individual data record.
2. Fill the help field *gs_colfield*. It contains two fields:
   - *fname*: the name of the field to be colored.
   - *color*: the color code to be used, compiled from the three subfields *col*, *int* and *inv*.
3. Appending the *gs_colfield* help field to the internal table *gs_booking-it_colfields* (which contains information about the cell color combinations.)
4. Write the work area *gs_booking* back to the internal data table using `MODIFY`

You need to perform steps 2 and 3 for each field that is to be colored.

Changing Cell Colors - Data Objects

```
* color constants
TYPE-POOLS: col.

* Type for row of internal table
TYPES: BEGIN OF gty_booking.
         INCLUDE   TYPE sbook.
* color information
TYPES:   it_colors TYPE lvc_t_scol,
       END OF gty_booking.

* internal table / work area
DATA: gt_bookings TYPE TABLE OF gty_booking,
      gs_booking  TYPE          gty_booking.

* work area for internal table for colors
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
DATA: gs_color TYPE lvc_s_scol.
```

Changing Cell Colors – Executable Statements

```
  LOOP AT gt_bookings INTO gs_booking.
*   field SMOKER shall be COL_GROUP if checked
    IF gs_booking-smoker = 'X'.
      CLEAR wa_colors.
      gs_color-fname     = 'SMOKER'.
      gs_color-color-col = col_group.
      gs_color-color-int = 1.
      APPEND gs_color TO gs_booking-it_colors.
    ENDIF.
    MODIFY gt_bookings FROM gs_booking
           TRANSPORTING it_colors.
  ENDLOOP.
...
* set color column
* preconditions:
* 1. go_columns is TYPE REF TO cl_salv_columns_table
* 2. COLUMNS object has been retrieved
  DATA: go_columns TYPE REF TO cl_salv_columns_table.
  go_columns->set_color_column( value = 'IT_COLORS' ).
```

Changing the Colors of Rows

The procedure for changing the color of a whole row is, in principle, the same as for changing the color of a cell. The color information is passed to the internal data table using the additional table for the row. The only difference: No value is specified for the *fname* field, so that the setting applies to the whole row. Regardless of this, however, you can still assign different color values to individual cells in this row.

Changing the Row Color – Executable Statements

```
  LOOP AT gt_bookings INTO gs_booking.
* Color information
* whole row shall be COL_NEGATIVE if booking has been canceled
    IF gs_booking-cancelled = 'X'.
      CLEAR gs_color.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
* no fname is set -> whole row is affected
      gs_color-color-col = col_negative.
      gs_color-color-int = 1.
      APPEND gs_color TO gs_booking-it_colors.
    ENDIF.
    MODIFY gt_bookings FROM gs_booking
           TRANSPORTING it_colors.
  ENDLOOP.
...
* setting of color column identical with cell example
...
```

Defining Cell Types

You can display various elements instead of text in the content of your ALV output. You can handle some of these elements as a click area for the user. You can display the following elements:

Checkbox

Differentiate here between activated and deactivated checkboxes. Activated checkboxes trigger an event when clicked.

Button

You can handle the user actions for this element with an event.

Hyperlink

These specify Internet addresses (URLs). The user can call the corresponding page directly in his or her Internet browser by clicking an Internet address of this type.

Click Area

You can handle the actions for this element with an event.

The setting for an interactive element is known as a cell type.

There are two options for assigning cell types:

- You can assign a cell type to an entire column using the method *SET_CELL_TYPE* of the class *CL_SALV_COLUMN_LIST*.
- You can assign a separate cell type to each individual cell.

Cell Types (Technical Names from Interface IF_SALV_C_CELL_TYPE)

Interface Constant	Description
TEXT	Text (default)
CHECKBOX	Checkbox

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Interface Constant	Description
BUTTON	Pushbutton
DROPDOWN	Dropdown
LINK	Link
HOTSPOT	Hotspot
CHECKBOX_HOTSPOT	Checkbox with hotspot

Assigning a Cell Type to an Entire Column

```
  DATA: go_columns TYPE REF TO cl_salv_columns_table,
        go_column  TYPE REF TO cl_salv_column_table.

* !!! precondition: the COLUMNS object has been retrieved
* column TELEPHONE
  go_column ?= go_columns->get_column( columnname = 'TELEPHONE' ).
  go_column->set_cell_type( value  = if_salv_c_cell_type=>hotspot ).
```

Assigning cell types by cell is done in the same way as assigning color information at cell level.

- **Extend the data table with an internal cell type table**
- **This internal table contains an entry for every field of the data row that has a particular cell type.**

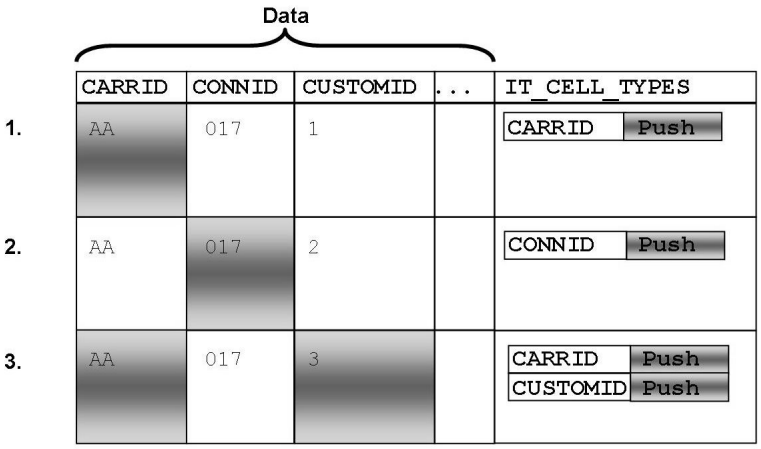


Figure 206: Changing the Cell Type (1)

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

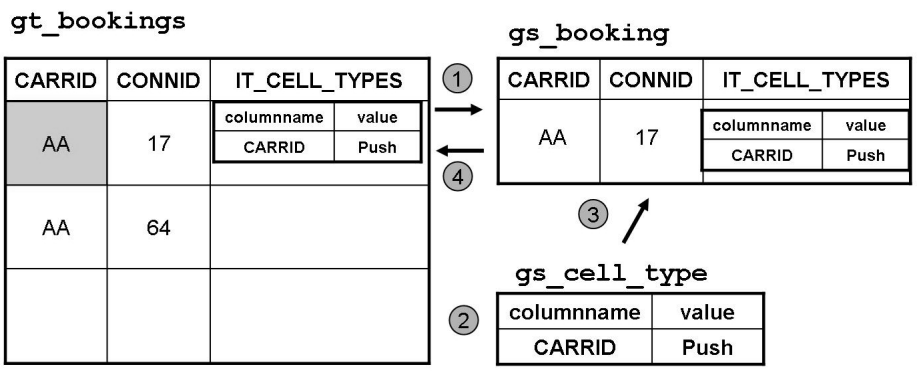


Figure 207: Changing the Cell Type (2)

Assigning a Cell Type to an Individual Cell (1)

```
TYPES: BEGIN OF gty_booking.
         INCLUDE       TYPE sbook.
* additional fields: cell type info
TYPES:   it_cell_types TYPE salv_t_int4_column,
       END OF gty_booking.

* internal table / work area
DATA: gt_bookings TYPE TABLE OF gty_booking,
      gs_booking  TYPE          gty_booking.
* work area for internal table for cell type info
DATA: gs_cell_type LIKE LINE OF gs_booking-it_cell_types.
```

Assigning a Cell type to an Individual Cell (2)

```
* calculate additional data
  LOOP AT gt_bookings INTO gs_booking.
**********************************************************************
* Cell type informations
**********************************************************************
* luggage weight gt 20 kg ? if yes, show as button
* (only useful in container)
    IF gs_booking-luggweight > 20 AND
       gs_booking-wunit = 'KG' AND
       pa_cont = 'X'.
      CLEAR gs_cell_type.
      gs_cell_type-columnname = 'LUGGWEIGHT'.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
      gs_cell_type-value      = if_salv_c_cell_type=>button.
      APPEND gs_cell_type TO gs_booking-it_cell_types.
    ENDIF.
    MODIFY gt_bookings FROM gs_booking TRANSPORTING it_cell_types.
  ENDLOOP.

*************************************************
* setting columns with special purposes
*************************************************
  DATA: go_columns TYPE REF TO cl_salv_columns_table,
* !!! Precondition: the COLUMNS object has been retrieved
* set cell type column
  go_columns->set_cell_type_column( value = 'IT_CELL_TYPES' ).
```

Displaying a Checkbox

A checkbox can be set in two ways: Either the indicator is set (the box is checked) or it is not set (the box is empty). Activation may overwrite both statuses: If a checkbox is activated (the field is white), the user can change its status by clicking it. If the checkbox is deactivated (the field is displayed in gray), a click does not change the status.

You can display checkboxes in all ALV tools. However, you cannot activate checkboxes in the classic ABAP list: the user cannot change the value of a checkbox in this case.

You can activate checkboxes, that is, indirectly permit the user to change the value of the checkbox. In the case of activated checkboxes, you can handle the event *LINK_CLICK* of the class *CL_SALV_EVENT_TABLE* using your application, and change the value of the clicked field in it.

The cells in which you want to display a checkbox need to have an alphanumeric data type. You control the display of the checkbox using the combination of cell type and the value of the cell:

- The cell type *IF_SALV_C_CELL_TYPE=>CHECKBOX* always displays a deactivated checkbox, the cell type *IF_SALV_C_CELL_TYPE=>CHECK-BOX_HOTSPOT* always displays an activated checkbox.
- The values “1” or “X” mean the checkbox indicator is set, while the values “0” or “ ” mean it is not set.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Checkbox	Description	Cell Type (IF_SALV_C_CELL_TYPE)	Value
☐	Checkbox activated Indicator not set	CHECKBOX_HOTSPOT	'0' or ' '
☑	Checkbox activated Indicator set	CHECKBOX_HOTSPOT	'1' or 'X'
☐	Checkbox not activated Indicator not set	CHECKBOX	'0' or ' '
☑	Checkbox not activated Indicator set	CHECKBOX	'1' or 'X'

Figure 208: Values of a Checkbox

Displaying a Pushbutton

You can display buttons in the ALV output. The content of the cell is displayed as the button's label. With buttons, the user triggers the event *LINK_CLICK* of the class *CL_SALV_EVENTS_TABLE*, which you can handle if necessary.

To display a cell as a button, use the cell type *BUTTON*.

Buttons are not permitted in the classic ABAP list.

Displaying a Click Area

You can define the content of a cell as a click area. If the user clicks this content, he or she triggers the event *LINK_CLICK* of the class *CL_SALV_EVENTS_TABLE*, which you can handle if necessary.

To define a cell as a click area, use the cell type *HOTSPOT*.

Defining Text Properties

You can control the way text is displayed in the ALV output. You can make the following settings for text properties:

- Determine the alignment of the text in a column
- Specify lower-case and upper-case text
- Determine the display of empty cells
- Modify the content using conversion routines

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Determining the Alignment of the Text in a Column

You can determine how the contents of a column are aligned:

- Left-aligned
- Centered
- Right-aligned

To do this use, the method *SET_ALIGNMENT* of the class *CL_SALV_COLUMN*. (Since precisely one column is affected by the change, the method needs to be performed on the object that represents this column.)

Determining the Alignment of Text in a Column

```
* precondition: COLUMN object has been determined
  DATA: go_columns TYPE REF TO cl_salv_columns_table,
        go_column  TYPE REF TO cl_salv_column_table.
...
* column PASS_NAME
  go_column ?= go_columns->get_column( columnname = 'PASS_NAME' ).

* set alignment
  go_column->set_alignment( value  = if_salv_c_alignment=>right ).
```

Overview: Determining Text Properties

All methods for changing text properties are listed below:

Overview: Determining Text Properties (Class CL_SALV_COLUMN)

Method	Function
SET_ALIGNMENT	Determine the alignment of text in a column
SET_LOWERCASE	Preset upper and lower case
SET_ZERO	Set representation for empty cells
SET_EDIT_MASK	Modify the content using conversion routines

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Influencing How Numeric Values are Displayed

You can make the following settings for numeric columns:

- Define properties such as the number of decimal places, unit of measure, currency, and rounding
- Show or hide the plus/minus sign
- Show or hide leading zeros

Defining Decimal Places, Unit of Measure, Currency, and Rounding

You define the number of decimal places and the unit of measure, currency, and rounding for each column. With each of these properties, you either move the decimal point by the desired number of places (and therefore change the value of the number), or you change the number of decimal places after the decimal point (the value of the number then remains unchanged). You can also combine two or more options.

You can use each of the properties in two ways:

- You assign the desired property with a specific value to the column that contains the value to be displayed. Each of the values in the column is changed and displayed in the same way.
- You use a separate column that applies the desired property to individual rows of the ALV output.

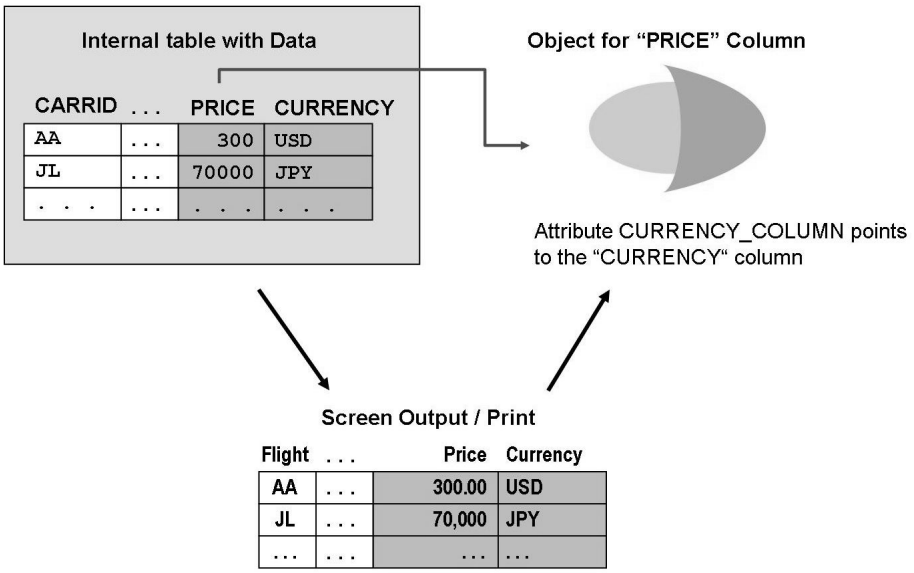


Figure 209: Displaying Output Appropriately for a Currency with a Control Column

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Note that this assignment is only required if the internal data table contains a column that has not been assigned a type using the Dictionary – for example, a column for currency amounts that is only assigned the type *P* (packed number) – or if the Dictionary typing is to be overridden.

Displaying Output Appropriately for a Currency with an Explicit Value Assignment

```
* precondition: COLUMN object has been retrieved
  DATA: go_columns TYPE REF TO cl_salv_columns_table,
        go_column  TYPE REF TO cl_salv_column_table.
...
* column FORCURAM
  go_column ?= go_columns->get_column( columnname = 'FORCURAM' ).
  go_column->set_currency( value = 'JPY'  ).
```

In the above code example, every currency amount in the column *FORCURAM* would be formatted in accordance with the formatting specifications for the currency *JPY*.

Displaying Output Appropriately for a Currency with a Control Column

```
* column FORCURAM
  go_column ?= go_columns->get_column( columnname = 'FORCURAM' ).
  go_column->set_currency_column( value = 'FORCURKEY' ).
```

In this code example, every currency amount in the column *FORCURAM* would be formatted in accordance with the formatting specifications for the currency in the column *FORCURKEY* in the same row. This variant is more flexible than the first variant.

Controlling the Display of Numerical Values

All methods for controlling the display of numerical values are listed below:

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Controlling the Display of Numerical Values (Class *CL_SALV_COLUMN*)

Method	Description
SET_DECIMALS	Define the number of decimal places (statically)
SET_DECIMALS_COLUMN	Define the number of decimal places (using a control column)
SET_QUANTITY	Define the unit of measure (statically)
SET_QUANTITY_COLUMN	Define the unit of measure (using a control column)
SET_CURRENCY	Define the currency (statically)
SET_CURRENCY_COLUMN	Define the currency (using a control column)
SET_ROUND	Define the rounding (statically)
SET_ROUND_COLUMN	Define the rounding (using a control column)
SET_SIGN	Showing and Hiding +/- Signs
SET_LEADING_ZERO	Showing and hiding leading zeroes

Determining the Column Width

You can make the following settings for the column width:

- Define the display width of a column
- Adjust the width to the space required by the values

This is done using the following methods:

Defining the Column Width (Class CL_SALV_COLUMN)

Method	Description
SET_OUTPUT_LENGTH	Define the column width
SET_OPTIMIZED	Optimize an individual column

Arranging a Data Record into Multiple Rows

With the **output format "classic ABAP list"**, you can arrange the values of a data record into multiple rows. In this way you can ensure, for example, that all data is visible at once, even if there is a very large number of columns.

You can achieve this by using the method *SET_ROW* of the class *CL_SALV_COLUMN*.

This setting has no effect for the other output formats.

© 2011 SAP AG. All rights reserved.

Arranging a Data Record in Multiple Rows

```
  DATA: go_columns TYPE REF TO cl_salv_columns_table,
        go_column  TYPE REF TO cl_salv_column_table.
...
* precondition: COLUMN object has been retrieved
* column LUGGWEIGHT
  go_column ?= go_columns->get_column( columnname = 'LUGGWEIGHT' ).
  go_column->set_row( value = 2 ).
* column WUNIT
  go_column ?= go_columns->get_column( columnname = 'WUNIT' ).
  go_column->set_row( value = 2 ).
* etc.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 28: ALV OM: Output Appearance

Exercise Objectives

After completing this exercise, you will be able to:

- Define the color of columns, rows, and cells
- Define an icon column
- Define an exception column
- Change the text properties of a column

Business Example

You are working as a programmer for ABC Travel. Your manager wants your ALV program to display the number of records read (in total and, if appropriate, for each group level). He or she also wants to use icons to be able to see at a glance how full each flight is and whether the flight is in the past or not. Finally, it is to be possible to highlight the "Aircraft Type" cell if desired, that is when it has particular content.

Task:

Create a counter column, an icon column, and an exception column for your ALV program. Highlight individual columns, rows, and cells in color.

1. Extend your program *ZBC405_##_OM*, or copy the program *BC405_OM_DESIGN_S3*.

   Model Solution: *BC405_OM_DESIGN_S4*

2. Create an exception column with which it is possible to see how full the flight is at a glance. Use the standard exception group *LIGHTS* with the traffic light icons.

   Create an additional column of the right type for the internal data table (suggested name: *usage*). After the data has been retrieved from the database, fill this column in accordance with the following conditions:

   If the number of free seats = 0, then a red light

   If the number of free seats <= 20, then a yellow light

   Otherwise, a green light

   Finally, use a suitable method to declare the created and filled column as the exception column of the ALV.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

3. Create an icon column with which you can see at a glance whether or not the flight is in the future. To do this, first create another column in the internal data table and fill it in each case with a suitable value for an icon (select two icons to display the circumstances as you see fit). Finally, use a suitable method to assign this additional column the property "icon column".
4. Provide information in the ALV about the number of rows (in total and, if appropriate, subtotals).
5. Highlight the **column** *Free Seats* in the color orange (*COL_GROUP*).
6. Highlight all **rows** in which more than 200 seats are free in the color blue (*COL_HEADING*).
7. **Optional:** Highlight all **cells** in the *Plane type* column which contain a plane type of your choice (for example, "747-400") green (*COL_POSITIVE*).

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 28: ALV OM: Output Appearance

Task:

Create a counter column, an icon column, and an exception column for your ALV program. Highlight individual columns, rows, and cells in color.

1. Extend your program *ZBC405_##_OM*, or copy the program *BC405_OM_DESIGN_S3*.

   Model Solution: *BC405_OM_DESIGN_S4*

   a) Self-explanatory

2. Create an exception column with which it is possible to see how full the flight is at a glance. Use the standard exception group *LIGHTS* with the traffic light icons.

   Create an additional column of the right type for the internal data table (suggested name: *usage*). After the data has been retrieved from the database, fill this column in accordance with the following conditions:

   If the number of free seats = 0, then a red light

   If the number of free seats <= 20, then a yellow light

   Otherwise, a green light

   Finally, use a suitable method to declare the created and filled column as the exception column of the ALV.

   a) See the complete solution after the last task step.

3. Create an icon column with which you can see at a glance whether or not the flight is in the future. To do this, first create another column in the internal data table and fill it in each case with a suitable value for an icon (select two icons to display the circumstances as you see fit). Finally, use a suitable method to assign this additional column the property “icon column”.

   a) See the complete solution after the last task step.

4. Provide information in the ALV about the number of rows (in total and, if appropriate, subtotals).

   a) See the complete solution after the last task step.

5. Highlight the **column** *Free Seats* in the color orange (*COL_GROUP*).

   a) See the complete solution after the last task step.

6. Highlight all **rows** in which more than 200 seats are free in the color blue (*COL_HEADING*).

   a) See the complete solution after the last task step.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

7. **Optional:** Highlight all **cells** in the *Plane type* column which contain a plane type of your choice (for example, “747-400”) green (*COL_POSITIVE*).

   a) See the complete solution after the last task step.

Result

Main program

```
REPORT bc405_om_create_s4.

INCLUDE bc405_om_design_s4_top.
INCLUDE bc405_om_design_s4_e01.
INCLUDE bc405_om_design_s4_o01.
INCLUDE bc405_om_design_s4_i01.
INCLUDE bc405_om_design_s4_f01.
```

TOP Include

```
* constants for icons / colors
TYPE-POOLS: icon, col.

* type for internal table
TYPES: BEGIN OF gty_flight.
         INCLUDE      TYPE sflight.
TYPES:   seatsfree    TYPE sflight-seatsocc,
         usage        TYPE n,              "exception
         icon_future  TYPE icon-id,        "icon
         line_counter TYPE i,              "line counter
         it_colors    TYPE lvc_t_scol,     "cell color information
       END OF gty_flight.

