

# BC410

## Developing User Dialogs

© SAP AG

- System: R/3
- System Requirements: SAP R/3, Basis Release 4.6C or higher
- Release 610
- 2002/Q3
- Material No.: 50054759

**Copyright 2002 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.**

© SAP AG 2002

## Notes on Trademarks:

- Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.
- 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 Server™ are registered trademarks of Informix Software Incorporated.
- UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.
- HTML, DHTML, XML, and 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 developed and implemented by Netscape.
- SAP, SAP Logo, R/2, RIVA, R/3, ABAP, 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 several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

### **Compulsory:**

- **SAPTEC - Basis Technology**
- **BC400 - ABAP Workbench Concepts and Tools**
- **BC401 - ABAP Objects**
- **BC430 - ABAP Dictionary**

### **Recommended:**

- **BC405 - Techniques of List Processing**
- **BC406 - Advanced Techniques of List Processing**

- **Participants:**  
**Programmers and Consultants**
- **Duration: Three days**



© SAP AG 2002

Notes to the user:

- The training materials are **not teach-yourself programs**. They **complement the course instructor's explanations**. There is space included on the page for you to write down additional information.

### Contents:

- **Course Goals**
- **Course Objectives**
- **Course Content**
- **Overview Diagram**
- **Main Business Scenario**

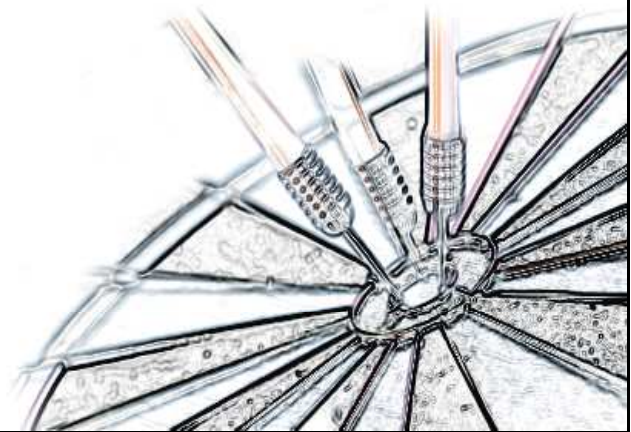


© SAP AG 1999

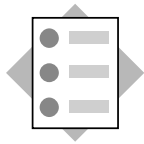


**This course will prepare you to:**

- **Program dynamic screen processing.**
- **Program user dialogs using the different screen elements in the SAP System.**



© SAP AG 2002



**At the conclusion of this course, you will be able to:**

- **Create a user interface for a program.**
- **Write user-friendly dialog programs.**
- **Use and process screen elements in the SAP System.**

---

<b>Unit 1</b>	<b>Course Overview</b>
Unit 2	<b>Introduction to Screen Programming</b>
Unit 3	<b>The Program Interface</b>
Unit 4	<b>Output Elements</b>
Unit 5	<b>Input/Output Elements</b>
Unit 6	<b>Subscreens and Tabstrips Controls Elements</b>
Unit 7	<b>Table Controls Elements</b>
Unit 8	<b>Context Menus</b>
Unit 9	<b>Lists on Screens</b>

---

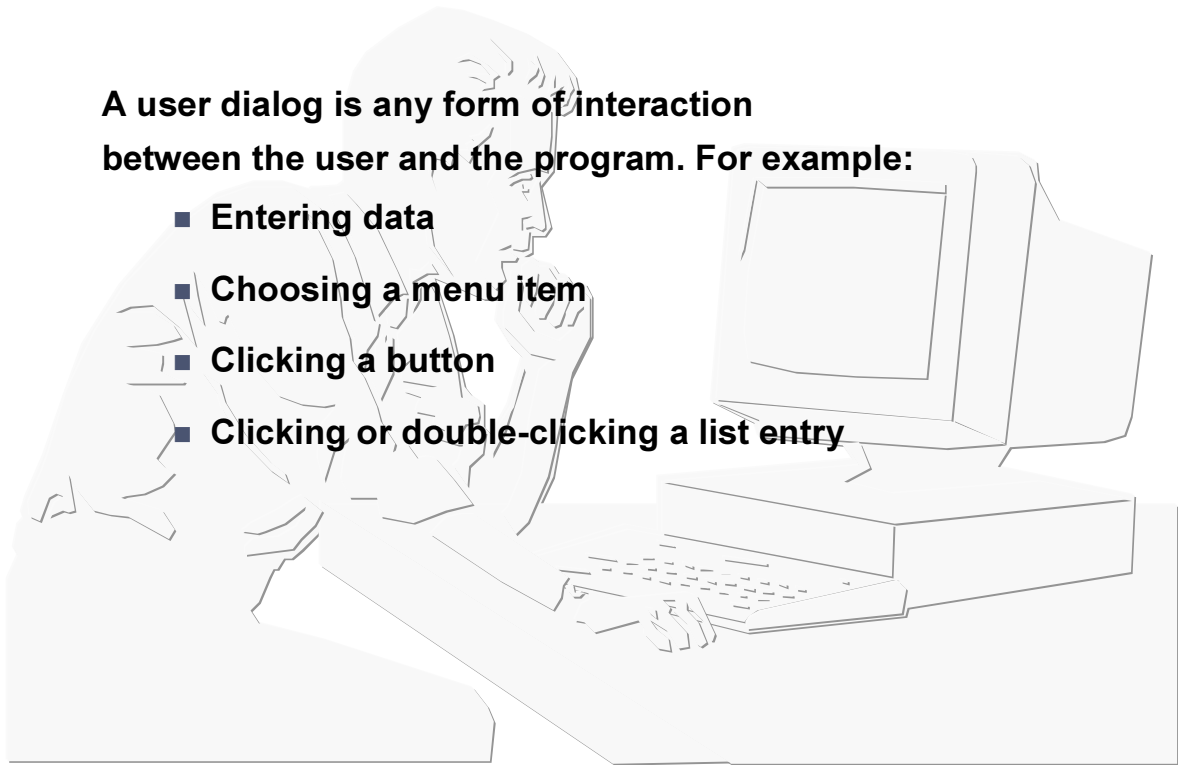
At the end of each unit are the relevant exercises and solutions.

## **Appendix**

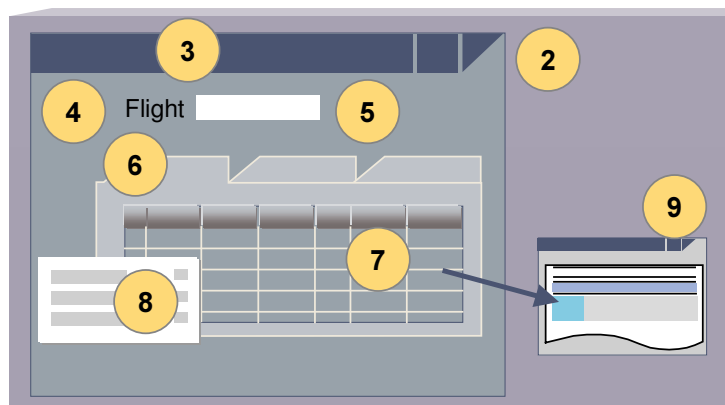
© SAP AG 2002

**A user dialog is any form of interaction between the user and the program. For example:**

- **Entering data**
- **Choosing a menu item**
- **Clicking a button**
- **Clicking or double-clicking a list entry**



© SAP AG 1999



© SAP AG 2002

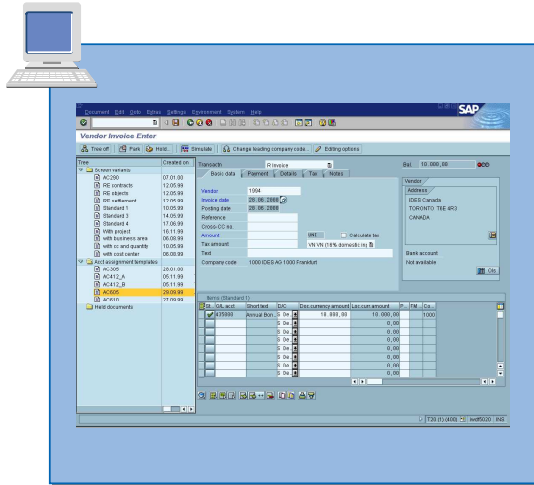
- |          |   |
|----------|---|
| ■ Unit 1 | Course Overview                                   |
| ■ Unit 2 | Introduction to Screen Programming                |
| ■ Unit 3 | The Program Interface                             |
| ■ Unit 4 | Screen Elements for Output                        |
| ■ Unit 5 | Screen Elements for Input/Output                  |
| ■ Unit 6 | Screen Elements: Subscreens and Tabstrip Controls |
| ■ Unit 7 | Screen Elements: Table Controls                   |
| ■ Unit 8 | Context Menus                                     |
| ■ Unit 9 | Lists in Screen Programming                       |



- During the course, each participant will write a complex transaction for maintaining flight and flight booking data with the following functions:
  - Display and change information for a selected flight
  - Display the bookings for selected flight



## Single-screen transaction



© SAP AG 2002

- Input screen and data screen combined in one window
- Switch between *Create*, *Change*, and *Display*
- Direct access to each object
- System retains context after saving

- The aim of the new SAP programming model is to replace long, nested screen sequences with single-screen transactions.
- The main advantage is that it improves the usability of the R/3 System. The transactions are much simpler for users to use.
- Input screen and data screen are combined in one window. This saves the user unnecessary navigation and ensures the correct business context.
- Single screen transactions provide the user with the program session that best fits his or her authorizations, allow the user to directly access the objects to be edited; and limit the selection area using a filter, a tree structure, or, for example, the last edited object.
- After saving the data to the database, the user can display the edited object again to check the changes he or she has made.

## Example Screen Layout: Transaction FB60

SAP

**Application functions**

**Object selection**

**Object ID**

**Details of object**

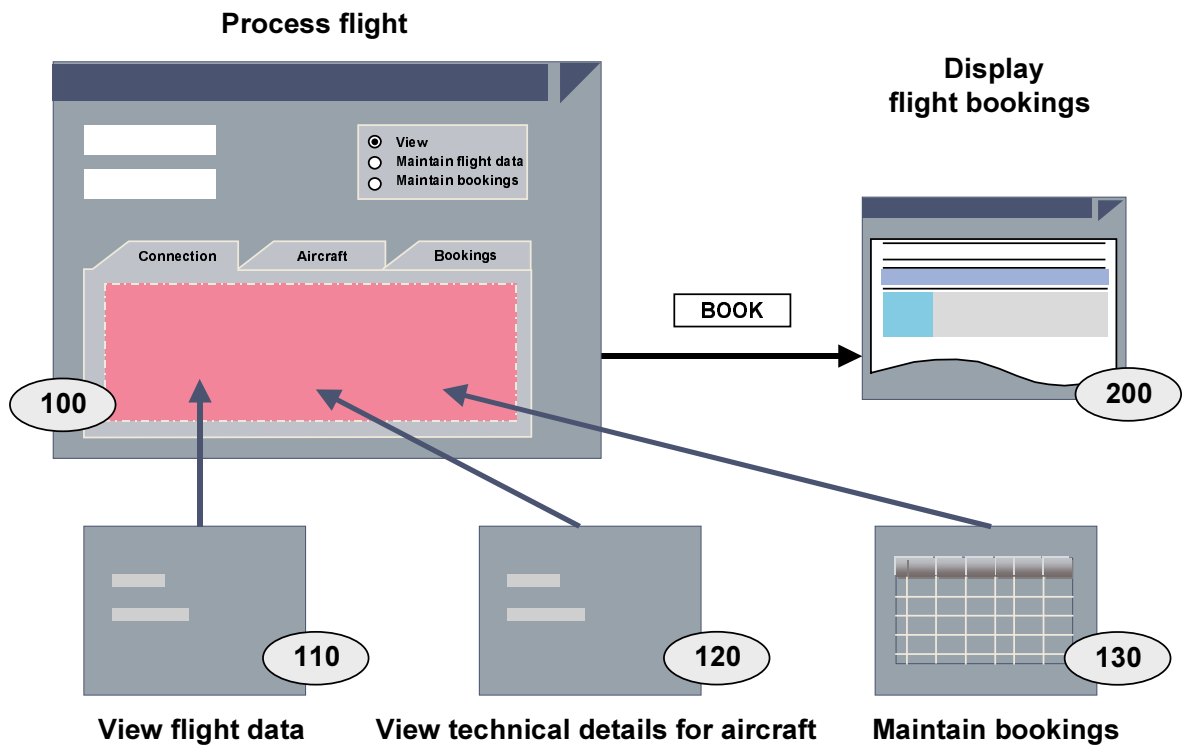
St.	GL acct	Short text	DIC	Doc. currency amount	Loc. curr. amount	P	FM	Co.
435800	Annual Bon.	S De.		10.000,00	10.000,00			1000
		S De.			0,00			
		S De.			0,00			
		S De.			0,00			
		S De.			0,00			
		S De.			0,00			
		S De.			0,00			
		S De.			0,00			

© SAP AG 2002

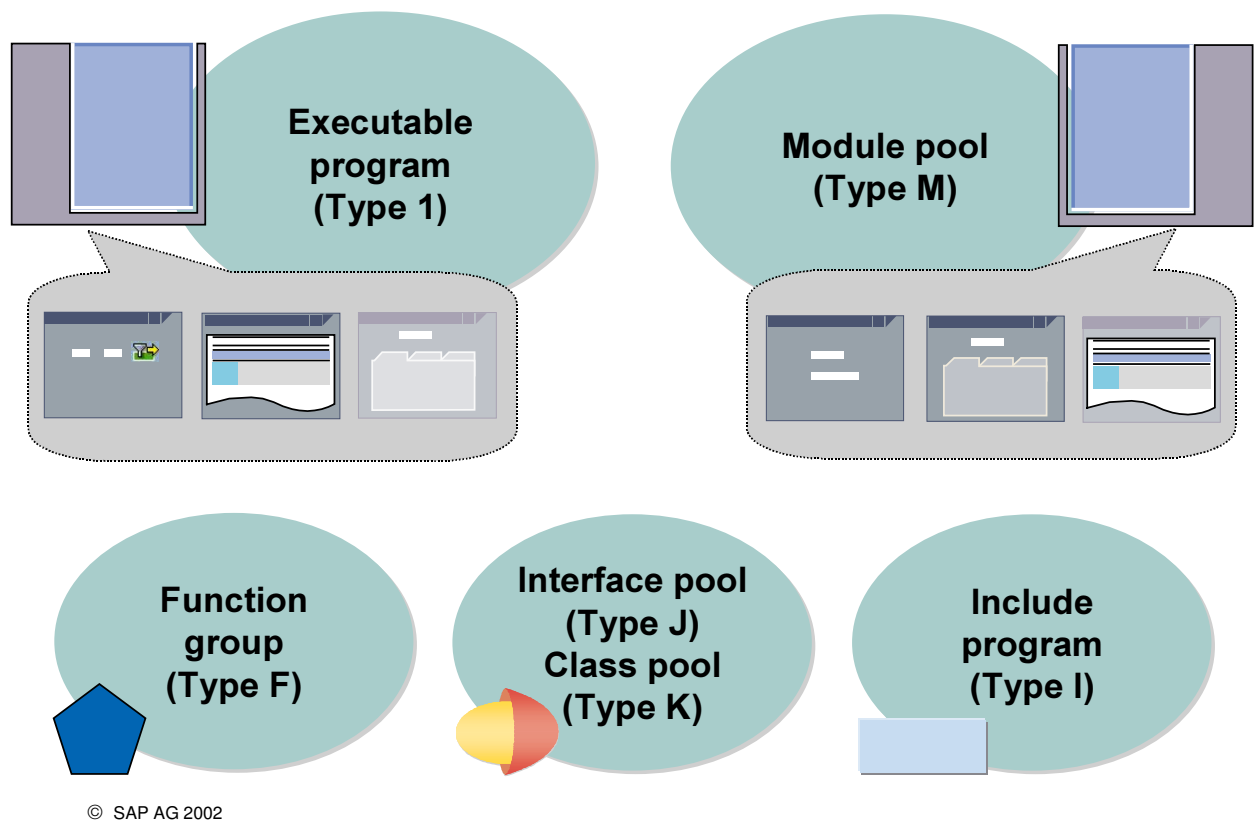
- Transaction FB60 has been redesigned in accordance with the new programming model.
- The screen is divided into four sections, each containing different functions:
  - **Object selection:** You can select the object you want to edit from a tree structure.
  - **Object ID:** You can edit the key data and attributes of the whole object.
  - **Details of object:** You can select subobjects for editing.
  - **Application functions:** Only a few functions are available from the application toolbar as a result of the new single screen transactions. These include display options, (such as showing and hiding screen areas), creating new objects with templates, or toggling between different sessions of a program.

## Exercise: Screen Programming

SAP



© SAP AG 2002



## ■ Executable program (type 1)

Executable programs are run directly from the ABAP Editor. A set of processing blocks is processed in a predefined order. You can use a standard selection screen. Type 1 programs normally create and display a list.

## ■ Module pool (type M)

For a type M program to be executable, you **must** create at least one transaction code in which you specify an initial screen. You can control the subsequent screen sequence either statically (in the screen attributes) or dynamically (in the program code).

## ■ The following types of programs **cannot** be executed directly. They serve as containers for modularization units, which you call from other programs. Whenever you load one of these modules, the system loads its entire main program into the internal session of the calling program.

### • Function group (type F)

A function group can contain function modules, local data declarations, and screens.

### • Include program (type I)

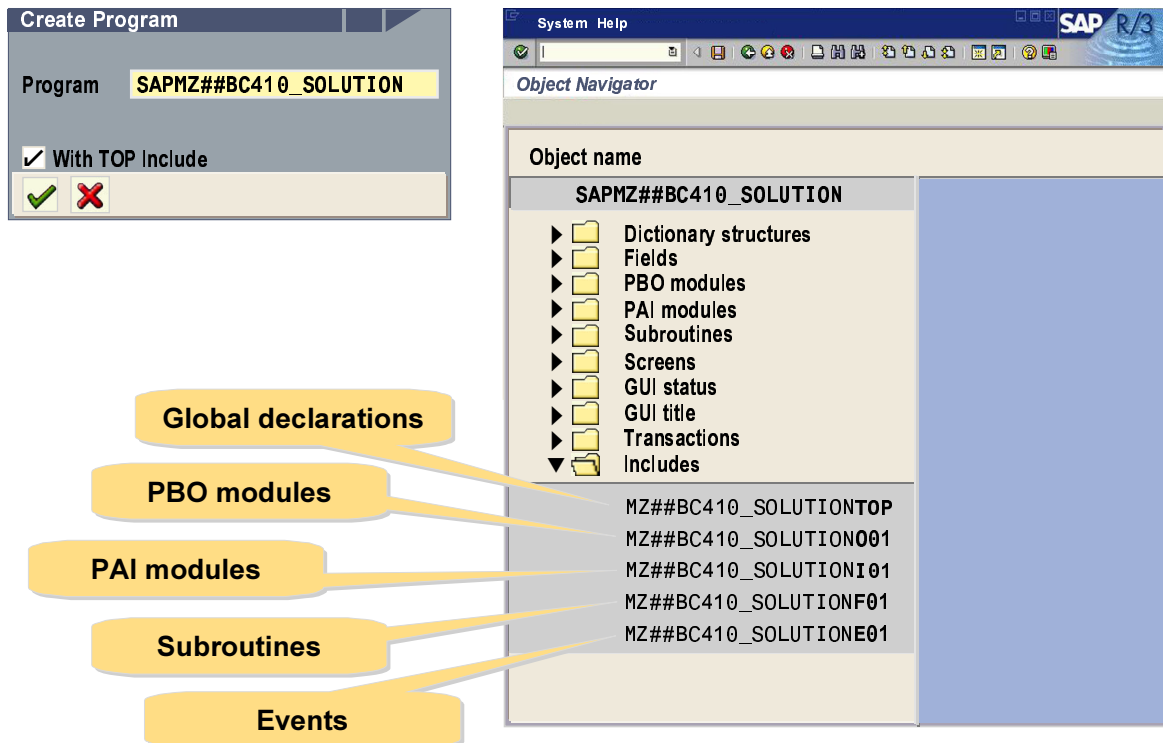
An include program can contain any ABAP statements.

### • Interface pool (type J)

An interface pool can contain global interfaces and local data declarations.

### • Class pool (type K)

A class pool can contain global classes and local data declarations.

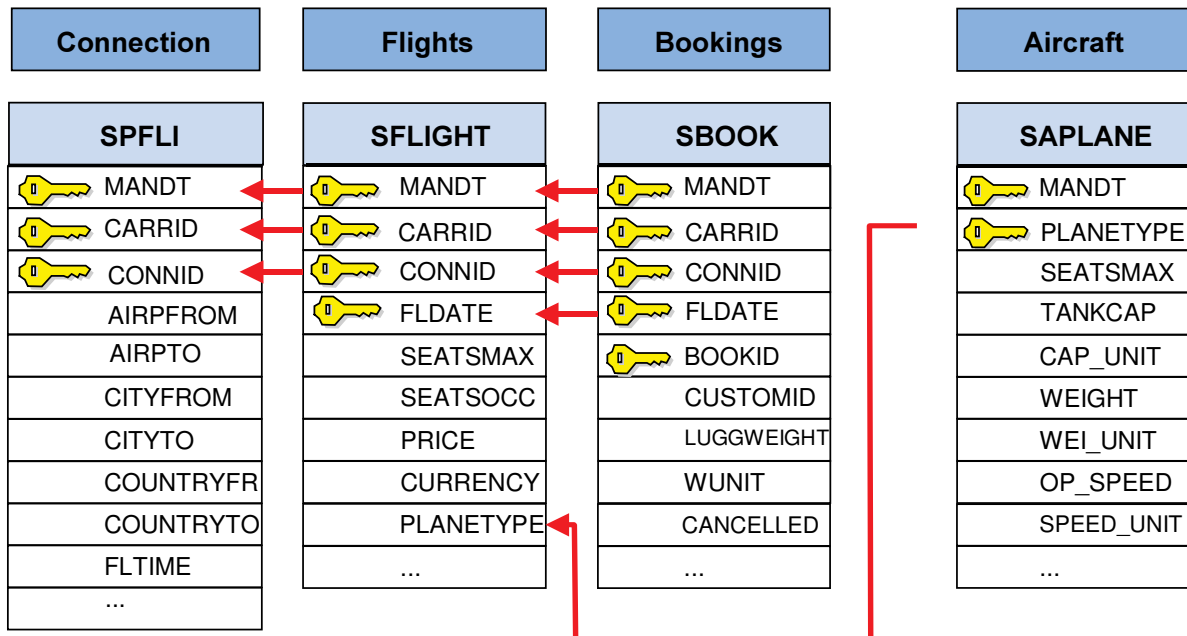


© SAP AG 2002

- In the simplest case, your program consists of a single source that contains all the necessary processing blocks. However, to make your program code easier to understand and to enable you to reuse parts of it in other programs (for example, for data declarations), you should use include programs.
- Whenever you create a program from the Object Navigator, the system proposes to create it **With TOP include**. Selecting this option will help you to create clearly structured programs.
- When you create processing blocks, the system automatically asks in which include program it should place the corresponding source code.
- If you specify an include program that does not yet exist, the system creates it and inserts a corresponding **INCLUDE** statement in the main program.

## Tables of the Flight Data Model (BC410)

SAP



© SAP AG 2002

### ■ The most important table fields used in this course and their meaning:

#### ■ SPFLI

- CARRID Airline identified
- CONNID Connection code
- AIRPFROM, AIRPTO Departure airport, arrival airport
- CITYTO, CITYFROM Arrival city, departure city

#### ■ SFLIGHT

- CARRID, CONNID See SPFLI
- FLDATE Flight date
- SEATSMAX, SEATSOCC Maximum capacity, occupied seats
- PRICE Basic flight price
- CURRENCY Currency

#### ■ SBOOK

- CARRID, CONNID, FLDATE See SFLIGHT
- BOOKID Reservation number
- CUSTOMID Customer number

#### ■ SAPLANE

- PLANETYPE Plane type
- SEATSMAX Maximum capacity

## Contents:

- Principles of screen programming
- Screen elements
- Screen processing
- Dynamic screen modifications
- Screen sequence

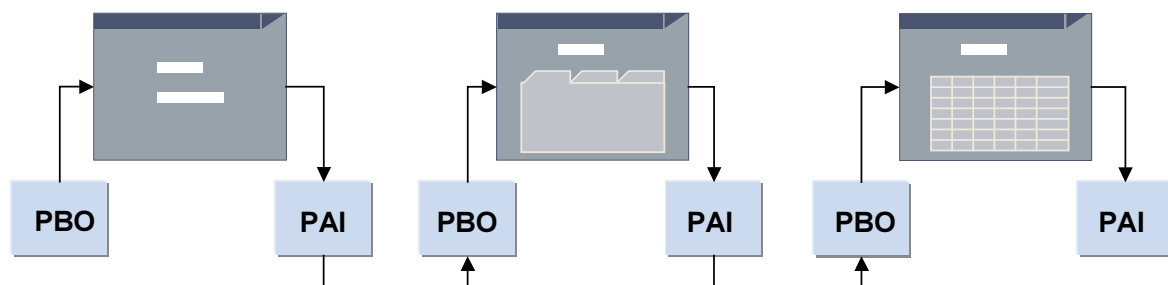


© SAP AG 2002

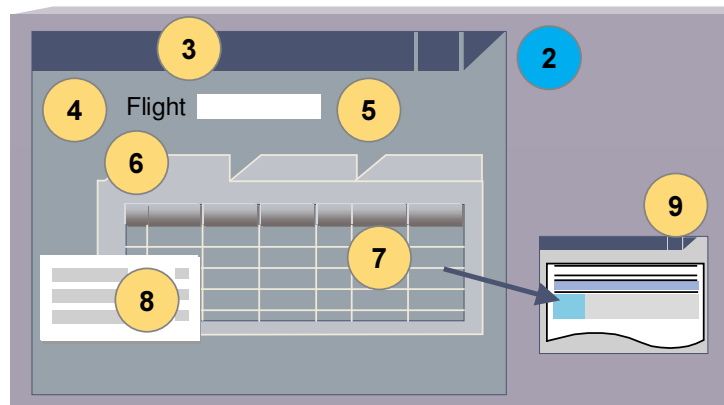


At the conclusion of this unit, you will be able to:

- Create and process screens
- Add ABAP Dictionary Screen elements
- Explain PBO and PAI processing
- Make dynamic screen modifications
- Insert screen sequences



© SAP AG 2002



© SAP AG 2002

- |          |   |
|----------|---|
| ■ Unit 1 | Course Overview                                   |
| ■ Unit 2 | <b>Introduction to Screen Programming</b>         |
| ■ Unit 3 | The Program Interface                             |
| ■ Unit 4 | Screen Elements for Output                        |
| ■ Unit 5 | Screen Elements for Input/Output                  |
| ■ Unit 6 | Screen Elements: Subscreens and Tabstrip Controls |
| ■ Unit 7 | Screen Elements: Table Controls                   |
| ■ Unit 8 | Context Menus                                     |
| ■ Unit 9 | Lists in Screen Programming                       |



**Principles of screen programming**

**Screen elements**

**Screen processing**

**Dynamic screen modifications**

**Screen sequence**

© SAP AG 2002

**Data entry  
with consistency  
checks**



**Screen**

**Table control**

	Flight	From	To	
	0400	Frankfurt	New	▲
	0402	Frankfurt	New	
	2407	Berlin	San F	▼

**Tabstrip control**

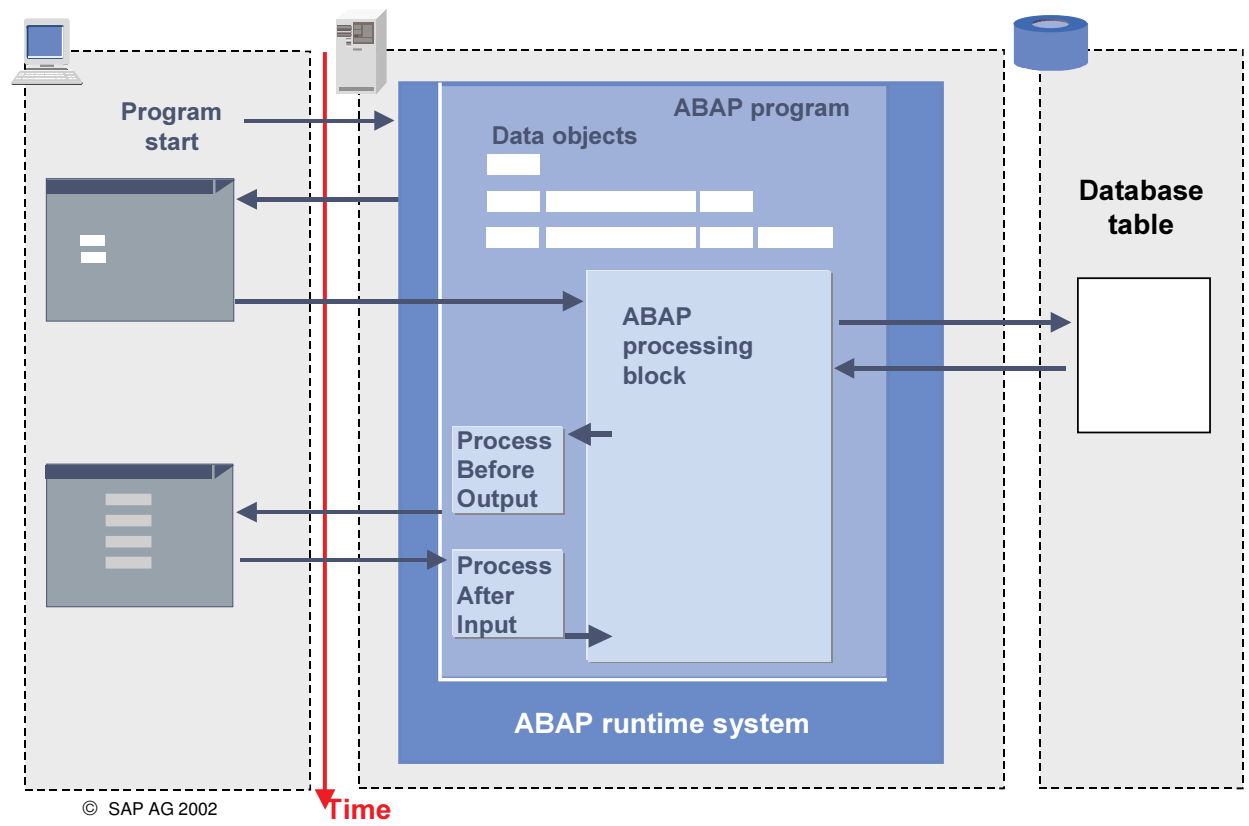
From	Arrival
Country	DE
City	Berlin
Time	10:10:00

© SAP AG 2002

- Screens allow you to enter and display data.
- One of their strengths is that they combine with the ABAP Dictionary to allow you to check the consistency of the data that a user has entered.
- Screens allow you to create user-friendly dialogs with pushbuttons, tabstrip controls, table controls, and other graphical elements.