* Workarea for cell color information
DATA: gs_color TYPE lvc_s_tcol.

...
```

E01 Include

```
*&---------------------------------------------------------------------*
*&   Event START-OF-SELECTION
*&---------------------------------------------------------------------*
START-OF-SELECTION.
* retrieve data into internal table
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
  SELECT * FROM  sflight
    INTO CORRESPONDING FIELDS OF TABLE gt_flights
    WHERE carrid IN so_car
    AND   connid IN so_con.

* fill additional data fields
  LOOP AT gt_flights INTO gs_flight.

* seats free
    gs_flight-seatsfree =
      gs_flight-seatsmax + gs_flight-seatsmax_b
      + gs_flight-seatsmax_f
      - gs_flight-seatsocc - gs_flight-seatsocc_b
      - gs_flight-seatsocc_f.

* usage (exception light)
    IF gs_flight-seatsfree = 0.
      gs_flight-usage = '1'.
    ELSEIF gs_flight-seatsfree <= 20.
      gs_flight-usage = '2'.
    ELSE.
      gs_flight-usage = '3'.
    ENDIF.

* icon "in the future/not in the future"
    IF gs_flight-fldate > sy-datum.
      gs_flight-icon_future = icon_positive.
    ELSE.
      gs_flight-icon_future = icon_negative.
    ENDIF.

* cell colors
* single cell if planetype is 747-400
    IF gs_flight-planetype = '747-400'.
      CLEAR gs_color.
      gs_color-fname     = 'PLANETYPE'.
      gs_color-color-col = col_positive.
      gs_color-color-int = 1.
      APPEND gs_color TO gs_flight-it_colors.
    ENDIF.

* whole line if seatsfree gt 200
    IF gs_flight-seatsfree >= 200.
      CLEAR gs_color.
      gs_color-color-col = col_heading.
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
      gs_color-color-int = 1.
      APPEND gs_color TO gs_flight-it_colors.
    ENDIF.

    MODIFY gt_flights FROM gs_flight
      TRANSPORTING
        seatsfree
        usage
        icon_future
        it_colors.
  ENDLOOP.

...
```

F01 Include

```
FORM set_columns
  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA: lo_columns TYPE REF TO cl_salv_columns_table,
        lo_column  TYPE REF TO cl_salv_column_table.
* help fields for title text
  DATA: lv_scrtext_s TYPE scrtext_s,
        lv_scrtext_m TYPE scrtext_m,
        lv_scrtext_l TYPE scrtext_l.
* help field for tooltip
  DATA: lv_lvc_tip TYPE lvc_tip.
* help field for column position
  DATA: lv_pos TYPE i.
* help field for column color
  DATA: ls_lvc_s_colo TYPE lvc_s_colo.

************************************************************************
* general columns' settings                                            *
************************************************************************
* get columns object
  lo_columns = po_alv_table->get_columns( ).

* fix key fields
  lo_columns->set_key_fixation(
*    VALUE  = IF_SALV_C_BOOL_SAP~TRUE
         ).

* optimize columns' width
  lo_columns->set_optimize(
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
*    VALUE  = IF_SALV_C_BOOL_SAP~TRUE
        ).

* bring column PLANETYPE to position 5
  lo_columns->set_column_position(
      columnname = 'PLANETYPE'
      position   = 5
        ).

* show column SEATSFREE in position 6
  lo_columns->set_column_position(
      columnname = 'SEATSFREE'
      position   = 6
        ).

**********************************************************************
* columns for special purposes                                       *
**********************************************************************
* set exception column
  lo_columns->set_exception_column(
      value     = 'USAGE'
*     GROUP     = SPACE
*     CONDENSED = IF_SALV_C_BOOL_SAP=>FALSE
        ).
* set count column
  lo_columns->set_count_column(
    value = 'LINE_COUNTER' ).

* set cell colors / line colors column
  lo_columns->set_color_column(
    value = 'IT_COLORS' ).

**********************************************************************
* single columns' settings                                           *
**********************************************************************
* Column SEATSFREE
* 1. get object
  TRY.
      lo_column ?= lo_columns->get_column(
                     columnname = 'SEATSFREE' ).
    CATCH cx_salv_not_found INTO go_alv_error.
      cl_sapbc405_exc_handler=>process_alv_error_msg(
        r_error     = g0_alv_error
        type        = 'A'
            ).
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
  ENDTRY.

* 2. change titles
*   (might as well be done via new data element)
* short
  lv_scrtext_s = text-sfr.
  lo_column->set_short_text( value = lv_scrtext_s ).
* medium
  lv_scrtext_m = text-sfm.
  lo_column->set_medium_text( value = lv_scrtext_m ).
* long
  lv_scrtext_l = text-sfl.
  lo_column->set_long_text( value = lv_scrtext_l ).

* 3. set tooltip
  lv_lvc_tip = text-sft.
  lo_column->set_tooltip( value = lv_lvc_tip ).

* set color to COL_GROUP
  ls_lvc_s_colo-col = col_group.
  ls_lvc_s_colo-int = 1.
  lo_column->set_color( value = ls_lvc_s_colo ).

* column MANDT
* 1. get object
  TRY.
      lo_column ?= lo_columns->get_column(
                     columnname = 'MANDT' ).
    CATCH cx_salv_not_found INTO go_alv_error.
      cl_sapbc405_exc_handler=>process_alv_error_msg(
         r_error = go_alv_error
         type    = 'A'
             ).
  ENDTRY.

* 2. set as technical
  lo_column->set_technical(
*    VALUE  = IF_SALV_C_BOOL_SAP=>TRUE
         ).

* column PRICE
  lo_column ?= lo_columns->get_column(
                 columnname = 'PRICE' ).
  lo_column->set_visible(
      value  = if_salv_c_bool_sap=>false
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
        ).

* column ICON_FUTURE
  lo_column ?= lo_columns->get_column(
                columnname = 'ICON_FUTURE' ).
  lo_column->set_icon( ).

  lv_scrtext_s = text-fut.
  lo_column->set_short_text( value = lv_scrtext_s ).
  lo_column->set_row( value = 2 ).

* column PAYMENTSUM
  lo_column ?= lo_columns->get_column(
                columnname = 'PAYMENTSUM' ).
  lo_column->set_row( value = 2 ).

* column CURRENCY
  lo_column ?= lo_columns->get_column(
                columnname = 'CURRENCY' ).
  lo_column->set_row( value = 2 ).

ENDFORM.                    " set_columns
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Explain the concept of columns with special technical meanings
- Define the color of columns, rows, and cells
- Define text properties
- Control the display of numerical values

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV OM: Functions, Events, and Interactive Elements

Lesson Overview

In this lesson, you learn how the generic functions are provided for the ALV and how you can define and handle your own functions. To do this, you learn about the available events of the ALV and how they are handled.

Essentially, there are three events:

- *added_function* for customers' own functions
- *double_click* for the double-click
- *link_click* for the clicking of a link

In this context, you learn about all interactive elements that are possible in the ALV. Clicking an element of this type triggers an associated event. There are the following interactive elements:

- Checkbox
- Pushbutton
- Internet link
- Click area

Lesson Objectives

After completing this lesson, you will be able to:

- Make generic ALV functions available with the desired scope
- Add customer functions to the ALV
- Handle ALV events such as the double-click or the call of a customer function
- Display interactive elements in the ALV output

Business Example

You are working as a programmer for ABC Travel. Your manager wants your booking report to have the customer function to assure that “only columns with customer data are displayed”. Using this function, all columns that do not directly contain customer information (for example, “travel agency number” or “point of sale”) are hidden.

It is also to be possible to obtain additional information about the customer – namely, the total number of bookings that the customer has already made – by double-clicking a booking.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Finally, there is to be a hyperlink to the Internet for every booking (for example, the customer's homepage). When the link is clicked, the user is taken directly to the target of the link.

Providing Functions

By default, no functions are available to the user for working with the ALV output. Even the basic functions such as sorting, filter, printing, and so on are not available with the simplest call of an ALV output. For the user to be able to use a function, you need to explicitly provide it.

Functions are divided into two groups:

Generic ALV Functions

You can activate these functions regardless of the objective of your application. Generic ALV functions are, for example, sorting, filtering, managing layouts, printing.

Self-Defined, Application-Specific Functions

You provide functions through your application that are specifically designed for the requirements of the application. You then provide buttons in the ALV output, with which the users can trigger the functions.

Using Generic ALV Functions

You can provide functions with which the user can operate the ALV output. The functions that the system provides for the ALV are known as generic ALV functions. Generic ALV functions are, for example, sorting, filtering, or making layout settings. Generic means here that the system checks whether a function is meaningful in a particular situation. It activates or deactivates the button or menu entry accordingly. On the other hand, the system has no influence on the activation status of self-defined functions.

The names of all of the generic ALV functions are contained as constants in the interface *IF_SALV_C_FUNCTION_LIST*. You can activate or deactivate every generic ALV function individually. However, most generic ALV functions belong to usefully-combined groups of functions. You can activate all functions of a group with a single command.

You can make the following settings for generic ALV functions:

- Get function settings
- Activate generic ALV functions
- Check the visibility and activation status of a generic ALV function

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

You generate an object of the class *CL_SALV_FUNCTION* with every function that you activate. You can manage all of these objects using the list of all function objects, which is an object of the class *CL_SALV_FUNCTIONS_LIST*. The method GET_FUNCTIONS of the main ALV class returns this list of all functions. You use this object to make all settings for generic ALV functions.

There is a setter method for every generic ALV functions, with which you can make the button or menu entry for a function visible to and usable by users.

You can display generic ALV functions in different ways:

- You activate every generic ALV function individually.
- You activate a function group. All of the functions of this function group are activated simultaneously.
- You activate all generic ALV functions simultaneously.

A predefined default selection of the most frequently used generic ALV functions is also available.

Activating Generic Functions (Class CL_SALV_FUNCTIONS_LIST)

Method	Description
SET_<generic ALV function>	Activate an individual generic ALV function
SET_GROUP_<function group>	Activate a function group
SET_ALL	Activate all generic ALV functions
SET_DEFAULT	Activate the predefined default selection of generic ALV functions

Activating Generic Functions (1)

```
  DATA: go_alv_table TYPE REF TO cl_salv_table.
  DATA: go_functions TYPE REF TO cl_salv_functions_list.
*
* Precondition: ALV main object has been retrieved !!!
*
  go_functions = go_alv_table->get_functions( ).
  go_functions->set_default( ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Example: Individual Methods for Activating Generic Functions (Class CL_SALV_FUNCTIONS_LIST)

Method	Description
SET_SORT_ASC	Activate the function “Sort in ascending order”
SET_AGGREGATION_MAXIMUM	Activate the function “Generate maximum”
SET_GROUP_SORT	Activate all functions of the “Sort” group
SET_GROUP_AGGREGATION	Activate all functions of the “Aggregation” group (total, maximum; minimum, average)

For a complete overview of all of the methods for activating individual functions or groups of functions, see the system documentation for the class *CL_SALV_FUNCTIONS_LIST*.

Activating Generic Functions (2)

```
  DATA: go_alv_table TYPE REF TO cl_salv_table.
  DATA: go_functions TYPE REF TO cl_salv_functions_list.

* Precondition: ALV main object has been retrieved
  go_functions = go_alv_table->get_functions( ).
  go_functions->set_default( ).
* ... but switch off filter functions
  go_functions->set_group_filter( value  = if_salv_c_bool_sap=>false ).
* ... and additionally set print preview
  go_functions->set_print_preview( ).
```

Screen Status

If you are displaying the ALV in a container, simply specify the desired functions with the appropriate setter method. The corresponding function button is then displayed in the ALV toolbar.

On the other hand, if you are displaying the ALV in the other display types, the ALV controls the entire screen. In this case, you can control which screen status is used and therefore also which functions are available to the user as menu entries and which as buttons. From a technical point of view, the ALV's own toolbar is not used in this case, but rather the application toolbar of the screen status. This knowledge is particularly important when you are providing customer functions.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

You determine the following settings in the fullscreen displays (fullscreen or classic ABAP list) of the ALV output using the screen status:

- Which generic ALV functions and which self-defined functions can the user use?
- Which functions are provided as buttons and which as menu entries?
- In what order are the buttons and menu entries displayed?

By default, the screen status *SALV_TABLE_STANDARD* of the function group *SALV_METADATA_STATUS* is used.

Do not change the original screen status. If you want to make changes to the screen status, for example, because you want to provide self-defined functions, use the original screen status as a template and adjust a copy of it to your requirements.

You can set a different screen status with the method *SET_SCREEN_STATUS* of the class *CL_SALV_TABLE*.

Implementing Self-Defined Application-Specific Functions

You can provide functions that have been specially designed for your application. So that the user can use these functions, generate buttons, or display menu entries in full screen display.

If the user chooses the button for a function that you have defined yourself, he or she triggers the event *ADDED_FUNCTION* of the class *CL_SALV_EVENTS_LIST*. If you register your ALV for this event, you then receive control in the method of the event handler and can handle the desired function code. For more information about this, see the section “Handling Events”.

You proceed differently to define the position of a button or of a menu entry, depending on whether you are using a self-defined function in an ALV output in a container or in a full screen display:

- In the container display, you can choose whether you want to position the buttons for self-defined functions to the left or the right of the buttons for the generic ALV functions. You then determine the sequence of the buttons using the sequence in which you then generate the self-defined functions. (In container display, you can only extend the application toolbar. You cannot generate menu entries using the ALV OM methods.)
- In the other display types, you use the screen status to define the position of a button or of a menu entry. The button is then displayed at the position defined here, regardless of the order in which you generate the functions.

Features

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The following features exist for self-defined functions:

- Get function settings
- Generate and display the self-defined function
- Check the visibility and activation status of a self-defined ALV function
- Run a generic ALV function instead of a self-defined function

Get function settings

You generate an object of the class *CL_SALV_FUNCTION* with every function that you activate. You can manage all of these objects using the list of all function objects, which is an object of the class *CL_SALV_FUNCTIONS_LIST*. There is the method *GET_FUNCTIONS* in the main ALV class *CL_SALV_TABLE* for this purpose. It returns the settings of all functions. You use this object to make all settings for self-defined functions.

Generating, Displaying, and Deleting Self-Defined Functions

Depending on whether you implement a self-defined function in the container or in a fullscreen display, you use different methods to display the function:

- In the case of container display, you explicitly generate a function object with the desired properties and display the button.
- In the other display types, you generate the function object indirectly by entering the self-defined function in the screen status. In this case, the function is displayed directly. You do not need to enter any additional commands to display it.

While you define the appearance, tooltip, and position of the function using the screen status in the case of full screen displays, you make these settings when generating the function object in the case of container display.

Methods for Generating, Displaying, and Deleting Self-Defined Functions (Class CL_SALV_FUNCTIONS_LIST)

Method	Description
ADD_FUNCTION	Generate and display the function object for a self-defined function
REMOVE_FUNCTION	Delete the function object
ADD_FUNCTION (Parameter TEXT)	Define the text on the button
ADD_FUNCTION (Parameter ICON)	Define the icon on the button
ADD_FUNCTION (Parameter POSITION)	Define the position within the application toolbar
ADD_FUNCTION (Parameter TOOLTIP)	Define the tooltip for the button

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Defining a Customer Function for the Container

```
************************************************************************
* User-defined functions (for container)
************************************************************************
  DATA: go_functions TYPE REF TO cl_salv_functions_list.
  DATA: gv_icon      TYPE string,
        gv_text      TYPE string,
        gv_tooltip   TYPE string.
...
* !!! precondition: FUNCTIONS object (go_functions) has been retrieved !!!
  IF pa_cont = 'X'. " ALV is to be displayed in Container
*   type conversions for method call
    gv_icon    = icon_customer.
    gv_text    = text-ocd.    " only customer data
    gv_tooltip = text-oct. " only columns with customer data

    go_functions->add_function(
        name     = 'ONLY_CD'
        icon     = gv_icon
        text     = gv_text
        tooltip  = gv_tooltip
        position = if_salv_c_function_position=>right_of_salv_functions
           ).
  ENDIF.
```

Defining a Customer Function for Full Screen Display Formats

```
PARAMETERS: pa_full RADIOBUTTON GROUP disp DEFAULT 'X', " fullcreen
            pa_list RADIOBUTTON GROUP disp.             " ABAP list
DATA: go_alv_table TYPE REF TO cl_salv_table.

************************************************************************
* User-defined functions via GUI status
* (for full screen grid or classical ABAP list)
************************************************************************
* customer-defined GUI status should be modified copy of
* GUI status SALV_TABLE_STANDARD of function pool SALV_METADATA_STATUS

* !!! precondition: ALV object (alv) has been retrieved !!!
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
  IF pa_full = 'X' OR pa_list = 'X'.
    go_alv_table->set_screen_status(
        report        = sy-cprog
        pfstatus      = 'UD_FULLSCREEN'
*       set_functions = cl_salv_model_base=>c_functions_all
           ).
  ENDIF.
```

Checking the Visibility and Activation Status of a Self-Defined ALV Function

You can show or hide the pushbuttons or menu entries of self-defined functions. In doing this, unlike with generic ALV functions, you can influence the usability of a self-defined function: You activate or deactivate the function.

You can define the visibility of a self-defined function in all output formats. You can only define the activation status, on the other hand, in the container.

Methods for Defining the Visibility and Activation Status of a Self-Defined Function (Class CL_SALV_FUNCTIONS_LIST)

Method	Description
SET_FUNCTION	Show or hide the self-defined function
ENABLE_FUNCTION	Activate/deactivate the self-defined function

Events and Event Handling

An object can announce that its state has changed by triggering events. Other classes can contain handler methods (as instance methods or class methods) that are executed when the event occurs.

An instance method is a method that can only be executed on an existing object (for example, the “cancel flight booking” method can only be used if a booking exists). A class method is a method that you can meaningfully execute irrespective of the existence of individual objects (example: “Create flight booking”).

For the handler methods of the instances or classes to be run, these instances or classes need to be entered into a list of the object that can trigger the event. Making the entry in this list is known as registration. At the point when the event is triggered, a Web AS service then ensures – on the basis of this list – that the registered methods are executed.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

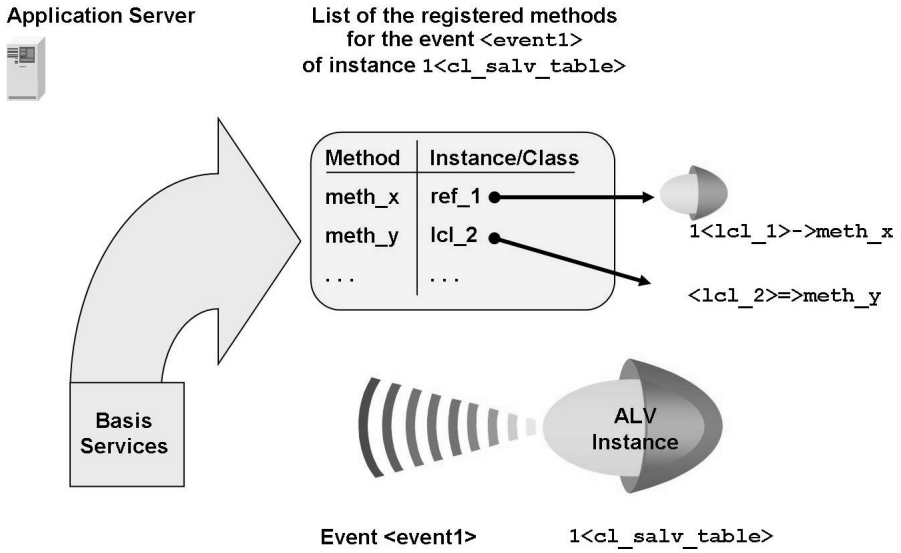


Figure 210: Processing ABAP Objects Events

Overview of the ALV Events

All of the events available for the ALV are defined in the class CL_SALV_EVENTS_TABLE.

The following events are available in the ALV:

ALV Events (Class CL_SALV_EVENTS_TABLE)

Event	Description
ADDED_FUNCTION	Control after executing a self-defined function
DOUBLE_CLICK	Control after double-click
LINK_CLICK	Control after single-clicking hotspot or button
BEFORE_SALV_FUNCTION	Control before executing a generic ALV function
AFTER_SALV_FUNCTION	Control after executing a generic ALV function
TOP_OF_PAGE	Control at the *TOP-OF-PAGE* event (list processing)
END_OF_PAGE	Control at the *END-OF-PAGE* event (list processing)

The following sections describe how each of these events can be handled.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Handling the Call of a Self-Defined Function

The following explains how to handle the call of a self-defined function. From a technical point of view, a call of this type is an event for the ALV event object, this being the *ADDED_FUNCTION* event of the *CL_SALV_EVENTS_TABLE* class. In order to react to this event (one can also say: in order to handle it), we have to implement the following:

Work Steps for Implementing Event Handling

1. Defining and implementing a handler method: In particular, define the *IMPORTING* parameters of your method. It is best to read the information about the parameters and their types in the Class Builder first.
2. Retrieve the ALV event object with the method *GET_EVENT* of the class *CL_SALV_TABLE*
3. Register the handler method for the event object of the ALV

The basic syntax for creating a handler method (both for an instance and a class method) is explained in the following.