## Screens in Dialog Programs

SAP



- Let us look at a simple dialog program with a selection screen as its initial screen and a screen for displaying information for a selected data record.
- When the program starts, the system loads its program context and prepares memory space for the program data objects. The selection screen is displayed.
- The user enters data on the selection screen and chooses *Execute*.
- In a processing block, the program reads data from the database. To do so, it passes information about the data requested by the user to the database. The database fills a structure with the required data record.
- The processing logic then calls a screen. This triggers a processing block belonging to the screen called Process Before Output (**PBO**). Once the PBO has been processed, the data is transferred to a structure that serves as an interface to the screen. It is then transferred to the screen and displayed.
- Any user action on the screen (such as entering data, choosing a menu entry or clicking a pushbutton ) returns control to the runtime system. The screen fields are then transported into the structure that serves as the interface between screen and program, and the runtime system triggers another processing block belonging to the screen, which is called Process After Input (**PAI**) and is always processed after a user interaction.

Principles of screen programming



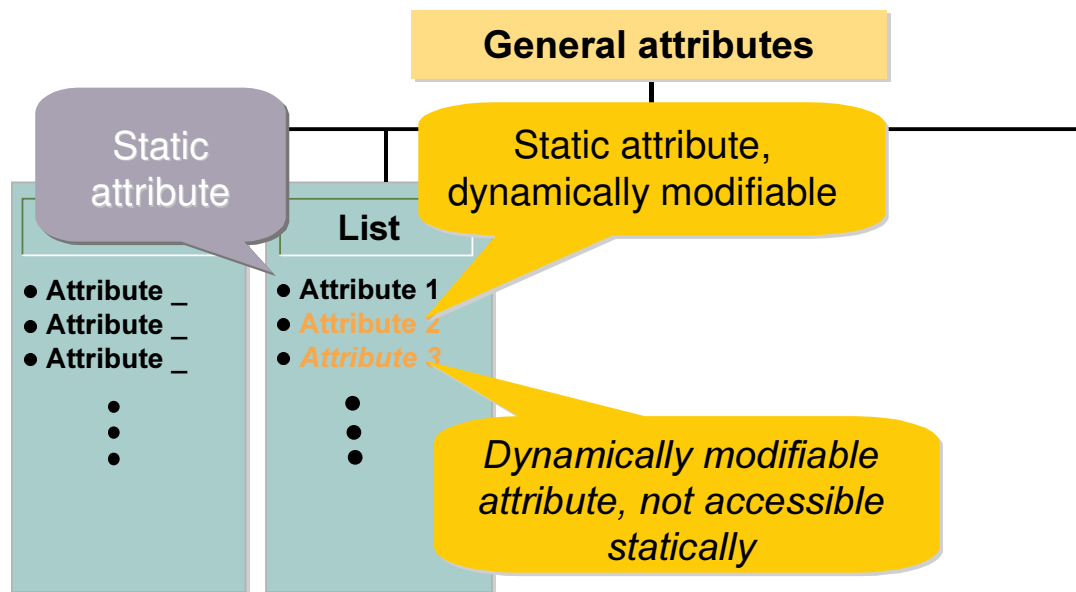
Screen elements

Screen processing

Dynamic screen modifications

Screen sequence

© SAP AG 2002



© SAP AG 1999

- The screen elements *text field*, *input/output field*, *status icon*, *group box*, *radio button*, *checkbox*, and *pushbutton* all have general attributes, Dictionary attributes, program attributes, and display attributes.
- The elements *subscreen*, *tabstrip control*, and *table control* have general attributes, and special attributes relating to the respective type.
- We can divide the attributes of an element into:
  - **Statically definable** attributes that **cannot be changed dynamically**
  - **Statically definable** attributes that **can be changed dynamically**
  - Attributes that **can only be changed dynamically**
- For complete documentation of the attributes of screen elements, see the online documentation (reference **DIA-2**).

Principles of screen programming

Screen elements



Screen processing

Dynamic screen modifications

Screen sequence

© SAP AG 2002

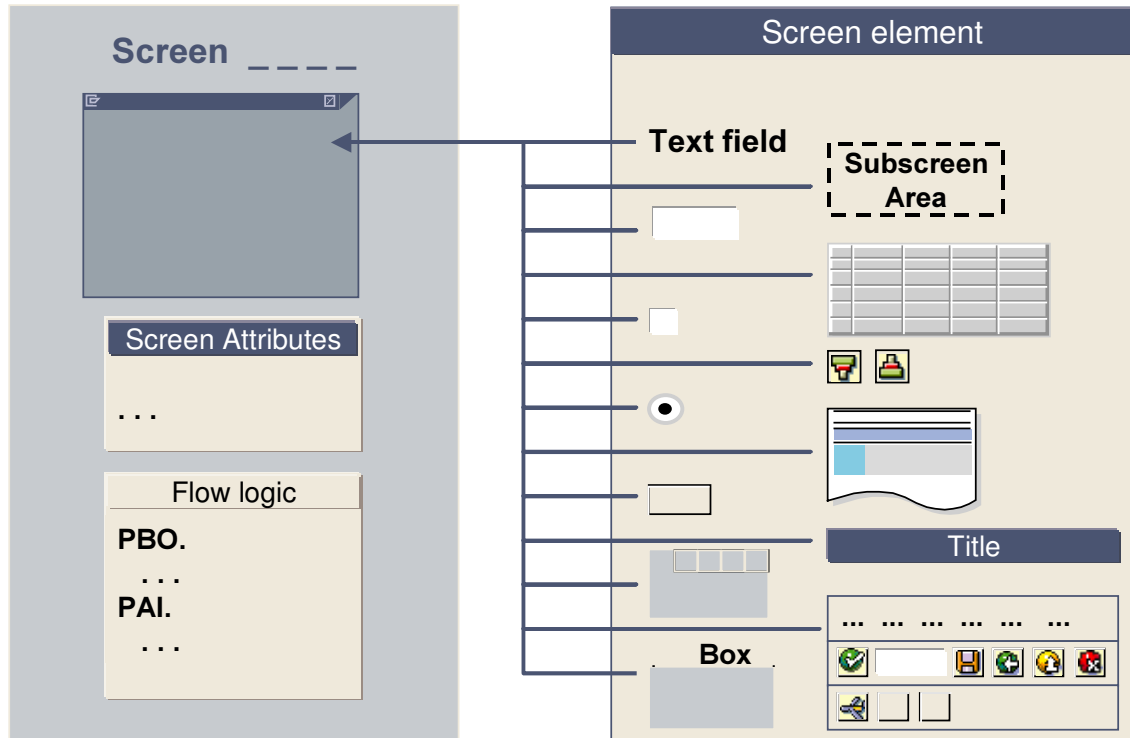
Internal Use SAP Partner Only



**SAP screen as container  
for other screen elements**

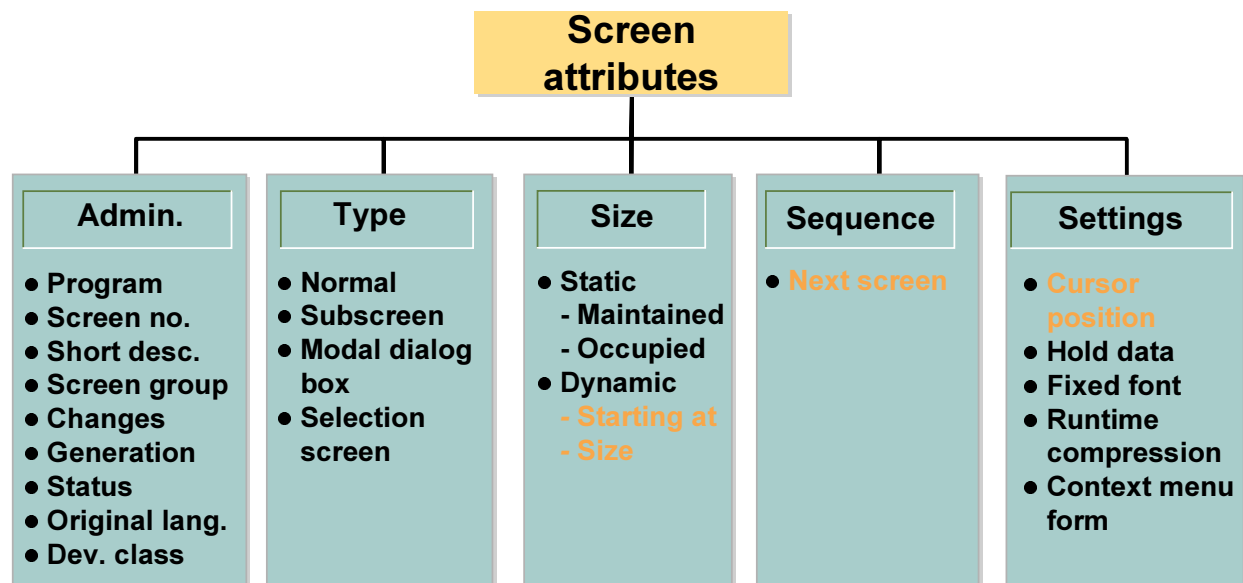
© SAP AG 2001

- Screens are freely definable objects that you can use to display or enter information through input and output fields, lists, and so on.
- They are a form of dialog between the user and the ABAP program.



© SAP AG 2001

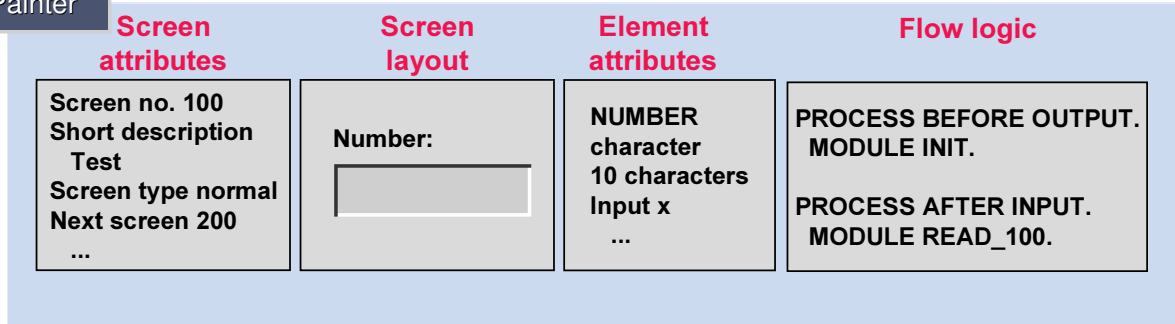
- A screen consists of a screen image and its flow logic. Strictly speaking, it is a program that controls the way the screen image is processed. For further information about programming screen flow logic, refer to the ABAP User's Guide.
- Screens have four components: the screen mask, the screen attributes, the element list, and the flow logic. The flow logic contains flow logic code (not ABAP statements).
- Screens are containers for other screen elements.



© SAP AG 2002

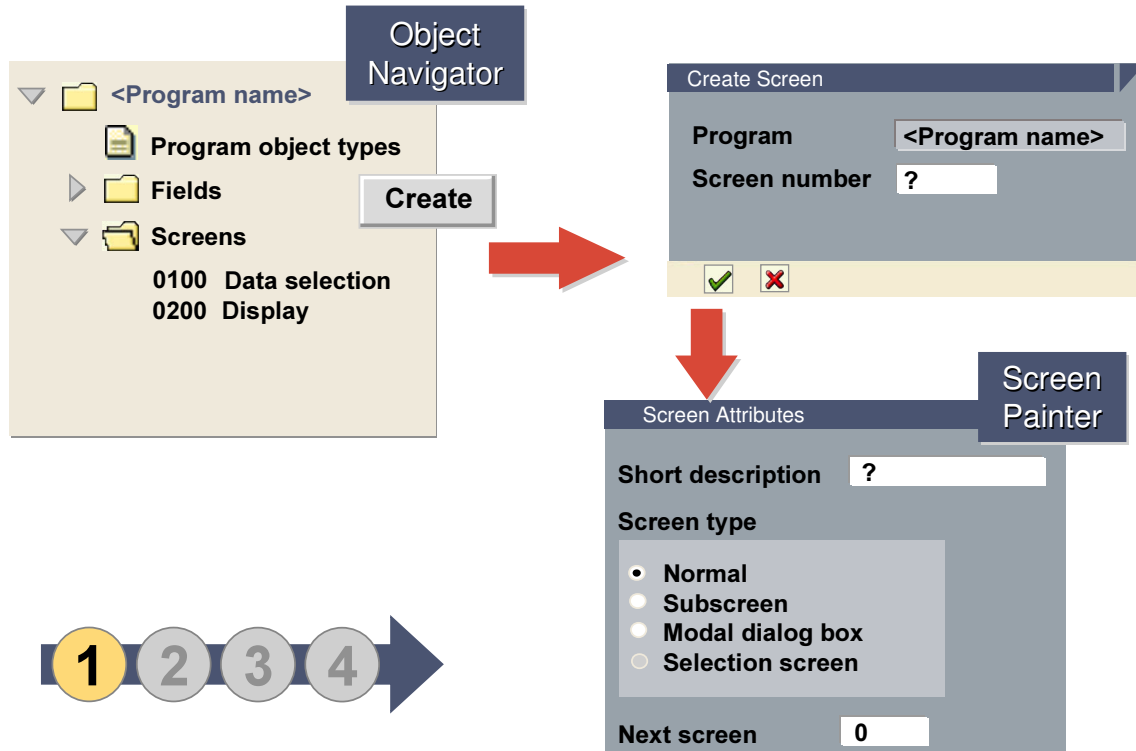
- Each screen has a set of administration attributes that specify its type, size, and the subsequent screen. It also has settings that influence other properties of the screen and of its components.
- The administration attributes *Program* and *Screen number* identify the screen by its number and the program to which it belongs.
- Screen numbers greater than 9000 are reserved for SAP System customers. Screen numbers 1000 through 1010 are reserved for the maintenance screens of ABAP Dictionary tables and the standard selection screens of executable programs.
- The screen type identifies the purpose of the screen. Certain other special attributes of a screen and its components depend on this attribute.
- The *Next screen* attribute allows you to specify the screen that should be processed after the current screen in a fixed sequence.
- For a full list of screen attributes with their meanings, refer to the online documentation path in appendix reference **DIA-3**.

Screen  
Painter



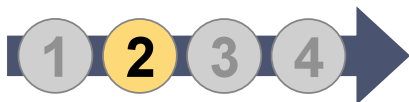
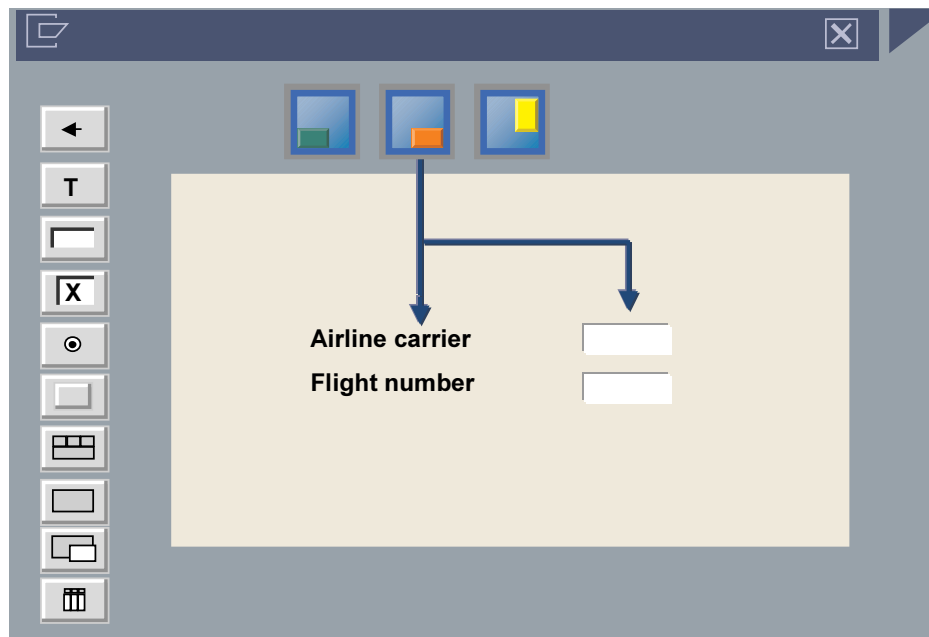
© SAP AG 2002

- When you create a screen, you must:
  - Set the general screen attributes (on the attribute screen).
  - Design the screen layout (in the layout editor).
  - Set the field attributes (in the field list).
  - Write the flow logic (in the flow logic editor).



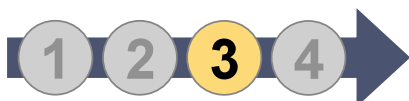
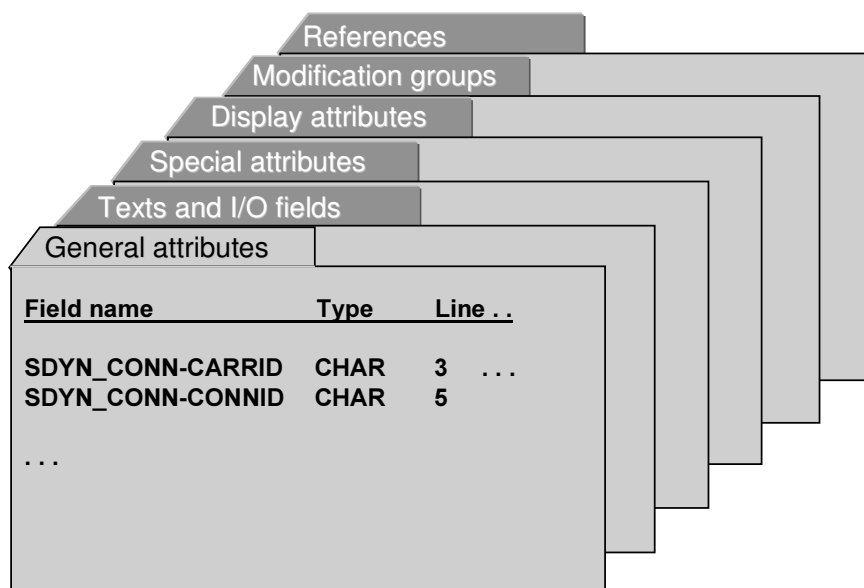
© SAP AG 2002

- To create a screen from the object list in the Object Navigator, create a new development object with the type *Screen*. Position the cursor on *Screens* and right-click.
- The Object Navigator automatically opens the Screen Painter.
- 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.
- You can also create a screen by writing a `CALL SCREEN <nnnn>` statement in the ABAP Editor and then double-clicking the screen number <nnnn>.



© SAP AG 2002

- You usually define screen elements by adopting the corresponding field descriptions from the ABAP Dictionary. However, you can also use field descriptions that you defined in your program. To do this, you must generate the program first.
- You can use the key word texts and templates either together or separately.
- The graphical layout editor provides an easy way of defining the various screen elements (such as input/output fields, key word texts, and boxes). You simply choose the element you require, and position it on the screen using the mouse.
- To delete a screen element, select it and choose *Delete*.
- You can move elements on the screen by dragging and dropping them with the mouse.
- **Note:** The graphical layout editor is available under Windows NT, Windows 95, Windows, 98, Windows 2000, and UNIX. If you use a different operating system, you must use the alphanumeric Screen Painter.



© SAP AG 2002

- To allow you to set the attributes of all screen elements, the Screen Painter contains an element list with six views. You can also display all of the attributes for a single element from any of the lists (*Attributes*). You can also maintain the attributes for an element from the layout editor using the *Attributes* function.
- In the Screen Painter, you work with external data types. These correspond to the data types in the ABAP Dictionary. For fields that you have chosen that are defined in the ABAP Dictionary, the system displays the external data type in the *Format* column. For elements (templates) that do not have an ABAP Dictionary reference, you must enter an external data type yourself.
- To find out the corresponding external data type for an internal data type (ABAP data type), see the keyword documentation for the ABAP **TABLES** statement. For example:

ABAP Dictionary Data Type ABAP Data Type

CHAR	C
NUMC	N

## Creating Screens: Flow Logic

SAP

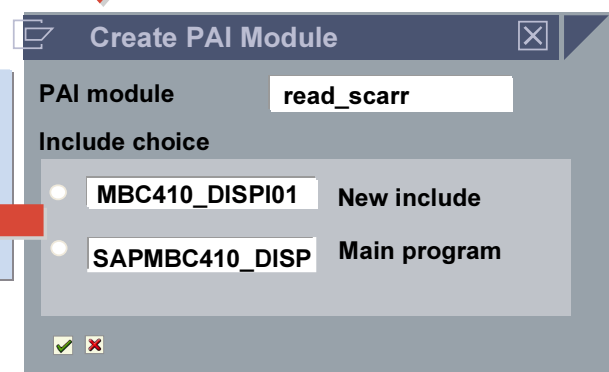
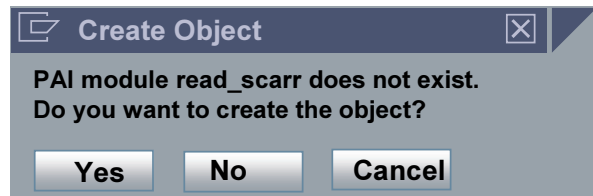
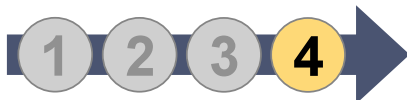
Screen Painter

```
PROCESS AFTER INPUT.
MODULE read_scurr.
```

Double-click

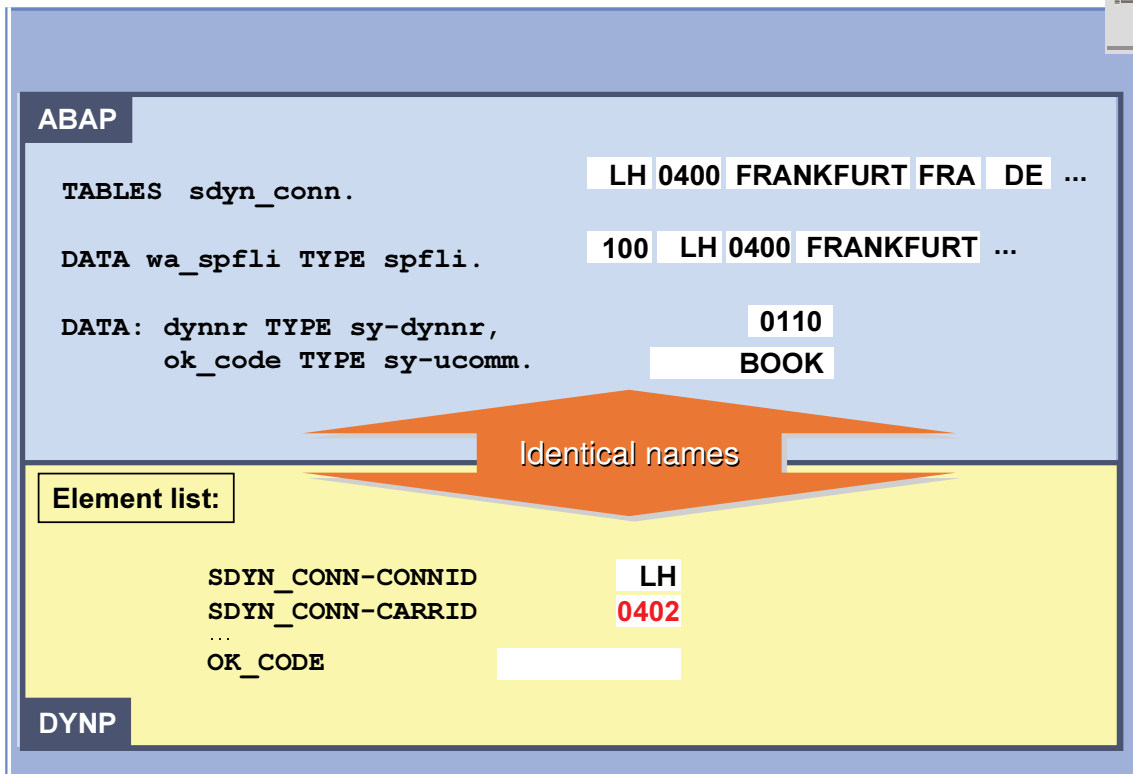
ABAP

```
MODULE read_scurr INPUT.
  SELECT SINGLE * FROM scarr
  WHERE carrid = sdyn_conn-carrid
ENDMODULE.
```



© SAP AG 2002

- Screens have their own set of keywords that you use in the PBO and PAI events of the flow logic. Refer to the online documentation **DIA4** for an overview of existing key words.
- In the flow logic, you write MODULE calls. The modules are components of the same ABAP program. They contain the ABAP statements that you want to execute.
- You can create a module by double-clicking the module name in the flow logic Editor.
- To create a module from the object list in the Object Navigator, choose the development module *PBO module* or *PAI module*.
- You can call the same module from more than one screen. If the processing depends on the screen number, you can retrieve the current screen number from the field SY-DYNNR.
- Note that the modules you call in the PBO processing block must be defined using the MODULE... OUTPUT statement; modules that you define using the statement MODULE... INPUT can be called only in the PAI event.

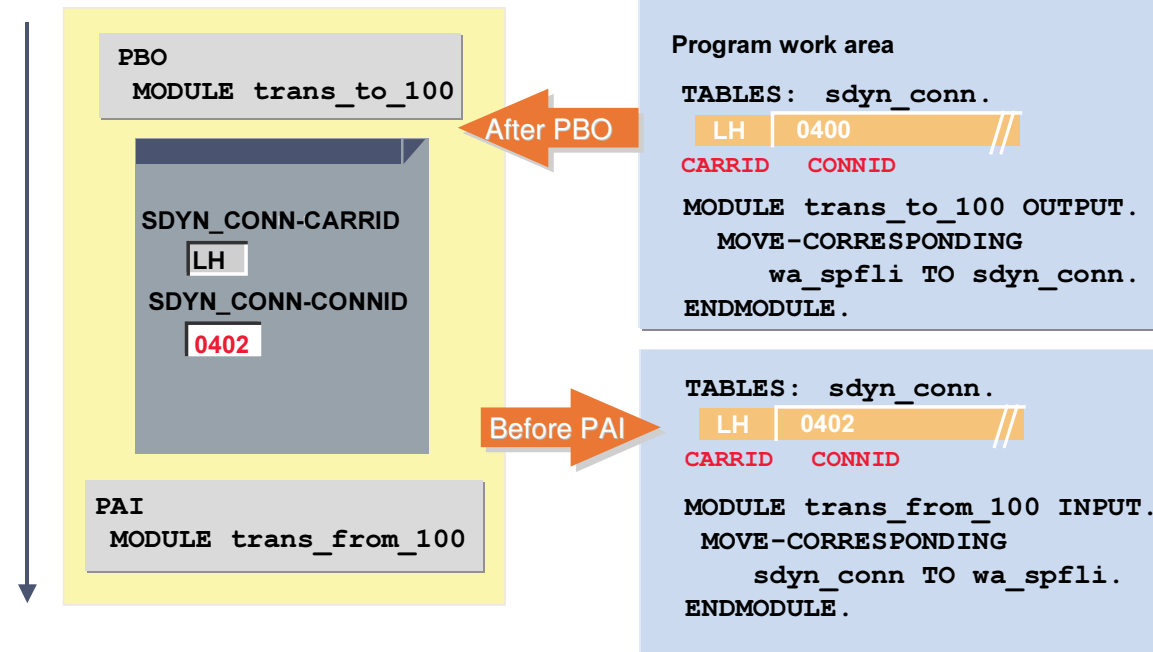


© SAP AG 2002

- There are many different software processors involved in the screen processing of your program. The ABAP processor controls the program flow within a module. The DYNP processor controls the flow logic and prepares data to be displayed on the screen.
- During this process, several sets of data are visible. You work with the global fields of your program within the module. Global fields are created in the TOP include using declarative statements, for example, TABLES or DATA.
- The fields that are recognized by the system from the element list are used to retrieve data to display on the screen, and also to transport data changed by the user. This occurs automatically, when you get fields from the ABAP Dictionary or from the program in the Layout Editor.
- It is necessary to copy the data because of the different sets of data fields. A system program carries out the copying process. At defined instances during the process, the identically named fields of DYNP and ABAP are compared.