```
CLASS <class_name> DEFINITION.
  [PUBLIC| PROTECTED| PRIVATE] SECTION.
    [CLASS-]METHODS:
       <method_name> FOR EVENT <event_name> OF <class_name>
           IMPORTING <p1> ... <pn> [SENDER],
       <method_name1> ...
ENDCLASS.
CLASS <class_name> IMPLEMENTATION.
  METHOD <method_name>.
    ...          " method implementation
  ENDMETHOD.
ENDCLASS.
```

Figure 211: Syntax: Handler Method for an Event

Defining a Handler Method (as a Class Method)

```
*---------------------------------------------------------------------*
*       CLASS lcl_event_handler DEFINITION
*---------------------------------------------------------------------*
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS:
```


© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
      on_added_function
        FOR EVENT added_function
        OF cl_salv_events_table
        IMPORTING e_salv_function.  " type: salv_de_function
ENDCLASS.                    "lcl_event_handler DEFINITION
```

Implementing a Handler Method

```
*---------------------------------------------------------------------*
*       CLASS lcl_event_handler IMPLEMENTATION
*---------------------------------------------------------------------*
CLASS lcl_event_handler IMPLEMENTATION.

  METHOD on_added_function.
    DATA: lo_columns TYPE REF TO cl_salv_columns_table,
          lo_column  TYPE REF TO cl_salv_column_table.

    CASE e_salv_function.
      WHEN 'ONLY_CD'.    " show only columns with customer data
*       set several columns invisible
        lo_columns = alv->get_columns( ).
*       hide column LEAVES_HOME
        lo_column ?= lo_columns->get_column(
                       columnname = 'LEAVES_HOME' ).
        lo_column->set_visible(
            value  = if_salv_c_bool_sap=>false ).
...
    ENDCASE.
  ENDMETHOD.                    "on_added_function

ENDCLASS.                    "lcl_event_handler DEFINITION
```

Retrieving the ALV Event Object and Registering the Handler Method

```
  DATA: go_alv_table TYPE REF TO cl_salv_table.
  DATA: go_event     TYPE REF TO cl_salv_events_table.

* get the EVENT object
* !!! precondition: ALV object is bound !!!
  go_event = go_alv_table->get_event( ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
  SET HANDLER:
    lcl_event_handler=>on_added_function FOR go_event.
```

Alternatively, you can define the handler method as an instance method.

Defining a Handler Method (as an Instance Method)

```
*---------------------------------------------------------------------*
*       CLASS lcl_event_handler DEFINITION
*---------------------------------------------------------------------*
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    METHODS:
      on_added_function
        FOR EVENT added_function
        OF cl_salv_events_table
        IMPORTING e_salv_function.  " type: salv_de_function
ENDCLASS.                    "lcl_event_handler DEFINITION

* !!! IMPLEMENATION is absolutely similar to class method !!!
```

There are no syntactic differences to the class method when implementing the instance method. Note, however, that for the registration, you now need to ensure that an object of the type of the class that contains the handler method must first be created (in this case, `lcl_event_handler`).

Registering the Handler Method (for Instance Methods)

```
  DATA: go_alv_table TYPE REF TO cl_salv_table.
  DATA: go_event     TYPE REF TO cl_salv_events_table.
  DATA: go_handler   TYPE REF TO lcl_event_handler.

* get the EVENT object
* !!! precondition: ALV object has been retrieved !!!
  go_event = go_alv_table->get_event( ).

  CREATE OBJECT go_handler.

  SET HANDLER:
    go_handler->on_added_function FOR go_event.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Handling the Call of a Generic Function

Normally, a programmer does not need to handle a generic function. However, if, in certain cases, you want to pass additional statements to the ALV before or after the originally intended function, handle the events *BEFORE_SALV_FUNCTION* or *AFTER_SALV_FUNCTION* of the class *CL_SALV_EVENTS_TABLE* to do this.

Handling a Double-Click

You can handle double-clicking or pressing the F2 key in two different ways:

1. You register your application for the event *DOUBLE_CLICK* (class *CL_SALV_EVENTS_TABLE*)
2. You directly specify the function code of the function that is executed at the double-click / F2. Use the method *SET_F2_CODE* of the class *CL_SALV_FUNCTIONAL_SETTINGS* to do this.

The event *DOUBLE_CLICK* returns the position of the cell in which the user performed the double-click. The second option, on the other hand, provides you with no information about where the double-click was placed.

The implementation of double-click handling using the *DOUBLE_CLICK* event is explained in the following. Note, in particular, that this event imports two parameters, *row*, and *column*. You can specify these by name in the interface of the handler method in the `IMPORTING` clause, and are then available as data objects with the same names for additional programming.

You can again determine the types assigned to these parameters in the Class Builder. In the Class Builder, call the class *CL_SALV_EVENTS_TABLE*, choose the *Events* tab, place the cursor on the event in which you are interested, choose the *Parameters* button, and then read about the type assigned to the parameters (column *Reference Type*).

Handling a Double-Click: Defining a Method

```
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS:
      on_double_click
        FOR EVENT double_click
        OF cl_salv_events_table
        IMPORTING
          row       " type salv_de_row
          column.   " type salv_de_column

ENDCLASS.                    "lcl_event_handler DEFINITION
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Handling a Double-Click: Implementing a Method

```
*---------------------------------------------------------------*
*       CLASS lcl_event_handler IMPLEMENTATION
*---------------------------------------------------------------*
CLASS lcl_event_handler IMPLEMENTATION.
  METHOD on_double_click.
    DATA: lv_text TYPE c LENGTH 70,
          lv_row  TYPE c LENGTH 10.

      lv_row = row.
      CONCATENATE
        'Row:' lv_row 'Column:' column ')'
        INTO lv_text SEPARATED BY space.

      MESSAGE lv_text TYPE 'I'.
    ENDIF.
  ENDMETHOD.                    "on_double_click
ENDCLASS.                   "lcl_event_handler DEFINITION

************************************************
* !!! DON'T FORGET TO SET METHOD AS HANDLER !!! *
************************************************
```

Handling a Single-Click

So that the user not only selects the cell with a single-click, but also triggers a specific function, you need to ensure the following:

- The cells for which this function is to be triggered are an interactive element (cell types *HOTSPOT*, *BUTTON*, *CHECKBOX*, *CHECKBOX_HOTSPOT*, *LINK*).
- You have registered your ALV for the event *LINK_CLICK* (class *CL_SALV_EVENTS_TABLE*) and programmed a suitable event handler.

Interactive Elements

You can display various elements instead of text in the content of your ALV output. You can handle some of these elements as a click area for the user. You can display the following elements:

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Checkbox

Differentiate here between activated and deactivated checkboxes. Activated checkboxes trigger an event when clicked.

Button

You can handle the user actions for this element with an event.

Hyperlink

These specify Internet addresses (URLs). The user can call the corresponding page directly in his or her Internet browser by clicking an Internet address of this type.

Click area

You can handle the actions for this element with an event.

The setting for an interactive element is known as a cell type.

There are two options for assigning cell types:

- You can assign a cell type to an entire column using the method *SET_CELL_TYPE* of the class *CL_SALV_COLUMN_LIST*.
- You can assign a separate cell type to each individual cell.

Cell Types (Technical Names from Interface IF_SALV_C_CELL_TYPE)

Interface Constant	Description
TEXT	Text (default)
CHECKBOX	Checkbox
BUTTON	Pushbutton
DROPDOWN	Dropdown
LINK	Link
HOTSPOT	Hotspot
CHECKBOX_HOTSPOT	Checkbox with hotspot

Handling a Link-Click (Single-Click)

The cell types checkbox, button (in container only), and click area all trigger the event *LINK_CLICK* of the class *CL_SALV_EVENTS_TABLE* when they are clicked. The event returns the parameters *row* and *column*, which contain the row number and the column name of the cell that was clicked.

The following example starts from the point that the cell type for the column *LUGGWEIGHT* was defined as *BUTTON* if the luggage weighs more than 20 kg. The fields *LEAVES_HOME* and *GOES_HOME* are displayed as changeable checkboxes (type *CHECKBOX_HOTSPOT*).

 © 2011 SAP AG. All rights reserved.

Handling a Link-Click – Defining the Method

```
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS:
      on_link_click
        FOR EVENT link_click
        OF cl_salv_events_table
        IMPORTING
          row       " type salv_de_row
          column.   " type salv_de_column
ENDCLASS.                    "lcl_event_handler DEFINITION
```

Handling a Link-Click – Implementing the Method

```
CLASS lcl_event_handler IMPLEMENTATION.
  METHOD on_link_click.
    CASE column.
      WHEN 'LUGGWEIGHT'.
        MESSAGE 'Too much luggage'(lwl) TYPE 'I'.
      WHEN 'LEAVES_HOME'.
*       change flag
        READ TABLE gt_bookings INTO gs_booking INDEX row.
        IF sy-subrc = 0.
          IF gs_booking-leaves_home = 'X'.
            CLEAR gs_booking-leaves_home.
          ELSE.
            gs_booking-leaves_home = 'X'.
          ENDIF.
          MODIFY gt_bookings FROM gs_booking
            INDEX row TRANSPORTING leaves_home.
          go_alv_table->refresh( ).
        ENDIF.
    ENDCASE.
  ENDMETHOD.                    "on_link_click
ENDCLASS.                    "lcl_event_handler IMPLEMENTATION
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Handling a Link-Click – Registering the Method

```
  DATA: go_event TYPE REF TO cl_salv_events_table.
* !!! Precondition: EVENT object has been retrieved !!!

  SET HANDLER:
    lcl_event_handler=>on_link_click FOR go_event.
```

Displaying an Internet Address

You can manage any number of Internet addresses as hyperlink objects. These hyperlink objects are each identified by a unique key.

You can display one of these hyperlink objects in any cell. Use the hyperlink column to do this. The hyperlink column is (in the same way as the color and cell type columns) a control column in which properties of the displayed cells are displayed.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The basic procedure for using hyperlinks is:

1. **Create an object** for technical settings (class *CL_SALV_FUNCTIONAL_SETTINGS*) and a subobject for all hyperlinks (class *CL_SALV_HYPERLINKS*).

   In order to better understand, recall the class hierarchy which applies here: *CL_SALV_TABLE* - *CL_SALV_FUNCTIONAL_SETTINGS* - *CL_SALV_HYPERLINKS* - *CL_SALV_HYPERLINK*

2. **Creating Repository Objects**: each hyperlink object represents a hyperlink (for example, www.sap.com). These hyperlink objects are reusable. That is, if a large number of cells are to point to www.sap.com, they can all use the associated hyperlink object.

   A hyperlink object is an object of the class *CL_SALV_HYPERLINK*. To be able to create an object of this type, a superobject of the class *CL_SALV_HYPERLINKS* must exist – as described above. The method *ADD_HYPERLINK* is applied to this superobject to generate a hyperlink object. A hyperlink object consists of a key (attribute *HANDLE*) and the actual URL.

3. **Add a hyperlink control column** (type *SALV_T_INT4_COLUMN*) to the internal data table of the ALV. Record the hyperlink control data for every output row in this column (for example, for field *x*, the hyperlink object *1*; for field *y*, the hyperlink object *2*).
4. **Define** the **hyperlink control column** of the internal data table using the method *SET_HYPERLINK_ENTRY_COLUMN* of the class *CL_SALV_COLUMNS_LIST*. This method is executed for the object for all columns (of type *CL_SALV_COLUMNS_TABLE*).

The following figure illustrates the hierarchy of the objects required for working with hyperlinks:

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

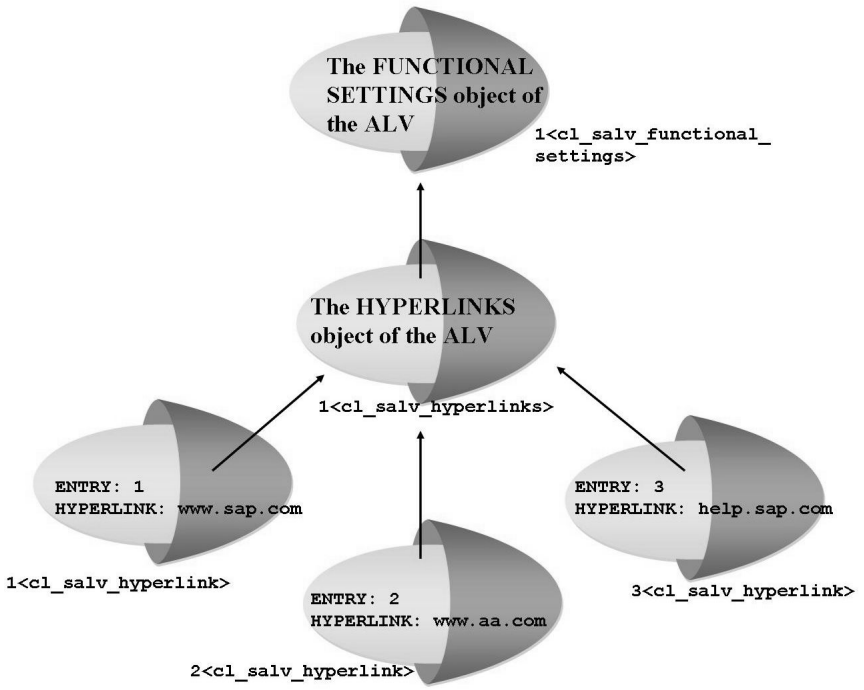


Figure 212: Creating Hyperlink Objects

The control column for the cells with hyperlinks follows a similar principle to the control columns for control information and for cell type information. The difference is the specification of a key of a hyperlink object instead of a color or cell type value predefined by SAP.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

- **Extend the data table with an internal hyperlink table**
- **For every field of the data row that has a hyperlink, this internal table contains an entry that points to a hyperlink object**

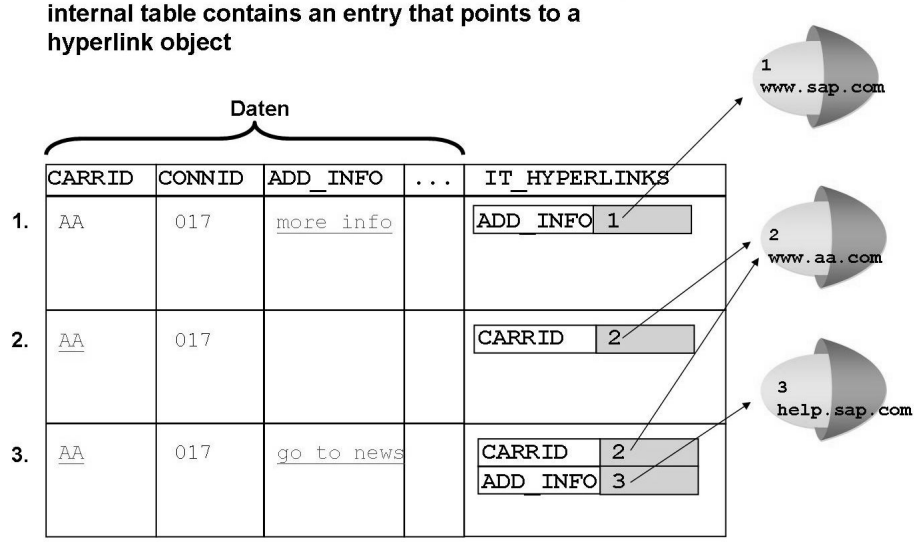


Figure 213: Hyperlink Control Column: Principle

As with the other control columns, an internal table is embedded in a cell of the internal ALV data table, in this case containing the control information for the hyperlinks of this ALV data row. The filling of this embedded internal table is described in the following figure:

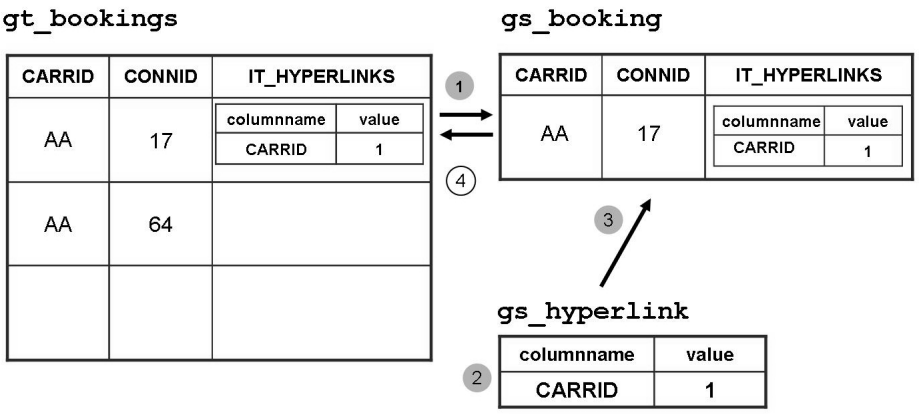


Figure 214: Filling Hyperlink Control Column with Data

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 29: ALV OM: Making Generic Functions Available

Exercise Objectives

After completing this exercise, you will be able to:

- Make generic ALV functions available with the scope that you desire

Business Example

You are working as a programmer for ABC Travel. You manager wants most of the generic functions to be made available. However, it should not be possible to calculate an average, and it should not be possible to export the data.

Task:

Make the generic functions available in precisely the scope that meets your needs.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_DESIGN_S4*.

   Model Solution: *BC405_OM_EVENT_S1*

2. Make all generic ALV functions available. The individual functions that are not to be made available are the calculating of an average value and any exporting (such as exporting to a word processing program or to a local file).

3. **Optional:** Create a new executable program *ZBC405_##_OM_EASY*. This is to display data from the table *SPFLI* (flight connections) in an ALV in full screen display. The common generic ALV functions are to be made available.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 29: ALV OM: Making Generic Functions Available

Task:

Make the generic functions available in precisely the scope that meets your needs.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_DESIGN_S4*.

   Model Solution: *BC405_OM_EVENT_S1*

   a) Self-explanatory

2. Make all generic ALV functions available. The individual functions that are not to be made available are the calculating of an average value and any exporting (such as exporting to a word processing program or to a local file).

   a) **Model Solution**

Main program

```
REPORT bc405_om_event_s1.

INCLUDE bc405_om_event_s1_top.
INCLUDE bc405_om_event_s1_e01.
INCLUDE bc405_om_event_s1_o01.
INCLUDE bc405_om_event_s1_i01.
INCLUDE bc405_om_event_s1_f01.
```

F01 Include

```
*&---------------------------------------------------------------------*
*&      Form  define_settings
*&---------------------------------------------------------------------*
FORM define_settings
  USING po_alv_table TYPE REF TO cl_salv_table.

  PERFORM: set_display    USING po_alv_table,
           set_columns    USING po_alv_table,
           set_sorts      USING po_alv_table,
           set_aggregs    USING po_alv_table,
           set_selections USING po_alv_table,
           set_layout     USING po_alv_table,
           set_functions  USING po_alv_table.
ENDFORM.                     " define_settings
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
*&---------------------------------------------------------------------*
*&      Form  set_functions
*&---------------------------------------------------------------------*
FORM set_functions
  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA: lo_functions TYPE REF TO cl_salv_functions_list.

* get the FUNCTIONS object
  lo_functions = po_alv_table->get_functions( ).

* offer all generic functions
  lo_functions->set_all(
*    VALUE  = IF_SALV_C_BOOL_SAP=>TRUE
         ).
* subtract average
  lo_functions->set_aggregation_average(
      value  = if_salv_c_bool_sap=>false
         ).
* subtract all export functions
  lo_functions->set_group_export(
      value  = if_salv_c_bool_sap=>false
         ).

ENDFORM.                    " set_functions
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

3. **Optional:** Create a new executable program *ZBC405_##_OM_EASY*. This is to display data from the table *SPFLI* (flight connections) in an ALV in full screen display. The common generic ALV functions are to be made available.

   a) **Model Solution:** ***BC405_OM_EVENT_S1_OPT***

Main program

```
REPORT  bc405_om_event_s1_opt.

DATA: gt_spfli TYPE TABLE OF spfli,
      gs_spfli TYPE          spfli.

DATA: go_alv_table TYPE REF TO cl_salv_table,
      go_functions TYPE REF TO cl_salv_functions_list.

SELECT-OPTIONS: so_car FOR gs_spfli-carrid.


START-OF-SELECTION.
* select data in internal table

  SELECT * FROM  spfli
    INTO TABLE gt_spfli
    WHERE carrid IN so_car.

* create the ALV object
  cl_salv_table=>factory(
    IMPORTING
      r_salv_table   = go_alv_table
    CHANGING
      t_table        = gt_spfli
         ).

* set default set of generic functions
  go_functions = go_alv_table->get_functions( ).
  go_functions->set_default( ).

* display it!
  go_alv_table->display( ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Exercise 30: ALV OM: Your Own Functions and Event Handling

Exercise Objectives

After completing this exercise, you will be able to:

- Make a function of your own available
- Handle the call of a user-defined function

Business Example

You are working as a programmer for ABC Travel. Your manager wants to include a customer function in your flight data report, to regroup and color the columns with the reservation information at the press of a button.

Task:

Make a complex customer function available.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_EVENT_S1*.

   Model Solution: *BC405_OM_EVENT_S2*

2. Add a customer function to the program, **only for display in the container**. This function is to display the three columns containing the reservation information from the three classes directly after the key for the flight and highlight them in color.

   First ensure that a function key is displayed on the ALV application toolbar for your function. The technical function name is to be *REORDER*. The function key is to be displayed to the right of the standard function keys. Use “Reservation” as the function text. Select an icon for the function key yourself.

3. Now create a handler method for your function.

   To do this, define a local class (suggested name *LCL_HANDLER*) with a class method for handling the event that is triggered if a customer function is to be executed. Specify the interface of the class method. For information about the interface (the `IMPORTING` parameter of the event), see the Class Builder for the event *ADDED_FUNCTION* of the class *CL_SALV_EVENTS_TABLE*.

   Then implement the method defined above. The three columns containing the reservation information from the three classes are to be displayed directly after the key for the flight and highlight them in color (*COL_TOTAL*). Call the appropriate methods to do this. (Reminder: Column items are managed

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

for all columns using the object (class *CL_SALV_COLUMNS_TABLE*). Column properties, like color, are managed for a single column using an object (class *CL_SALV_COLUMNS_TABLE*).

4. Finally, register your handler method for the ALV object. To do this, first retrieve the event object of the ALV and then perform the actual registration.

5. **Optional:** You can also implement the customer function for the fullscreen and list output. Copy the standard status *SALV_TABLE_STANDARD* of the function group *SALV_METADATA_STATUS* to a screen status of your own (suggested name: *FULLSCREEN*). (This screen status should be a subobject of your own program.) Extend this status *FULLSCREEN* with a function. This must also have the technical name *REORDER*.

   Assign a function key, a new menu entry, and a button in the application toolbar to this function.

   Set the new status with the method *SET_SCREEN_STATUS* of the class *CL_SALV_TABLE*.

   Test whether the function that you defined yourself now also works correctly in the full screen and ABAP list modes.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 30: ALV OM: Your Own Functions and Event Handling

Task:

Make a complex customer function available.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_EVENT_S1*.

   Model Solution: *BC405_OM_EVENT_S2*

   a) Proceed in the usual manner.

2. Add a customer function to the program, **only for display in the container**. This function is to display the three columns containing the reservation information from the three classes directly after the key for the flight and highlight them in color.

   First ensure that a function key is displayed on the ALV application toolbar for your function. The technical function name is to be *REORDER*. The function key is to be displayed to the right of the standard function keys. Use "Reservation" as the function text. Select an icon for the function key yourself.

   a) See complete answer at the end of the task (main task).

3. Now create a handler method for your function.

   To do this, define a local class (suggested name *LCL_HANDLER*) with a class method for handling the event that is triggered if a customer function is to be executed. Specify the interface of the class method. For information about the interface (the `IMPORTING` parameter of the event), see the Class Builder for the event *ADDED_FUNCTION* of the class *CL_SALV_EVENTS_TABLE*.

   Then implement the method defined above. The three columns containing the reservation information from the three classes are to be displayed directly after the key for the flight and highlight them in color (*COL_TOTAL*). Call the appropriate methods to do this. (Reminder: Column items are managed for all columns using the object (class *CL_SALV_COLUMNS_TABLE*). Column properties, like color, are managed for a single column using an object (class *CL_SALV_COLUMNS_TABLE*).

   a) See complete answer at the end of the task (main task).

4. Finally, register your handler method for the ALV object. To do this, first retrieve the event object of the ALV and then perform the actual registration.

   a) See complete answer at the end of the task (main task).

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

5. **Optional:** You can also implement the customer function for the fullscreen and list output. Copy the standard status *SALV_TABLE_STANDARD* of the function group *SALV_METADATA_STATUS* to a screen status of your own (suggested name: *FULLSCREEN*). (This screen status should be a subobject of your own program.) Extend this status *FULLSCREEN* with a function. This must also have the technical name *REORDER*.

   Assign a function key, a new menu entry, and a button in the application toolbar to this function.

   Set the new status with the method *SET_SCREEN_STATUS* of the class *CL_SALV_TABLE*.

   Test whether the function that you defined yourself now also works correctly in the full screen and ABAP list modes.

   a) Create the *FULLSCREEN* status, add the *REORDER* function.

   b) See complete answer at the end of the task (additional task).

Result

MAIN TASK (tasks 1-4)

Your own functions in the container

Main program

```
REPORT bc405_om_event_s2.

INCLUDE bc405_om_event_s2_top.
INCLUDE bc405_om_event_s2_k01.
INCLUDE bc405_om_event_s2_e01.
INCLUDE bc405_om_event_s2_o01.
INCLUDE bc405_om_event_s2_i01.
INCLUDE bc405_om_event_s2_f01.
```

K01 Include

```
CLASS lcl_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS:
      on_added_function
        FOR EVENT added_function
          OF cl_salv_events_table
        IMPORTING e_salv_function.  " type salv_de_function

ENDCLASS.                    "lcl_handler DEFINITION
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
*&---------------------------------------------------------------------*
*       CLASS lcl_handler IMPLEMENTATION
*&---------------------------------------------------------------------*
CLASS lcl_handler IMPLEMENTATION.
  METHOD on_added_function.
    DATA: lo_columns TYPE REF TO cl_salv_columns_table,
          lo_column  TYPE REF TO cl_salv_column_table.
    DATA: ls_lvc_s_colo TYPE lvc_s_colo.

    CASE e_salv_function.
      WHEN 'REORDER'.
* get the COLUMNS object
        lo_columns = go_alv_table->get_columns( ).
* positions: (MANDT column + 3 key colums)
        lo_columns->set_column_position(
            columnname = 'SEATSOCC'
            position   = 5
               ).

        lo_columns->set_column_position(
            columnname = 'SEATSOCC_B'
            position   = 6
               ).

        lo_columns->set_column_position(
            columnname = 'SEATSOCC_F'
            position   = 7
               ).

* prepare color info
        ls_lvc_s_colo-col = col_negative.
        ls_lvc_s_colo-int = 0.
        ls_lvc_s_colo-inv = 0.
* column SEATSOCC
        lo_column
          ?= lo_columns->get_column(
               columnname = 'SEATSOCC' ).
        lo_column->set_color(
          value = ls_lvc_s_colo ).

* column SEATSOCC_B
        lo_column
          ?= lo_columns->get_column(
               columnname = 'SEATSOCC_B' ).
        lo_column->set_color(
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
          value = ls_lvc_s_colo ).

* column SEATSOCC_F
        lo_column
          ?= lo_columns->get_column(
               columnname = 'SEATSOCC_F' ).
        lo_column->set_color(
          value = ls_lvc_s_colo ).

    ENDCASE.
  ENDMETHOD. "on_added_function
ENDCLASS.    "lcl_handler IMPLEMENTATION
```

F01 Include

```
*&---------------------------------------------------------------------*
*&      Form  define_settings
*&---------------------------------------------------------------------*
*       main form to call specialized forms
*----------------------------------------------------------------------*
*      -->po_alv_table  ALV object
*----------------------------------------------------------------------*
FORM define_settings
  USING po_alv_table TYPE REF TO cl_salv_table.

  PERFORM: set_display USING po_alv_table,
           set_columns USING po_alv_table,
           set_sorts USING po_alv_table,
           set_aggregs USING po_alv_table,
           set_selections USING po_alv_table,
           set_layout USING po_alv_table,
           set_functions USING po_alv_table,
           set_events USING po_alv_table.
ENDFORM. " define_settings

FORM set_functions
  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA: lo_functions TYPE REF TO cl_salv_functions_list.

* help variables for method call ADD_FUNCTION
  DATA: lv_icon    TYPE string,
        lv_text    TYPE string,
        lv_tooltip TYPE string.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
* get the FUNCTIONS object
  lo_functions = po_alv_table->get_functions( ).
* offer all generic functions
  lo_functions->set_all(
*    VALUE  = IF_SALV_C_BOOL_SAP=>TRUE
        ).
* subtract average
  lo_functions->set_aggregation_average(
      value  = if_salv_c_bool_sap=>false
        ).
* subtract all export functions
  lo_functions->set_group_export(
      value  = if_salv_c_bool_sap=>false
        ).
* customer defined function:
*   display occupied seats columns most left
*   and highlight them
  IF pa_cont = 'X'.
    lv_icon = icon_insert_row.
    lv_text = text-tot.
    lv_tooltip = text-ttt.
    lo_functions->add_function(
        name     = 'REORDER'
        icon     = lv_icon
        text     = lv_text
        tooltip  = lv_tooltip
        position = if_salv_c_function_position=>right_of_salv_functions
           ).
  ENDIF.

ENDFORM.                    " set_functions

*&---------------------------------------------------------------------*
*&      Form  set_events
*&---------------------------------------------------------------------*
FORM set_events
  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA: lo_event TYPE REF TO cl_salv_events_table.

* get the EVENT object
  lo_event = po_alv_table->get_event( ).

  SET HANDLER lcl_handler=>on_added_function
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
    FOR lo_event.

ENDFORM.                    " set_events
```

ADDITIONAL TASK (task 5)

Your own function in the fullscreen and in the list

E01 Include

```
*&---------------------------------------------------------------------*
*&   Event START-OF-SELECTION
*&---------------------------------------------------------------------*
START-OF-SELECTION.
* retrieve data into internal table
  SELECT        * FROM  sflight
    INTO CORRESPONDING FIELDS OF TABLE gt_flights
         WHERE  carrid  IN so_car
         AND    connid  IN so_con.

* fill additional data fields
  LOOP AT gt_flights INTO gs_flight.
    ...
  ENDLOOP.
...
* adjustments for ALV type
  CASE 'X'.
    WHEN pa_full OR pa_list.

* decide about list display
      IF pa_list IS NOT INITIAL.
        gv_list_display = if_salv_c_bool_sap=>true.
      ELSE.
        gv_list_display = if_salv_c_bool_sap=>false.
      ENDIF.

* create ALV instance
      TRY.
          cl_salv_table=>factory(
           EXPORTING
             list_display   = gv_list_display
*     R_CONTAINER    =
*     CONTAINER_NAME =
            IMPORTING
              r_salv_table   = go_alv_table
```

Continued on next page

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

```
            CHANGING
              t_table        = gt_flights
                ).
        CATCH cx_salv_msg INTO gr_error.
          cl_sapbc405_exc_handler=>process_alv_error_msg(
            r_error = gx_alv_error ).

      ENDTRY.
* set customer defined status
      go_alv_table->set_screen_status(
          report        = sy-cprog
          pfstatus      = 'FULLSCREEN'
*    SET_FUNCTIONS = C_FUNCTIONS_NONE
             ).

* define settings
      PERFORM define_settings USING go_alv_table.
* display ALV
      go_alv_table->display( ).

    WHEN pa_cont.
      CALL SCREEN 100.
  ENDCASE.

...
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 31: ALV OM: Handling Single- and Double-Clicks

Exercise Objectives

After completing this exercise, you will be able to:

- Handle the ALV events double-click and link-click

Business Example

You are working as a programmer for ABC Travel. Your manager wants to be able to display the number of free seats per class in a dialog box by double-clicking a flight (until now, only the total number of free seats has been known).

The contents of cells in the "Airline" column are also now to be displayed underlined (therefore as a link). When you click a cell of this type, detailed data about the airline is to be displayed in a dialog box.

Task:

Handle the events *DOUBLE_CLICK* and *LINK_CLICK*.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_EVENT_S2*.

   Model solution: *BC405_OM_EVENT_S3*

2. When the user double-clicks a data row, a dialog box is to appear, showing the number of free seats in each class (Economy, Business, and First).

   Now define and implement a corresponding handler method for the *DOUBLE_CLICK* event. Take the number of the selected data row from the event interface, and read the associated data record from the internal data table (using `READ TABLE`).

   Calculate the three values and display it with an information message. Link all output values, including descriptive texts, to a single character string `CONCATENATE` and output them with `MESSAGE message_text TYPE 'I'`.

   Register your method.

3. Display the contents of the "Airline" column as a click area (*HOTSPOT* format). To do this, change the cell type of the corresponding column with a suitable method.

4. Now handle the *LINK_CLICK* event, which is triggered by clicking a cell in the "Airline" column.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

As previously with the handling of the double-click, first retrieve the affected data record. Then retrieve the following detailed data from the database table *SCARR* for the airline in which you are interested:

- Short name

- Airline name

- Local currency

Display the collected data with an information message again.

Remember to register the method.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Solution 31: ALV OM: Handling Single- and Double-Clicks

Task:

Handle the events *DOUBLE_CLICK* and *LINK_CLICK*.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_EVENT_S2*.

   Model solution: *BC405_OM_EVENT_S3*

   a) Self-explanatory

2. When the user double-clicks a data row, a dialog box is to appear, showing the number of free seats in each class (Economy, Business, and First).

   Now define and implement a corresponding handler method for the *DOUBLE_CLICK* event. Take the number of the selected data row from the event interface, and read the associated data record from the internal data table (using `READ TABLE`).

   Calculate the three values and display it with an information message. Link all output values, including descriptive texts, to a single character string `CONCATENATE` and output them with `MESSAGE message_text TYPE 'I'`.

   Register your method.

   a) See the complete answer at the end of the task.

3. Display the contents of the “Airline” column as a click area (*HOTSPOT* format). To do this, change the cell type of the corresponding column with a suitable method.

   a) See the complete answer at the end of the task.

4. Now handle the *LINK_CLICK* event, which is triggered by clicking a cell in the “Airline” column.

   As previously with the handling of the double-click, first retrieve the affected data record. Then retrieve the following detailed data from the database table *SCARR* for the airline in which you are interested:

   - Short name

   - Airline name

   - Local currency

   Display the collected data with an information message again.

   Remember to register the method.

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

a) See the complete answer at the end of the task.

Result

Main program

```
REPORT bc405_om_event_s2.

INCLUDE bc405_om_event_s2_top.
INCLUDE bc405_om_event_s2_k01.
INCLUDE bc405_om_event_s2_e01.
INCLUDE bc405_om_event_s2_o01.
INCLUDE bc405_om_event_s2_i01.
INCLUDE bc405_om_event_s2_f01.
```

K01 Include

```
CLASS lcl_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS:
      on_added_function
        FOR EVENT added_function
          OF cl_salv_events_table
        IMPORTING
          e_salv_function,  " type salv_de_function
      on_double_click
        FOR EVENT double_click
          OF cl_salv_events_table
        IMPORTING
          row,   " type salv_de_row = int4
      on_link_click
        FOR EVENT link_click
          OF cl_salv_events_table
        IMPORTING
          row    " type salv_de_row = int4
          column." type salv_de_column = char(30)

ENDCLASS.                    "lcl_handler DEFINITION
*&---------------------------------------------------------------------*
*       CLASS lcl_handler IMPLEMENTATION
*&---------------------------------------------------------------------*
CLASS lcl_handler IMPLEMENTATION.
  METHOD on_added_function.
    ...
  ENDMETHOD.                    "on_added_function
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
METHOD on_double_click.
  DATA: lv_message_text TYPE c LENGTH 70,
        lv_seatsfree_e  TYPE c LENGTH 10,
        lv_seatsfree_b  TYPE c LENGTH 10,
        lv_seatsfree_f  TYPE c LENGTH 10.

  READ TABLE gt_flights INTO gs_flight INDEX row.
  CHECK sy-subrc EQ 0.
  lv_seatsfree_e =
    gs_flight-seatsmax - gs_flight-seatsocc.
  lv_seatsfree_b =
    gs_flight-seatsmax_b - gs_flight-seatsocc_b.
  lv_seatsfree_f =
    gs_flight-seatsmax_f - gs_flight-seatsocc_f.

  CONCATENATE
    'Free Seats:'(frs)
    'Economy:'(eco) lv_seatsfree_e
    'Business'(bus) lv_seatsfree_b
    'First'(fst) lv_seatsfree_f
    INTO lv_message_text SEPARATED BY space.

  MESSAGE lv_message_text TYPE 'I'.

ENDMETHOD.  "on_double_click

METHOD on_link_click.
  DATA: lv_carrname     TYPE scarr-carrname,
        lv_currcode     TYPE scarr-currcode,
        lv_message_text TYPE c LENGTH 70.

  READ TABLE gt_flights INTO gs_flight INDEX row.
  CHECK sy-subrc EQ 0.
  CASE column.
    WHEN 'CARRID'.
      SELECT SINGLE carrname currcode
        INTO (lv_carrname, lv_currcode)
        FROM scarr
        WHERE carrid = gs_flight-carrid.
      IF sy-subrc EQ 0.
        CONCATENATE
          'Airline:'(air) gs_flight-carrid
          'Name:'(nme) lv_carrname
          'Currency:' lv_currcode
          INTO lv_message_text
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
              SEPARATED BY space.

          MESSAGE lv_message_text TYPE 'I'.
        ENDIF.
    ENDCASE.
  ENDMETHOD.                    "on_link_click
ENDCLASS.    "lcl_handler IMPLEMENTATION
```

F01 Include

```
*&---------------------------------------------------------------------*
*&      Form  set_events
*&---------------------------------------------------------------------*
FORM set_events
  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA: lo_event TYPE REF TO cl_salv_events_table.

* get the EVENT object
  lo_event = po_alv_table->get_event( ).

  SET HANDLER:
    lcl_handler=>on_added_function FOR lo_event,
    lcl_handler=>on_double_click FOR lo_event,
    lcl_handler=>on_link_click FOR lo_event.
ENDFORM.  " set_events


*&---------------------------------------------------------------------*
*&      Form  set_columns
*&---------------------------------------------------------------------*
FORM set_columns

...

* column carrid: set cell type
  lo_column ?= lo_columns->get_column(
                columnname = 'CARRID' ).
  lo_column->set_cell_type(
      value  = if_salv_c_cell_type=>hotspot ).

ENDFORM.                    " set_columns
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Make generic ALV functions available with the desired scope
- Add customer functions to the ALV
- Handle ALV events such as the double-click or the call of a customer function
- Display interactive elements in the ALV output

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson: ALV OM: Print Design and the Header and Footer

Lesson Overview

In this lesson, you learn about the special settings that are available for printing from the ALV, such as displaying the sorting and filter criteria, or selection criteria.

You also learn how you can format the output on the screen and when printed by adding header and footer areas (for each page or for the output overall).

Lesson Objectives

After completing this lesson, you will be able to:

- Make settings for printing from the ALV
- Handle the events *TOP_OF_PAGE* and *END_OF_PAGE*
- Compile design objects and assign a header or footer area

Business Example

You are working as a programmer for ABC Travel. Your manager wants there to be a page break for each new “Airline” control level when printing the booking list, and the key of the airline to be displayed in the header area of the list.

A fixed text is also to be displayed once before the start of the output of the data rows.

Making Settings for Printing

The user can print the ALV output on a printer or view it as a print output on a screen. You can use the ALV object model to make various settings to control this print output.

Independent of which display type you choose for displaying the ALV output on the screen: When printing, the ALV output is always converted to the classic ABAP list (or in a list in the style of the ABAP list) and printed in this form.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

You can make the following settings for the printing of the ALV output:

- Output additional information
- Output standard title of the program
- Define that the ALV is called in print mode
- Note actual list size when printing
- Optimize the column widths in the printout
- Reserve rows for the *END_OF_PAGE* event

Outputting Additional Information

You can output two areas in the printout in addition to the actual ALV output:

List information

Information about which filters and which sorting have been used on the ALV output and the number of data records to which the ALV output is restricted.

Selection information

Information about the application-specific selection criteria with which the user called the ALV output (such as company code or material number)

If you activate one or both pieces of additional information, the corresponding information blocks are printed before the ALV output.

You can define whether a page break is to be generated after these information blocks, so that the additional information acts as a cover sheet on a separate page, or whether the ALV output is to be appended directly after the additional information.

You can output the standard title of the program. This consists of the current date, the title of the ALV output, and the number of pages.

Methods for Print Settings (Class CL_SALV_PRINT)

Method	Description
SET_LISTINFO_ON	Output list information
SET_SELECTIONINFO_ON	Output selection information
SET_COVERPAGE_ON	Output additional information on a separate page

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Method	Description
SET_REPORT_STANDARD_HEADER_ON	Output standard title of the program
SET_PRINT_ONLY	Print the ALV output immediately (without displaying it on the screen first)
SET_PRINT_PARAMETERS_ENABLED	Note actual list size when printing

Note actual list size when printing

You can use the application to define how wide the list is to be on the printout. If the width of the ALV output has changed (for example, because the user has shown additional columns), it may be that the space on the page is not sufficient to print all information. You can specify that the ALV determines the current size before printing, and adjusts the print parameters accordingly.

Print Settings

```
DATA: go_alv_table TYPE REF TO cl_salv_table
DATA: go_print     TYPE REF TO cl_salv_print.
...
* !!! Precondition: the ALV object is bound !!!
* get the PRINT object
  go_print = go_alv_table->get_print( ).

* set listinfo on
  go_print->set_listinfo_on( value = 'X' ).

* set selectioninfo on
  go_print->set_selectioninfo_on( callback_program = sy-cprog  ).

* set coverpage on
  go_print->set_coverpage_on( value = 'X' ).

* set standard header on
  go_print->set_report_standard_header_on( value = 'X' ).

* regard actual listsize (optimize column width)
  go_print->set_print_parameters_enabled( value = 'X' ).
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Header and Footer of the ALV Output

You can use various elements to design the areas above and below the ALV output. You use **design objects** to do this.

There are various **elements** available, which you combine to make a design object. You then display it in the desired position.

You need to make various decisions – in addition to deciding on the design issues – when doing so:

- In which display type (full screen, in a container) are you using the design object? Not all of the options described below are available in every variant.
- Which design object do you use in which output medium? You can handle the output differently on screen and in the printout.
- At what position in the entire ALV output should the design objects appear? You can handle the following positions:
  - At the start of the entire ALV output (TOP_OF_LIST)
  - At the end of the entire ALV output (END_OF_LIST)

  For ALV outputs of multiple pages:

  - At the start of a page (TOP_OF_PAGE)
  - At the end of a page (END_OF_PAGE)

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

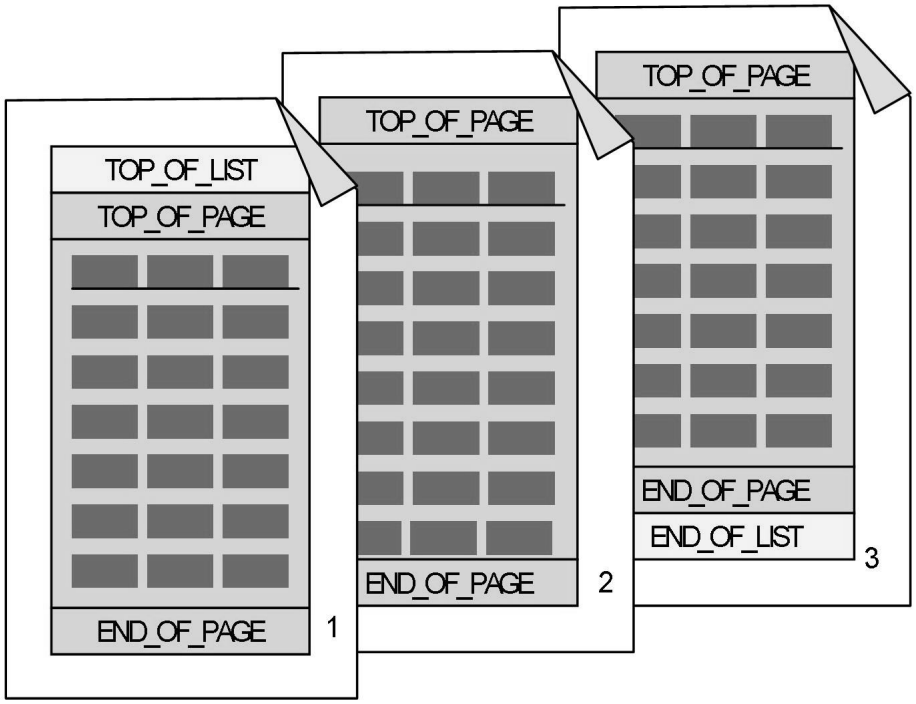


Figure 215: Header and Footer of the ALV Output

Compiling Design Objects

With the design object, you design the areas around the ALV output. You compile a design object from a variety of elements of different types and then display it.