## IDENTICAL NAMES

Time



© SAP AG 2002

- For a screen and its ABAP program to be able to communicate, the fields on the screen and the corresponding fields in the program **must have identical names**.
- **After** it has processed all of the modules in the **PBO** processing block, the system copies the contents of the fields in the ABAP work area to their corresponding fields in the screen work area.
- **Before** it processes the first module in the **PAI** processing block, the system copies the contents of the fields in the screen work area to their corresponding fields in the ABAP work area.
- You should use your own structures (such as SDYN\_CONN) for transporting data between the screen and the ABAP program. This ensures that the data being transported from the screen to the program and vice versa is exactly the data that you want.

Principles of screen programming

Screen elements

Screen processing



Dynamic screen modifications

Screen sequence

© SAP AG 2002

## Attributes

### General

- Object name
- Modif. groups
- Size
- Dynamic

### Program

- Dialog behavior
- Input
- Output
- Required

### Display

- Bright
- Invisible

## SCREEN

SCREEN-NAME

SCREEN-GROUP1  
SCREEN-GROUP2  
SCREEN-GROUP3  
SCREEN-GROUP4

SCREEN-LENGTH

SCREEN-INPUT

SCREEN-OUTPUT

SCREEN-REQUIRED

SCREEN-INTENSIFIED

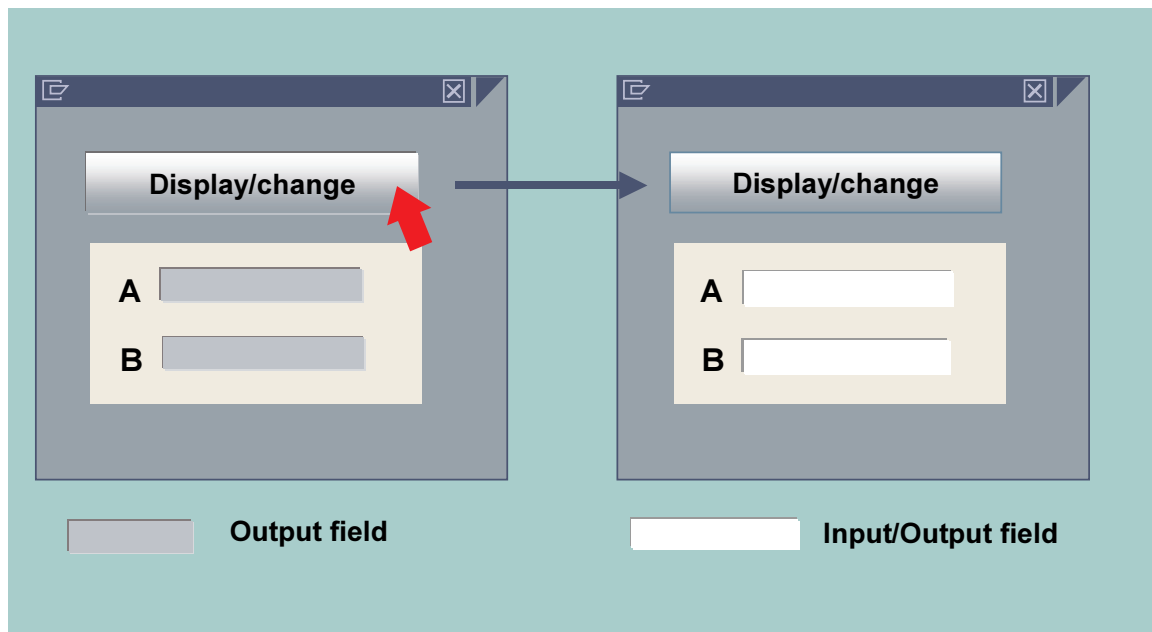
SCREEN-INVISIBLE

SCREEN-ACTIVE



© SAP AG 2002

- At the beginning of the PBO, the runtime system reads the statically-created and dynamically-modifiable attributes of each screen element on the current screen into a system table with the line type SCREEN.
- For a complete definition of the SCREEN structure, refer to the Help on...(Ctrl-F8) documentation on *Table, Structure, or View* in the ABAP Editor (Ctrl-F8).
- The graphic shows the assignment of the fields in the system table SCREEN to the names of the statically created attributes of the screen elements.



© SAP AG 2001

- Dynamic changes to the attributes of screen elements are **temporary**.
- Using this technique to modify the attributes of a screen element (for example, to change whether an input/output field is ready for input), you can replace long sequences of separate screens, which are more costly in terms of both programming time and run time.

## The SCREEN System Table

SAP

NAME	GROUP 1	GROUP 2	GROUP 3	GROUP 4	LENGTH	INPUT	OUTPUT	REQUIRED	INTENSIFIED	INVISIBLE	ACTIVE
FIELD1					20	1	1	1	1	0	1
RADIO1	ADM				1	1	1	0	0	0	1
RADIO2	ADM				1	1	1	0	0	0	1
RADIO3	ADM				1	1	1	0	0	0	1
P_TOG					35	0	0	0	0	0	1
FIELDA	SEL				15	1	1	0	0	0	1
FIELDDB	SEL				15	1	1	0	0	0	1

© SAP AG 2002

- The system table with line type SCREEN is called **SCREEN system table** in the following unit.
- When a screen is processed, the SCREEN system table contains an entry for each element created in the Screen Painter for that screen.

## Initializing the SCREEN System Table

SAP

NAME	INPUT	OUTPUT	REQ. FIELD	...
FIELD1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
P_TOG	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
FIELDA	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
FIELDDB	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Copy the static attributes before PBO

Element list  
screen 100



NAME	INPUT	OUTPUT	REQUIRED	ACTIVE	...
FIELD1	1	1	1	1	
P_TOG	0	1	0	1	
FIELDA	1	1	0	1	
FIELDDB	1	1	0	1	
...					

© SAP AG 2002

- The system table SCREEN is initialized at the start of the PBO event for the current screen. To do this, a system program copies the statically defined attributes of the individual screen elements into the table.
- You can then change the dynamically modifiable attributes of the elements on the screen in a module at PBO using the following statements:  
 LOOP AT SCREEN.  
 ...  
 MODIFY SCREEN.  
 ENDLOOP.  
 To do this, you use the structure SCREEN, which is created automatically by the system, and filled with the values of each successive line of the system table in the loop. Set attributes have the value '1', attributes that are not set have the value '0'. To change the system table, use MODIFY SCREEN within the loop.
- To find the element whose attributes you want to modify, you can use a LOOP on the SCREEN table and query one of the following fields: SCREEN-NAME or SCREEN-GROUP1 to SCREEN-GROUP4.

Screen Painter

Element list: Modification groups				
Name	Group 1	Group 2	Group 3	Group 4
FIELD1		ADM	TRA	
P_TOG		ADM		
FIELDA	SEL		TRA	
FIELDDB	SEL			
...				

© SAP AG 2002

- You can change the attributes of several screen elements simultaneously at run time, by including them in a modification group in the Screen Painter. Assign all elements that will be changed within a single processing step to a group in the Screen Painter. To do this, enter a group name for each of the relevant elements in one of the GROUP1 to GROUP4 fields.
- You can include each element in up to four modification groups. You can choose any three-character sequence for the group name. You can assign elements to a modification group either in the element list or the layout editor in Screen Painter.

```
PROCESS BEFORE OUTPUT.
.
.
MODULE modify_screen.
.
.
```

Screen  
Painter

```
MODULE modify_screen OUTPUT.
...
LOOP AT SCREEN.
  IF screen-group1 = 'SEL'.
    screen-input = 1.
  ENDIF.
  IF screen-name= 'FIELD1'.
    screen-active= 0.
  ENDIF.
  MODIFY SCREEN.
ENDLOOP.
ENDMODULE.
```

ABAP

© SAP AG 2002

- You must program your screen modifications in a module that is processed during the PROCESS BEFORE OUTPUT processing block.
- You use a loop through the table SCREEN to change the attributes of an element or a group of elements. (LOOP AT SCREEN WHERE . . . and READ TABLE SCREEN are not supported).
- To activate and deactivate attributes, assign the value 1 (active) or 0 (inactive), and save your changes using the MODIFY SCREEN statement.
- Note that elements you have defined statically in the Screen Painter as invisible cannot be reactivated with SCREEN-ACTIVE = 1. Instead, use the statement SCREEN-INVISIBLE = 0. However, elements that you have statically defined as visible in the Screen Painter can dynamically be made invisible with SCREEN-ACTIVE = 0. This has the same effect as the three statements SCREEN-INVISIBLE = 1, SCREEN-INPUT = 0, SCREEN-OUTPUT = 0.

Principles of screen programming

Screen elements

Screen processing

Dynamic screen modifications

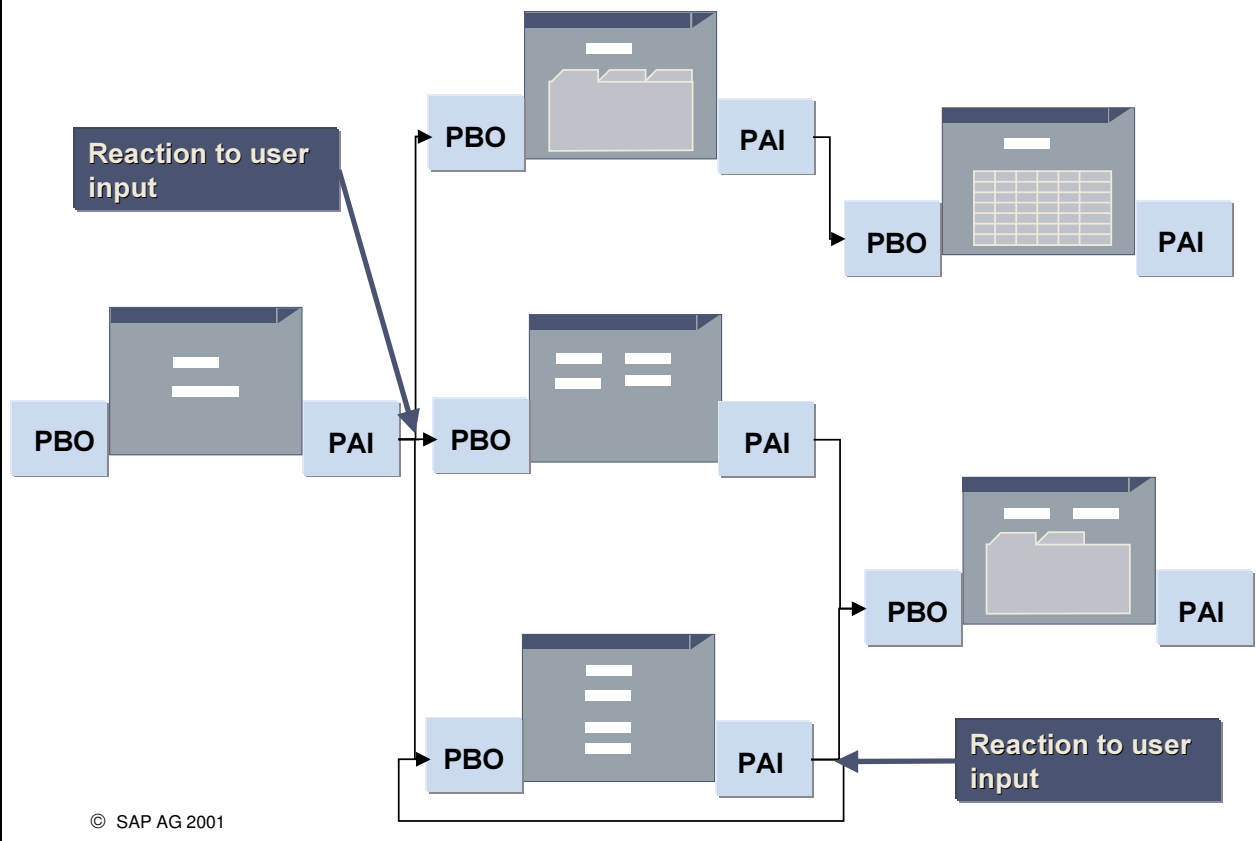


Screen sequence

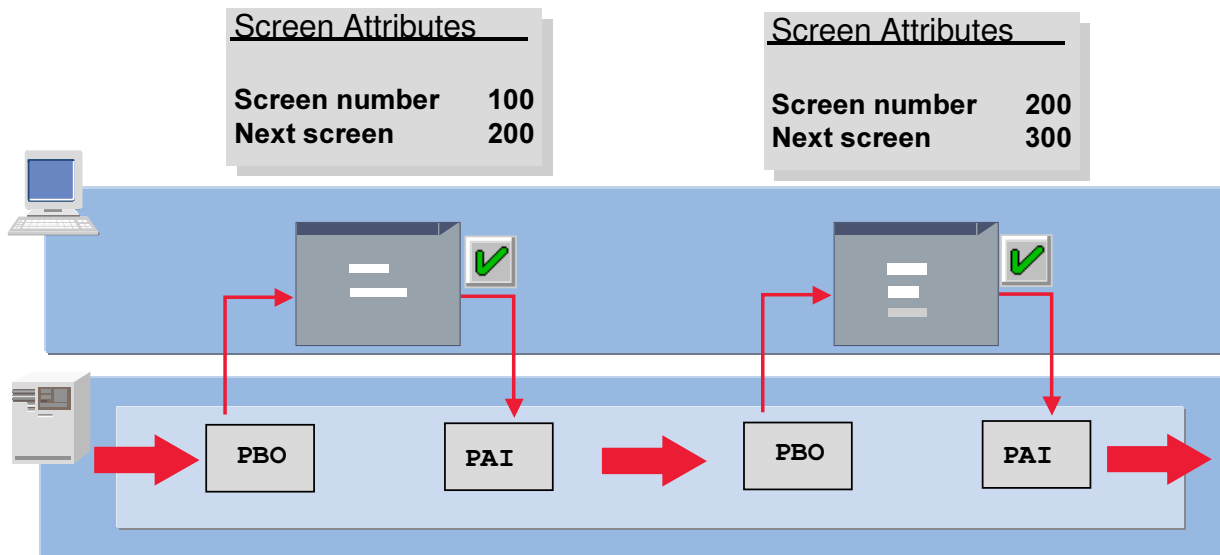
© SAP AG 1999

## Determining the Next Screen

SAP



- For complex transactions it may be necessary to use multiple screens. The initial screen is determined when creating the transaction code. Each screen determines the next screen according to the user input.
- The next screen is entered statically in the screen attributes. At run time you can temporarily override the static next screen using the `SET SCREEN <nnnn>` statement.

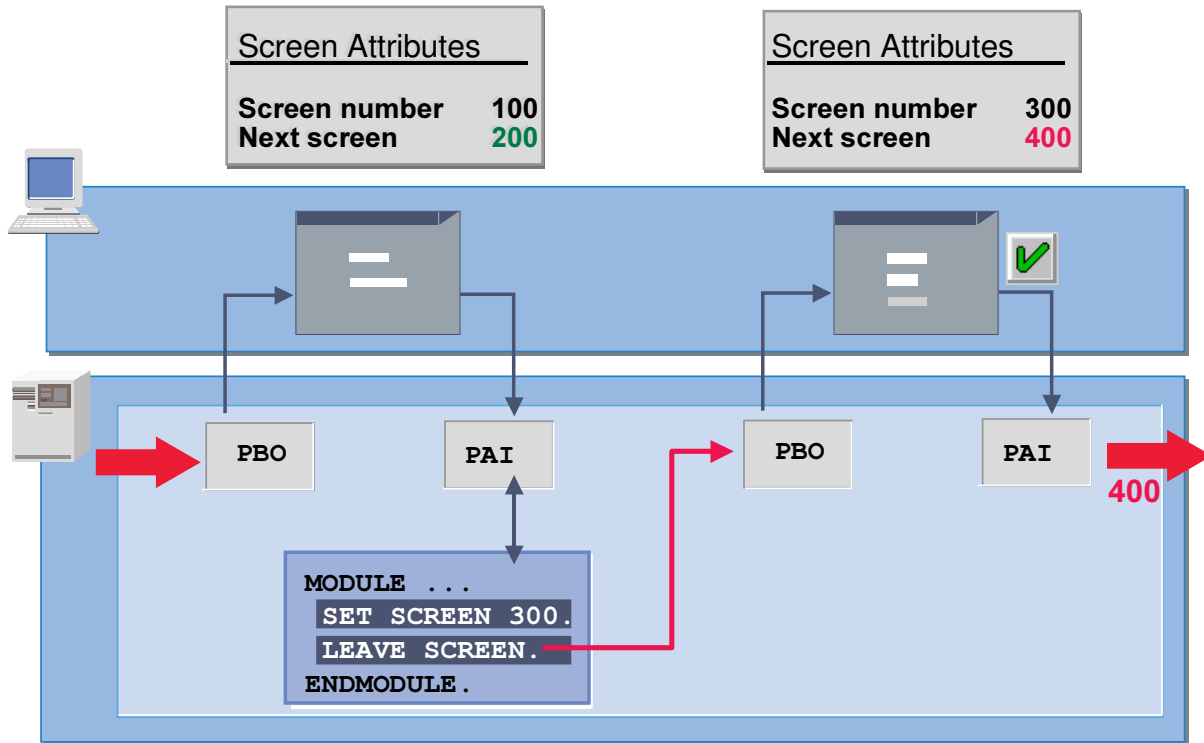


© SAP AG 1999

- You can establish a static sequence of screens by entering a value in the *Next screen* field of the screen attributes.
- If you enter 0 (or no value) as the next screen, the system resumes processing from the point at which the screen was initiated, once it has finished processing the screen itself.

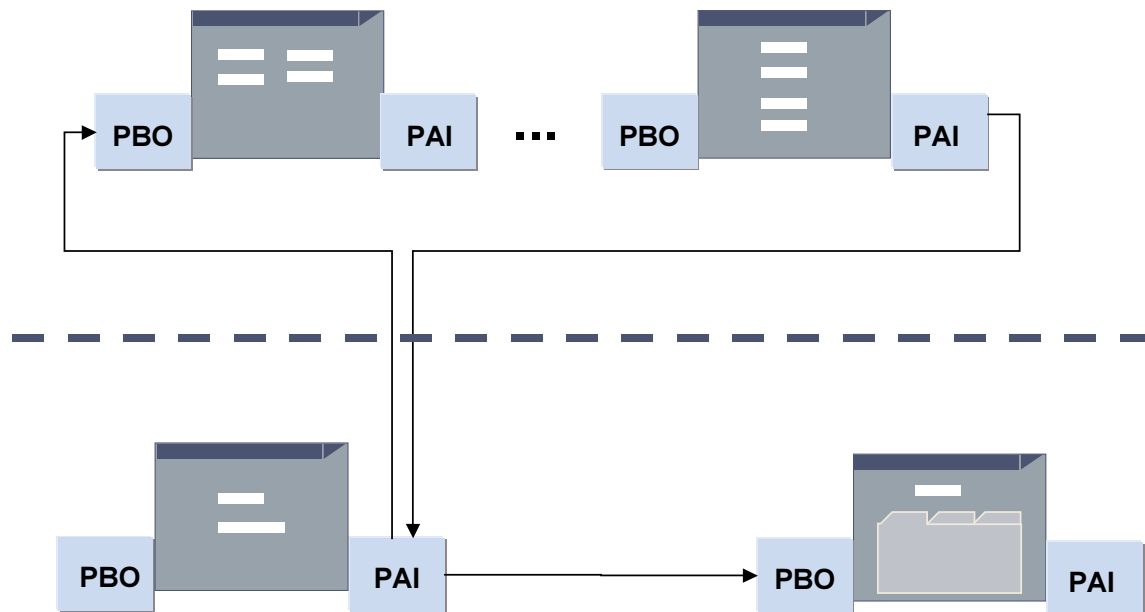
## Setting the Next Screen Dynamically

SAP



© SAP AG 2002

- The SET SCREEN <nnnn> statement **temporarily** overwrites the *Next screen* attribute.
- The screen <nnnn> must belong to the same program.
- The next screen is processed either when the current screen processing ends, or when you terminate it using the LEAVE SCREEN statement.
- To specify the next screen and leave the current screen in a single step, use the LEAVE TO SCREEN <nnnn> statement.



© SAP AG 2002

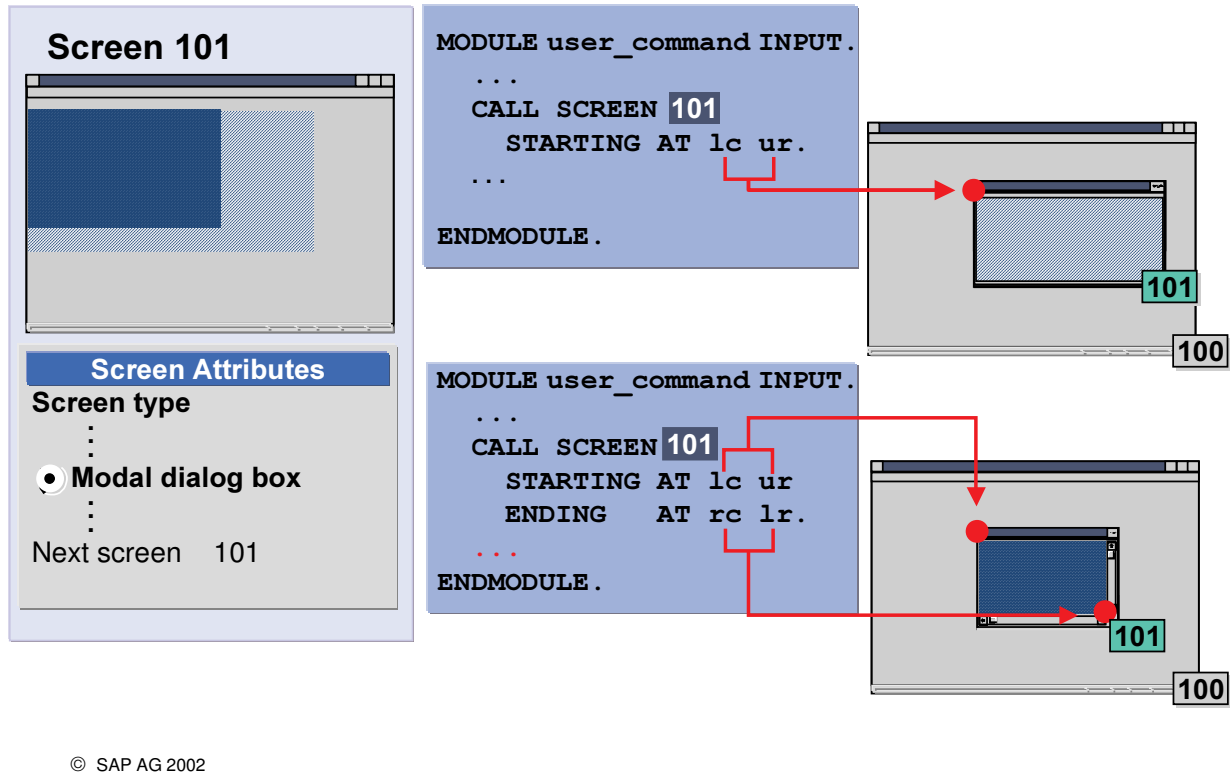
- You can insert a screen sequence. This adds another layer to a stack.
- You insert a screen sequence using the `CALL SCREEN <nnnn>` statement.
- Note: Layers created in this way must be removed afterwards. You can do this by setting the next screen statically or dynamically to the initial value (0) at the end of the inserted screen sequence.



- 
- 2-32

## Calling a Dialog Box Dynamically

SAP

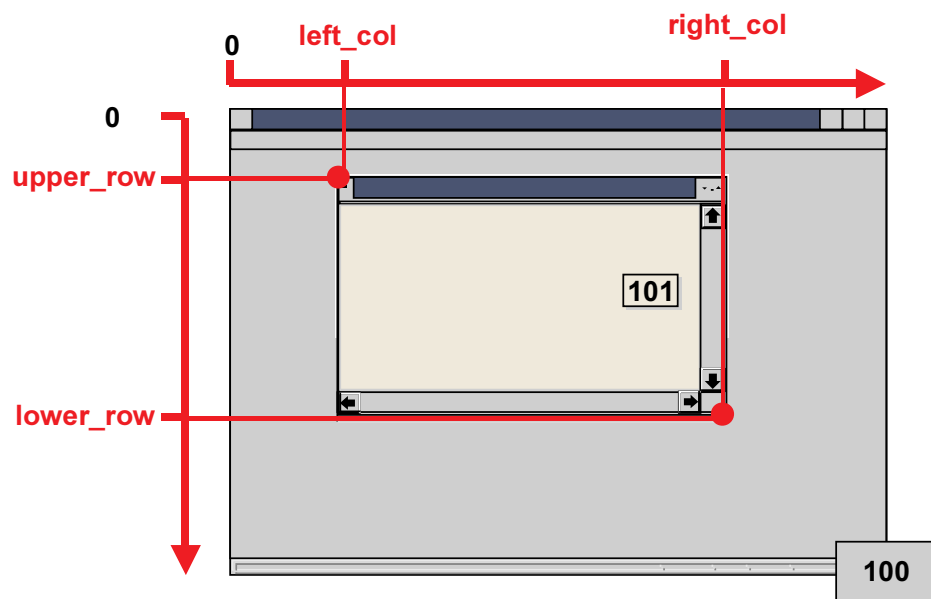


- In the CALL SCREEN statement, you can use the STARTING AT and ENDING AT additions to specify the position and size of the screen that you are calling. The screen in the CALL SCREEN statement must be defined as a modal dialog box (to comply with the SAP System's ergonomic standards).
- If you omit the ENDING AT statement, the size of the dialog box is determined by the *Used size* in its screen attributes. The system then determines the dialog box size using the *Occupied size* screen attribute.
- If you use the ENDING AT addition, the system displays as much of the dialog box as will fit into the available space. If there is not enough room to show the entire dialog box, it appears with scrollbars.

## Window Coordinates

SAP

```
CALL SCREEN 101  
  STARTING AT left_col upper_row  
  ENDING   AT right_col lower_row.
```



© SAP AG 2002

- The starting position (origin) of every SAP System window is its top left-hand corner.
- The values that are used for the variables left\_col, upper\_row, right\_col and lower\_row in the following statement relate to the R/3 Systemscreen from which you display the second screen with CALL SCREEN (screen 100 in the example in the figure).

```
CALL SCREEN <nnnn>
```

```
  STARTING AT left_col upper_row
```

```
  ENDING AT right_col lower_row.
```

## Setting the Cursor Position Dynamically

SAP

```
SET CURSOR  
FIELD <f> [OFFSET <o>] .
```

```
PROCESS BEFORE OUTPUT.  
MODULE set_cursor.
```

Screen  
Painter

```
MODULE set_cursor OUTPUT:  
  sdyn_conn-carrid = 'LH'.  
  SET CURSOR  
    FIELD 'SDYN_CONN-CONNID'.  
  ...  
ENDMODULE.
```

ABAP

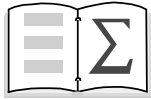
A screenshot of an SAP screen titled "Airline". It contains three input fields. The first field, labeled "Flight number", contains the text "LH". The second field, labeled "Flight date", contains a question mark "?". The third field, which is unlabeled, also contains a question mark "?". The cursor is positioned in the "Flight number" field.

© SAP AG 2002

- When the system displays a screen, it automatically places the cursor in the first input field. If you want the cursor to appear always in a different field, you can enter the corresponding element name in the *Cursor position* field of the screen attributes.
- You can also tell the system in the PBO event to position the cursor in a particular field. This makes your application easier to use.
- You can set the field in which the cursor should appear in the program using the ABAP statement:

```
SET CURSOR FIELD <object_name> OFFSET <position>.
```

- <field\_name> can be a unique name in quotation marks, or a variable containing the object name. To place the cursor at a certain position within a field, use the OFFSET parameter, specifying the required position in <position>.
- The system then places the cursor at the corresponding offset position, counting from the beginning of the field.



**You are now able to:**

- **Create and process screens**
- **Add ABAP Dictionary Screen elements**
- **Explain PBO and PAI processing**
- **Make dynamic screen modifications**
- **Insert screen sequences**

© SAP AG 2002

# Exercises



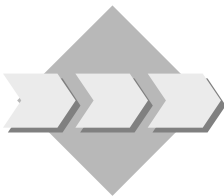
## Unit: Introduction to Screen Processing

### Topic: Creating screens



At the conclusion of these exercises, you will be able to:

- Create screens and use them in your programs



Create a screen and use it in your program.

1-1 Create a screen transaction.

1-1-1 Create development class **ZBC410\_##** where ## is your group number and assign it to the change request provided by your instructor.

1-1-2 Create program **SAPMZ##BC410\_SOLUTION** with a TOP include. Accept the system's proposal for the name of the TOP include. Use the sample solution **SAPMBC410ADIAS\_DYNPRO** as a guide.

1-2 Create the following program object:

Screen	0100	<b>Description:</b> Maintenance screen <b>Type:</b> Normal <b>Next screen:</b> 0100
--------	------	---

1-3 In the PAI event of screen 100, call a module **user\_command\_100**. Use forward navigation to create the module in a new include. Accept the system's proposal for the include (**MZ##BC410\_SOLUTIONI01**). In this module, use a statement to terminate the program and return to the point from which it was called.

1-4 Assign a transaction code **Z##BC410\_SOLUTION** to the program.