You can use the following element types:

- Header elements (header info)
- Text element with or without a label (text label)
- Action information (action info)

The elements differ only in appearance. No functions are connected with the various element types.

You can arrange these elements within your design object. To do this, choose between two forms of layout:

Row-type layout

All elements are lined up in a row.

Table-type layout

You arrange the individual elements in rows and columns.

You can combine the two layout forms with one another as you wish, that is, you can insert rows into a table and vice versa.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

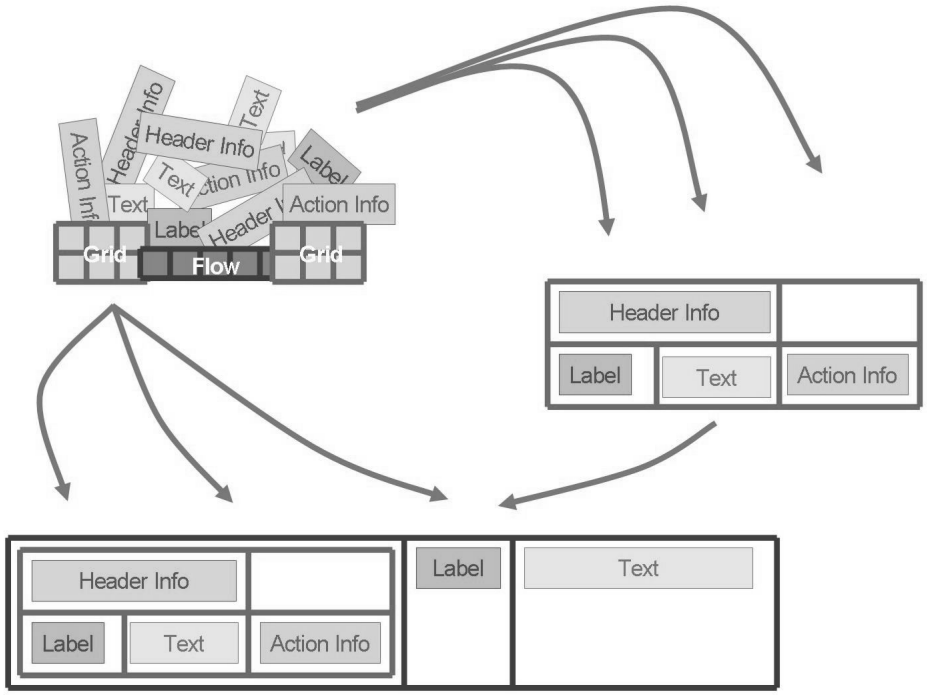


Figure 216: Compiling Design Objects

Integration

A separate **class** is available for each element type:

Layout Classes

Class	Layout
CL_SALV_FORM_LAYOUT_FLOW	Row-type layout
CL_SALV_FORM_LAYOUT_GRID	Table-type layout

Element Type Classes

Class	Element Type
CL_SALV_FORM_ACTION_INFO	Action information
CL_SALV_FORM_HEADER_INFO	Header
CL_SALV_FORM_LABEL	Label
CL_SALV_FORM_TEXT	Text

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

You must proceed in different ways depending on the number of elements that make up your design object:

- If you only want to display one element (such as a text), generate the corresponding object.
- If you want to display multiple elements, generate the desired elements with the methods for the row- or table-type layout elements and specify the desired position within the layout element.

Features

The following functions are available for the **layout forms**:

Layout Functions

Class CL_SALV_FORM_...	Method	Function
LAYOUT_FLOW or *LAYOUT_GRID*	*CREATE_**	Generate an element of the desired type
	SET_ELEMENT	Move an element
LAYOUT_GRID	*CREATE_** *SET_ELEMENT* (parameter *COLSPAN* or *ROWSPAN*)	Link cells
	SET_GRID_LINES	Show lines between columns and rows
	SET_COLUMN_LABEL_FOR	Define which column contains the labels and which column contains the associated texts
	ADD_ROW	Add a row

You have quite similar functions available for the **element types**:

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Element Type Functions

Class CL_SALV_FORM_...	Method	Function
ACTION_INFO and *HEADER_INFO* and *LABEL* and *TEXT*	*SET_TEXT*	Enter a text
	SET_TOOLTIP	Enter a tooltip
	SET_HORIZONTAL_ ALIGNMENT	Set alignment within a cell
LABEL	*SET_LABEL_FOR*	Determine the text element to which the label belongs

Example

The following example shows how you display an element of type *Text* at the end of the ALV output:

Element of Type Text at the End of the ALV Output

```
* Precondition: the ALV object is bound !!!
DATA: go_alv_table TYPE REF TO cl_salv_table.
DATA: go_text      TYPE REF TO cl_salv_form_text.

  CREATE OBJECT go_text
    EXPORTING
      text   = text-nbt " there ain't no better travel
       .
  go_alv_table->set_end_of_list( go_text ).
```

The following example shows how to implement a design object with table-type layout at the beginning of the ALV output:

Element of Type Grid at the Start of the ALV Output

```
* Precondition: the ALV object is bound !!!
```

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
DATA: go_alv_table   TYPE REF TO cl_salv_table.
DATA: go_layout_grid TYPE REF TO cl_salv_form_layout_grid.

  CREATE OBJECT go_layout_grid.

* add header: booking list
  go_layout_grid->create_header_information(
      row     = 1
      column  = 1
      colspan = 2
      text    = text-hdi  " Booking List
     ).
* add text: please check bookings
  go_layout_grid->create_text(
      row     = 2
      column  = 1
      colspan = 2
      text    = text-pls   " Please check bookings
     ).
* set grid lines
  go_layout_grid->set_grid_lines(
    value  = if_salv_form_c_grid_lines=>lines
         ).

  go_alv_table->set_top_of_list( go_layout_grid ).
```

Displaying Design Objects for the Entire ALV Output

When you have compiled a design object, you can display it at different positions in your ALV output:

- Before or after the ALV output
- For ALV outputs of multiple pages, at the beginning or end of each page (page header, footer)

Choosing the Output Medium

If you would like to design the area either above or below the ALV output, you must decide on the output medium for which you would like to implement your design object:

- For display on a screen (online)
- In a printout (print)

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Caution: If you are displaying the ALV in a container, output on the screen (both above and below the ALV output is not possible.

The design object will appear differently depending on the output medium you select.

- If you only use a design object for display on the screen, it will appear on both the screen and the printout.
- If you specify a design object for the display on the screen and another one for the printout, one design object will appear on the screen, and the other in the printout.
- If you only use a design object for the printout, the design object will only appear in the printout.

In order to display a design object on the screen, use the following methods:

Methods for Displaying a Design Object on the Screen or in the Printout (Class CL_SALV_TABLE)

Methods	Function
SET_TOP_OF_LIST	Before output, on screen
SET_END_OF_LIST	After output, on screen
SET_TOP_OF_LIST_PRINT	Before output, in the printout
SET_END_OF_LIST_PRINT	After output, in the printout

Displaying Page Header and Footer

Caution: Your specifications for the page footer are only displayed in the printout or in the print preview of your ALV output with all display types. The page header, on the other hand, is also displayed on the screen with the display type “classic ABAP list”.

Using Events

Unlike the design object above or below the entire ALV output, (*TOP_OF_LIST* and *END_OF_LIST*) the page header or footer is an event: If ALV creates a page break while creating the print output, the *END_OF_PAGE* or *TOP_OF_PAGE* event occurs. ALV checks whether a design object is defined for this event, and inserts it at the correct position.

You can use the parameter *PAGE* in both the *TOP_OF_PAGE* and the *END_OF_PAGE* events to determine the current page number. In the *TOP_OF_PAGE* event, you can also use the *TABLE_INDEX* parameter to get the table index of the entry. The indexed entry is then displayed first on the current page.

 © 2011 SAP AG. All rights reserved.

To be able to display the desired design object in the *END_OF_PAGE* event, you need to explicitly reserve the required number of lines. To do this use the method *SET_RESERVE_LINES* from the *CL_SALV_PRINT* class. With the *GET_RESERVE_LINES*, you can determine how many lines are reserved.

You can handle the events in the usual way. You can either implement design objects as previously discussed or work with the classic *WRITE* statement.

The following example shows the handling of the *TOP_OF_PAGE* event of the class *CL_SALV_EVENTS_TABLE*:

Handling the TOP_OF_PAGE Event – Defining a Method

```
*---------------------------------------------------------------------*
*       CLASS lcl_event_handler DEFINITION
*---------------------------------------------------------------------*
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS:
      on_top_of_page
        FOR EVENT top_of_page
        OF cl_salv_events_table
        IMPORTING
          r_top_of_page  " cl_salv_form; references for the t-o-p area
          page           " sypagno
          table_index.   " syindex; 1st row in current page
ENDCLASS.                    "lcl_event_handler DEFINITION
```

In this example, current data from the first data record of the new page is to be displayed in each case in the header line of the page. This can be useful when using control level changes.

Handling the TOP_OF_PAGE Event – Implementing a Method

```
  METHOD on_top_of_page.
    DATA: lv_header_text TYPE        c LENGTH 70.
    DATA: lo_content     TYPE REF TO cl_salv_form_element.
    DATA: lo_text        TYPE REF TO cl_salv_form_text.

* read data set #1 of current page
    READ TABLE gt_bookings INTO gs_booking INDEX table_index.
```


© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
    IF sy-subrc EQ 0.
* concatenate heading text
      CONCATENATE
        'Airline'(air) gs_booking-carrid
        'Connection'(con) gs_booking-connid
        INTO lv_header_text
        SEPARATED BY space.
* create text object (of class cl_salv_form_text)
      CREATE OBJECT lo_text
        EXPORTING
          text   = lv_header_text.

* up cast
* (cl_salv_form_element is superclass for cl_salv_form_text)
      lo_content = lo_text.

* set content of top-of-page-area
* ( go_top_of_page points to it -
*  we got hold of it via the event's interface )
      go_top_of_page->set_content( lo_content ).
    ENDIF.

  ENDMETHOD.                    "on_top_of_page
```

Finally, you need to register the method as a handler method.

Handling the TOP_OF_PAGE Event – Registering a Method

```
DATA: go_alv_table TYPE REF TO cl_salv_table.
DATA: go_event     TYPE REF TO cl_salv_events_table.

* get the EVENT object
* !!! Precondition: ALV object is bound !!!
  go_event = go_alv_table->get_event( ).

  SET HANDLER:
    lcl_event_handler=>on_top_of_page FOR go_event.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

(Reminder: The group change properties are set together with the sorts using the *SET_GROUP_ACTIVE* method of the *CL_SALV_SORTS* class and for the second using the *GROUP* parameter of the *ADD_SORT* method of the *CL_SALV_SORTS* class.)

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Exercise 32: ALV OM: Print Design and the Header and Footer

Exercise Objectives

After completing this exercise, you will be able to:

- Make settings for printing from the ALV
- Generate output in the header and footer area of the ALV

Business Example

You are working as a programmer for ABC Travel. Your manager wants the printout of the ALV contents to contain information about the chosen selections, sortings and filterings. A particular introduction is also to be output at the start of the ALV output (on screen and in the printout).

Task:

Display the selection and list information when printing the ALV. Generate a header area for the output.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_EVENT_S3*.

   Model Solution: *BC405_OM_OTHER_S1*

2. Display the selection and list information when printing the ALV. To do this, retrieve a corresponding object of type *CL_SALV_PRINT*. Apply the relevant methods to this object.

   Resolve the following requirements for each suitable method call:

   The additional information is to be output on a separate cover sheet. The standard title of the report is to be output on every page. (Important: To do this, you do **not** have to handle the *TOP_OF_PAGE* event.)

   To test the result, you need to choose the *Print* function. Specify the output device *LOCL*. For the *Time of Print*, specify *Send to SAP Spooler Only for Now*. Display the printout with transaction SP02.

3. Generate a header area for the output (on the screen and in the printout). Important: When using the “container” representation, this is not visible on the screen.

   Output a header that consists of two rows:

   1. Row: “Flight list group ##”
   2. Row: “Please check the utilization for the individual flights.”

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Solution 32: ALV OM: Print Design and the Header and Footer

Task:

Display the selection and list information when printing the ALV. Generate a header area for the output.

1. Extend your program *ZBC405_##_OM* or copy the program *BC405_OM_EVENT_S3*.

   Model Solution: *BC405_OM_OTHER_S1*

   a) Self-explanatory

2. Display the selection and list information when printing the ALV. To do this, retrieve a corresponding object of type *CL_SALV_PRINT*. Apply the relevant methods to this object.

   Resolve the following requirements for each suitable method call:

   The additional information is to be output on a separate cover sheet. The standard title of the report is to be output on every page. (Important: To do this, you do **not** have to handle the *TOP_OF_PAGE* event.)

   To test the result, you need to choose the *Print* function. Specify the output device *LOCL*. For the *Time of Print*, specify *Send to SAP Spooler Only for Now*. Display the printout with transaction SP02.

   a) See the complete answer at the end of the task.

3. Generate a header area for the output (on the screen and in the printout). Important: When using the "container" representation, this is not visible on the screen.

   Output a header that consists of two rows:

   1. Row: "Flight list group ##"
   2. Row: "Please check the utilization for the individual flights."

   a) See the complete answer at the end of the task.

Result

Main program

```
REPORT bc405_om_event_s2.

INCLUDE bc405_om_event_s2_top.
INCLUDE bc405_om_event_s2_k01.
```

Continued on next page

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

```
INCLUDE bc405_om_event_s2_e01.
INCLUDE bc405_om_event_s2_o01.
INCLUDE bc405_om_event_s2_i01.
INCLUDE bc405_om_event_s2_f01.
```

F01 Include

```
*&---------------------------------------------------------------------*
*&      Form  define_settings
**&---------------------------------------------------------------------*
FORM define_settings
  USING po_alv_table TYPE REF TO cl_salv_table.

  PERFORM: set_display    USING po_alv_table,
           set_columns    USING po_alv_table,
           set_sorts      USING po_alv_table,
           set_aggregs    USING po_alv_table,
           set_selections USING po_alv_table,
           set_layout     USING po_alv_table,
           set_functions  USING po_alv_table,
           set_events     USING po_alv_table,
           set_print      USING po_alv_table,
           set_header     USING po_alv_table.

ENDFORM.                    " define_settings

*&---------------------------------------------------------------------*
*&      Form  set_print
*&---------------------------------------------------------------------*
FORM set_print
  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA: lo_print TYPE REF TO cl_salv_print.

* get the PRINT object
  lo_print = po_alv_table->get_print( ).
* set selectioninfo on
  lo_print->set_selectioninfo_on( callback_program = sy-cprog  ).
* set listinfo on (sorts, filters)
  lo_print->set_listinfo_on( value = 'X' ).
* print these infos on a separated coverpage
  lo_print->set_coverpage_on( value = 'X' ).
  lo_print->set_report_standard_header_on( value = 'X' ).

ENDFORM.                    " set_print
```

Continued on next page

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

```
*&---------------------------------------------------------------------*
*&      Form  set_header
*&---------------------------------------------------------------------*
FORM set_header
  USING po_alv_table TYPE REF TO cl_salv_table.

  DATA: lo_layout_grid TYPE REF TO cl_salv_form_layout_grid.

* 1st step: create the grid object
  CREATE OBJECT lo_layout_grid.

* 2nd step: add some text items
  lo_layout_grid->create_header_information(
      row     = 1
      column  = 1
      text    = 'Flight List Grp ##'(fll)
         ).

  lo_layout_grid->create_text(
      row     = 2
      column  = 1
      text    = 'Please check utilization'(uti)
         ).

* 3rd step: declare grid as header of list
  po_alv_table->set_top_of_list( value = lo_layout_grid ).

ENDFORM.                    " set_header
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Lesson Summary

You should now be able to:

- Make settings for printing from the ALV
- Handle the events *TOP_OF_PAGE* and *END_OF_PAGE*
- Compile design objects and assign a header or footer area

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Unit Summary

You should now be able to:

- Create an ALV with the output formats “full screen” and “classic ABAP list”
- Display data in the ALV
- Display an ALV as a subarea of a dynpro
- Explain the class hierarchy of the ALV OM
- Describe the basic procedure to change properties of an ALV
- Set the title of an ALV
- Set other general display properties of the ALV
- Set display properties that apply to all output columns
- Define properties of individual columns (principle)
- Add an additional column to the ALV and format it (principle)
- Define a column title
- Define or change key columns
- Define or change a DDIC reference
- Trigger sorting, filtering, and aggregation in the ALV from the application
- Activate selection mode and make selections
- Work with layout variants
- Explain the concept of columns with special technical meanings
- Define the color of columns, rows, and cells
- Define text properties
- Control the display of numerical values
- Make generic ALV functions available with the desired scope
- Add customer functions to the ALV
- Handle ALV events such as the double-click or the call of a customer function
- Display interactive elements in the ALV output
- Make settings for printing from the ALV
- Handle the events *TOP_OF_PAGE* and *END_OF_PAGE*
- Compile design objects and assign a header or footer area

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Test Your Knowledge

1. Which statements are correct?
   Choose the correct answer(s).
   - □ A To change any property of the ALV, it is always sufficient to change only the ALV main object.
   - □ B Subobjects of the ALV are fetched with GET methods.
   - □ C Subobjects of the ALV are generated with SET methods.
   - □ D Subobjects of the ALV are generated with CREATE OBJECT methods.
   - □ E There is a maximum of subobjects of the first level for the ALV main object.

2. What is the name of the main class for implementing an ALV for simple, two-dimensional tables?

   _____
   _____
   _____
   _____

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Answers

1. Which statements are correct?

   Answer: B

   To change any property of the ALV, you always need to work with subobjects of the ALV main object. Subobjects of the ALV are fetched with GET methods. They are not generated with SET methods, nor with CREATE OBJECT. There is a maximum of three subobject levels, that is in the most extreme case, there can be an ALV main object, a subobject of the first level for it, a subobject of the second level for it, and then finally a subobject of the third level.

2. What is the name of the main class for implementing an ALV for simple, two-dimensional tables?

   Answer: *CL_SALV_TABLE*

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Internal Use SAP Partner Only
Internal Use SAP Partner Only

Course Summary

You should now be able to:

- Design a selection screen with complex screen elements
- Create an SAP List Viewer (ALV)
- Define the initial layout of an ALV
- Define additional columns for an ALV
- Work with events and methods to extend the functions of an ALV
- Read data from the database using logical databases
- Read data from the database using your own OPEN SQL accesses
- Call executable programs from an ALV

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Appendix 1

Detail Lists

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Since the introduction of the ALV, the classic list (generated with `WRITE`) is actually obsolete. The following information is therefore only intended for when you have to maintain old programs that still use this technique.

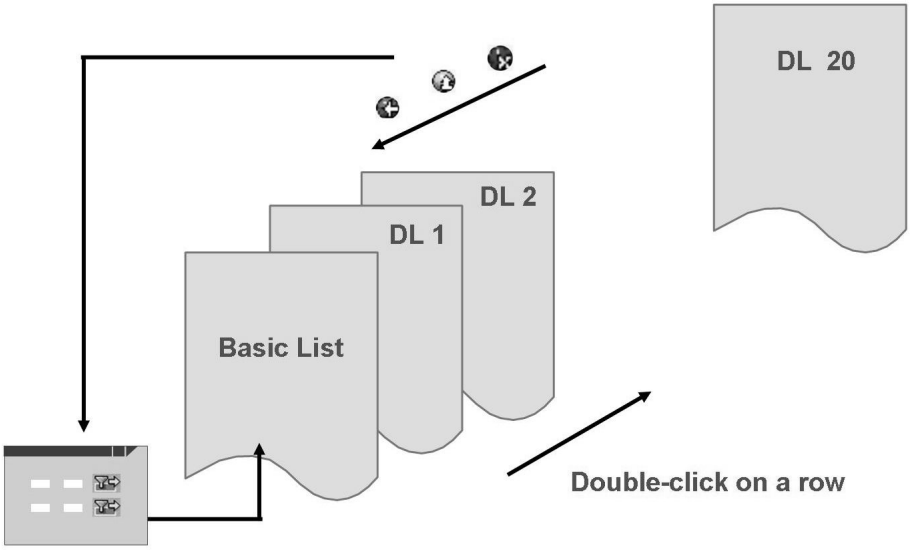


Figure 217: The List System

For each basic list up to 20 parallel detail lists can exist. Each list has its own individual memory area called a list buffer.

The same functions are available in detail lists as are available in basic lists.

Choose ***Back*** (the green arrow) or ***Cancel*** (the red cross) to leave the current list. This releases the corresponding list buffer and returns to the preceding list.

Use the ***Exit*** function (yellow arrow) to return to the selection screen.

© 2011 SAP AG. All rights reserved.

```
*  Basic list

START-OF-SELECTION.
  [                    ]
GET ... .
  [                    ]
END-OF-SELECTION.
  [                    ]
TOP-OF-PAGE.
  [                    ]

*     Details lists

AT LINE-SELECTION.
  [                    ]
AT USER-COMMAND.
  [                    ]
TOP-OF-PAGE DURING LINE-SELECTION.
  [                    ]
```

Figure 218: Events: Overview

The **START-OF-SELECTION, GET, END-OF-SELECTION** and **TOP-OF-PAGE** events are only available for generating the basic list. Once you leave a basic list, these events are no longer processed.