## Unit: Introduction to Screen Processing

### Topic: Creating a screen

#### Model Solution SAPMBC410ADIAS\_DYNPRO

##### Main program

```
INCLUDE MBC410ADIAS_DYNPROTOP.
```

```
INCLUDE MBC410ADIAS_DYNPROI01.
```

##### Flow logic screen 100

```
PROCESS BEFORE OUTPUT.
```

```
* MODULE STATUS_0100.
```

```
*
```

```
PROCESS AFTER INPUT.
```

```
MODULE user_command_0100.
```

##### Top include

```
PROGRAM sapmbc410adidas_dynpro.
```

##### PAI module include

```
MODULE user_command_0100 INPUT.
```

```
LEAVE TO SCREEN 0.
```

```
ENDMODULE. " user_command_0100 INPUT
```

### Contents:

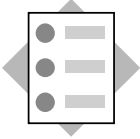
- Overview: GUI titles and GUI statuses
- Creating a GUI status
- Using a GUI status



© SAP AG 1999

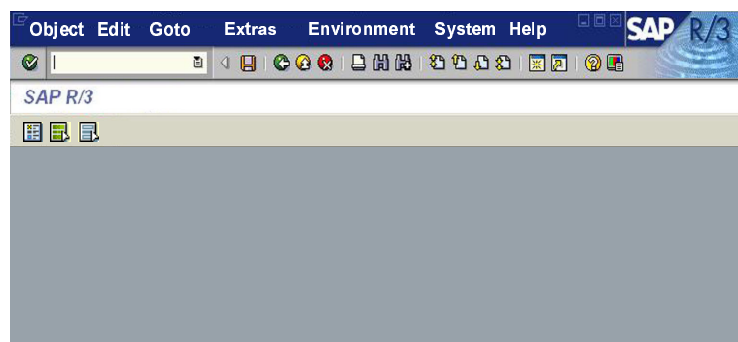
## Unit Objectives

SAP

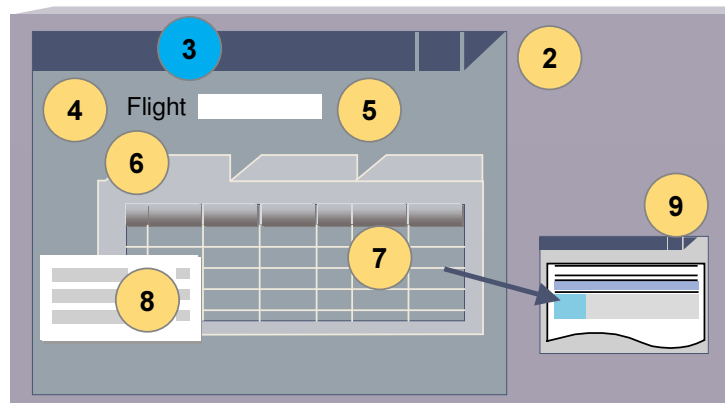


**At the conclusion of this unit, you will be able to:**

- **Create a user interface for a program and use it in user dialogs.**



© SAP AG 2002



© SAP AG 2002

- Unit 1 Course Overview
- Unit 2 Introduction to Screen Programming
- Unit 3 **The Program Interface**
- Unit 4 Screen Elements for Output
- Unit 5 Screen Elements for Input/Output
- Unit 6 Screen Elements: Subscreens and Tabstrip Controls
- Unit 7 Screen Elements: Table Controls
- Unit 8 Context Menus
- Unit 9 Lists in Screen Programming



Overview: GUI titles and GUI statuses

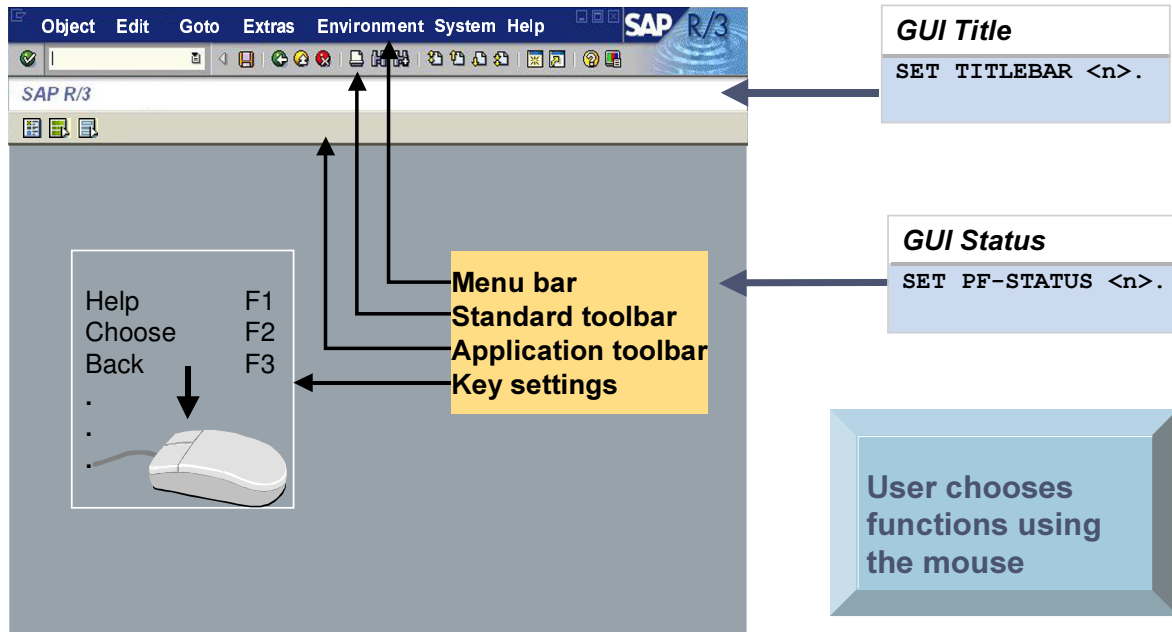
Creating a GUI status

Using a GUI status

© SAP AG 1999

## Overview: Interface

SAP



© SAP AG 2002

- A GUI status is made up of a menu bar, a standard toolbar, an application toolbar, and of function key settings. Each screen can have one or more GUI statuses. For example, an editor program might have two statuses: one for display mode and one for change mode.
- The elements of a GUI status allow users to choose functions 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 System and Help menus are present on every screen in the R/3 System. They always have identical functions and cannot be changed or hidden.
- The application toolbar contains icons for frequently used functions. The standard toolbar, which is the same on every screen in the R/3 System, contains a set of icons, each of which has a fixed assignment to a corresponding function key. If a function in the standard toolbar is not available on the current screen, the icon is grayed out.
- The application toolbar allows the user to choose frequently used functions by clicking the corresponding button.
- You use the function key settings to assign functions such as *Find*, *Replace*, or *Cut* to the function keys.
- All of a program's GUI titles and statuses taken together make up its user interface. Whenever you add a new title or status, you must regenerate the user interface.

```
MODULE status_100 OUTPUT.
...
SET TITLEBAR 'TITLE'.
```



Double-click

**Create Object**

GUI titles TITLE does not exist  
Do you want to create the object?

Yes No ☒ Cancel

**Create Title**

Program

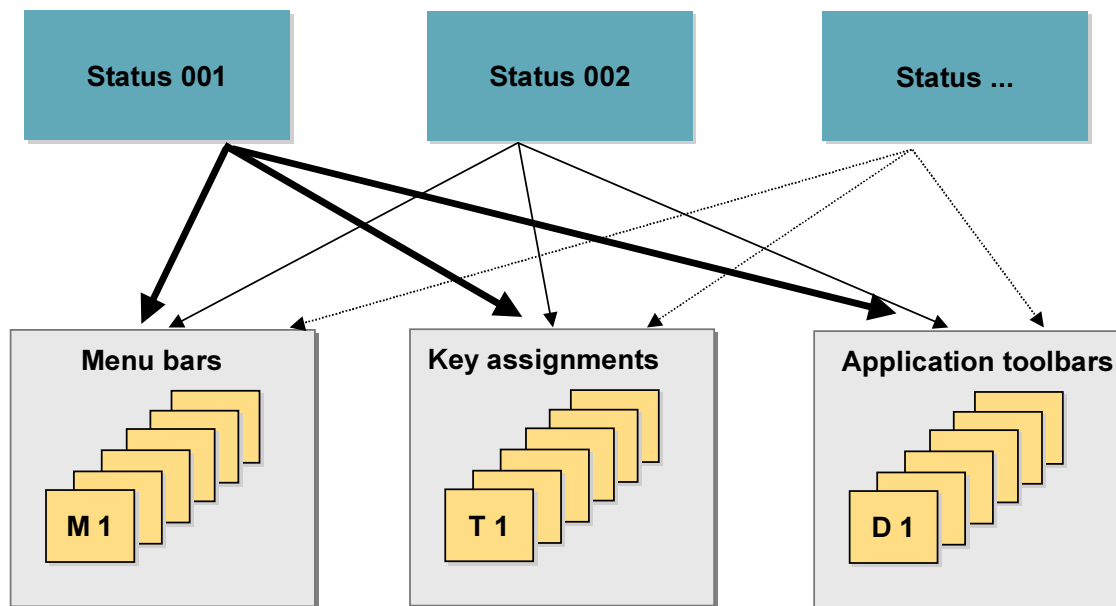
Title code

Title

☒ All title ☒

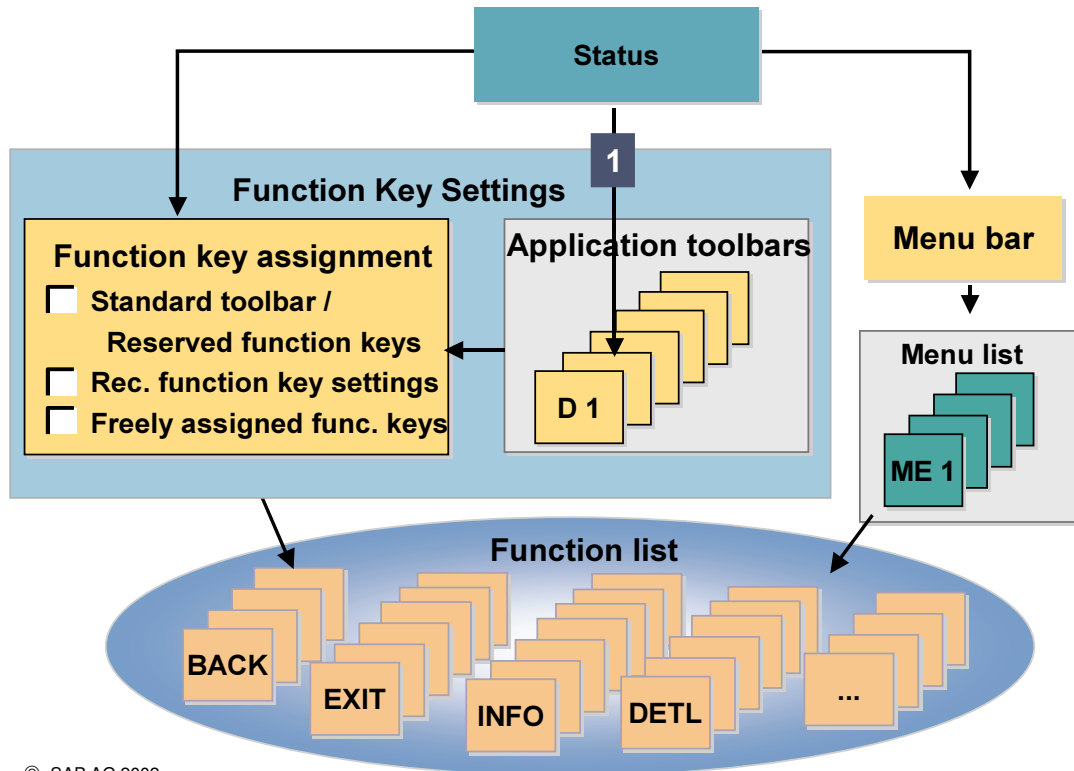
© SAP AG 2002

- There are three ways to create a title: from the object list in the Object Navigator, from the Menu Painter, or by forward navigation from the ABAP Editor.
- The name of a title can be up to 20 characters long.
- You should set an appropriate title for each screen in your application.
- You can use variables in titles that are set dynamically at runtime by including the ampersand character (&) as a placeholder. At runtime, the ampersand is replaced by a value that you specify. You can use up to nine variables by placing digits after the ampersand.  
To set a title that contains variables, use the statement:  
SET TITLEBAR <title\_name> WITH <&1> ... <&9>.
- A title bar remains in place until you set another one. At runtime, the system variable sy-title contains the current title. Title bars are also known as GUI titles.



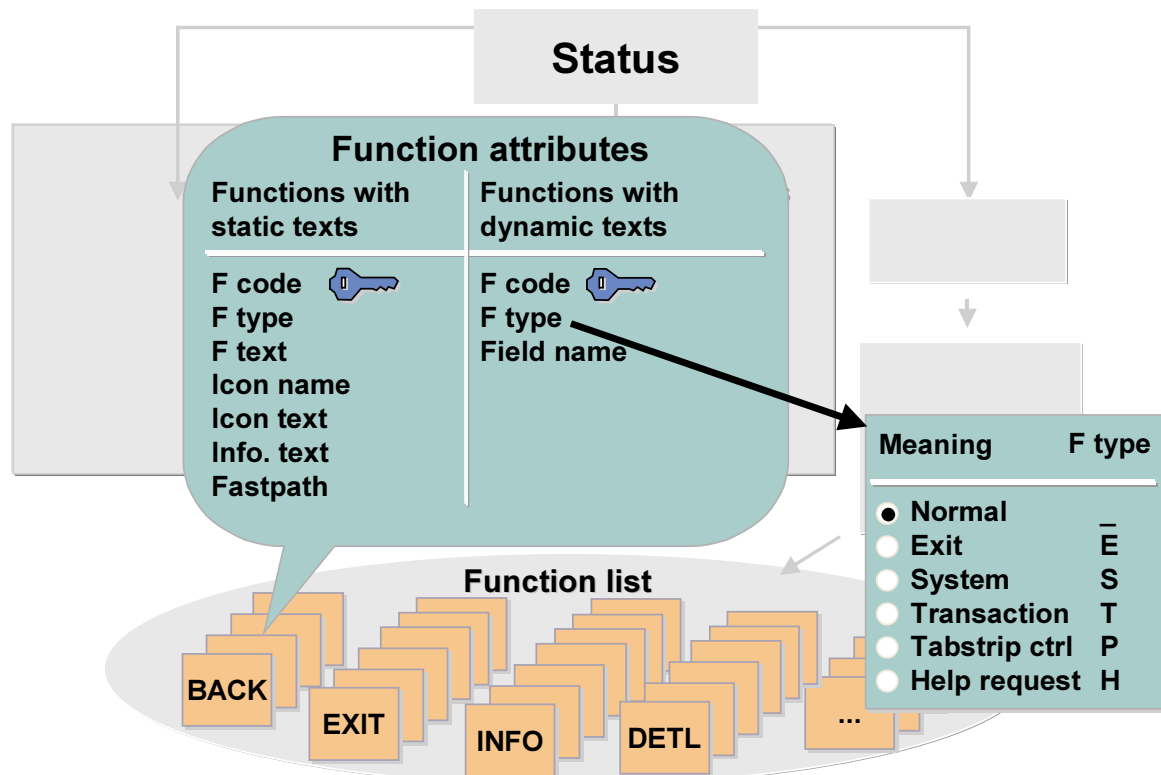
© SAP AG 1999

- From a technical point of view, a status is a **reference** to a menu bar, to certain key assignments, and to an application toolbar.
- A single component (such as a menu bar) can be used by more than one GUI status.
- GUI statuses are ABAP program objects that can be displayed on screens and lists.
- You should set a status for every screen in your application.



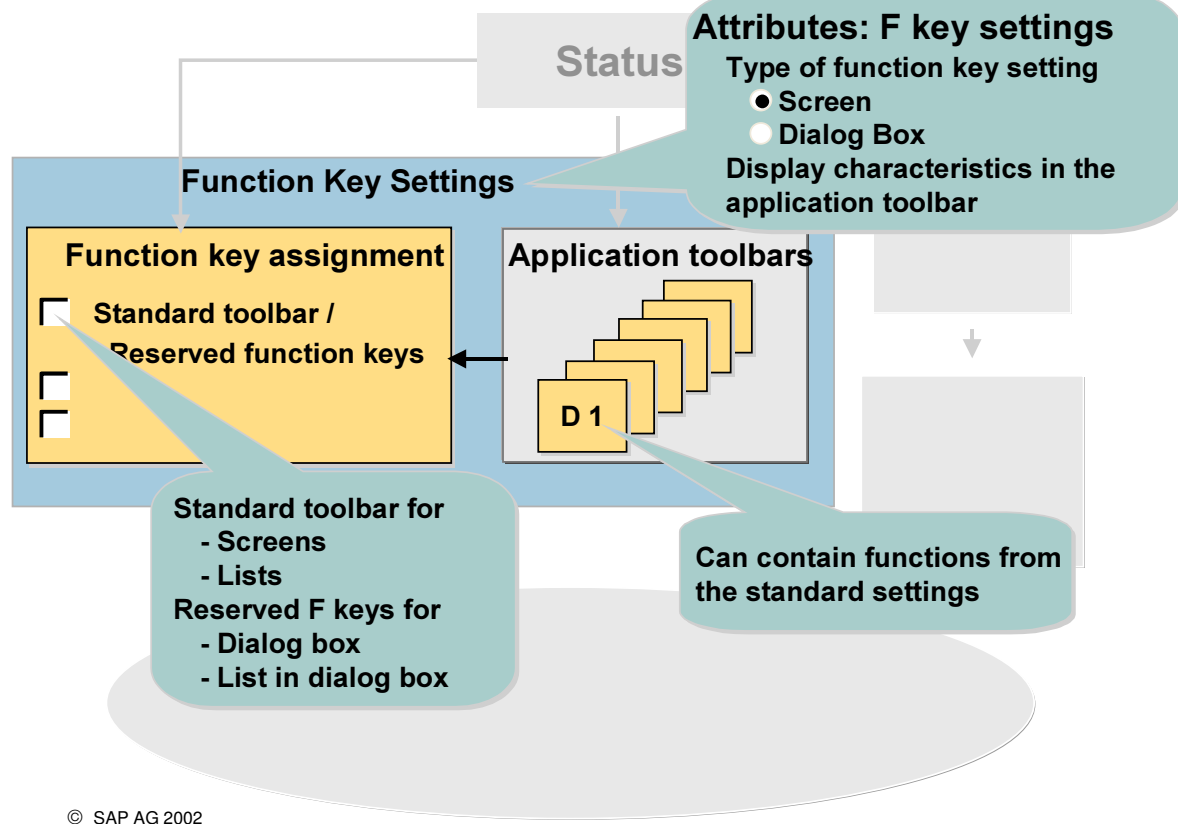
© SAP AG 2002

- A menu bar is made up of individual menus.
- Key assignments and application toolbars are subobjects of the function key settings.
- You can create a set of application toolbars for a single key setting by choosing the menu path **GUI-1**. Functions must be assigned to a function key before you can assign them to a pushbutton. Each status contains a **single** application toolbar.
- All program menus and key assignments refer to the set of all interface functions (function list). These functions can be reached using F4 help. The application toolbar refers to the functions indirectly via the standard settings.
- A function within a status can be either active or inactive.

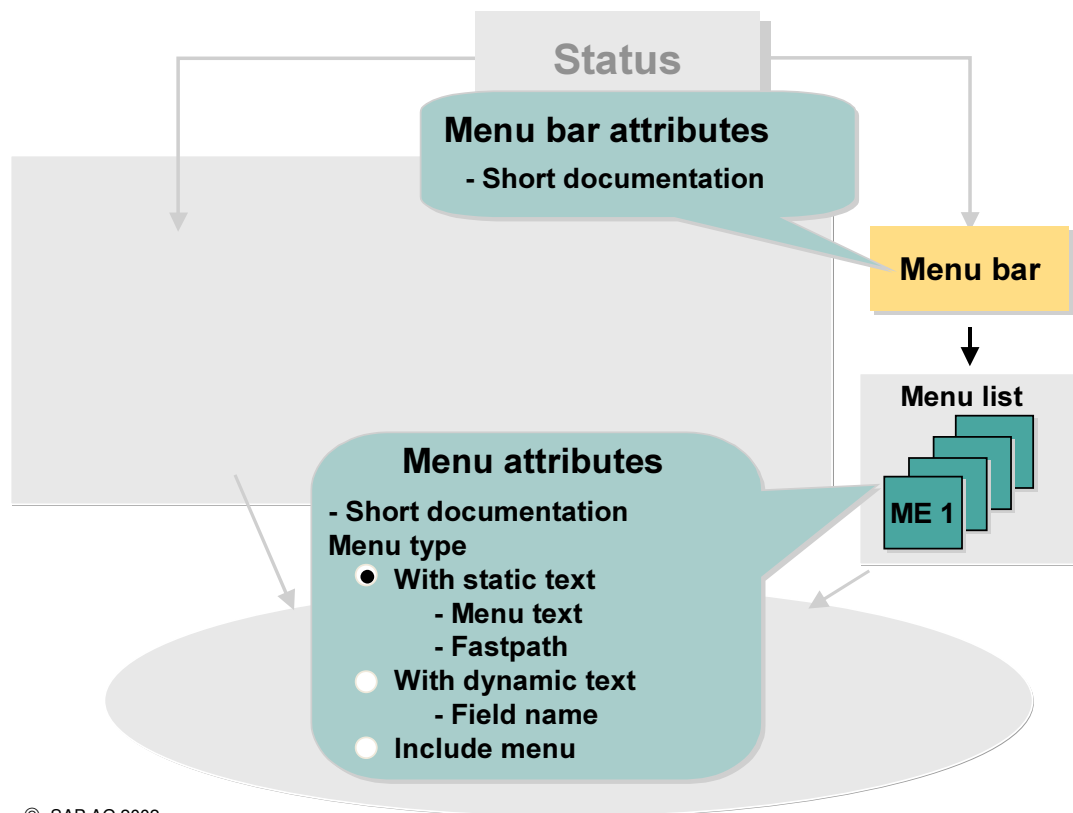


© SAP AG 2002

- Functions are identified by their *function codes*.
- The attribute *function type* determines the intended purpose of a function. Use the function types ' ' (space), E, and P for pushbuttons, which you place on a screen using the Screen Painter, and for tab titles. Function types S and H are reserved for internal use by the SAP System. Function type T indicates a transaction code. When a function of this type is triggered, the system leaves the calling program and calls the new program.
- Functions can be created with static texts or dynamic texts.
- If a function has a static text, you can assign an icon to it (*Icon name* attribute). If the function is already assigned to a pushbutton, an icon is displayed instead of the static text. The static text is used when you assign the function to a menu entry. The function text belonging to the function is used as quick info text. The contents of the *Infotext* attribute appear in the status bar of the screen when the user chooses the function. If you want to display text as well as the icon, enter the text in the *Icon text* attribute.
- You can use the *Fastpath* attribute to specify the letters that allow you to choose a function from the menu bar without using the mouse.
- For further information, refer to the online documentation path in appendix reference **GUI-2**.



- Functions can be assigned to individual function keys or buttons.
- Function key settings consist of a key assignment and an application toolbar pushbutton.
- The type of function key settings (screen and dialog box) determines the exclusive technical purpose of the function key setting. In addition, you can set options for the implementation of context menus and input help on lists. For more information on using context menus in lists, see the *Context Menus* unit later in this course.
- Key assignments consist of *reserved functions keys*, *recommended functions keys*, and *freely assigned function keys*. *Reserved functions keys* are function keys whose assigned values cannot be changed in the SAP system. You may activate and deactivate their functions, but you cannot change the icons and texts assigned to them. Reserved function keys appear in the standard toolbar on screens and lists. *Recommended function keys* contain proposals, which comply with the SAP System's ergonomic standards.
- Functions that have been assigned to function keys can also be assigned to buttons in the application toolbar.
- An application toolbar can contain up to 35 buttons. You can insert vertical separators in the button bar to group buttons visually. You can control the display of inactive functions in the application toolbar by choosing *Goto → Attributes → Pushbutton settings*.



© SAP AG 2002

- A menu can contain up to 15 entries.
- Possible entries are functions, separators, and menus (cascading menus).
- Menus can be up to three levels deep. The third level may contain only functions and separators.
- Menus can be created with static or dynamic text. If you want to use dynamic text, you must assign a field to the menu. The contents of this field will be displayed as the menu text.
- The menu type *Include menu* allows you to reference menus in other programs. When you do this, you must specify the name of the program and status from which you want to include the menu next to the *Short documentation* field.
- Include menus can be accessed only using 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.

Overview: GUI titles and GUI statuses



Creating a GUI status

Using a GUI status

© SAP AG 2002

Internal Use SAP Partner Only

## Creating a GUI Status

SAP

```
MODULE status_100 OUTPUT.
```

```
...  
SET PF-STATUS 'BASE'.
```



Double-click

Create Object

GUI status BASE does not exist.  
Do you want to create the object?

Yes

Create Status

Program: xxxxx

Status: BASE

Status attributes

Short text: Status for flight

Status type

- ☒ Online status
- ☐ Dialog box
- ☐ Context menu

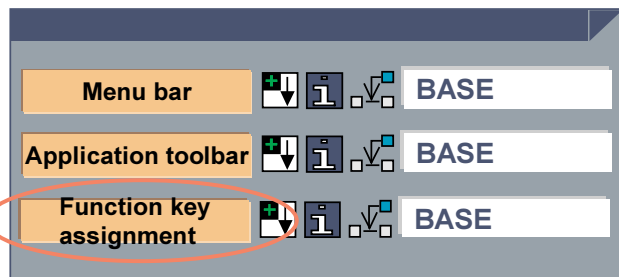
✓ ✗

© SAP AG 2002

- 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 that can be displayed with a right-click. We deal with them separately in the Context Menus unit.
- You can create a status by creating links to existing components or by creating a new blank status. You can also combine the two techniques. If you want to create an entirely new status, you must then create your own menu bars, menu functions, and other elements. Changes to a status only affect that status.
- When you use the reference technique, you create menu bars, application toolbars, and function key assignments as independent elements. You then create your own status and refer to the menu bar, application toolbar, and any function key assignment you want. The Menu Painter stores and maintains these references so that any changes in the menu bar, application toolbar, or function key assignments automatically take effect in all statuses referring to them.
- The linking technique is particularly effective for ensuring consistency in very large applications that use several statuses. The links ensure that the user can access functions in the same way whatever status is set.

## Creating a GUI Status: Function Key Assignment

SAP



### Standard toolbar/ Recommended function key settings

- Corresponding status type
- Adjust list functions

### Freely assigned function keys

- Enter F code, max. 20 characters
- Double-click to maintain attributes

### Function attributes

#### Functions with static texts

F code  
F type  
F text  
Icon name  
Icon text  
Info. text  
Fastpath

#### Functions with dynamic texts

F code  
F type  
Field name

© SAP AG 2002

- In a key setting, you assign individual functions to function keys and pushbuttons. Function key settings consist of a key assignment and a set of application toolbars.
- Key settings can have various types (screen, dialog box, list, and list in dialog box).
- You can attach functions to reserved function keys, recommended function keys, and freely assigned function keys. Make sure that they conform to the SAP System's ergonomic standards, which can be found in the *Environment* menu in the Menu Painter.
- Reserved function keys appear in the standard toolbar on screens and lists.
- If a function is important, and you have already assigned it to a function key, you can also assign it to a pushbutton in the application toolbar. The application toolbar may contain up to 35 buttons.

## Standard Toolbar: Automatic Assignments



Icon	Function key	Meaning
	Enter	
	Ctrl-S	Save
	F3	Back
	Shift-F3	Exit (program)
	F12	Cancel (screen)
	Ctrl-P	Printing
	Ctrl-F	Find
	Ctrl-G	Find next
	Ctrl-Page up	First page
	Page up	Previous page
	Page down	Next page
	Ctrl-Page down	Last page
	F1	Help













© SAP AG 2002
















- When you assign a function to the standard toolbar it is also automatically assigned to a reserved function key.
- To find out the function keys to which these functions are assigned in the current status, select the *Information* in the Menu Painter.
- For more information about how key combinations such as Ctrl-P are converted into internal function key numbers (for example, for batch input), follow menu path **GUI-3** in the Menu Painter.

## Creating a GUI Status: Application Toolbar

SAP

Maintain Status

Menu bar	   	BASE
Application toolbar	   	BASE
Function key assignment	   	BASE

Items	1 - 7	FUN1			
Items	8 -14				
Items	15 -21				
Items	22 -28				
Items	29 - 35				



© SAP AG 2002

- You can use a function in the application toolbar only if you have already assigned it to a function key.
- Use the F4 help to select functions.
- If you assign an icon to a function with a static text (*Icon name* attribute), the system displays the icon instead of the static text in the application toolbar. The function text belonging to the function is used as quick info text. The contents of the *Infotext* attribute appear in the status bar of the screen when the user chooses the function. If you want to display additional text with an icon, it should be entered in the *Icon text* attribute.
- To insert a separator in the application toolbar, use the *Insert* menu in the Menu Painter.

## Creating a GUI Status: Menu Bar

SAP

Maintain Status		
Menu bar	[Function Code]	BASE
Application toolbar	[Function Code]	BASE
Function key assignment	[Function Code]	BASE

### Menu bar attributes

- Short documentation

### Menu bar

- Display standards
- Maintain menus (<List>)

### Menu attributes

- Short documentation

#### Menu type

With static text

- Menu text
- Fastpath

With dynamic text

- Field name

Include menu






© SAP AG 2002

- A menu entry can be a function, a separator, or another menu (cascading menu).
- To add a function to a menu, enter its function code in the left-hand column. If the function already exists in the function list and has a text assigned to it, this is entered automatically in the text field. If not, double-click the right-hand field to enter a text.
- To insert a separator, use the Insert menu, fill the function text field with minus signs at the appropriate position or use the Context menu.
- To create a submenu, simply enter its name in the right field of the menu entry.

## Displaying Standards


SAP

Select reserved  
function keys

Icon	Function key	Function code
	Enter	
	Ctrl-S	SAVE
	F3	BACK
	Shift-F3	EXIT
	F12	CANCEL



Menu bar

 Display standards

Objects Edit Goto Extras Environ.

Application toolbar

Function key assignment

Status\_100

Status\_100

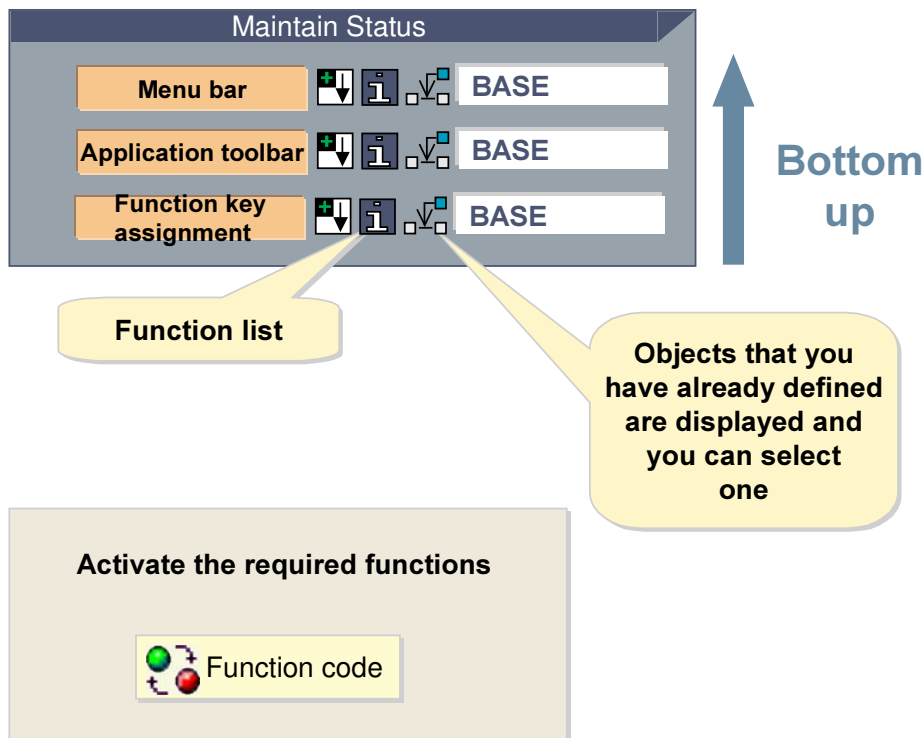
Status\_100

© SAP AG 2002

- To ensure consistency, you should reuse existing menu bars, application toolbars, and key settings wherever possible. The Menu Painter administers the links you establish between these objects so that any changes apply to all other statuses that use them. You can also use a set of standard menu entries as a template and modify them.
- When you assign functions to the reserved function keys in the standard toolbar, you should adhere to the SAP System standards. This makes your program easier for users to understand and for you to maintain.

## Including Existing Elements

SAP



© SAP AG 2002

- Using the Menu Painter in a status, you can include key settings, application toolbars, or menu bars that you have already defined elsewhere. If you do this, work from the bottom upward. If there is more than one application toolbar defined for your key setting, you can choose the appropriate one.
- Initially, all functions are inactive. Activate only the functions that are relevant in the current status.
- When you create a new function, you can decide whether all statuses that refer to the same object should also be changed. The new functions are initially inactive.

Overview: GUI titles and GUI statuses

Creating a GUI status

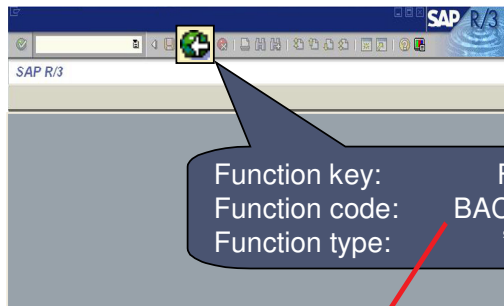


Using a GUI status

© SAP AG 2002

## Processing the Function Code

SAP



PROCESS AFTER INPUT.  
...  
MODULE user\_command\_100.

Screen Painter

DATA:ok\_code TYPE sy-ucomm.

ABAP

```
MODULE user_command_100 INPUT.
  CASE ok_code.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.
```

General attributes	
Field name	Type ...
...	
OK_CODE	OK

Screen Painter  
Element list

© SAP AG 2002



The ok\_code ABAP data field still contains the old function code after processing

- When the user triggers a function with type '' using a button, menu entry, or function key, the system places the relevant function code in the OK\_CODE field of the screen.
- To allow you to process this field in the PAI event, you must assign a name to the field, which you then enter in the element list in the Screen Painter. You must then create a field in your ABAP program with the same name. During the automatic field transport at the beginning of the PAI event, the function code is passed from the screen to the corresponding field in the 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 identically named ABAP field. The initial value is then automatically transported to the corresponding command field at PBO.

## PBO

or

## PAI

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

Screen Painter

```
PROCESS AFTER INPUT.
...
MODULE save_ok_code.
MODULE user_command_0100.
```

Screen Painter

```
DATA:ok_code TYPE sy-ucomm.

MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.

MODULE user_command_100 INPUT.
  CASE ok_code.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.
```

ABAP

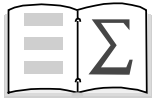
```
DATA:ok_code TYPE sy-ucomm,
      save_ok LIKE ok_code.

MODULE save_ok_code INPUT.
  save_ok = ok_code.
  CLEAR ok_code.
ENDMODULE.

MODULE user_command_100 INPUT.
  CASE save_ok.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.
```

ABAP

- You can initialize the command field at PAI or at PBO.
- The simplest method is to insert a module at the end of PBO in which the identically named ABAP field is initialized. The automatic data transport initializes the command field.
- Another method is to copy the OK\_CODE field explicitly to a field of similar type, and then to initialize it.
- Note: You cannot combine the two methods. If you do this, errors may occur in the processing of the screens that initialize the OK\_CODE field at PAI.



**You are now able to:**

- **Create a user interface for a program and use it in user dialogs.**

# Exercises



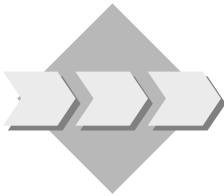
## Unit : The Program Interface

### Topic: Creating a GUI status and using it with a screen



At the conclusion of these exercises, you will be able to:

- Create online statuses and use them in your programs.



Use Menu Painter to create GUI Status and GUI Title.

- 1-1 Create a GUI Title and a GUI Status for a screen.

Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410ADIAS\_DYNPRO**). You can use the model solution **SAPMBC410ADIAS\_GUI** for orientation.

- 1-2 In the PBO event of screen 100, call a module **status\_0100**. Use forward navigation to create the module in a new include. Accept the system's proposal for the name (**MZ##BC410\_SOLUTIONO01**). In this module, set the GUI status **STATUS\_100** and GUI title **TITLE\_100** (Flight data). You can create the status and title by forward navigation.

Choose *Online status* for the Status type and *Maintenance screen* for Short text. Activate the standard function BACK (F3) with the function type ' ' (space).

- 1-3 Assign the name **OK\_CODE** to the function code field on your screen, and create a corresponding variable in the top include of your program.

- 1-4 Implement the command field processing. Use forward navigation to create the PAI modules in a new include. Accept the system's proposal for the name (**MZ##BC410\_SOLUTIONI01**). Ensure that the user returns from screen 100 to the point from which it was called if he or she chooses BACK (F3) **only**.

- 1-5 To avoid unnecessary navigation, initialize the command field in a module at the PBO event of the screen.



## Unit 3: The Program Interface

### Topic: Creating a GUI status and using it with a screen

## Solution SAPMBC410ADIAS\_GUI

### Main program

```
INCLUDE MBC410ADIAS_GUITOP.  
  
INCLUDE MBC410ADIAS_GUIII01.  
  
INCLUDE MBC410ADIAS_GUIIO01.
```

### Flow logic for screen 100

```
PROCESS BEFORE OUTPUT.  
  
    MODULE status_0100.  
  
    MODULE clear_ok_code.  
*  
PROCESS AFTER INPUT.  
  
    MODULE user_command_0100.
```

### Top include

```
PROGRAM sapmbc410adidas_gui.  
  
DATA ok_code TYPE sy-ucomm.
```

## PAI module include

```
MODULE user_command_0100 INPUT.  
  CASE ok_code.  
    WHEN 'BACK'.  
      LEAVE TO SCREEN 0.  
  ENDCASE.  
ENDMODULE.                                " user_command_0100  INPUT
```

## PBO module include

```
MODULE status_0100 OUTPUT.  
  SET PF-STATUS 'STATUS_100'.  
  SET TITLEBAR 'TITLE_100'.  
ENDMODULE.                                " status_0100  OUTPUT  
  
MODULE clear_ok_code OUTPUT.  
  CLEAR ok_code.  
ENDMODULE.                                " clear_ok_code  OUTPUT
```

### Contents:

- Text fields
- Status icons
- Group boxes

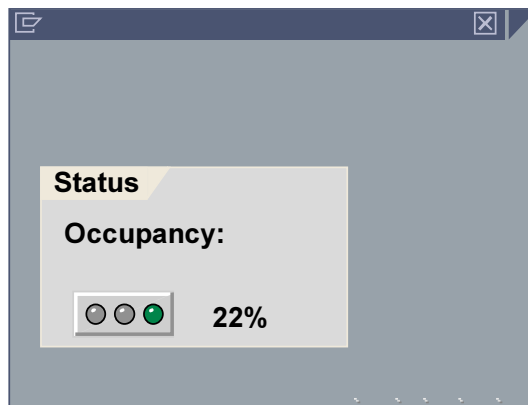


© SAP AG 1999

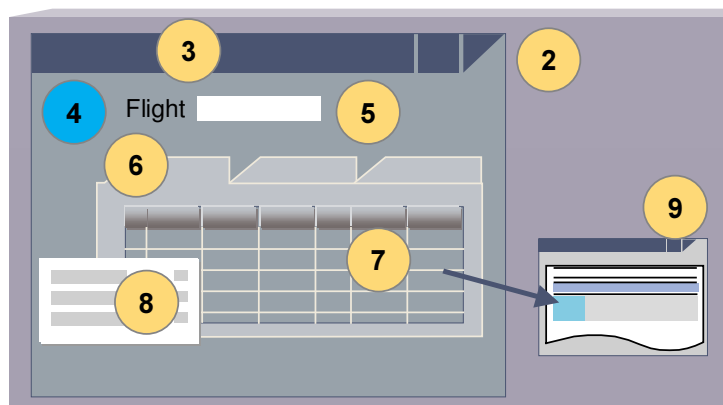


**At the conclusion of this unit, you will be able to:**

- **Create and change text fields, status icons, and group boxes.**



© SAP AG 2002



© SAP AG 2002

- |          |   |
|----------|---|
| ■ Unit 1 | Course Overview                                   |
| ■ Unit 2 | Introduction to Screen Programming                |
| ■ Unit 3 | The Program Interface                             |
| ■ Unit 4 | <b>Screen Elements for Output</b>                 |
| ■ Unit 5 | Screen Elements for Input/Output                  |
| ■ Unit 6 | Screen Elements: Subscreens and Tabstrip Controls |
| ■ Unit 7 | Screen Elements: Table Controls                   |
| ■ Unit 8 | Context Menus                                     |
| ■ Unit 9 | Lists in Screen Programming                       |

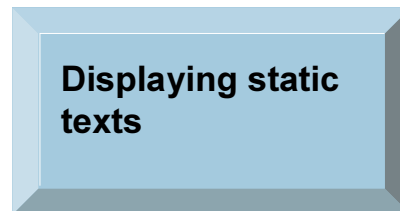
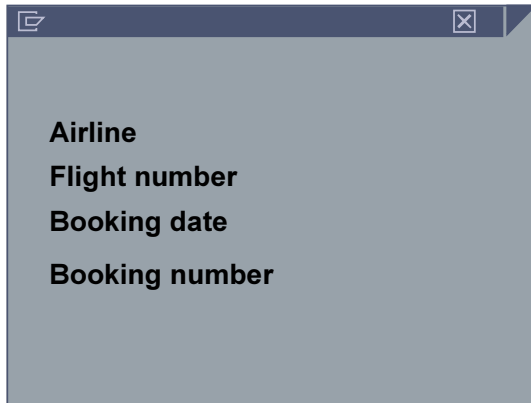


**Text fields**

**Status icons**

**Group boxes**

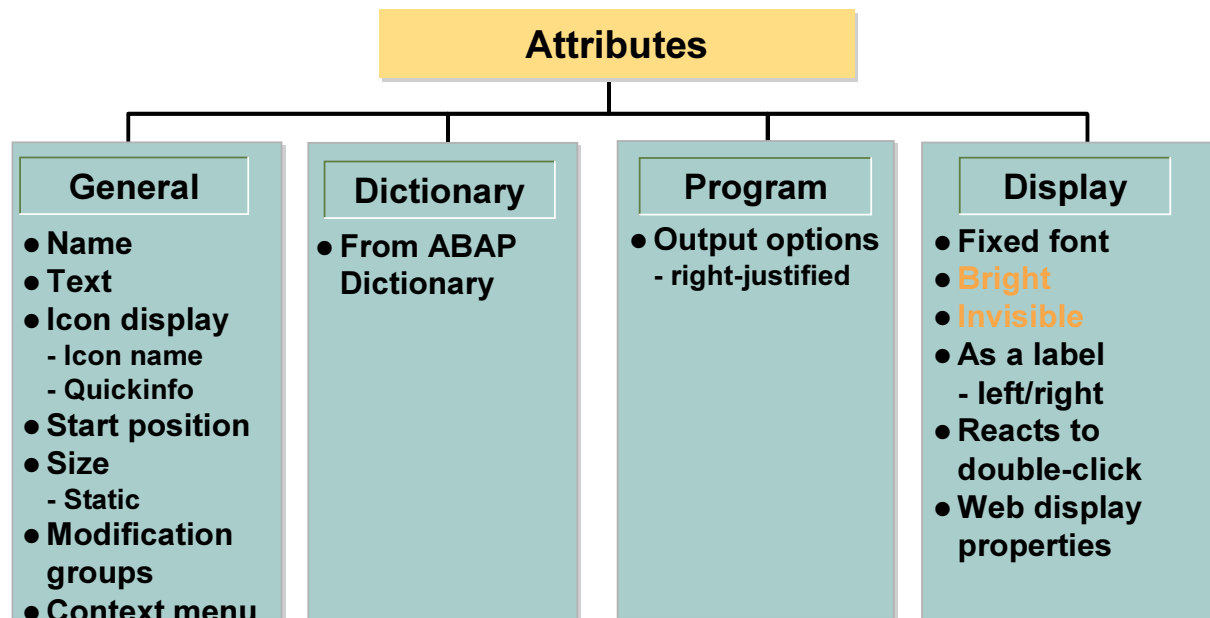
© SAP AG 1999



- **Multilingual**

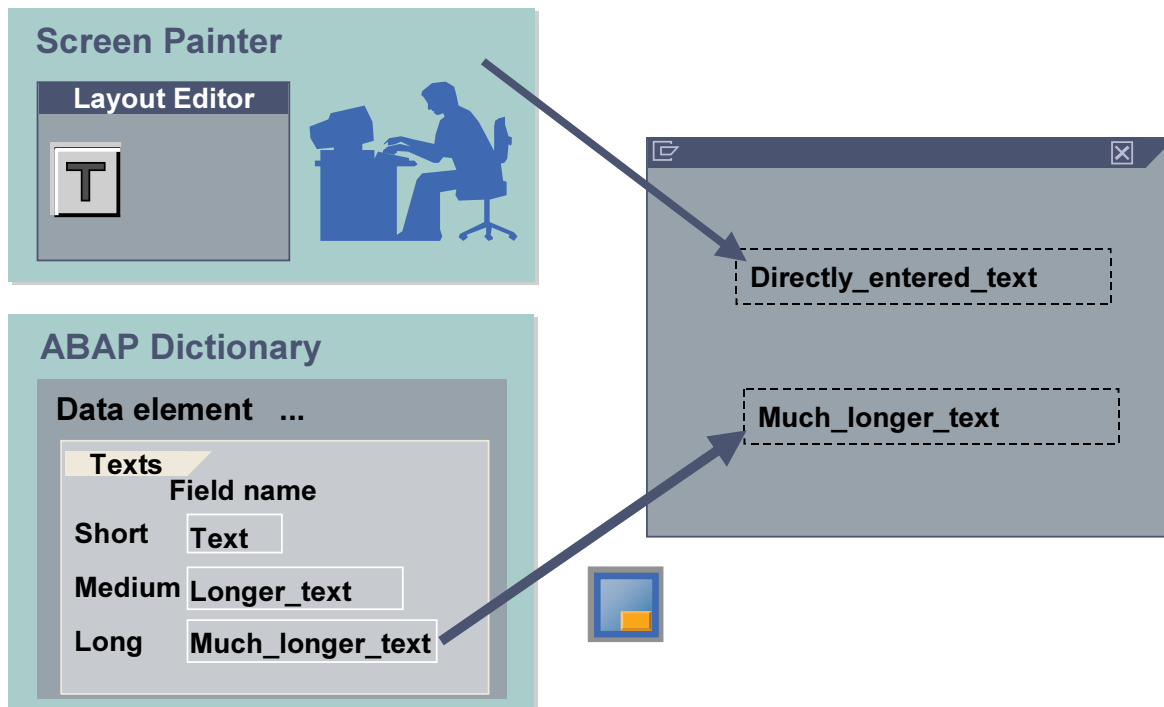
© SAP AG 2002

- A text field is a rectangular area on a screen in which the system displays text.
- Text fields contain labels for other elements. These labels (sometimes called keywords) are purely for display. The user cannot change these labels at run time. Text fields are displayed in a fixed position on the screen.
- Text fields can also contain lines, icons, and other static elements. They can contain any alphanumeric characters, but may not begin with an underscore (\_) or a question mark (?). If you use text to label a radio button or check box, you can specify whether the label is to have a *Left button* or a *Right button*.
- If your text consists of more than one word, use underscore characters as separators. This enables the system to recognize that the different words in fact belong together. The system interprets spaces as separators between two different text fields.
- Text fields can be translated. They then appear in the user's logon language. To do this, follow the menu path under **OUT-1**.



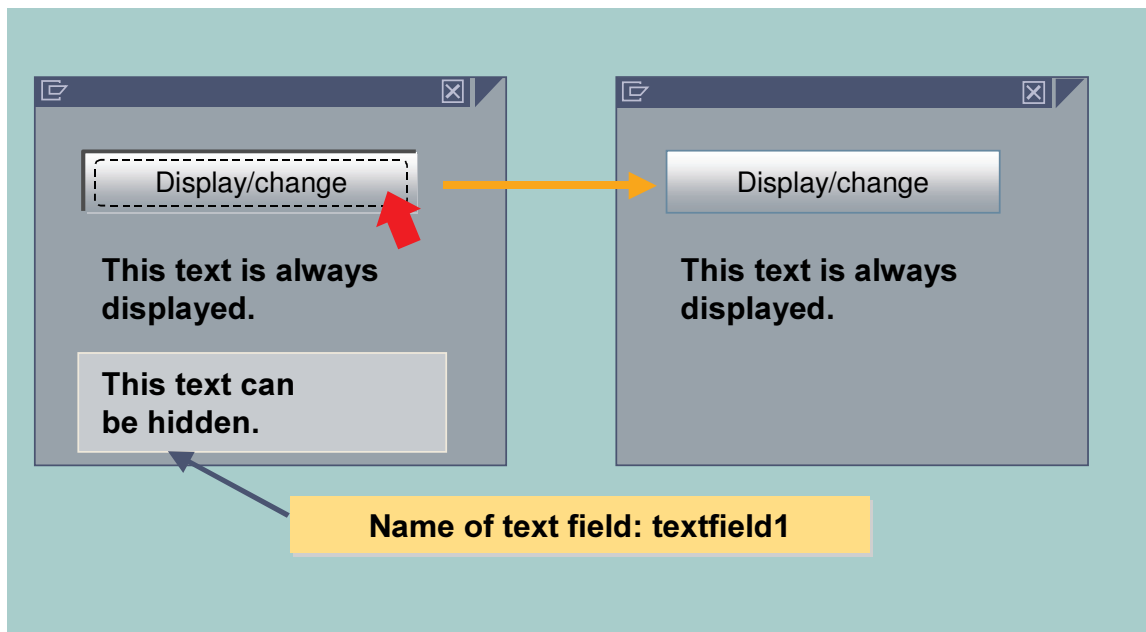
© SAP AG 2002

- At run time, you can change the size (*visible length*) of a text field and the display attributes *Bright* and *Invisible*. To do this, use the SCREEN-LENGTH, SCREEN-INTENSIFIED, and SCREEN-INVISIBLE (or SCREEN-ACTIVE) fields.



© SAP AG 2002

- You can create text fields in either of the following ways:
  - Directly in the layout editor, by placing a text field object in the work area and entering the text in the Text attribute.
  - By using the accompanying text of a data element from the ABAP Dictionary.
- When you use fields from ABAP Dictionary structures on the screen, the system normally displays the data element and the template for the input/output fields on the screen.



© SAP AG 2002

- You can make text fields invisible at run time.
- If you make an element invisible that is enclosed in a box, the box is not displayed either.



SCREEN-NAME	Object name
SCREEN-GROUP1	Modification group 1
SCREEN-GROUP2	Modification group 2
SCREEN-GROUP3	Modification group 3
SCREEN-GROUP4	Modification group 4
SCREEN-REQUIRED	Required entry
SCREEN-INPUT	Ready for input
SCREEN-OUTPUT	Ready for output
SCREEN-INTENSIFIED	Intensified
SCREEN-INVISIBLE	Invisible
SCREEN-LENGTH	Object length
SCREEN-ACTIVE	Active object
SCREEN-DISPLAY_3D	Display object in 3D
SCREEN-VALUE_HELP	Field with value help
SCREEN-REQUEST	Input exists (only in PAI)



© SAP AG 2002

- When the PBO module is processed, the system table with the line type SCREEN is initialized by the runtime environment, and filled with the static attributes from the Screen Painter.
- To hide a text field at run time, you modify the system table. Use LOOP AT SCREEN. ... MODIFY SCREEN. ENDLOOP.
- To make a text field invisible, use SCREEN-INVISIBLE = 1 or SCREEN-ACTIVE = 0.

```
PROCESS BEFORE OUTPUT.
.
.
MODULE modify_screen.
.
.
```

Flow  
logic

```
MODULE modify_screen OUTPUT.
...
LOOP AT SCREEN.
  IF screen-name = 'TEXTFIELD1'.
    screen-active = 0.
    MODIFY SCREEN.
  ENDIF.
ENDLOOP.
ENDMODULE.
```

ABAP

© SAP AG 2002

- To dynamically hide the TEXTFIELD1 field, you can call a module in the PROCESS BEFORE OUTPUT processing block that sets the invisible attribute for that field.
- To do this, set the contents of the SCREEN-ACTIVE field to 0.
- Note: SCREEN is a system internal table. The system does **not** support the statements LOOP AT SCREEN WHERE... and READ TABLE SCREEN.

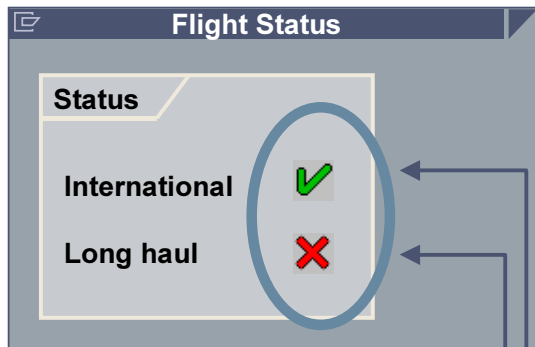
Text fields



Status icons

Group boxes

© SAP AG 1999



## Displaying icons

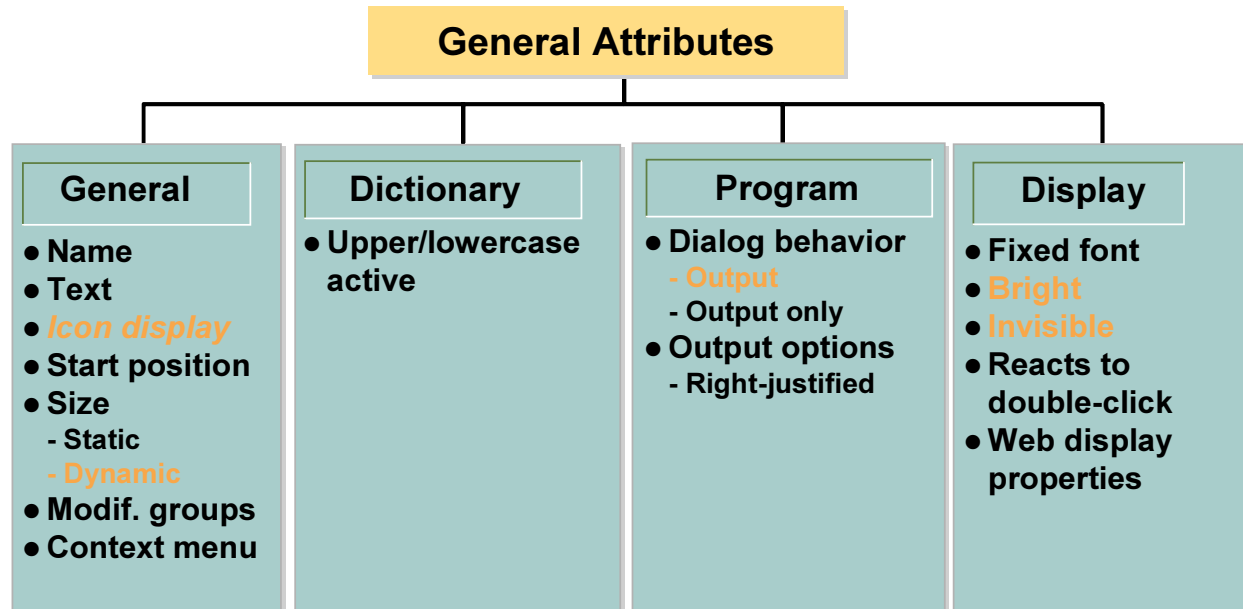
- Icon selection at runtime (dynamic icons)
- Graphical elements improve screen layout

### Conditions:

countryfr NE countryto  
distance GE 1000

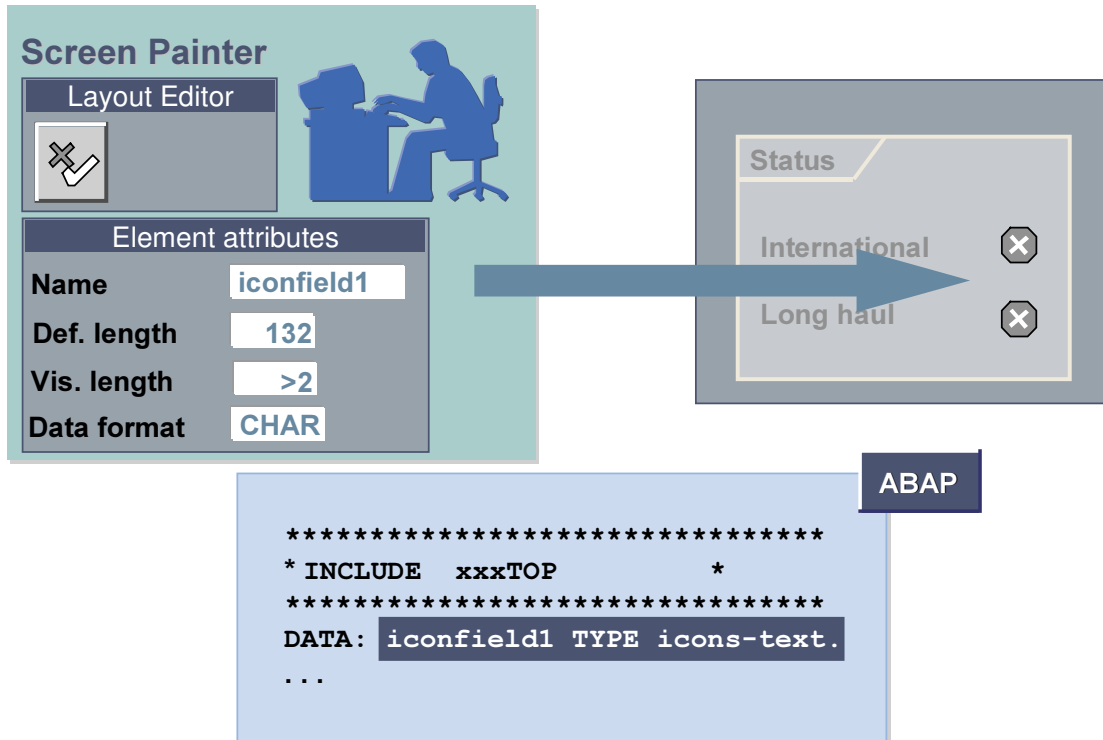
© SAP AG 2002

- A status icon is an output field that contains an icon. You choose the relevant icon at run time. Icons allow you to indicate a status in your application. They are predefined in the system, and take up between two and four characters.
- For information about the available icons, see the online documentation (reference **OUT-2**).