Detail lists are created using two basic events: **AT LINE-SELECTION** and **AT USER-COMMAND**. You use the **TOP-OF-PAGE DURING LINE-SELECTION** event to create headings for each of the detail lists. If the application displays multiple detail lists, use the `CASE` statement and system field *sy-lsind* to specify which detail list headings to display.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

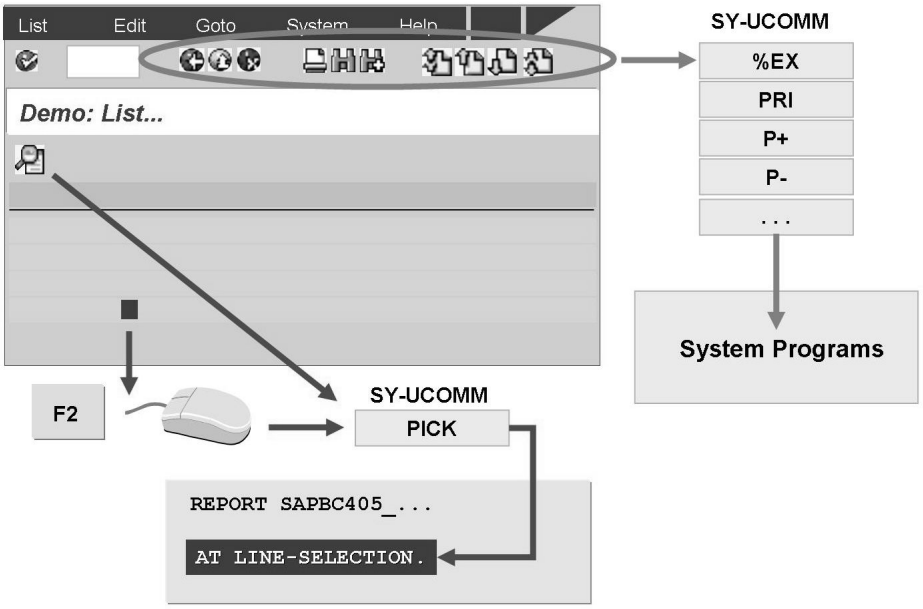


Figure 219: Program Flow in Detail Lists

When a user has triggered an action in a list, the function code defined for this action is triggered and copied to system field *sy-ucomm*.

If the *PICK* function is triggered (double-click or F2), the program is given control and interactive event `AT LINE-SELECTION` is processed. All other functions of the standard list status are processed by system programs.

If you want to offer the user your own functions, you have to maintain your own GUI status in the *Menu Painter* and set it in the program using `SET PF-STATUS <status>`. When such a function is triggered, the program is given control, and interactive event `AT USER-COMMAND` is processed.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

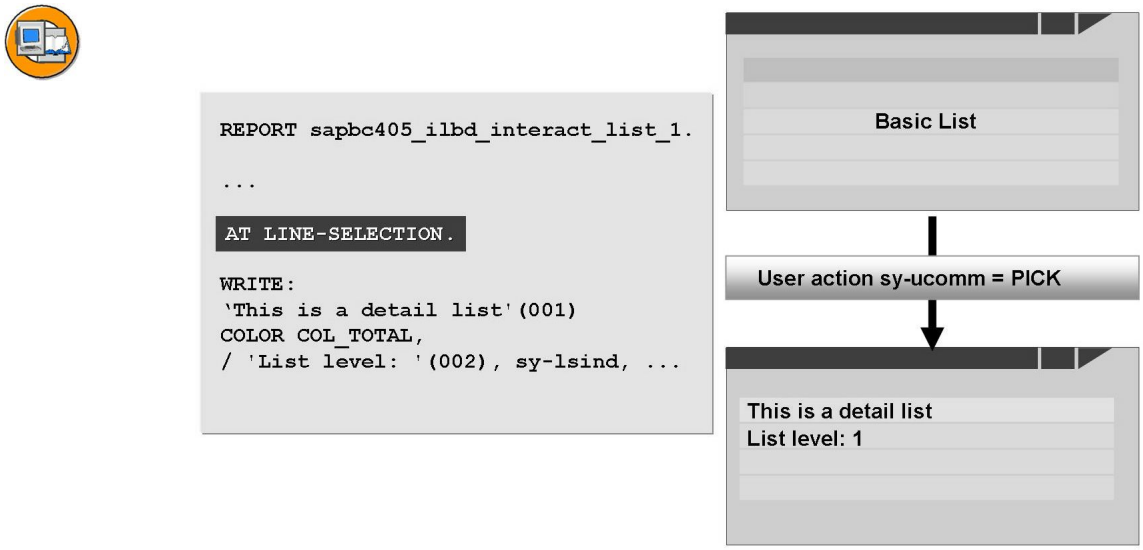


Figure 220: Creating Detail Lists I

If the *PICK* function is triggered, the program is given control and interactive event `AT LINE-SELECTION` is processed. All subsequent output (`WRITE, ULINE, SKIP`) is output in a detail list.

The system field *sy-lsind* indicates the current list level and is increased by one whenever event `AT LINE-SELECTION` or `AT USER-COMMAND` is processed.

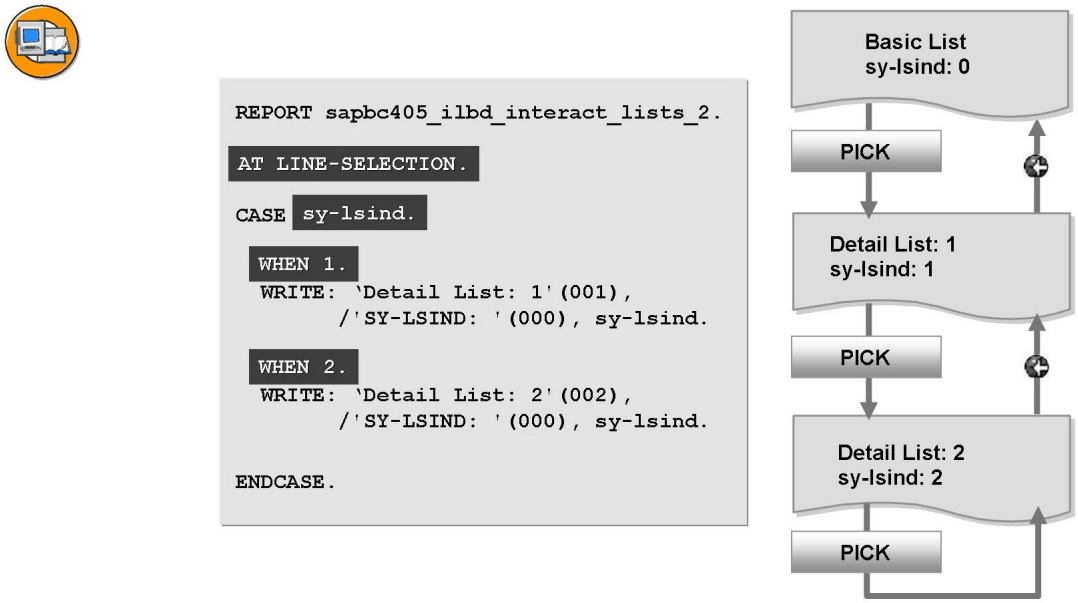


Figure 221: Creating Detail Lists II

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The system field *SY-LSIND* is always increased by one when events `AT USER-COMMAND` or `AT LINE-SELECTION` are triggered. You can determine the number of the new list buffer in a *CASE* query of the system field *SY-LSIND*.

You control the maximum number of displayable detail lists. In the example, no third list is generated: If the `AT LINE-SELECTION` event is triggered while viewing list 2, the value of SY-LSIND is increased to 3, but since no `WRITE`, `ULINE` or `SKIP` is executed, the system resets *SY-LSIND* to 2, and detail list remains visible.

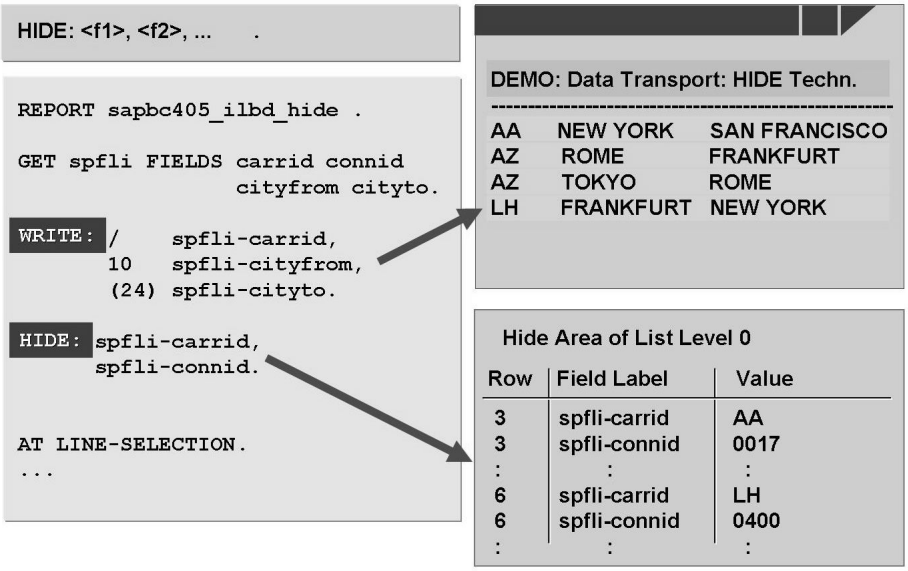


Figure 222: Hide Area

Use the `HIDE` statement to store global field names and their values so they are available when the user selects a list row. The runtime system automatically records the name and content of the data object in relation to its row position in the list currently being created. If a line is selected, the system places the hidden values for this row back into the corresponding fields in the program.

You normally use the `HIDE` statement to save key fields or other information that you will need in the next step.

The `HIDE` keyword must be programmed **immediately** after the last `WRITE` statement on the output line.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

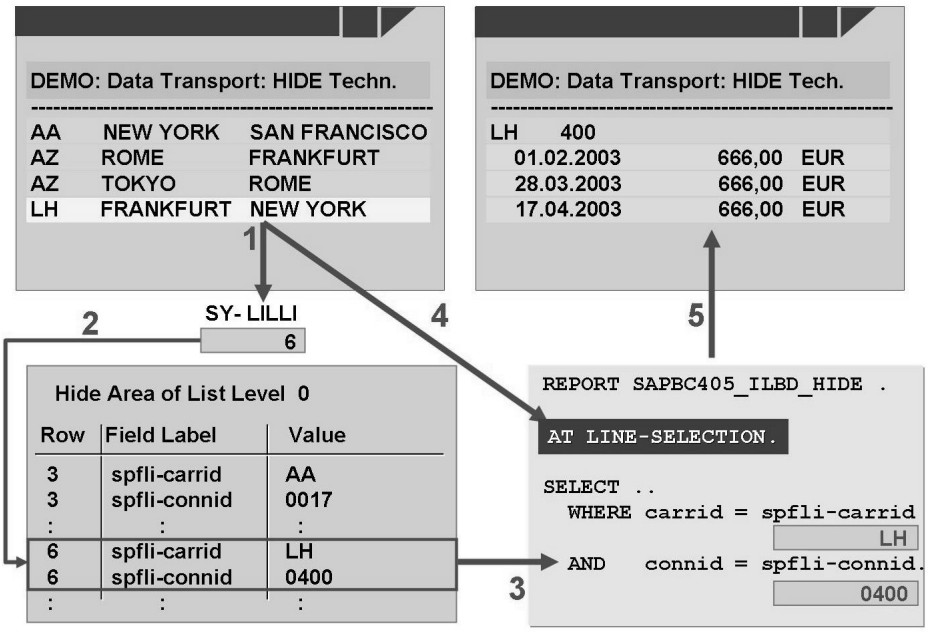


Figure 223: Row selection: The HIDE Technique

The individual steps of the above example are described below:

1. The user selects a row for which data has been stored in the `HIDE` area. He or she does this by double-clicking a row. If the field or row was written using `HOTSPOT ON`, a clicking once has the same effect as a double-click. The runtime system evaluates the *SY-LILLI* field to determine which row was selected: in this case, row 6.
2. The runtime system jumps to the point in the `HIDE` area where data for this row is stored.
3. The runtime system then inserts all values stored for the selected row in the `HIDE` area into their corresponding fields.
4. The runtime system processes the event `AT LINE-SELECTION` and its corresponding program block.
5. A detail list is created.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

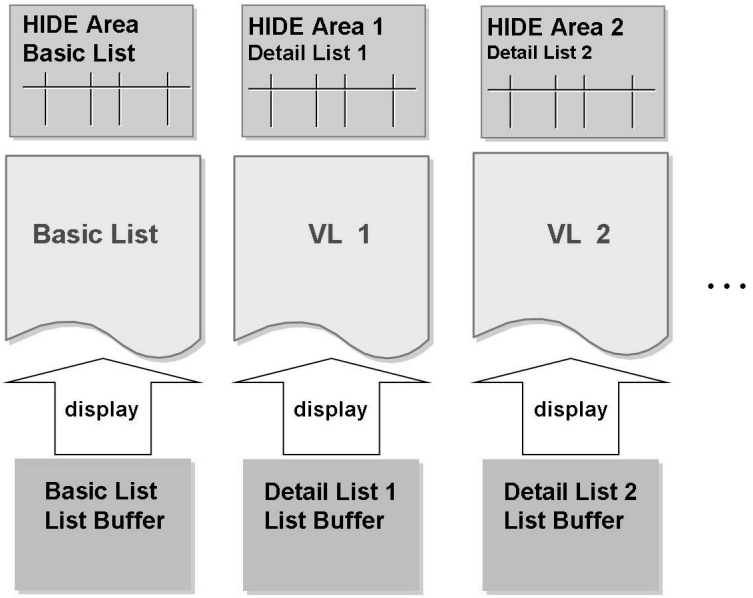


Figure 224: Buffering Detail Lists

All the output for a list is saved in a list buffer. As a result, each list remains saved in the system until the list buffer is deleted. All the detail lists whose list level is greater than the current list level are deleted automatically by the runtime system, that is, their list buffer is deleted. If, for example, you go from detail list 5 to detail list 3, the list buffers for detail lists 4 and 5 are deleted.

Each list level is assigned its own `HIDE` area.

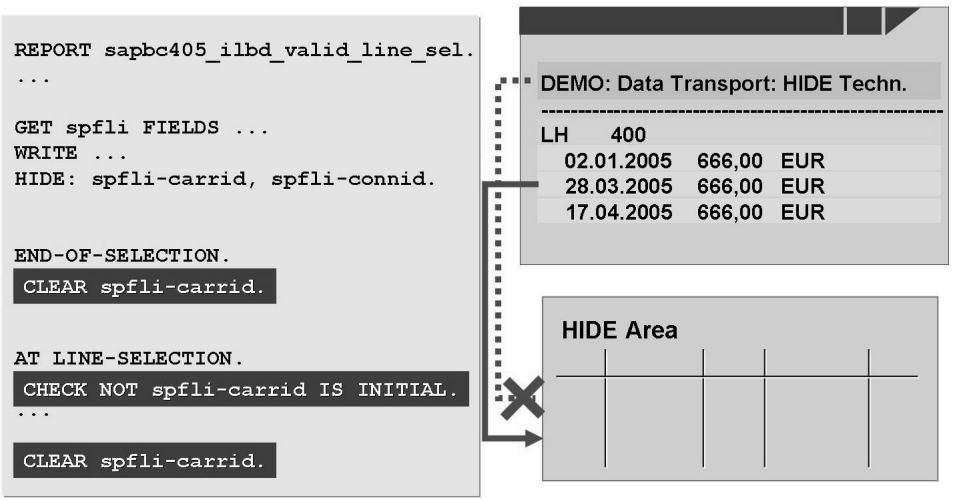


Figure 225: Selecting Valid Rows

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

You must ensure that when you select a row for which no data has been stored in the `HIDE` area, this is interpreted as an invalid row selection. Invalid lines include headers and footers.

When necessary, you can ensure that no detail lists are created for invalid row selections.

The field (*spfli-carrid*), whose contents have been stored in the `HIDE` area, is initialized in the programming example above. If a user selects an invalid row, the runtime system will not find a value in the `HIDE` area. The field retains its initial value. The `CHECK` statement then reads this initial value and the program exits the current processing block (`AT LINE-SELECTION`). No detail list is created, and the current list display is retained.

Other Techniques for Interactive Lists

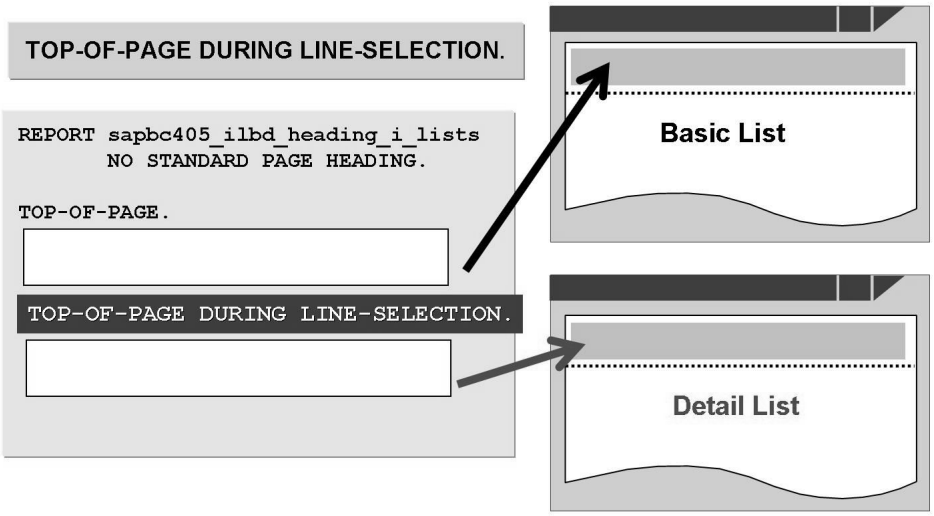


Figure 226: Page Headings for Detail Lists

The `TOP-OF-PAGE` event is only processed when you create a basic list.

You use the `TOP-OF-PAGE DURING LINE-SELECTION` event to create headings for each of the detail lists. Evaluate the system field *sy-lsind* to determine which heading you need to display.

`TOP-OF-PAGE DURING LINE-SELECTION` is triggered by the first output (such as `WRITE`) on each new list page.

`END-OF-PAGE` is processed for both the basic list and the detail lists. You therefore need to query *sy-lsind*.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

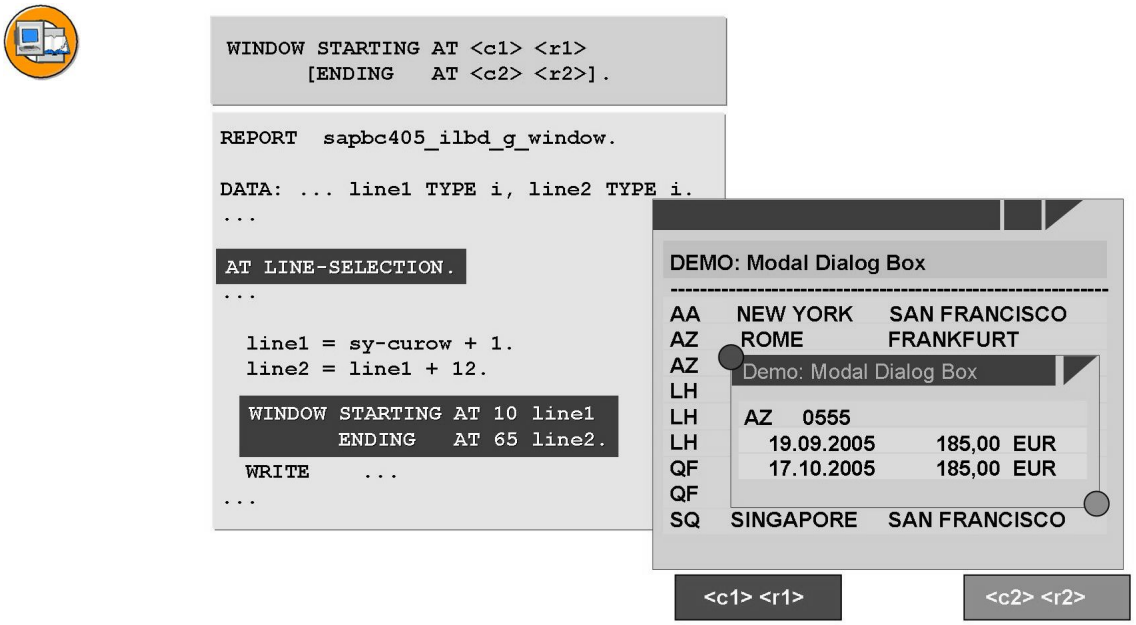


Figure 227: Creating a List in a Modal Dialog Box

You can display the current detail list in a modal dialog box with statement *WINDOW STARTING AT*.

The upper left corner of a modal dialog box is positioned at column <c1> row <r1>. The count begins at zero/zero in the upper left corner of the main window. If <r1> is less than or equal to zero, the list is displayed in full screen format instead of a modal dialog box.

You can use the optional qualifier `ENDING` to set the length and width of a modal dialog box: The lower-right corner is positioned at column <c2> row <r2>.

You can use `SET TITLEBAR <TITLE>` to set a title for the modal dialog box.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

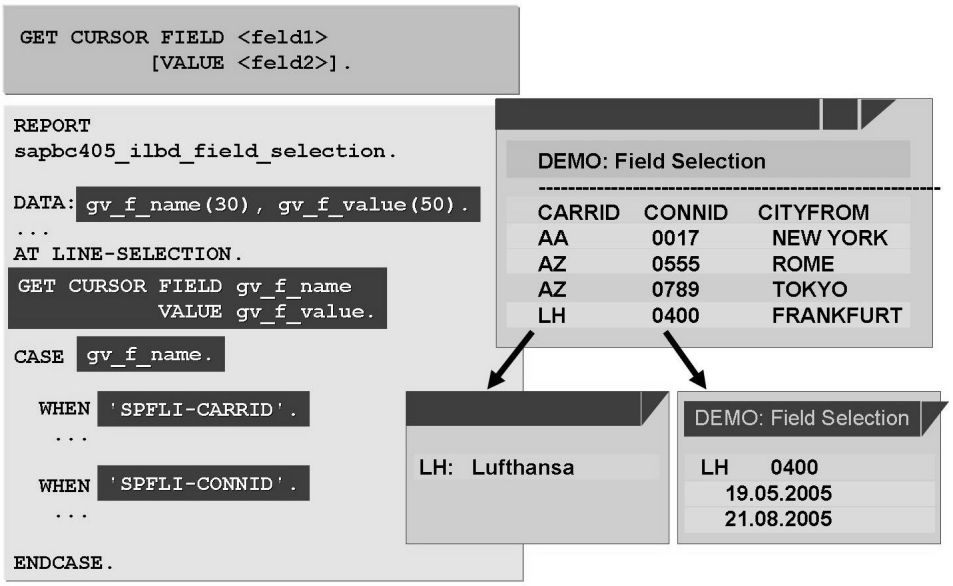


Figure 228: Field Selection

You can use the `GET CURSOR` statement to create detail lists according to the cursor position. If a user clicks one of the CARRID fields in the example above, the name of the corresponding airline is displayed. On the other hand, if a user clicks one of the CONNID fields, all flight dates for that particular flight number are displayed instead.

The parameter *FIELD* returns the name of an output field. The parameter *VALUE* returns the output value.

The return value SY-SUBRC is set to:

- Value 0: The cursor is positioned on a field
- Value 4: The cursor is not positioned on a field

Hint: Do not use the value of parameter *VALUE* in the *WHERE* condition of a *SELECT* statement, as this can lead to type conversions for non-character fields that produce undesirable results. It is better to use the HIDE technique in this case.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

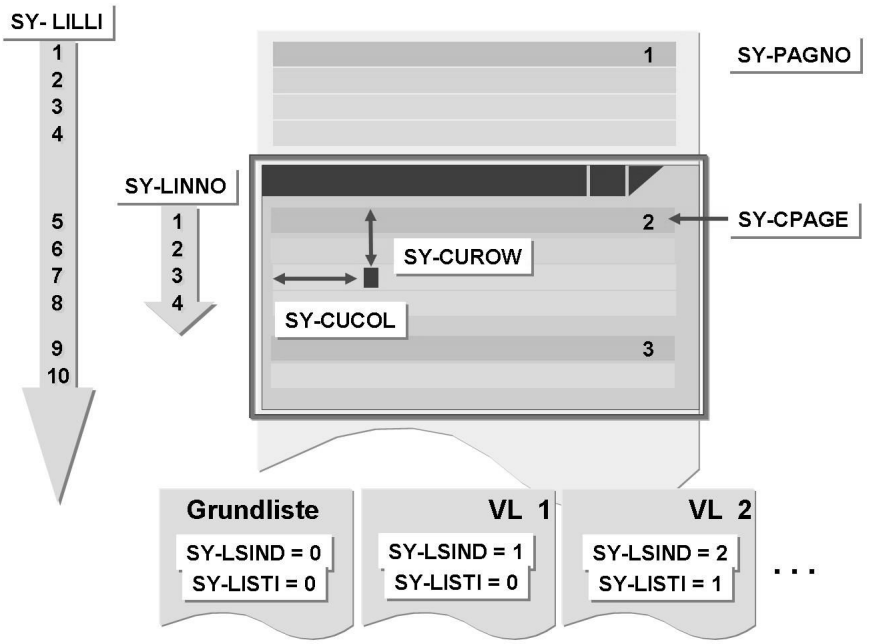


Figure 229: System Fields for Interactive Lists

The following system fields are automatically supplied with values at each interactive event:

System Fields for Interactive Reporting

SY-LSIND	Index of the current list
SY-LISTI	Index of the last list displayed
SY- LILLI	Absolute number of a selected row in the displayed list
SY-CPAGE	Page number of the top displayed page in the displayed list
SY-CUCOL	Number of the column in the window where cursor was last positioned in the displayed list
SY-CUROW	Row number in the window where cursor was last positioned in the displayed list
SY-STACO	Number of the first displayed column in the displayed list
SY-STARO	Number of the first visible row in the top displayed page (SY-CPAGE) in the displayed list (not including header lines)
SY-UCOMM	Function code that triggered the interactive event in the displayed list
SY-PFKEY	Status of the displayed list

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Appendix 2

Customer GUI Status for Interactive Lists

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Interface: Introduction

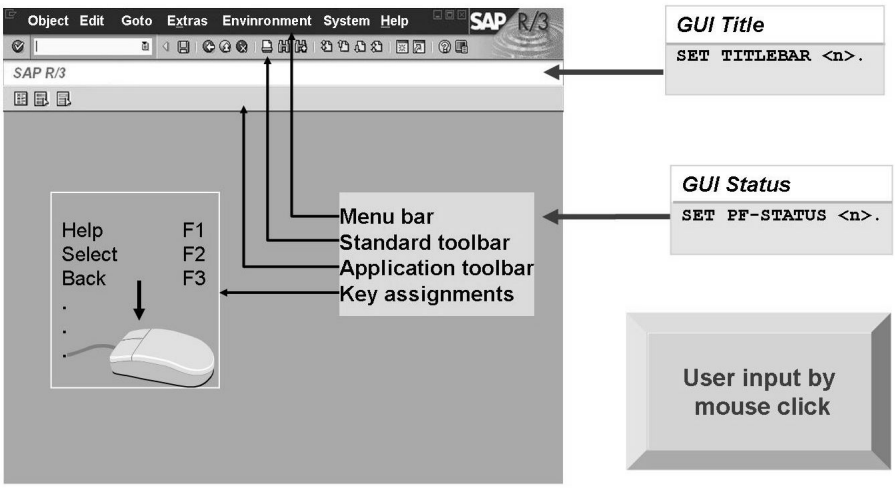


Figure 230: Overview: User interface

A GUI status consists of a link to menu bar, a standard toolbar, an application toolbar, and of function key settings. There can be different statuses for a screen. An editor program can have two statuses, for example: one for display mode and one for change mode.

GUI status elements allow the user to determine which actions a program performs by using the mouse.

Menus are control elements that allow the user to choose which functions will be processed by an application program. Menus can also contain submenus. The menus “System” and “Help” are displayed on every screen in the SAP system. They always contain the same functions and cannot be changed or suppressed.

© 2011 SAP AG. All rights reserved.

The standard toolbar contains icons for frequently used functions. The standard toolbar contains the same icons on all screens of the SAP system, with a fixed assignment of function keys. The icons for functions that cannot be executed on the current screen are displayed in gray.

The application toolbar can be used to execute frequently used function keys of a screen with a mouse click.

A function key assignment defines the function keys that are assigned to certain user interface functions such as *Replace, Find*, or *Cut*.

All of a program's GUI titles and statuses taken together make up its user interface. You need to regenerate the interface when you change or add a new title or status.

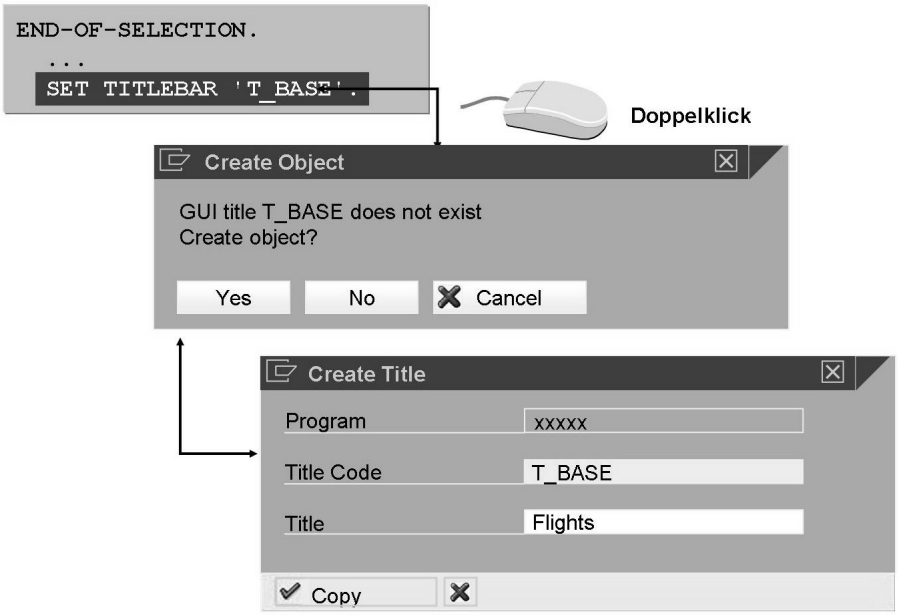


Figure 231: GUI Title

Create a status either from the object list of the *Object Navigator*, directly using the *Menu Painter*, or by forward navigation from the *ABAP Editor*.

The title code can have a maximum of 20 characters.

The title can have a maximum of 70 characters.

You should set a descriptive title for every screen of your application.

If you want to use variables in a title, you need to use the ampersand (&) as a placeholder. The & is then replaced at runtime by a value that you specify. You can use up to nine variables by placing digits after the & character. In this case, use the following statement to set the title:

```
SET TITLEBAR <title_name> WITH <&1> ... <&9>.
```

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Title bars are displayed until they are explicitly changed. At runtime, the current title is in the system variable *sy-title*. Title bars are also known as GUI titles.

GUI titles are created in the *Menu Painter* and can be translated from there. If there is a translation, the respective title is displayed in the logon language. If the runtime system does not find a translation in the logon language, the titles are displayed in the original language of the program.

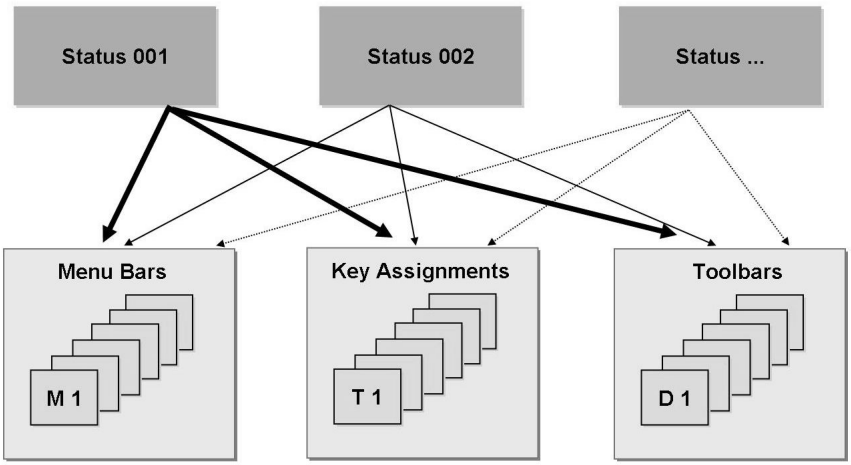


Figure 232: Status: Technical View (1)

From a technical point of view, a status is a **reference** to a menu bar, to a key setting, and to an application toolbar.

Different GUI statuses can work with the same subobjects, such as a menu bar.

GUI statuses are ABAP program objects that can be displayed in lists and on selection screens.

You should define a status for each screen.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

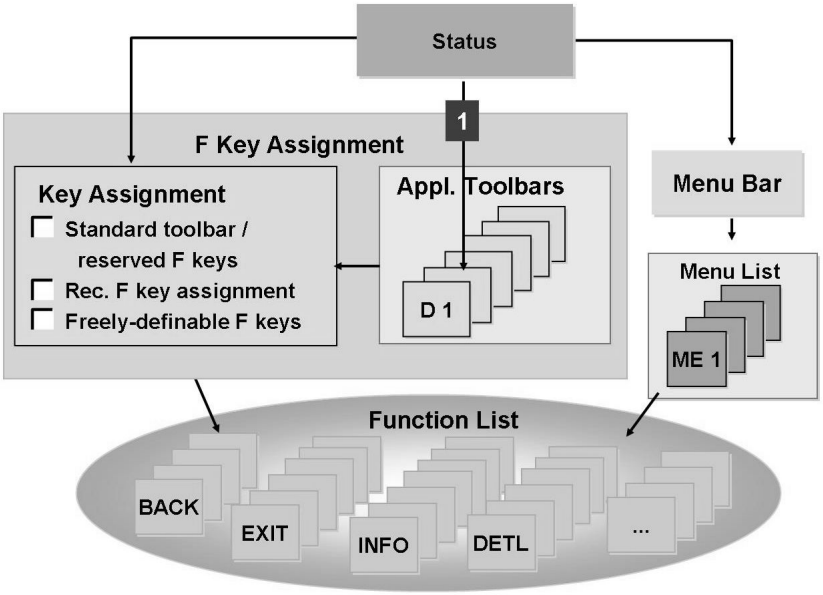


Figure 233: Status: Technical View (2)

A menu bar is made up of individual menus.

Key settings and application toolbars are sub-objects of the function key settings.

Create a set of application toolbars for an individual key setting. You need to assign a function to a function key before you can assign a button. Every status contains a single application toolbar.

All menus and key assignments of a program refer to the set of all interface functions (function list). This list can be reached using F4 help. Application toolbars refer to the functions indirectly by way of the key assignment.

Functions can be active or inactive for each status.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

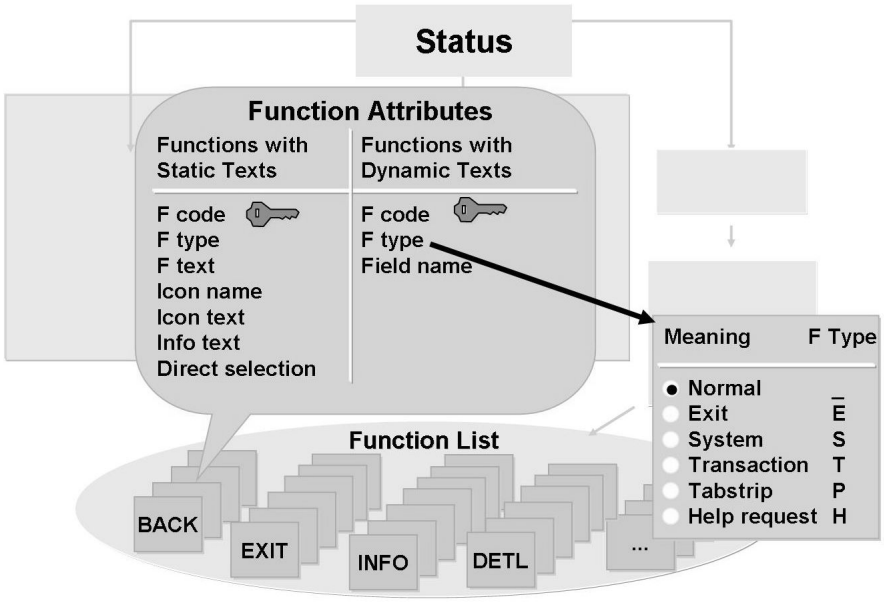


Figure 234: Functions

Functions are identified using their *function codes* and can be assigned to function keys, buttons, and menu entries.

The *function type* attribute determines the intended purpose of a function. You can use the function types " " (space), E, and P for buttons and for tab titles. The function types S and H are used internally by SAP. The function type "T" indicates a transaction code. When the user chooses a function of this type, the system exits the calling program and calls the new program.

If a function has a static text, you can assign an icon to it using the *icon name*. When the function is placed in the application toolbar, this icon is displayed. If you use an icon, the *function text* is displayed as a tool tip. *Info text* is displayed in the status bar when you choose the pushbutton. If you specify an *icon text*, both the icon and the text are displayed on the button.

In the case of functions with dynamic text, you define the text using the Dictionary structure SMP_DYNTXT.

However, the menu displays only the *function text*.

You can use the *fastpath* attribute to specify the letters that allow you to choose a function without using the mouse.

The menu path *Utilities → Help Texts → Standards/proposals* shows a list of suggested function keys. Double-clicking the individual texts displays the function keys and their use.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

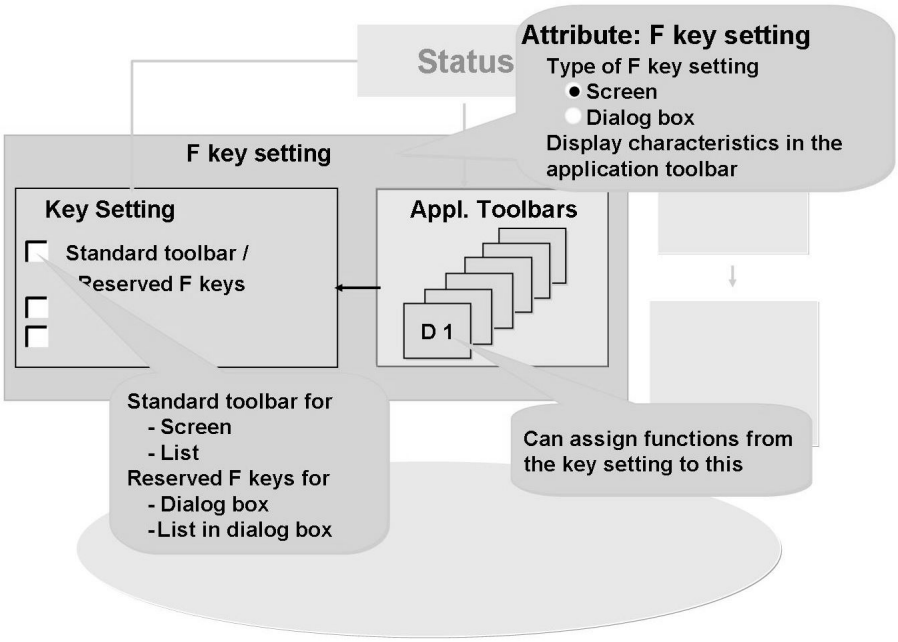


Figure 235: Function Key Settings

Function key settings consist of key assignments and buttons o the application toolbar.

A function key assignment's type (possible values: screen or dialog box) only serves to define the technical use of the function key assignment. You can also set options for the implementation of context menus and input help for lists.

Key assignments consist of *Reserved Functions Keys, Recommended Function Keys* and *Freely Assigned Function Keys*. *Reserved Function Keys* are function keys whose values cannot be changed in the SAP system. You can allocate functions to these; the assignment to the icons and texts cannot be changed. The allocated reserved function keys are displayed on screens and lists in the standard toolbar. *Recommended function keys* contain assignment proposals that satisfy SAP's ergonomic standards.

Functions that have been assigned to function keys can also be assigned to buttons in the application toolbar.

Functions that have been assigned buttons in the toolbar must be assigned function keys.

Buttons from the application toolbar allow easy access to the most frequently used functions of an application. The buttons should be a subset of the functions in the menu.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

An application toolbar can contain up to 35 buttons. You can insert vertical separators in the application toolbar for optical partitioning. You can control the display of deactivated functions in the application toolbar. To do this, choose *Goto → Attributes → Pushbutton Assignment.*

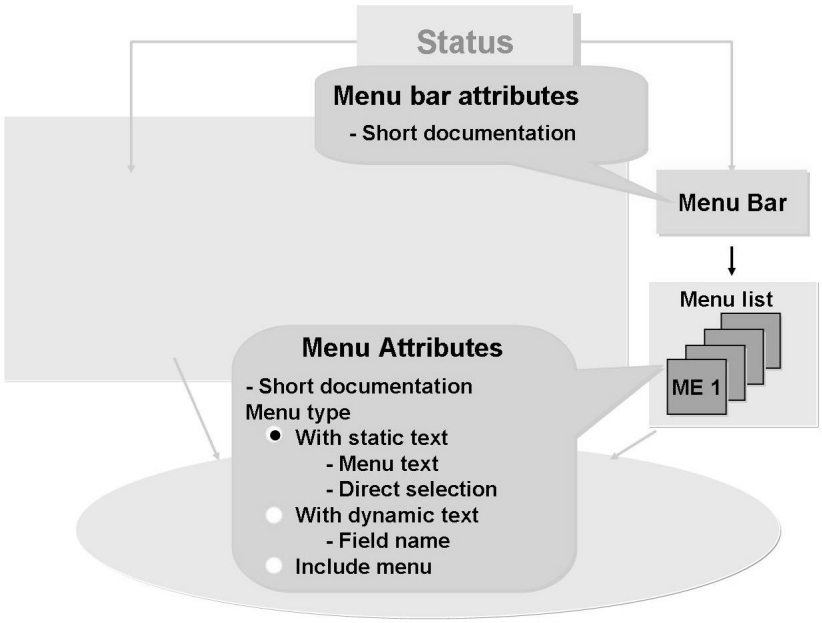


Figure 236: Menus and Menu Bars

A menu can contain up to 15 entries.

Possible entries are functions, separators, and menus (cascading menus).

The nesting depth for menus is restricted to three levels. The last level can only contain functions and separators.

Menus can be created with static or dynamic text. If you want to use dynamic text, you need to assign a field to the menu by choosing *Edit → Insert → Menu with dyn. text*. The system prompts you to enter a field name. Create a 20 character character field in the ABAP program. The content of this field is then displayed as the menu text.

The menu type *Include menu* allows you to reference menus from other programs. To do this, you need to specify not only the appropriate *short documentation*, but also the name of the program whose menu is being referenced and the name of the status that displays the menu as well.

Include menus can only be referenced in the menu bar.