© SAP AG 2002

- Status icons are special output fields that display icons. The system sets the attributes *Output field* and *2 dimensional*, which cannot be changed. The default data format is CHAR.
- You can change the *Visible length*, *Intensified*, and *Invisible* attributes of a status icon dynamically.

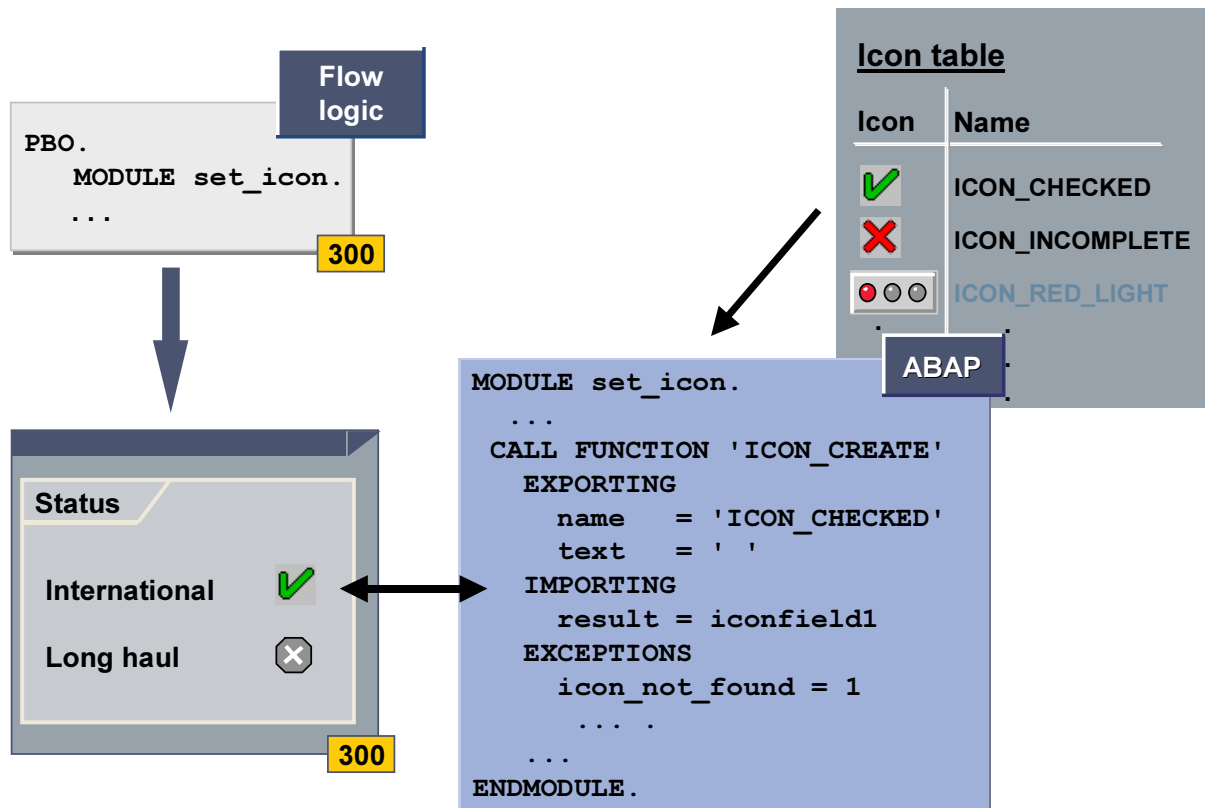


© SAP AG 2002

- You can only define a status field in the graphical layout editor. A status field is an output field with an icon. You use it to display an icon, which you specify dynamically at run time.
- To ensure that you can display the *Quick info* that might be longer, define the field with defined length of 132 and visible length two.
- In the ABAP program, define a field with the same name as the screen field using the TEXT field from the ICONS structure. At run time, this field contains the name of the icon you want to display.
- At run time, assign the required icon to this field using the **ICON\_CREATE** function module.

## Filling a Status Icon

SAP



© SAP AG 2002

- You select the icon you want to display from the ABAP program. Before the screen is displayed, you need to find out the technical name of the icon. You do this by calling a module in the PBO event.
- You retrieve the technical name of an icon using the `ICON_CREATE` function module. You must pass the name of the icon you want to display to the function module. You can also pass a text to be displayed with the icon. The function module returns the technical name of the icon.
- For further details about this function module, refer to its documentation.

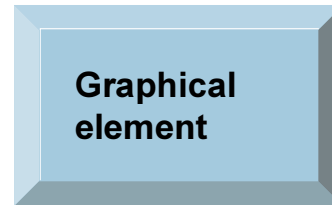
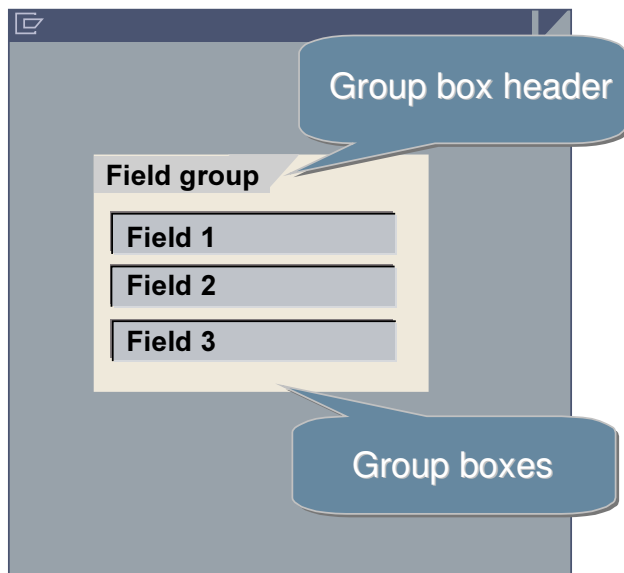
Text fields

Status icons



Group boxes

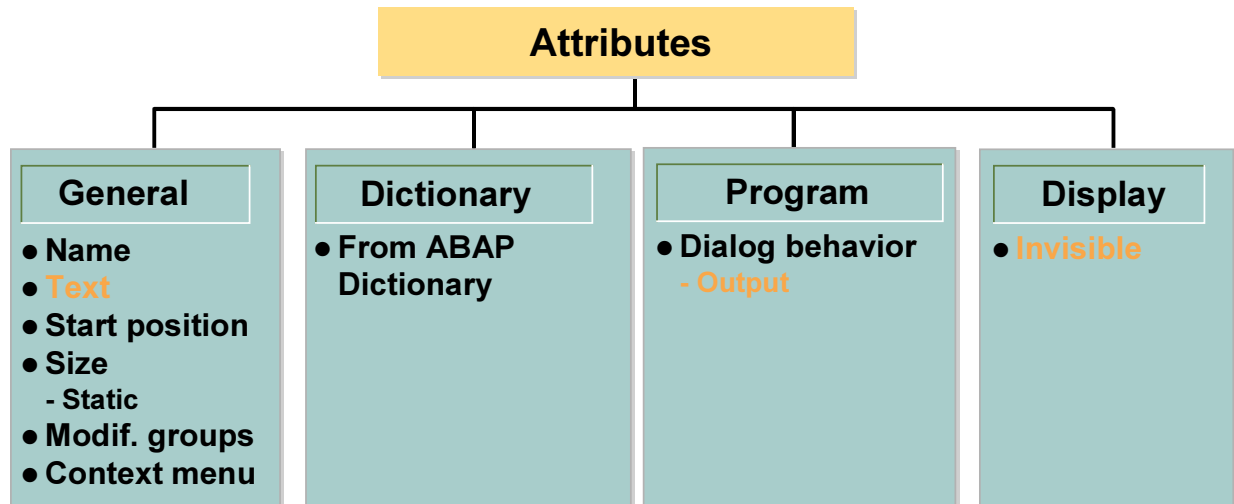
© SAP AG 1999



- Visual bundling of objects on the screen
- Graphical elements improve screen layout

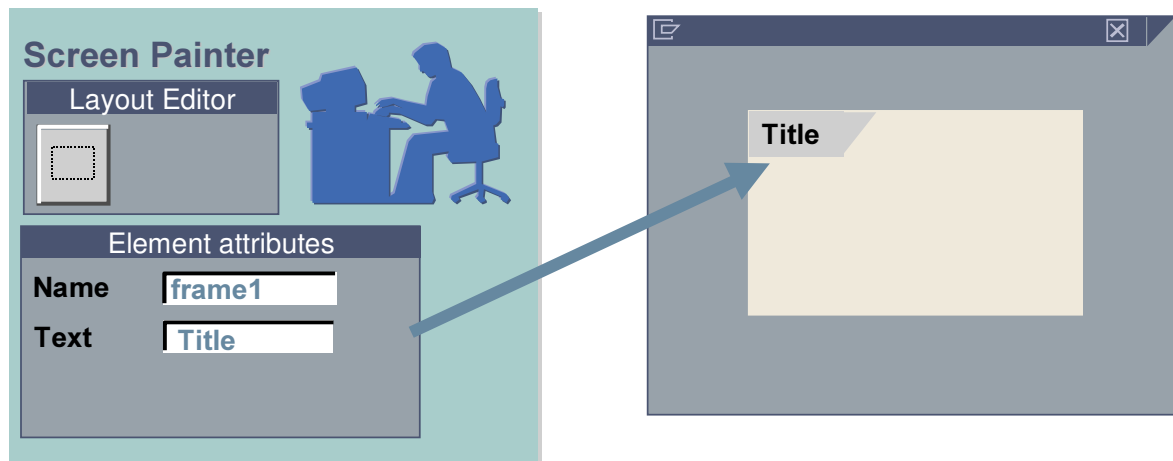
© SAP AG 2002

- Group boxes enclose a selection of elements that belong together (for example, a group of fields or a radio button group). They are purely display elements, which help the user to identify which elements on the screen belong together in a group.
- You can use group boxes to make sure that all fields within a box have the same context menu assigned to them. For further information, refer to the *Context Menus* unit.
- Group boxes can have a title.



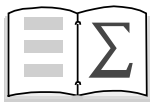
© SAP AG 2002

- You can change the *Visible length* and *Invisible* attributes using the `SCREEN` system table.
- A group box may contain other screen elements.
- At run time, if the box contains only invisible elements and the screen attribute *Runtime compression* is set, the box itself is not displayed.



© SAP AG 2002

- You define a group box in the layout editor. The object must have a name; you may also assign a heading to the box.
- You can change the group box text dynamically. To do this, you should activate the *output field* attribute and create a global data field in the ABAP program with the same name. Because the Screen Painter field and the program field have the same name, any changes to the field contents will be immediately visible on the screen (similarly to input/output fields).



**You are now able to:**

- **Create and change text fields, status icons, and group boxes.**

© SAP AG 2002

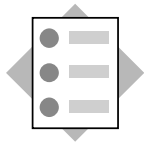
- The main purpose of output elements is to make your screens clearer.
- Text fields allow you to add labels to input and output fields. Make sure the text field and the input/output field have the same name. If you deactivate an input/output field, the system automatically hides the text field associated with it.
- Status icons are an excellent way of presenting information so the user can grasp it at a glance.
- Group boxes group fields that belong together logically. Runtime compression ensures that empty group boxes are hidden.
- Static texts on the screen can be translated. They then appear in the user's logon language. To make dynamically assigned texts available in several languages, use ABAP program text elements.

### Contents:

- Input/output fields
- Input help
- Checkboxes and radio button groups
- Pushbuttons

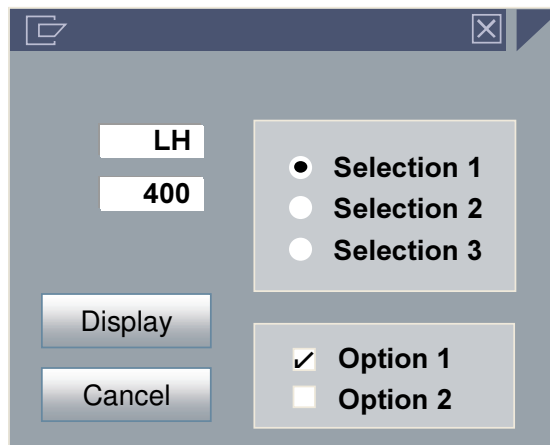


© SAP AG 1999



**At the conclusion of this unit, you will be able to:**

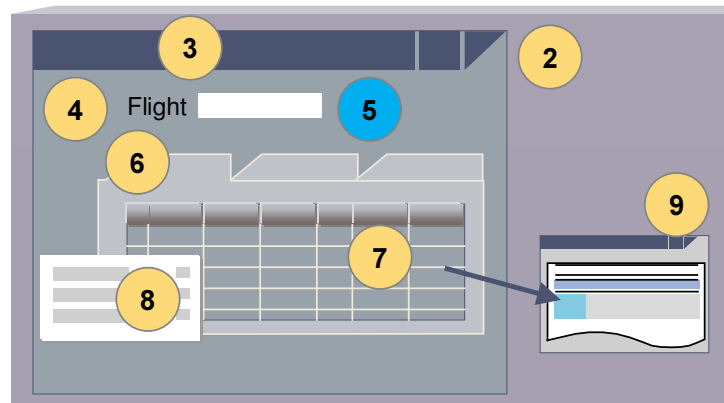
- **Use input/output fields, checkboxes, radio button groups, and pushbuttons in your programs.**



The screenshot shows a standard SAP dialog box with a title bar containing a copy icon and a close button. The dialog contains the following elements:

- Two input fields: the first contains the text "LH" and the second contains the number "400".
- Two pushbuttons: "Display" and "Cancel".
- A radio button group with three options: "Selection 1" (selected), "Selection 2", and "Selection 3".
- A checkbox group with two options: "Option 1" (checked) and "Option 2" (unchecked).

© SAP AG 2002



© SAP AG 2002

- Unit 1 Course Overview
- Unit 2 Introduction to Screen Programming
- Unit 3 The Program Interface
- Unit 4 Screen Elements for Output
- Unit 5 **Screen Elements for Input/Output**
- Unit 6 Screen Elements: Subscreens and Tabstrip Controls
- Unit 7 Screen Elements: Table Controls
- Unit 8 Context Menus
- Unit 9 Lists in Screen Programming

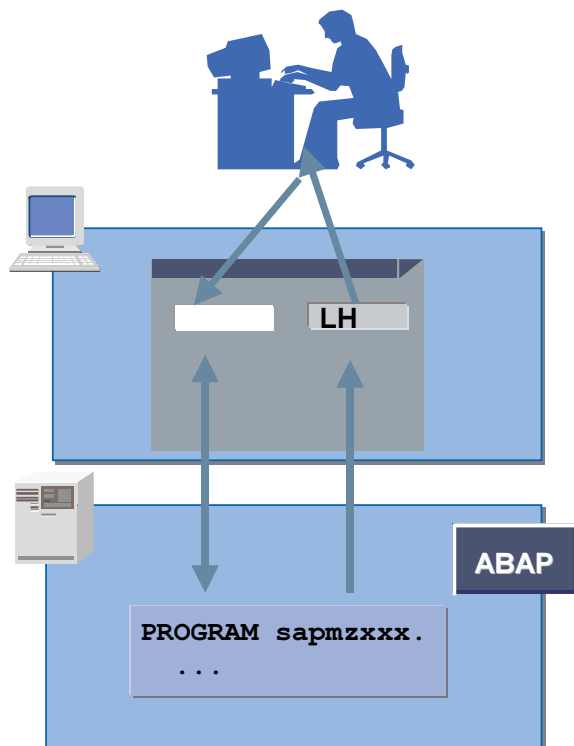


**Input/output fields**

**Input help**

**Checkboxes and radio button groups**

**Pushbuttons**

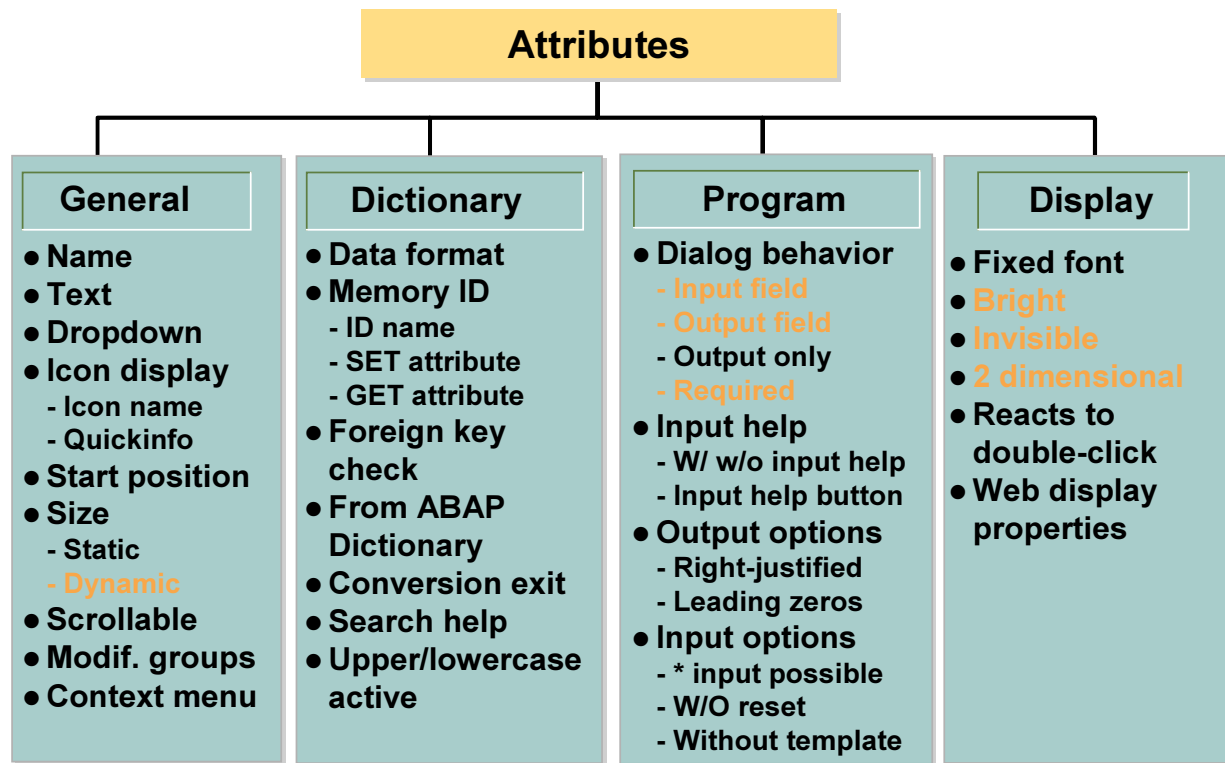


Displaying and receiving data at the front end

- Automatic field input checks
- Data consistency checks (check table)
- Input helps

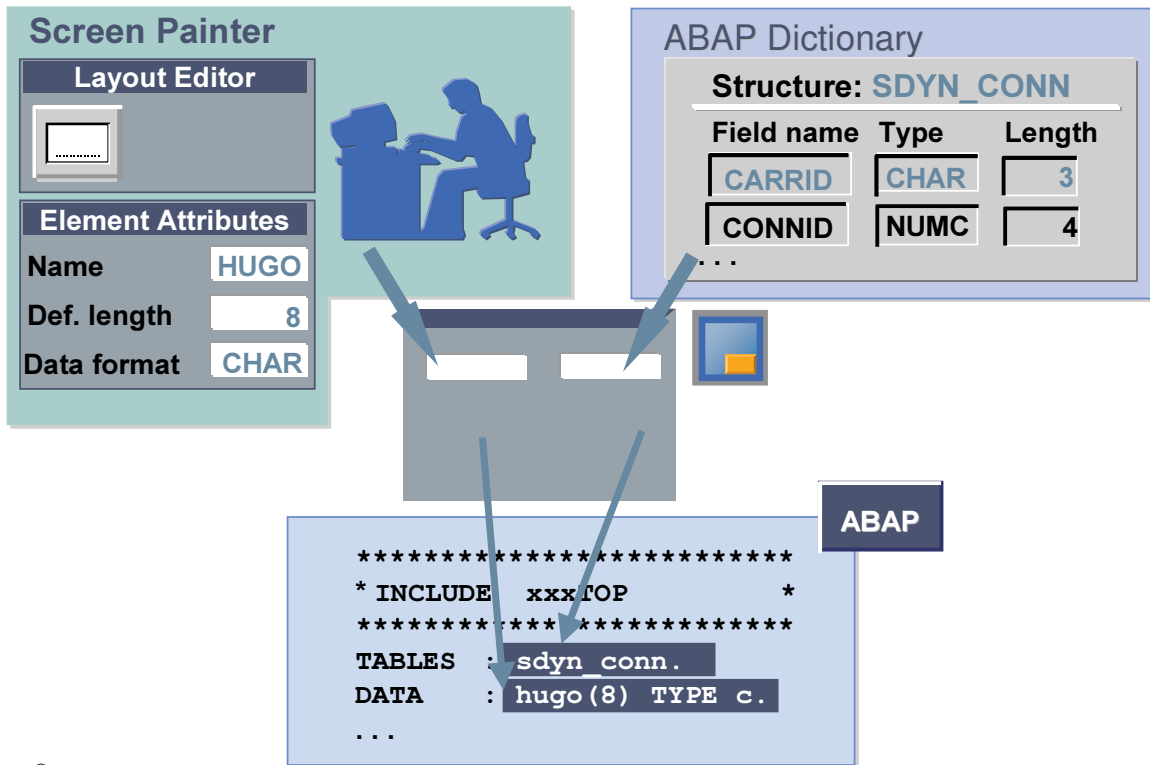
© SAP AG 2002

- An input field is a rectangular screen element in which users enter data.
- An output field is a rectangular screen element in which the system displays text or other data.
- Input/output fields are also known as templates.
- Input fields may have automatic field input checks that relate to their data type (for example, date fields will allow you to enter a valid date only).
- Input fields that you create with reference to ABAP Dictionary fields may have built-in data consistency checks (foreign key checks, and value sets).
- Input fields may have possible values help.
- For further information about input/output fields, see the online documentation, **INP-1**.



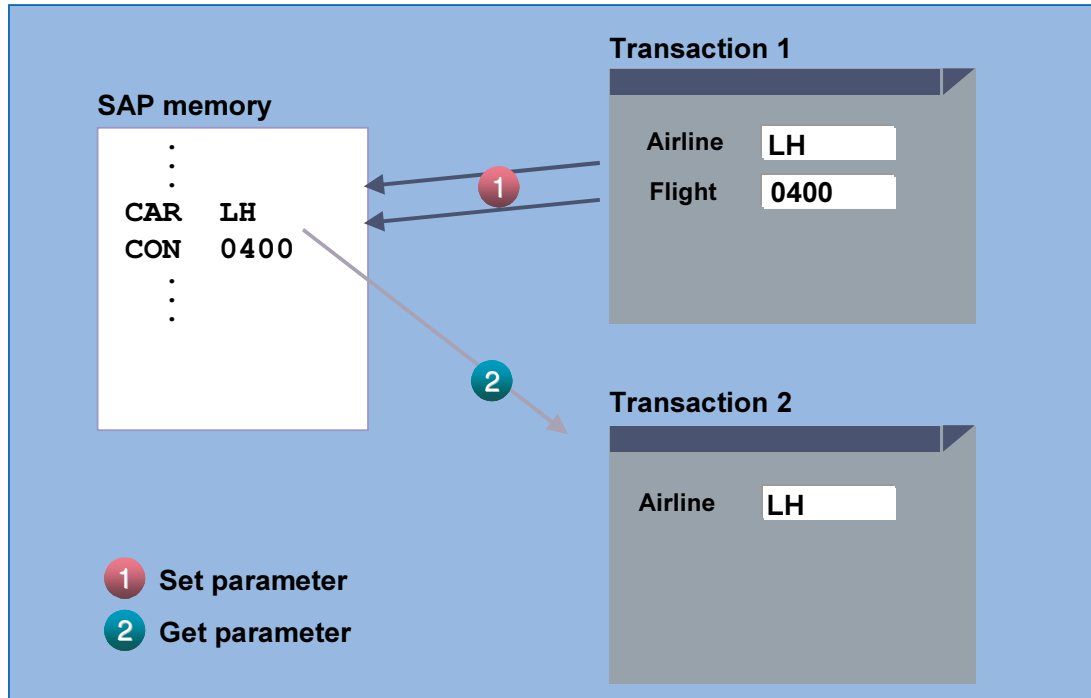
© SAP AG 2002

- You can temporarily change the object attributes marked in gray using the SCREEN system table.
- It may not be possible to activate all possible combinations of attributes. This depends on the format of the input/output field. For example, you cannot activate the *Leading zeros* attribute for a field with the data format CHAR, since it is only relevant for numeric fields.
- For further information about the *Data format* attribute, refer to the online documentation (reference INP-2).



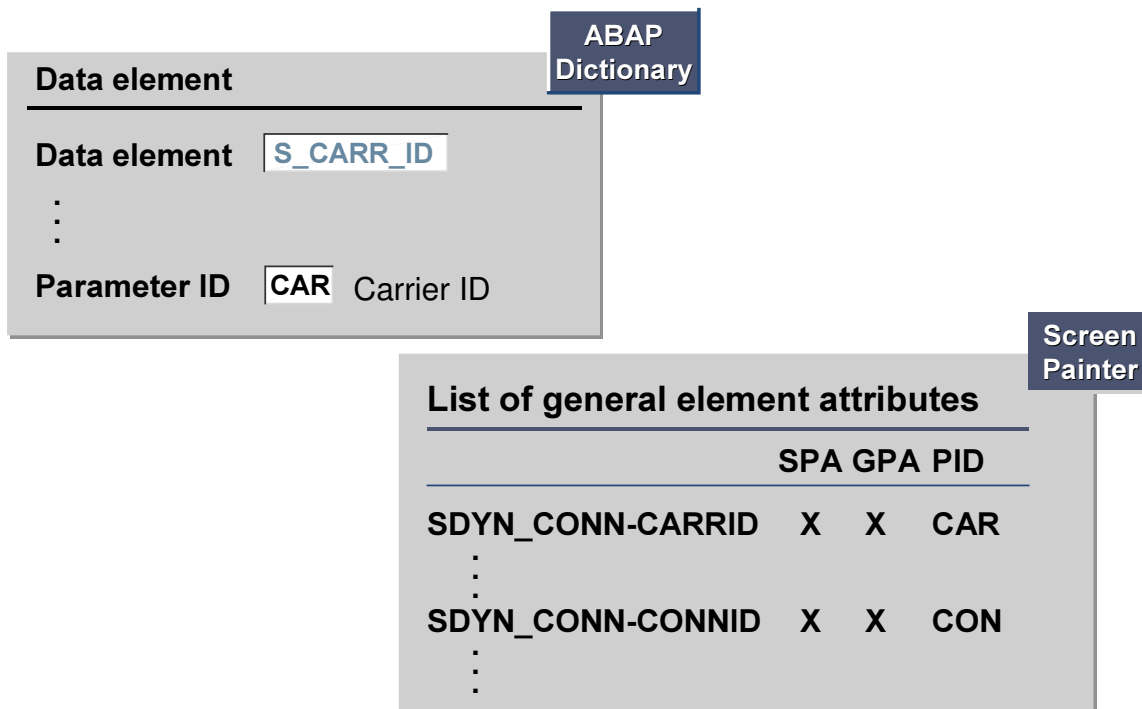
© SAP AG 2002

- You can create input/output fields in two ways:
  - By entering them directly in the layout editor. You determine the size of the field by the number of underscore characters in the object text attribute. For numeric values, you can specify a comma as a separator, and a period as a decimal point. As the last character in the input/output field, you can enter V as a placeholder for a plus or minus sign.
  - By using a template from the ABAP Dictionary. To do this, choose *Dict/Program fields*.
- If you want to use the contents of an input/output field in your ABAP program, you must declare the field globally using the DATA or TABLES statement.



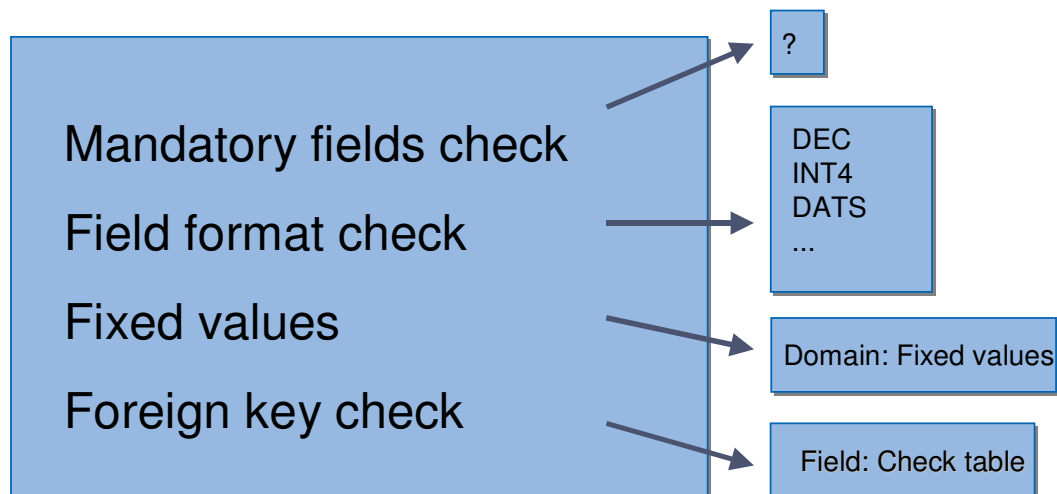
© SAP AG 2002

- You can save values in the SAP memory using a parameter ID. These are user and terminal-session specific, but available to **all internal and external sessions**.
- SET Parameter copies the corresponding field contents into the SAP System memory in the PAI processing block.
- GET Parameter copies the corresponding field contents from the SAP memory at the end of the PBO processing block (after data has been transferred from the program), if the screen field still has its initial value.