A menu bar can contain up to eight different menus. Up to six of these can be freely assigned. The system automatically adds both the “System” menu and the “Help” menu to every menu bar.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Interface: Using a GUI Status

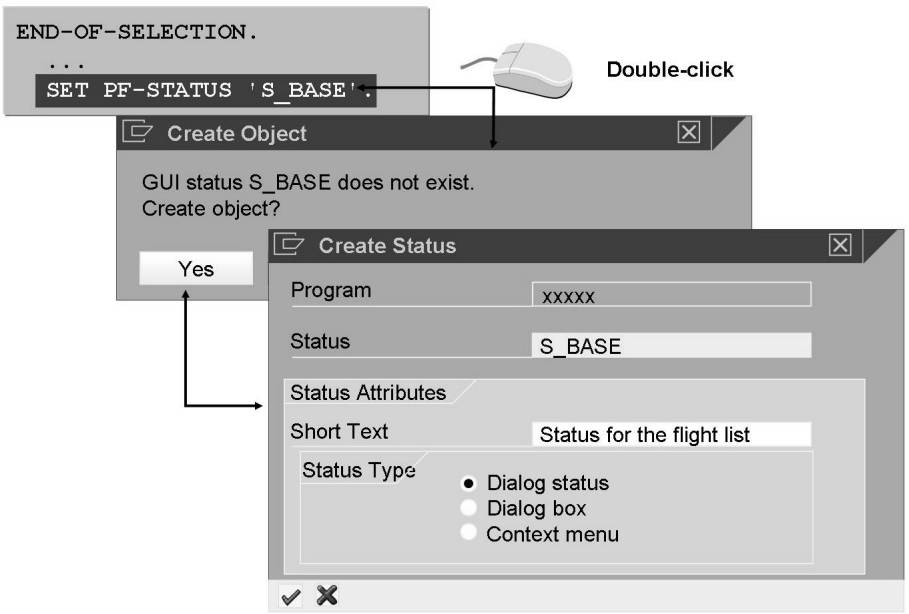


Figure 237: Creating a GUI Status

`SET PF-STATUS "'XXX'" [OF PROGRAM YYY]`. XXX is the status name and YYY is the program name.

`SET PF-STATUS " "`: This will revert back to the standard list status.

You can create or maintain a status either from the object list of the *Object Navigator*, directly using the *Menu Painter*, or by forward navigation.

The status type refers to the technical properties of a status. You can choose here between online status (status for full screen mode) and a dialog box (status for modal dialog box). Context menus are special collections of functions that can be displayed using the right mouse button.

You can create a new status by creating connections to existing components or creating a new "empty" status. You can also combine these two techniques. To create a completely new status, first create it and then add menu bars and menu functions, and so on. The status is then complete. Changes to this status apply only to this status.

When you use the reference technique, you create menu bars, application toolbars, and function key assignments in a status. You then create another status and refer to any menu bar, application toolbar, and function key assignment you want. The *Menu Painter* maintains these references so that any changes to a menu bar, application toolbar, or function key assignments automatically take effect in all statuses referring to them.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

In particular with very large applications, you can use references to ensure good consistency. By using the references, you ensure that certain functions are consistent across statuses.

If no GUI interface has been set for list processing, a standard user interface is displayed. You can explicitly set the standard list status with SET PF-STATUS space. You can deactivate functions at runtime with the addition EXCLUDING <fcode>. If you want to deactivate several function codes at the same time, you must first transfer these to an internal table.

```
DATA: gt_fcodes LIKE TABLE OF sy-ucomm
APPEND 'SAVE' TO gt_fcodes.
APPEND 'CANCEL' TO gt_fcodes.
SET PF-STATUS 'LIST' EXCLUDING gt_fcodes.
```

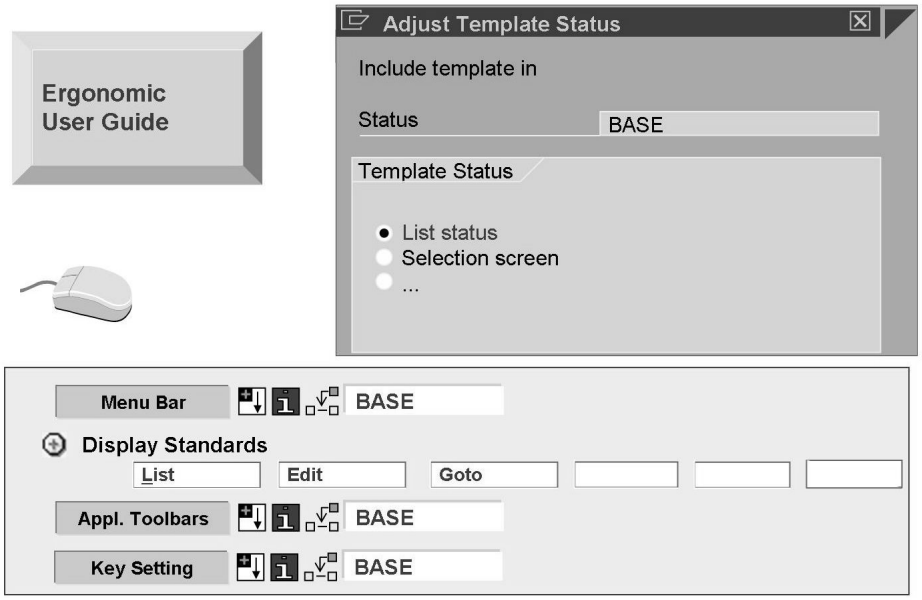


Figure 238: Adjusting the Status

You can use the function *Include Template* from the *Extras* menu to add standardized function codes to your own status and, in this way, you can mix objects from any status into the current status. In particular, you can select the standards for list status or selection screens or use any status from another ABAP program.

If you want to, the system can also propose standards for the menu bar, which you can also modify.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

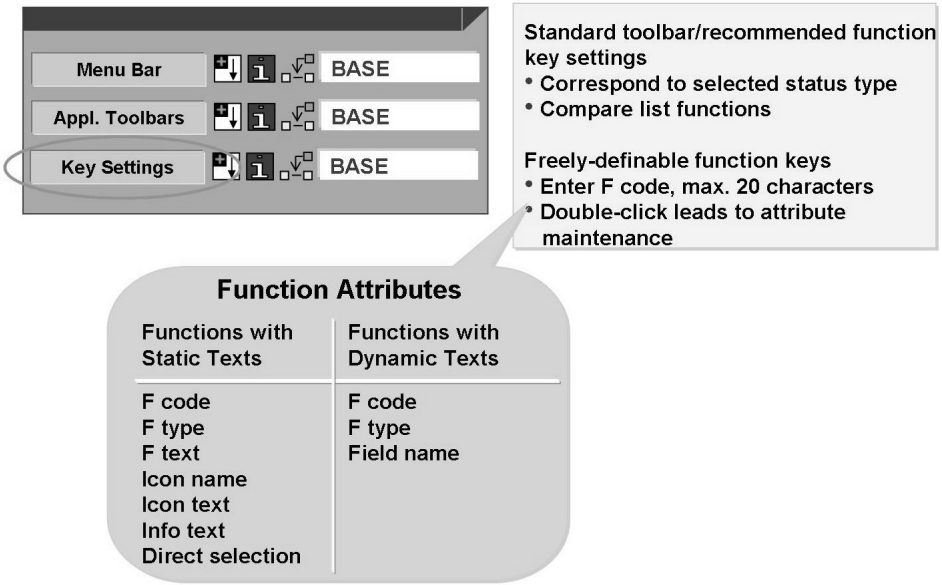


Figure 239: Creating a GUI Status: Function Key Settings

Function key settings consist of a key assignment and a set of application toolbars. Functions are assigned to function keys. You can also assign buttons in the application toolbar to the most commonly used functions.

A function key assignment's type (possible values: screen, dialog box, list, list in a dialog box) determines where the key can be used.

You can connect functions with reserved function keys, recommended function keys, and freely-definable function keys. Ensure that you meet SAP's ergonomics standards, which you can find in the *Environment* menu of the *Menu Painter*.

The allocated reserved function keys are displayed on screens and lists in the standard toolbar.

If a function is important and you have already assigned a function key, you can also assign a button in the application toolbar to it. An application toolbar can contain up to 35 buttons.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Symbol	Function Key	Meaning
	Enter	
	Control-S	Save
	F3	Back
	Shift-F3	Exit (program)
	F12	Cancel (screen)
	Control-P	Print
	Control-F	Find
	Control-G	Find again
	Control-Page Up	First page
	Page Up	Previous page
	Page Down	Next page
	Control-Page Down	Last page
	F1	Help

Figure 240: Standard toolbar: automatic assignment

The function codes that you assign to the standard toolbar are automatically assigned their reserved function keys.

To find out which assignments are made for the functions of the current status, choose the information icon in the *Menu Painter*.

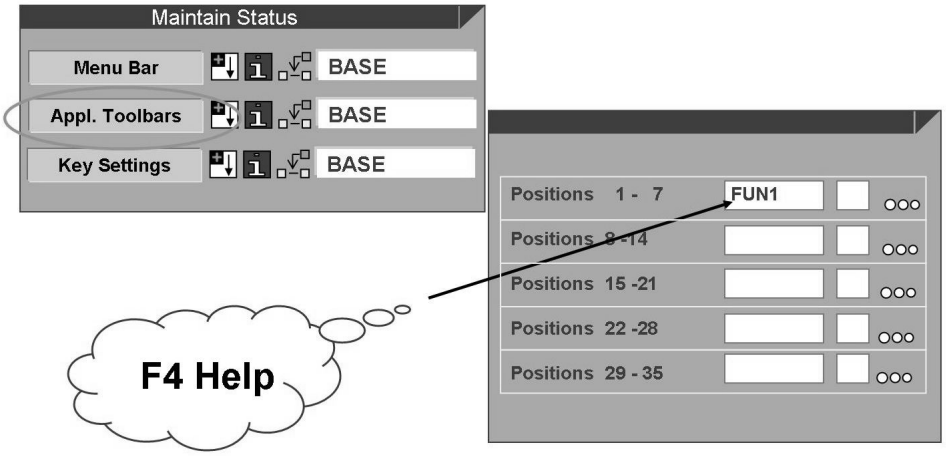


Figure 241: Creating a GUI Status: Application Toolbar

If you create a button in the application toolbar, you also need to assign the function to a function key.

Use the F4 help to select the function.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

If you assign an icon (icon name) to a function with static text, the application toolbar displays the icon instead of the static text. The function text associated with the function is used as the “quick info text”. The contents of the *Info. text* attribute appear in a screen’s status line whenever the function is triggered. If you want to display additional text with an icon, it should be entered in the *Icon text* attribute.

You can use the *Insert* menu of the *Menu Painter* to insert separator lines into the toolbar.

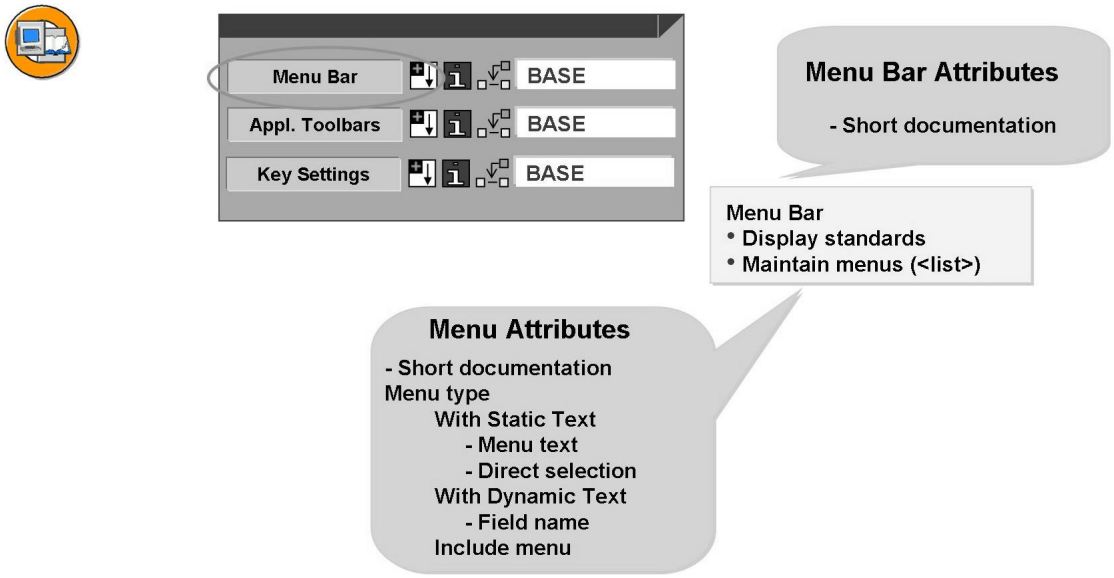


Figure 242: Creating a GUI Status: Menu Bar

Possible menu entries are functions, separator lines, and other menus.

You enter functions in a menu by writing the function code in the corresponding input field on the left. If appropriate, the function text is then taken from the function list. Otherwise, after double-clicking, you can enter a text in the input field on the right.

You can add separator lines using the *Insert* menu,or by filling the corresponding field on the right with a minus sign.

You can obtain submenus simply by entering the name in the field on the right of the menu entry, without entering a function code. By double-clicking he name, you can expand the submenu. You can create two submenus for every menu entry. Every menu bar and every submenu can contain a total of 15 menu elements.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

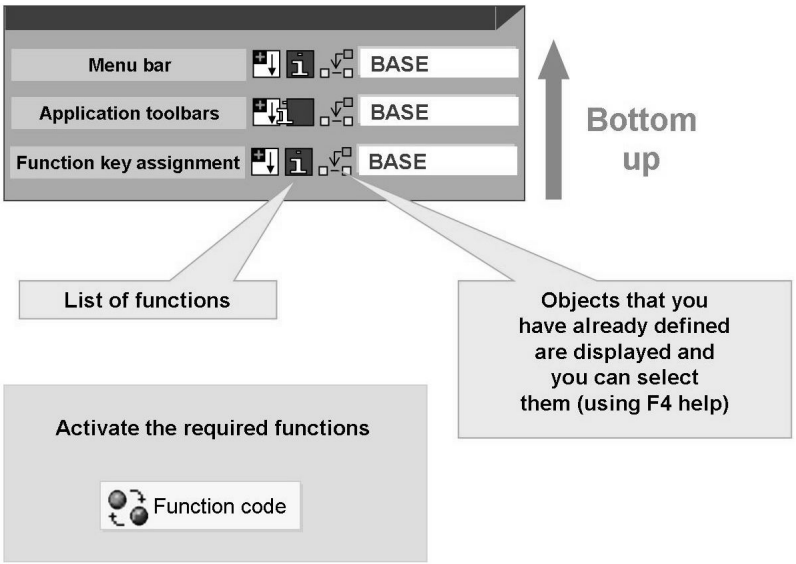


Figure 243: Including Existing Elements

If you are using the *Menu Painter* for a status, you can use existing button assignments, application toolbars, and menu bars. When doing so, you should work from the bottom up. If multiple application toolbars are assigned for your key assignment, you can select the appropriate ones.

Initially, all functions are inactive. You can therefore activate only the desired functions in each case.

If you create a new function, you can choose whether all statuses that refer to the same objects are also to be changed. The new functions are then initially inactive.

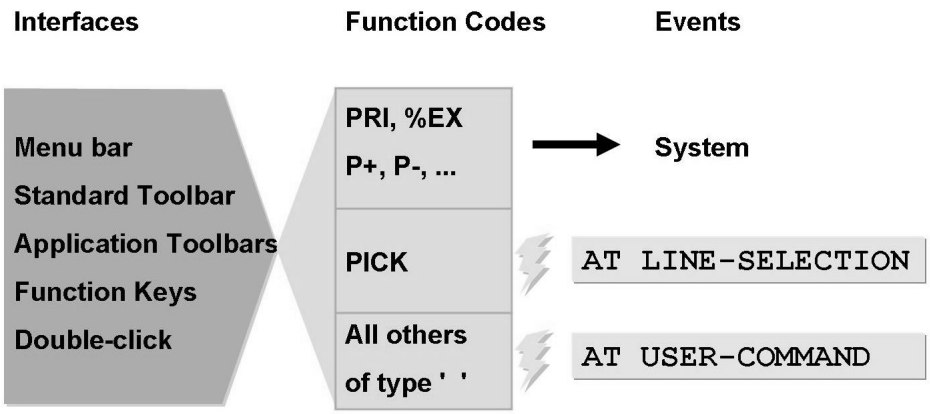


Figure 244: Flow Control in Detail Lists

After a user chooses an action from a list, it triggers that action's function code. This function code, in turn, triggers a corresponding event.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

The system function codes are reserved and do not trigger any interactive event blocks. The system does not go to the program, but rather runs the corresponding system function.

All function codes, with the exception of “PICK”, as well as all codes reserved for system use, such as “PRI” and “%EX” trigger the event AT USER-COMMAND.

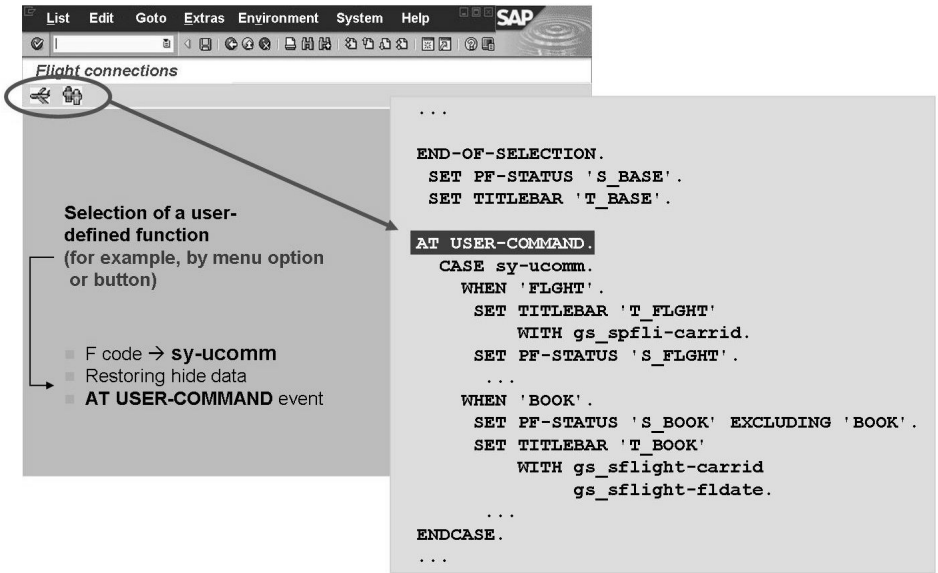


Figure 245: Event AT USER-COMMAND

You program *AT USER-COMMAND* as an interactive event and query the system field *sy-ucomm* in a CASE control structure. The current function code is entered in this field.

Data corresponding to the cursor position is read from the *HIDE* area to the associated global data fields.

Internal Use SAP Partner Only
Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Index

Numerics/Symbols

->, 196
=>, 196
?=, 430

A

Accessibility, 5–6
Aggregation, 450
ALV, 5
ALV Object Model, 75
Application Toolbar, 93
AT SELECTION-SCREEN ON VALUE-REQUEST, 456
AT SELECTION-SCREEN OUTPUT, 28
Average, 450

B

Background processing, 346
Batch processing, 346
BC405_LOGDB_D2, 257
Buffering, 301

C

CALL METHOD, 363
CALL SCREEN, 89
CALL SELECTION-SCREEN, 31
CALL TRANSACTION, 328
Cast, 430
Cell Colors, 472–473
Checkboxes, 23, 476, 479
cl_gui_custom_container, 377
CL_SALV_TABLE, 74, 359
CLASS-METHODS, 195
Cluster tables, 297
Colors, 475
Column color, 171
Column position, 171
Column width, 484
Columns, 428, 465
CONSTRUCTOR, 107, 381
Container, 87
Container Control, 377
Context menu, 234
Control Framework, 85, 188
Control Level Change, 449
Controls, 85
Counter column, 466
CREATE OBJECT, 107, 381, 383
Currencies, 483
Custom Container control, 91

D

Data element, 22
Database views, 284
Decimals, 482
Define job, 347
Detail Lists, 561
Dialog box, modal, 31
Double-click, 193, 509
Downcast, 430
Dynamic selections, 259, 261

E

END-OF-SELECTION, 257–258
Event handler method, 196
Events, 190, 504

F

F1 Help, 47
Factory Method, 359, 386
Field catalog, 161
Filter, 449
Flow logic, 88
Flow Logic, 92
Footer, 541

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

FOR ALL ENTRIES, 287
Full Screen, 366
Function Key Setting, 93

G

GET, 256–257
GET LATE, 257
GUI status, 236
GUI Status, 93

H

Handler methods, 504
Header, 541
Hotspot, 194
Hyperlink, 476
Hyperlinks, 476

I

Icons, 172, 466
Includes, 10
Index, 301
InfoSet Query, 3
INITIALIZATION, 28–29, 59

J

Job overview, 347
Joins, 285

L

Layout Variant, 453
LEAVE TO SCREEN, 96
LEAVE TO TRANSACTION, 328
List buffer, 8
Logical databases, 251
LOOP AT SCREEN, 61

M

Modification groups, 61
Modules, 92
Multiple rows, 484

N

Next screen, 111
NODES, 256

P

PAI, 87
PARAMETERS, 22
PBO, 87
Performance, 284
Pooled tables, 297
Print list, 203
Pushbuttons, 59, 196, 476, 480

Q

Quick info text, 173
QuickViewer, 3

R

Radio buttons, 23
REJECT, 258
Required entry fields, 22
Row Colors, 475

S

SAP GUI, 85
SAP List Viewer, 5
SAP Memory, 22, 26
SAP Query, 3
Screen, 87
Screen number, 31
Screen Painter, 90
SELECT-OPTIONS, 23
Selection screen variants, 48
Selection view, 261
SELECTION-SCREEN, 29
Selections, 451
SET HANDLER, 196
SET PF STATUS, 94
SET/GET parameters, 22
Sort, 300
Sorting, 446
SQL, 283
SQL trace, 284
sscrfields, 59
Standard Toolbar, 93
START-OF-SELECTION, 429
STOP, 258
Subscreens, 32–33
Subtotals, 79
Symbol, 172
Symbols, 466

Internal Use SAP Partner Only
Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

T
Tab pages, 32
Tabstrips, 32
Title, 93
Toolbar, 196
Totals, 450
Traffic Light Icons, 469
Transport, 49
TVARVC, 50

V
Variants, 48

W
Work processes, 346
WRITE, 8

Internal Use SAP Partner Only

Internal Use SAP Partner Only

 © 2011 SAP AG. All rights reserved.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.

Feedback

SAP AG has made every effort in the preparation of this course to ensure the accuracy and completeness of the materials. If you have any corrections or suggestions for improvement, please record them in the appropriate place in the course evaluation.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

© 2011 SAP AG. All rights reserved.