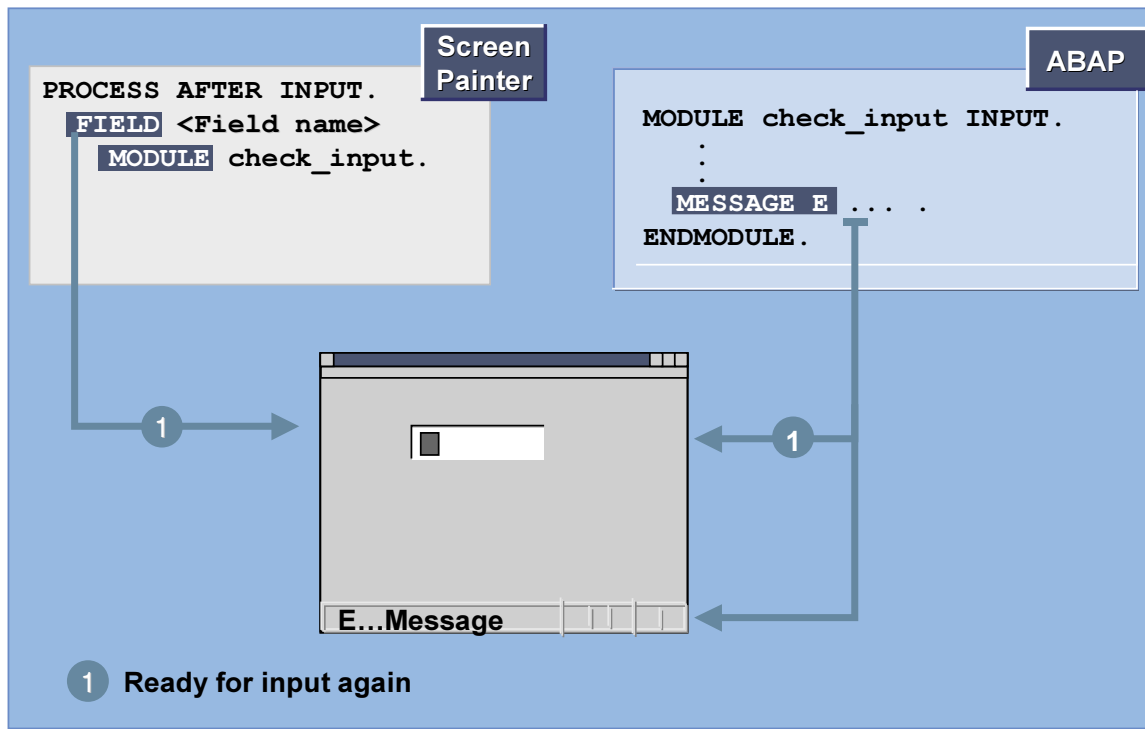
© SAP AG 2002

- You can link an input/output field to an area of the SAP memory in the ABAP Dictionary.
- When you use an input/output field that is defined in the ABAP Dictionary, its parameter ID is displayed in the Dictionary attribute *Parameter ID* in the Screen Painter.
- The SET Parameter and GET Parameter attributes (SPA and GPA in the table) allow you to enable the SET and GET parameter functions separately.
- You can define parameter IDs in table TPARA.



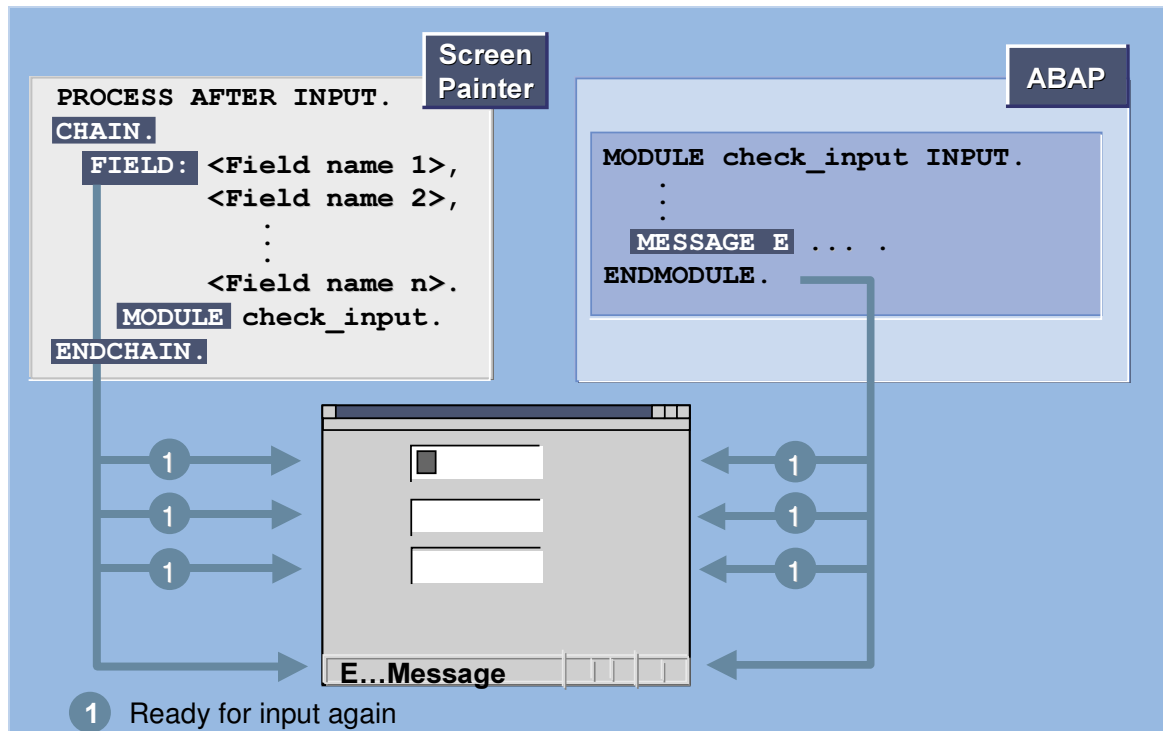
© SAP AG 2002

- After the screen is displayed but before the PAI modules are processed, the system automatically checks the values the user enters on the screen.
- The first check is to ensure that all required fields have been filled.
- The system can perform a foreign key check only if a screen field refers back to an ABAP Dictionary field for which a check table has been defined. The foreign key check attribute must also be set.
- The F4 help function is also active. The system displays the possible entries from which the user can choose.



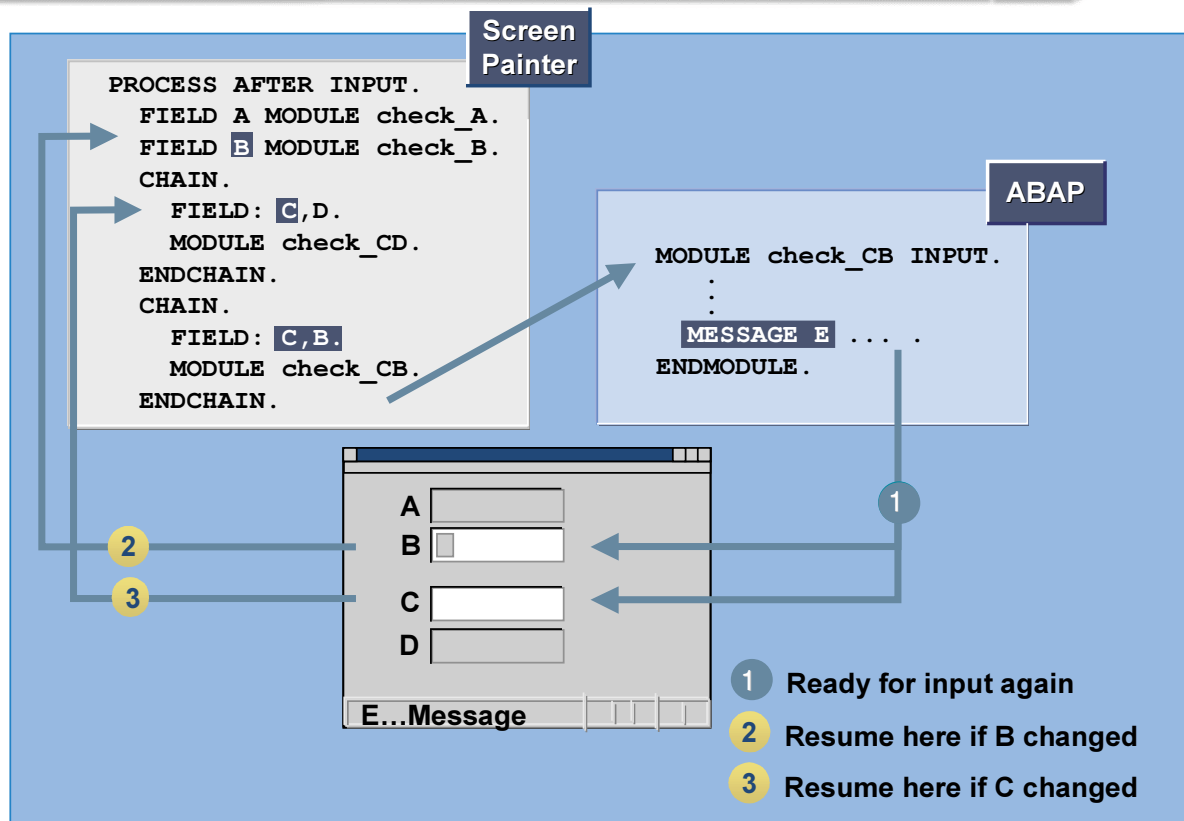
© SAP AG 2002

- If the automatic field input checks are insufficient for your requirements, you can program your own in the PAI event. To do this, use the **FIELD** statement with the **MODULE** addition. This means that the module you specify is processed only for the field specified in the **FIELD** statement.
- If an error or warning message occurs during the module, the system sends the screen again, but without processing the PBO module. The message is displayed; only the field to which the check was applied is ready for input.
- Note: The **FIELD** statement is responsible for making the field ready for input again. If you use a message in a module that is not called from within a **FIELD** statement, the system displays the message, but does not make the field ready for input again.



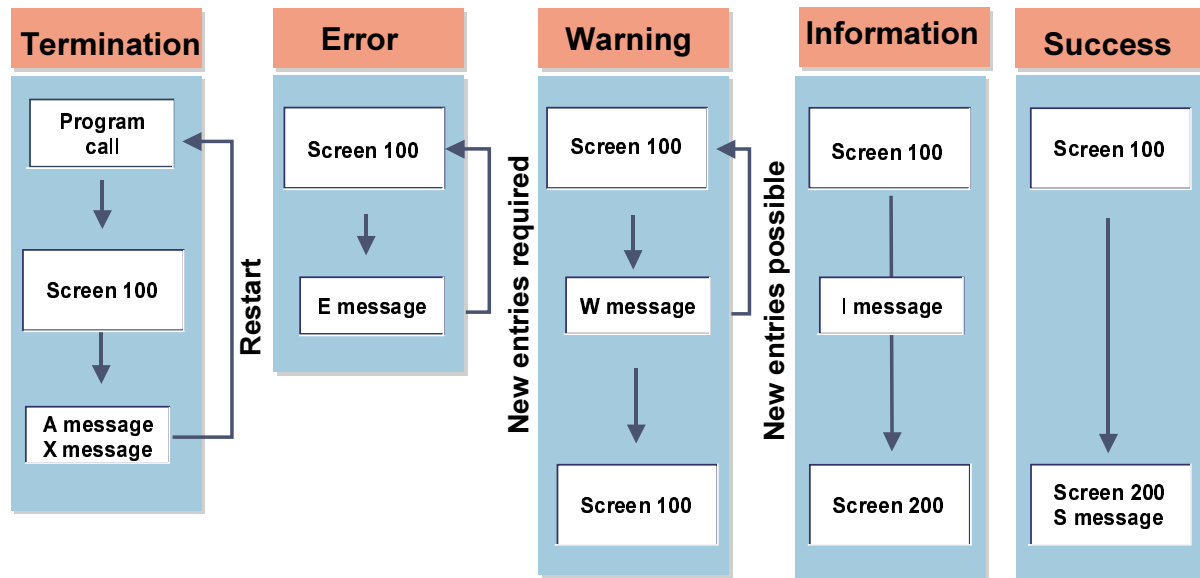
© SAP AG 2002

- If you want to ensure that more than one field is ready for input following an error dialog, you must list all of the relevant fields in the FIELD statement, and include both that and the MODULE statement in a CHAIN ... ENDCHAIN block.
- You can include individual fields in more than one CHAIN ... ENDCHAIN block.
- Note that the FIELD statement does not only make the field ready for input again. It also means that field contents changed during the current PAI processing are visible only if the field in question was also included in the FIELD statement of the current CHAIN block.



© SAP AG 2002

- If the system sends an error or warning message, the current screen is sent again but the PBO is not processed again.
- Only the fields to which the module is assigned are ready for input again.
- After the user has entered new values, the PROCESS AFTER INPUT module is not completely reprocessed, but restarted somewhere within the processing block.
- The system finds out which field the user changed and resumes processing at the first corresponding FIELD statement.
- If the user merely confirms a warning message (without changing the field's contents), the system restarts the PAI processing after the MESSAGE statement where the error was triggered.



© SAP AG 2002

- Messages are divided into six categories: A, X, E, W, I, and S. The differences between each class are as follows:

A	Termination	The processing terminates and the user must restart the transaction.
X	Exit	Like message type A, but with short dump MESSAGE_TYPE_X.
E	Error	Processing is interrupted, and the user <b>must</b> correct the entry
W	Warning	Processing is interrupted and the user can correct the entries (works like an Error message). However, it is also possible to confirm the existing entries by selecting <i>Enter</i> (works like an I message).
I	Information	Processing is interrupted, but continues when the user has confirmed the message (by selecting <i>Enter</i> ).
S	Success	Information is displayed on the next screen.

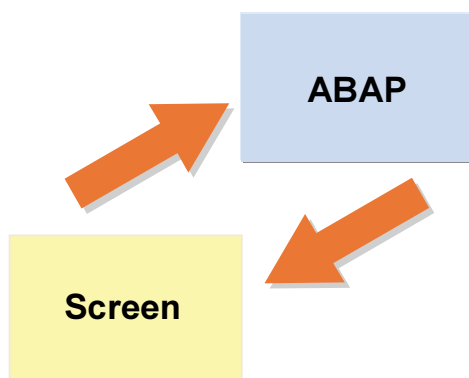
## The FIELD Statement and Data Transport

SAP

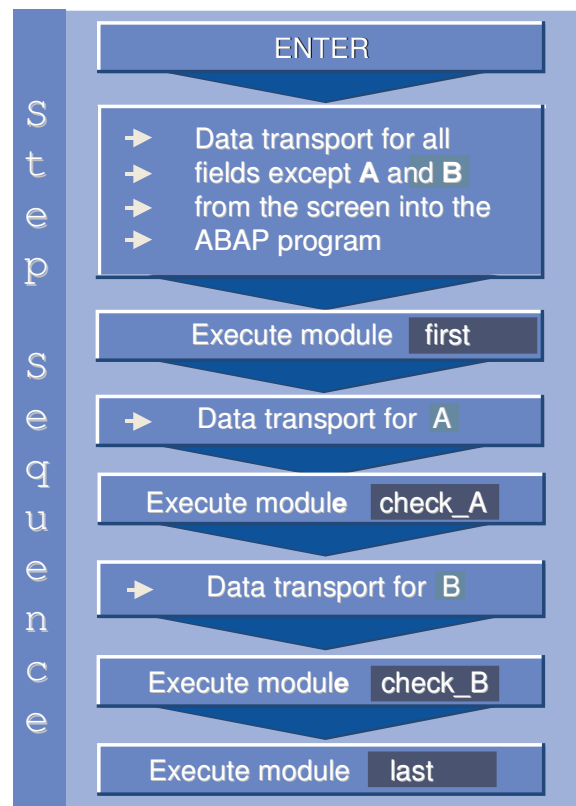
Screen Painter

```

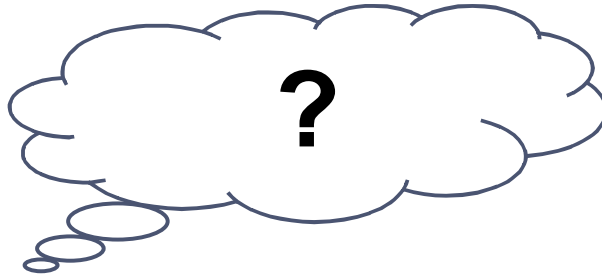
PROCESS AFTER INPUT.
MODULE first.
FIELD A MODULE check_A.
FIELD B MODULE check_B.
MODULE last.
    
```



© SAP AG 2002



- The system transports data from screen fields into the ABAP fields with the same name in the PAI processing block. First, it transports all fields that are not contained in any FIELD statements. The remaining fields are transported when the system processed the relevant FIELD statement.
- If an error or warning message occurs in a module belonging to a FIELD statement, the current values of all fields in the same CHAIN structure are automatically transported back into their corresponding screen fields.



- **How can I avoid unnecessary field checks?**
- **How can I leave the screen without any automatic field checks?**
- **How can I avoid data loss when the user navigates?**

© SAP AG 2002

- Field input checks usually require access to the database. Avoiding them where possible improves the performance of your program.
- If the user has strayed onto the screen by mistake, he or she are not usually able to make a consistent set of entries that will satisfy the input checks. You should therefore make it possible for a user to leave a screen without the field checks taking place.
- To protect the user from losing data that he or she has already entered if they leave the screen unintentionally, program security prompts.

```
PROCESS AFTER INPUT.
  FIELD <Field name>
    MODULE <module> ON INPUT.
  :
```

Screen  
Painter

```
PROCESS AFTER INPUT.
  CHAIN.
    FIELD: <Field name 1>,
          <Field name 2>,
          :
          <Field name n>.
    MODULE <module> ON CHAIN-INPUT.
  ENDCHAIN.
  :
```

Screen  
Painter

Called when  
field contents  
are not equal  
to initial value

© SAP AG 2002

- If you use the ON INPUT addition in a MODULE statement after FIELD, the module is called only if the field contents have changed from their initial value.
- Within a CHAIN block, you must use the ON CHAIN-INPUT addition. The module is then called if the contents of at least one screen field within the CHAIN block have changed from their initial value.
- You may use the ON INPUT addition only if the MODULE statement is contained in a FIELD statement.

```
PROCESS AFTER INPUT.
  FIELD <Field name>
    MODULE <module> ON REQUEST.
  .
  .
```

Screen  
Painter

Execution when  
input is new

```
PROCESS AFTER INPUT.
  CHAIN.
    FIELD: <Field name 1>,
          <Field name 2>,
          .
          <Field name n>.
    MODULE <module> ON CHAIN-REQUEST.
  ENDCHAIN.
  .
  .
```

Screen  
Painter

© SAP AG 2002

- If you use the ON REQUEST addition in a MODULE statement after FIELD, the module is called only if the user enters a new value in that field.
- Within a CHAIN block, you must use the ON CHAIN-REQUEST addition. The module is then called if the user changes the contents of at least one screen field within the CHAIN block.
- You may use the ON REQUEST addition only if the MODULE statement is contained in a FIELD statement.

```
PROCESS AFTER INPUT.
MODULE exit AT EXIT-COMMAND.
.
.
```

Screen  
Painter

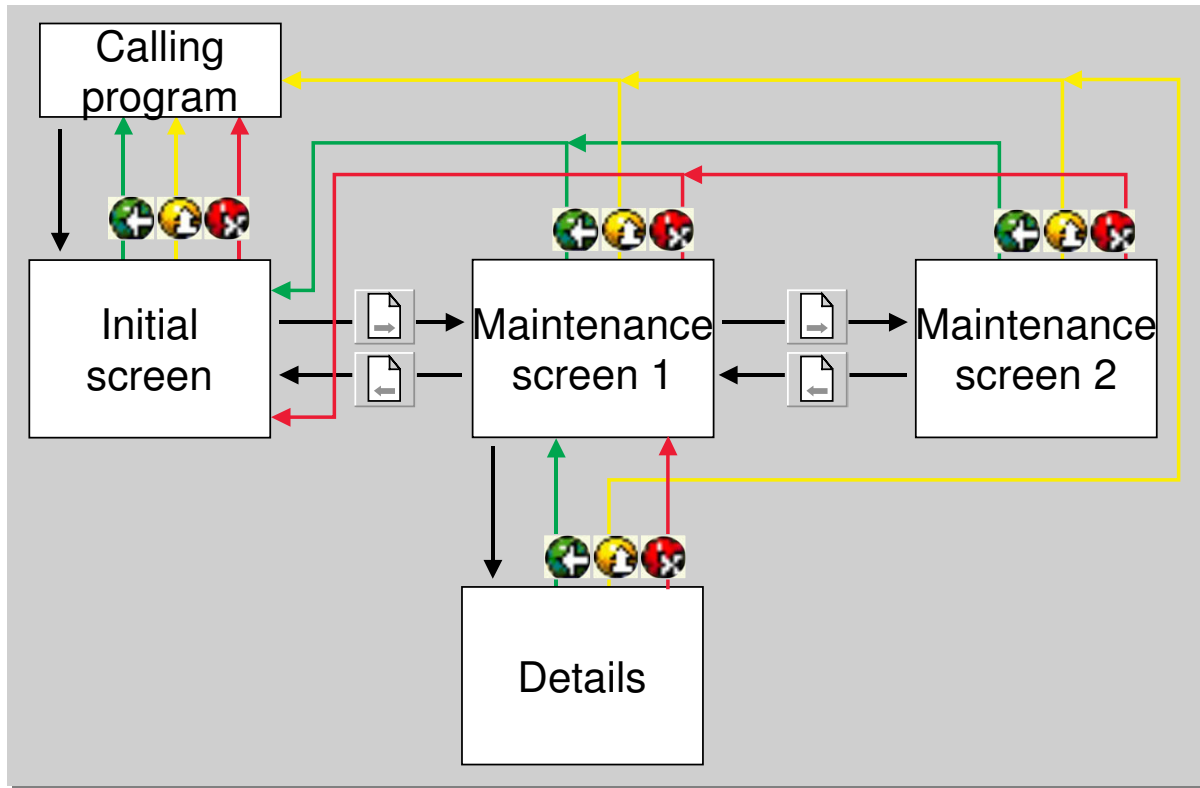
Execution when  
function has  
type E

```
MODULE exit INPUT.
CASE ok code.
WHEN 'CANCEL'.
CLEAR ok_code.
LEAVE TO SCREEN 0.
WHEN 'EXIT'.
LEAVE PROGRAM.
ENDCASE.
ENDMODULE. " EXIT INPUT
```

ABAP

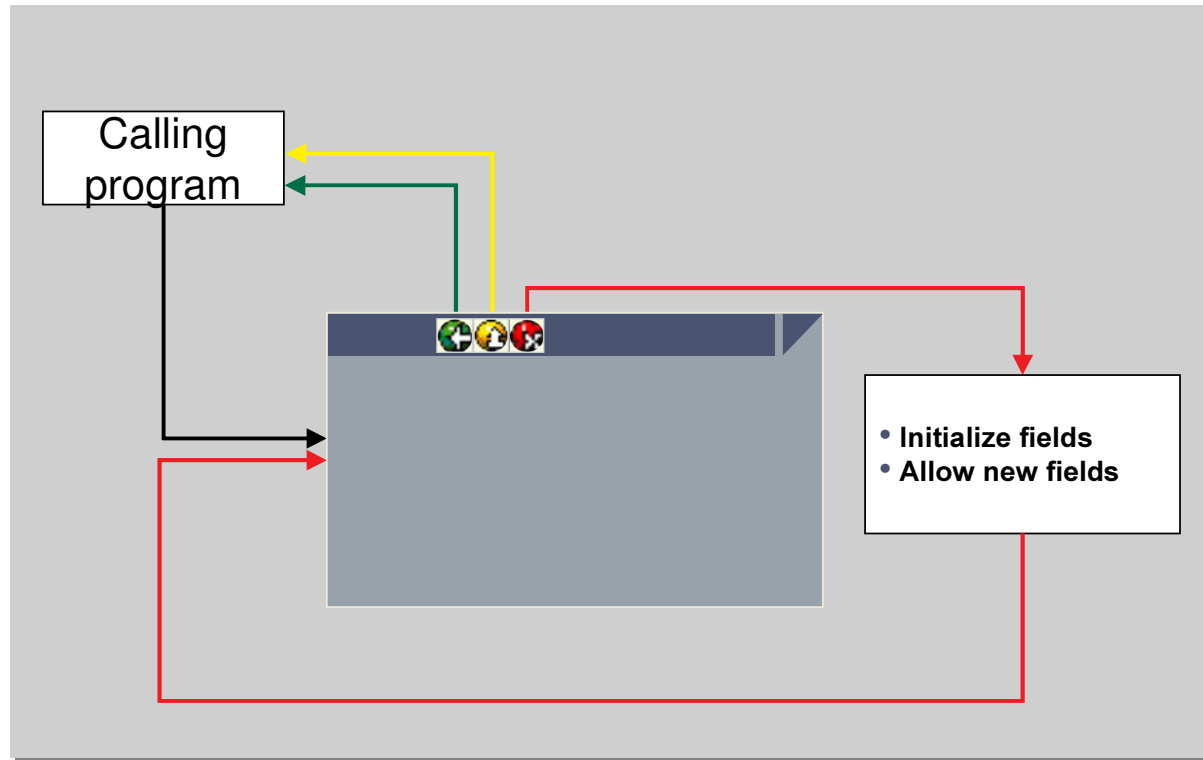
© SAP AG 2002

- The module with the AT EXIT-COMMAND addition is processed before the automatic field input checks. You can use it for navigation. You may use the AT EXIT-COMMAND addition with only one module for each screen. It may not have an associated FIELD statement.
- If you do not leave the screen from this module, the automatic field checks are processed after it, followed by the rest of the PAI event.







© SAP AG 2002

- The *Back* and *Cancel* functions should lead one logical level backward. From screens on the same level as the initial screen, they lead back to the initial screen. From screens that contain detailed information, they lead back to the screen that called the current screen.
- The *Cancel* function differs from *Back* in its dialog behavior.
- The *Exit* function should return to where the processing unit was called.
- On the initial screen of a program, all three functions -(*Back*, *Exit*, and *Cancel*) lead back to the screen from which the current program was called.
- For further information, see the online documentation, **INP-3**.



© SAP AG 2002

- *Back* exits the current transaction and returns to the calling program (for example, the Workplace). The function works like *Exit*.
- *Exit* exits the current transaction and returns to the calling program (for example, the Workplace).
- The *Exit* and *Back* functions are different in terms of their dialog behavior to prevent losing input data.
- *Cancel* displays the screen again with initialized data fields and allows the user to select a new object.

	Back 	Exit 	Cancel 
	Change Session 		
Saves dialog	Yes	Yes	No
Checks entries	Yes	Yes	No
Sequence	Check, then save dialog	First save dialog, then check	-
Function type	' '	'E'	'E'
Example	Save data?	Save data?	Unsaved data will be lost; cancel?
Function module for dialog	<code>popup_to_ confirm_step</code>	<code>popup_to_ confirm_step</code>	<code>popup_to_ confirm_ loss_of_data</code>

© SAP AG 2002

- If the user has entered data on the screen (sy-datar = X or your own flag), you can avoid accidental loss of data by using a predefined security prompt.
- For the *Exit* and *Cancel* functions, you first send a dialog box to the user. Then (in the case of the *Exit* function), the system checks the input on the screen. The functions in question must be function type E.
- In the case of the *Back* function, the input checks come before the dialog.
- Note that unsaved data may also be lost, for example, when switching from *Change* to *Display* mode. If the user chooses not to save, the system will display the original data stored in the database.
- The R/3 System contains a series of function modules that you can use for the user dialogs. Flow logic diagrams for the implementation of the individual functions are included in the appendix.
- For further information, see the online documentation, **INP-4**.



Input/output fields

Input help

Checkboxes and radio button groups

Pushbuttons

**Airline** Lufthansa  
Lauda Air  
Delta Airlines

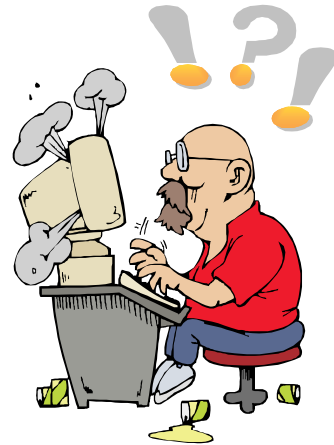
**Flight date** 09.09.2002

✓ ✗

Airline: LH  
Flight number: 0400

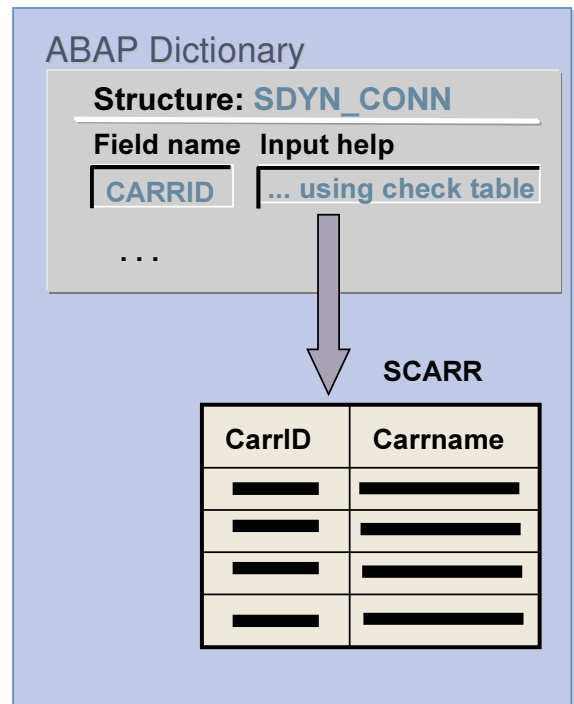
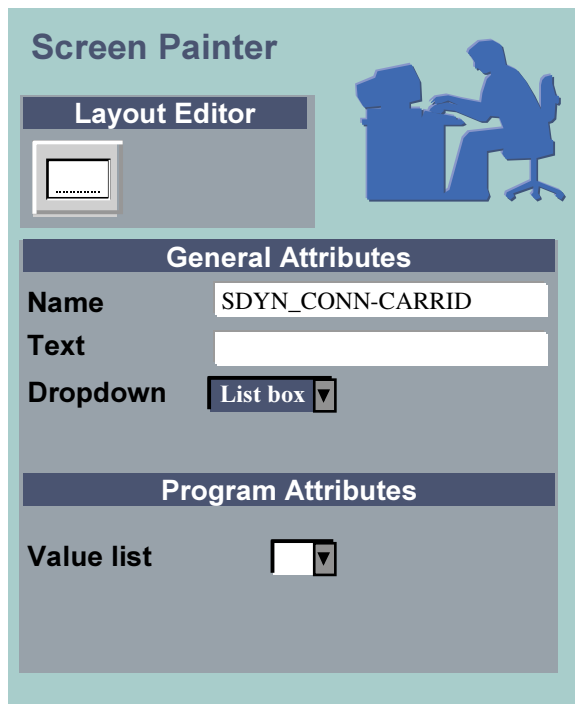
**Flight date**  
09.09.2002  
10.10.2002  
11.29.2002  
12.02.2002  
12.19.2002  
12.21.2002

F4



© SAP AG 2002

- You can help the user with input by using dropdown list boxes containing the possible entries.
- Input help (F4 help) is a standard function in the R/3 System. It allows the user to display a list of possible entries for a screen field. If the field is ready for input, the user can place a value in it by selecting it from the list.
- If a field has input help, the possible entries button appears on its right. The button is visible whenever the cursor is placed in the field. You can start the help either by clicking the button or by pressing the F4 key.
- In addition to the possible entries, the input help displays relevant additional information about the entries. This is especially useful when the field requires a formal key.
- Since the input help is a standard function, it should have the same appearance and behavior throughout the system. Utilities in the ABAP Workbench allow you to assign standardized input help to a screen field.
- The precise description of the input help of a field usually arises from its semantics. Consequently, input help is usually defined in the ABAP Dictionary.



© SAP AG 2002

- Dropdown boxes allow the user to choose an entry from a pull-down list containing the possible entries. The user cannot enter values freely, but must choose a value from the list.
- To create a dropdown box for an input field, you must do the following in the Screen Painter:
  - Set the *Dropdown* attribute to *List box*.
  - Change the *Visible Length* attribute to the displayed length of the descriptive text.
  - Set the *Value list* attribute to ' ' to use value help from the ABAP Dictionary.
  - If required, set the function code for the selection. Like a menu entry, this function code triggers the PAI; you can interpret the function code using the OK\_CODE field.
- Important: The **visible length of the field determines the width of the field** (including the button) and the selection list. You must change the width of the field when you convert the field to a dropdown box.
- The values are filled automatically using the search help assigned to the ABAP Dictionary field. The ABAP Dictionary field must have a search help (check table) with two columns or a table of fixed values.

Input/output fields

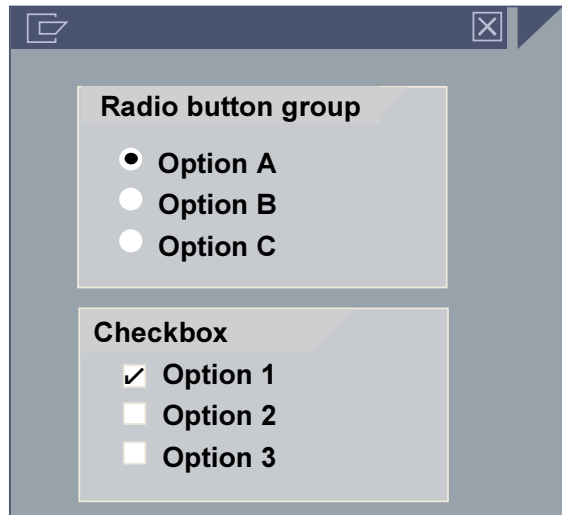
Input help



Checkboxes and radio button groups

Pushbuttons

© SAP AG 2001



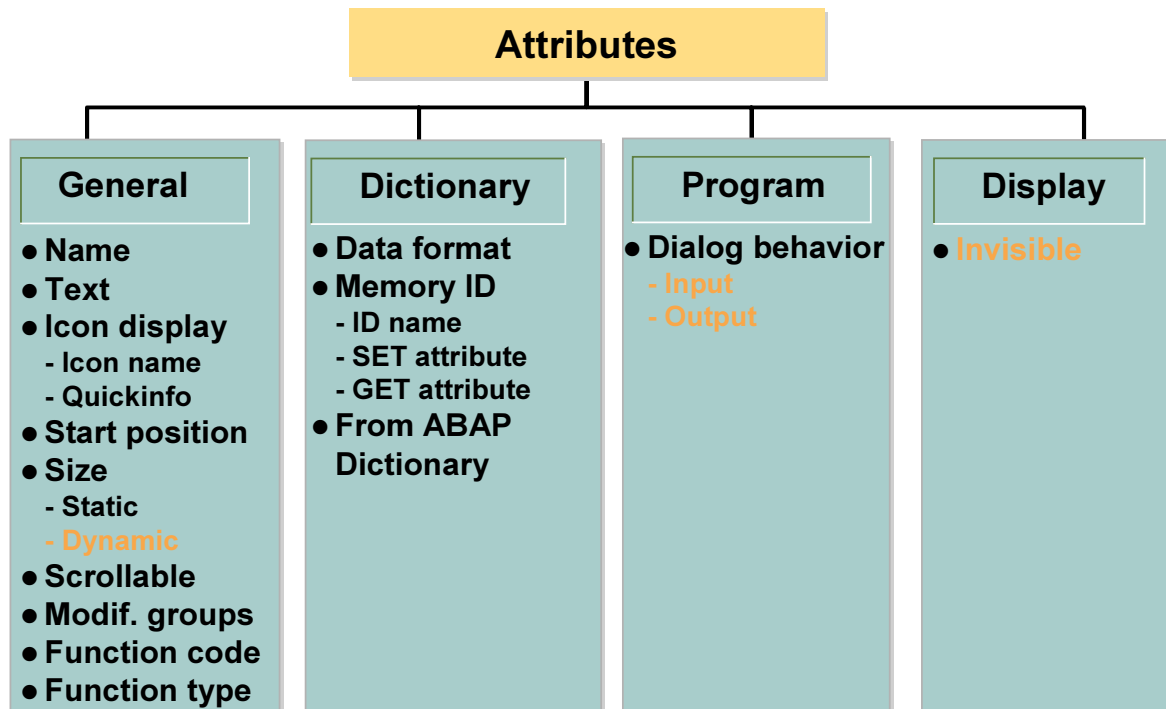
The screenshot shows a dialog box with a title bar containing a copy icon and a close button. Inside the dialog, there are two sections. The first section is titled "Radio button group" and contains three radio buttons labeled "Option A", "Option B", and "Option C". The second section is titled "Checkbox" and contains three checkboxes labeled "Option 1", "Option 2", and "Option 3". The "Option 1" checkbox is checked, while "Option 2" and "Option 3" are unchecked.

**User chooses  
functions using  
the mouse**

- **Simple display of all possible options**

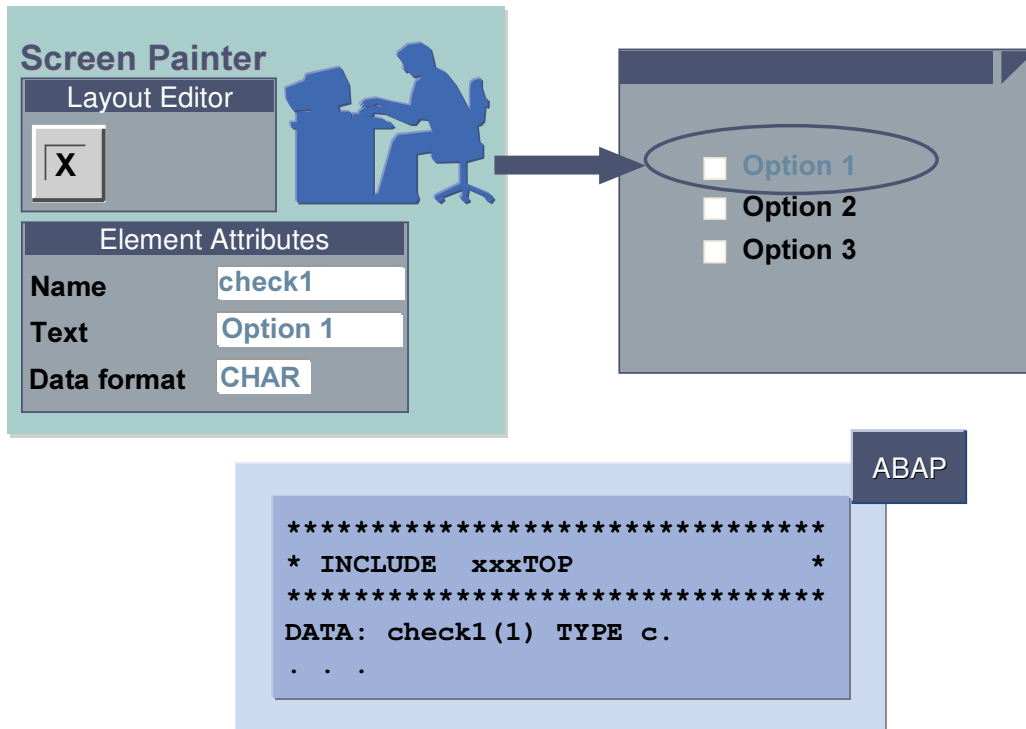
© SAP AG 2002

- Use radio buttons when you want to allow a user to choose only a single element from a group of fields.
- Use checkboxes when you want to allow the user to choose one or more elements from a group of fields.
- With radio buttons, one selection rules out all other options within the group. When the user selects one, all of the others are automatically deselected.



© SAP AG 2002

- You must attach a name to checkboxes and radio buttons.
- In addition to the input/output field, you can display text and icons for them. The text is contained in the *Text* field in the attributes. To display an icon, enter its name in the *Icon name* attribute. A quick info for the icon then appears in the appropriate field.
- You can change the *Input field* and *Invisible* attributes dynamically using the SCREEN system table.

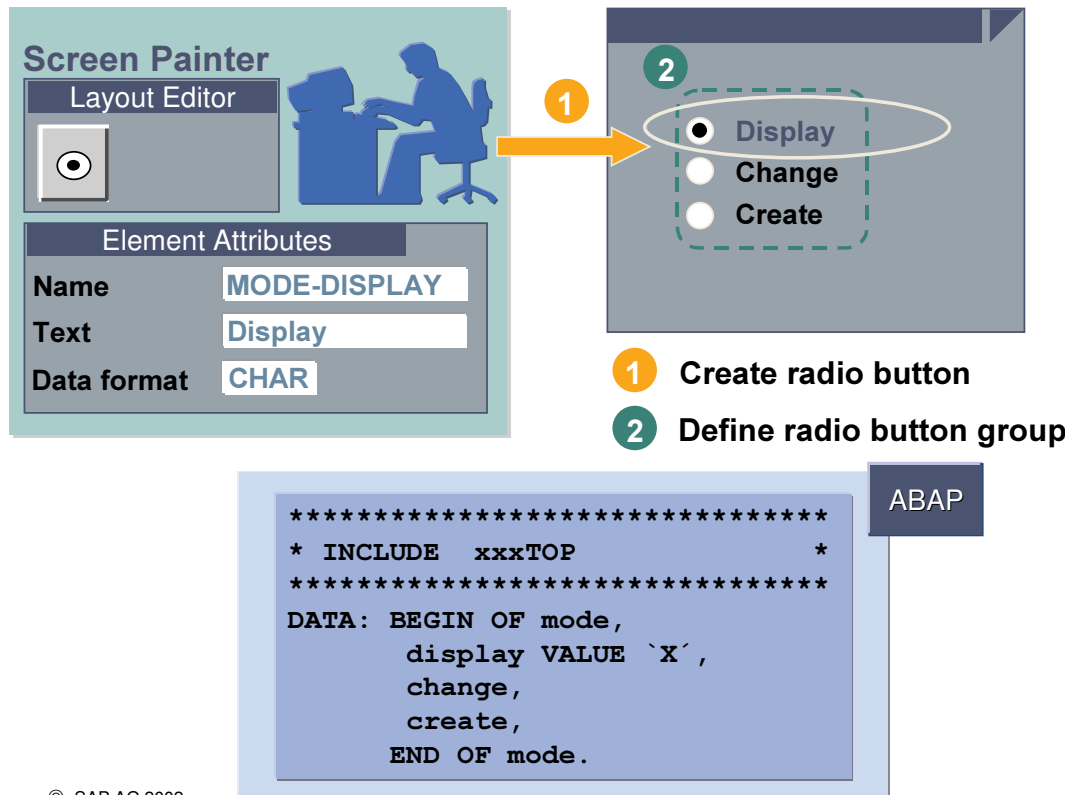


© SAP AG 2002

- You create checkboxes in the fullscreen editor of the Screen Painter. To do this, choose the checkbox object from the object list and place it on the screen. You must assign a name to each checkbox. In the ABAP program, create a field with the same name, type C, and length one.
- You can find out whether a user has chosen a checkbox in the ABAP program by querying the field contents. If a checkbox is not selected, its field value is initial.
- You can assign a function code and function type to a checkbox. When the user selects it, the PAI event is triggered and the function code is placed in the command field (that is, the OK\_CODE field).

## Creating a Radio Button Group

SAP



© SAP AG 2002

- You create checkboxes in the layout editor of the Screen Painter. There are two steps involved:
  - Create the radio buttons as individual elements. Choose *radio button* from the object list and place it on the screen. You must assign a name to each radio button. In the ABAP program, create a field with the same name, type C, and length one. To make your programs easier to read and maintain, create a structure associated with each radio button group.
  - You can also combine a collection of radio buttons into a radio button group. To do this, select the radio buttons in the layout editor and then choose *Edit → Group → Radio button group → Define*.
- You can find out which radio button a user has chosen by querying the field contents in the ABAP program. If a radio button is not selected, the field value is initial.
- You can assign a function code and function type to a radio button group. When the user selects one of the radio buttons, the PAI event is triggered and the function code is placed in the command field (that is, the OK\_CODE field).

## Program Flow for Radio Buttons and Checkboxes

SAP

Screen  
Painter

Screen Painter

Element List		
Name	Type	FctCode
check1		CHK1
check2		
check3		
mode-display		TOGGLE
mode-change		TOGGLE
mode-create		TOGGLE
ok_code	OK	

PROCESS AFTER INPUT.

...

MODULE user\_command\_100.

...

```

DATA ok_code TYPE sy-ucomm.
CONSTANTS marked VALUE 'X'.
...
MODULE user_command_100 INPUT.
CASE ok_code.
  WHEN 'CHK1'.
    IF NOT check1 IS INITIAL.
      ...
    ENDIF.
  WHEN 'TOGGLE'.
    CASE marked.
      WHEN mode-display. ...
      WHEN mode-change. ...
      WHEN mode-create. ...
    ENDCASE.
  ENDCASE.
ENDMODULE.
    
```

ABAP

© SAP AG 2002

- Depending on whether or not you have assigned a function code to a checkbox or radio button, when you select the field the system either triggers or does not trigger a PAI event.
- You can assign a function code to a radio button after you have defined a radio button group. The system then assigns the same function code to all radio buttons of the group.

Input/output fields

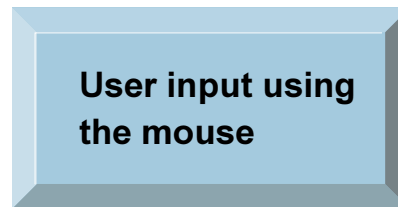
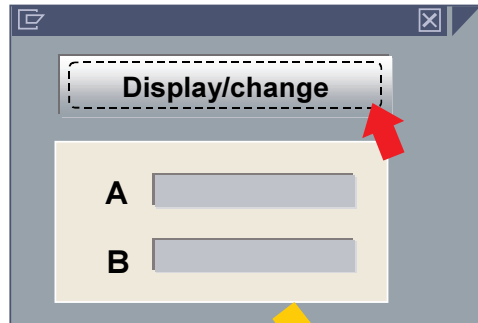
Input help

Checkboxes and radio button groups

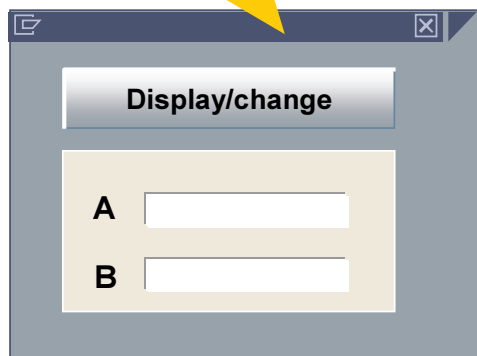


Pushbuttons

© SAP AG 1999

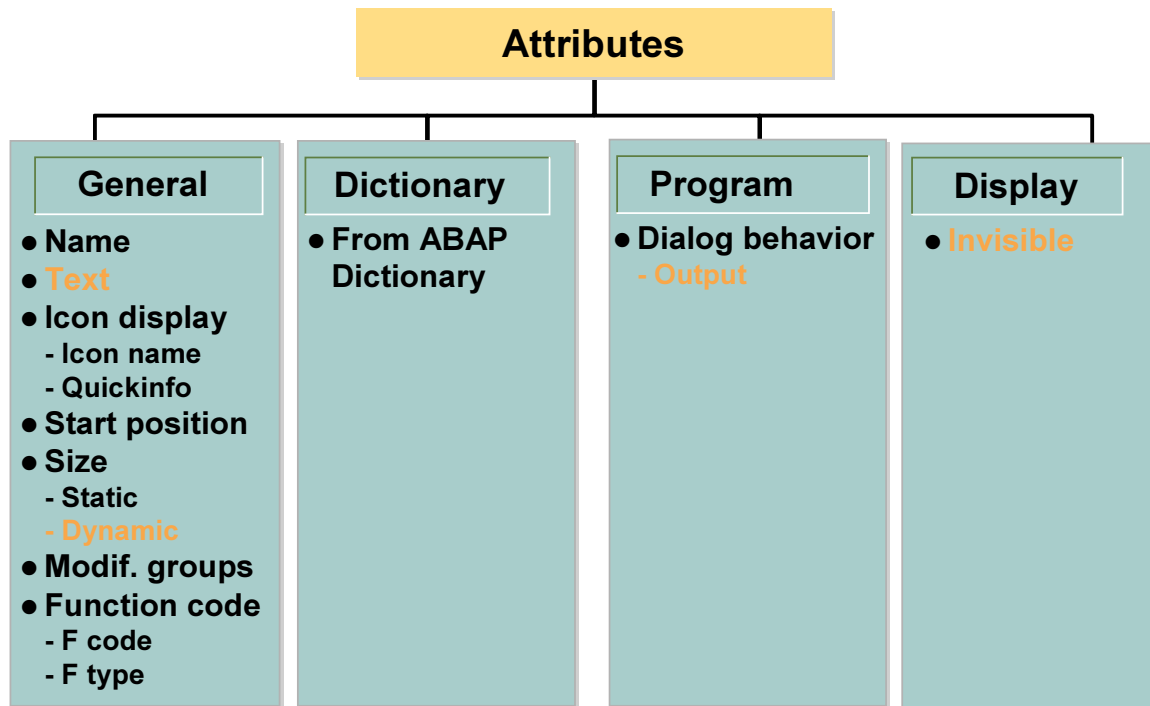


- User-determined program flow
- Functions that relate to individual screen elements or groups of screen elements



© SAP AG 2002

- Pushbuttons are input fields for the command field (that is, the OK\_CODE field).
- Using the mouse, users can quickly access functions that relate to individual screen elements or groups of screen elements.
- Use pushbuttons in the data area of your screen to show or hide further information.
- If a pushbutton relates to a single field or a small group of fields, make sure that the pushbutton is as close to them as possible. If the function relates to a group, make this clear using a group box.
- If pushbuttons relate to a table displayed on the screen, place them underneath it in a horizontal row, close together, with a blank line between them and the table.
- When the user chooses a pushbutton, the system tells the program which function is chosen. At this point, control of the program passes back to a work process on the application server, which processes the PAI processing block.

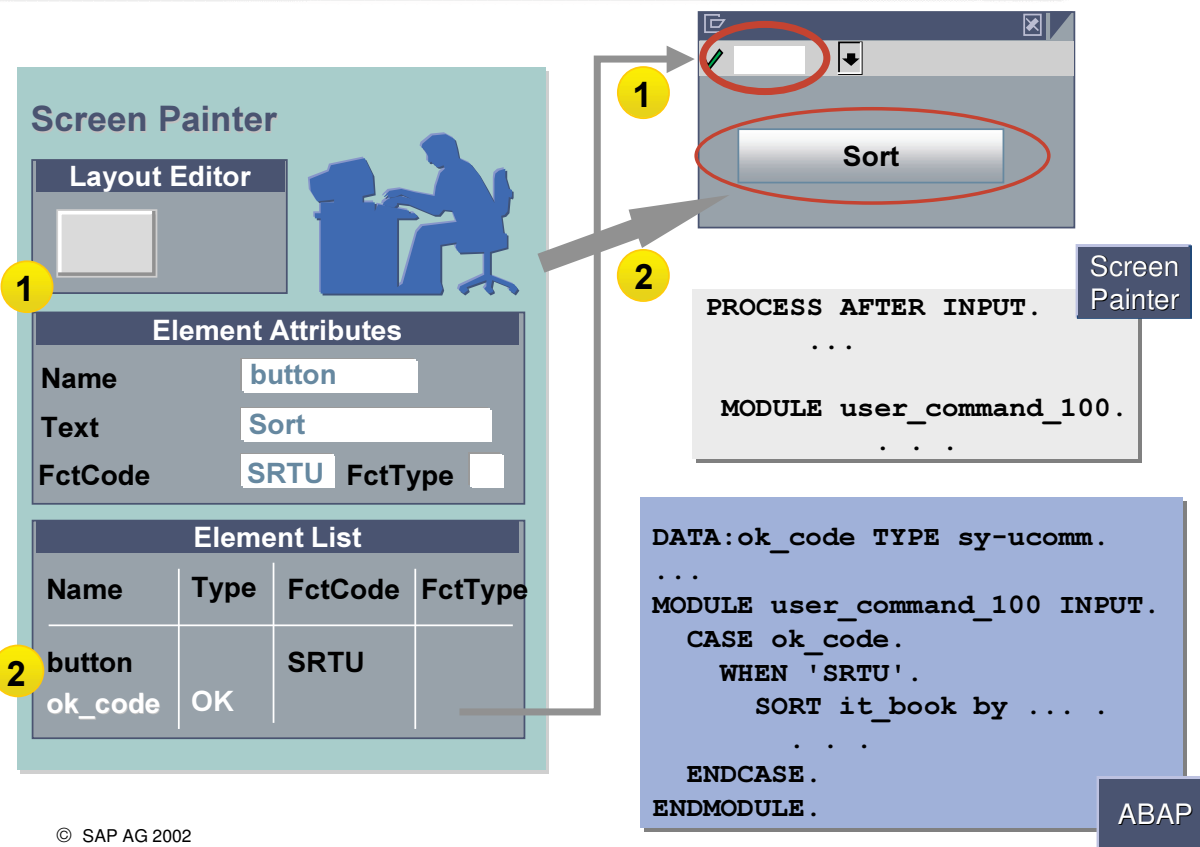


© SAP AG 2002

- Pushbuttons may contain text (*Text* attribute), an icon, or both. You can either specify an icon statically or dynamically, using the function module `ICON_CREATE`.
- You can change the *visible length*, *output field*, and *invisible* attributes dynamically using the system table `SCREEN`.
- You can change the text on a pushbutton dynamically. To do this, set the *Output field* attribute in the Screen Painter to active, and create a global field with the same name in your ABAP program. Because the Screen Painter field and the program field have the same name, any changes to the field contents will be immediately visible on the screen (similarly to input/output fields).

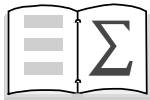
## Creating and Processing Pushbuttons

SAP



© SAP AG 2002

- When you create a pushbutton, you must:
  - **Create a pushbutton:** Choose the pushbutton object from the Screen Painter element list, place it on the screen, and assign a name to it. You can enter a static text in the *Text* attribute. Enter a function code for the pushbutton in the *Function code* attribute. This is placed in the OK\_CODE field automatically when the user chooses the pushbutton on the screen.
  - **Activate the command field (OK\_CODE field):** You must give the field a name in the element list of the Screen Painter, then declare an identically-named field in the ABAP program with reference to the system field sy-ucomm.
- When the user chooses a function on the screen, the system places the corresponding function code into the OK\_CODE field. You can then query the field and use the result to trigger the appropriate coded processing block.
- If the user chooses a pushbutton that has the function type '' (space), the PAI event is processed.
- If the user chooses a pushbutton that has the function type E, the system processes a module with the AT EXIT-COMMAND addition. This happens before the automatic field transport and the field input checks. The system places the function code that has been triggered into the OK\_CODE field, which you can then query in the module.
- After the AT EXIT-COMMAND module, the system continues processing the screen normally (field input checks, followed by PAI processing).



**You are now able to:**

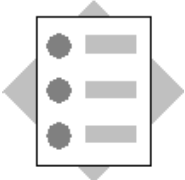
- **Use input/output fields, checkboxes, radio button groups, and pushbuttons in your programs.**

# Exercises



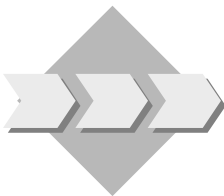
## Unit: Screen Elements for Input/Output

### Topic: Input/output fields on screens, input checks, input help, and mode selection using a radio button group



At the conclusion of these exercises, you will be able to:

- Create input/output fields for screens.
- Make input checks.
- Use input helps in your programs.
- Create radio button groups and program the relevant logic.
- Make dynamic changes to screens.
- Add input/output fields to your maintenance transaction for flight information. The airline, flight number, and flight date fields should be ready for input.
- Support the user by checking the entries and providing input help.
- Allow the user to switch between different program modes:
  - Display mode
  - Flight data maintenance mode (the user can change the aircraft type)
  - Maintain bookings (you will use this later)
- If the user changes the aircraft type, he or she should be able to save the changed value.



1-1 Put the input/output fields on the screen.

1-1-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410ADIAS\_GUI**).

You can use the model solutions **SAPMBC410AINPS\_INPUT\_FIELD**.

1-1-2 Use the ABAP Dictionary structure **SDYN\_CONN** in the **TABLES** statement (in the TOP include) to create a structure with the same name for transporting data.

- 1-1-3 Create the following fields on the screen. Use the facility for using fields from the ABAP Dictionary.

Screen 100	<b>I/O Fields, Text fields:</b> SDYN_CONN -CARRID -CONNID -FLDATE	<b>Attributes for fields:</b> <b>Input:</b> ON <b>Output:</b> ON <b>Required entry:</b> ON <b>SET parameter:</b> ON <b>GET parameter:</b> ON
	<b>I/O Fields, Text Fields:</b> SDYN_CONN -PRICE -CURRENCY -PLANETYPE -SEATSMAX -SEATSOCC -PAYMENTSUM	<b>Attributes for each field:</b> <b>Input:</b> OFF <b>Output:</b> ON

- 1-1-4 In the PAI event of screen 100, call a module **read\_sflight**. Create the module using forward navigation. Try to read the corresponding data record from table **sflight**, and analyze the return code **sy-subrc**. If the data record does not exist, display message **038** from class **BC410** as an information message and refresh the output fields.
- 1-2 Execute the input checks and extend the navigation functions on the screen to include the *Exit* function.
- 1-2-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise or copy the corresponding model solution **SAPMBC410AINPS\_INPUT\_FIELD**. You can use the model solution **SAPMBC410AINPS\_CHECK\_INPUT** for orientation.
- 1-2-2 Change the information message in 5-1-4 to an error message. Read the database table only if the user changes one or more entries on the screen. Make sure that the fields are ready for input again if the data record does not exist.
- 1-2-3 Assign the function codes EXIT and CANCEL to the standard keys SHIFT-F3 (*Exit*) and F12 (*Cancel*). Ensure that these functions are processed **before** the automatic input checks. If the user chooses *Exit*, leave the program. If the user chooses *Cancel*, initialize the input/output fields and display the screen again.
- 1-3 Make the user's task easier by providing input help.
- 1-3-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise or copy the corresponding model solution **SAPMBC410AINPS\_CHECK\_INPUT**. You can use the model solution **SAPMBC410AINPS\_HELP\_FOR\_INPUT** for orientation.

- 1-3-2 On screen 100, set the *Dropdown* attribute to *List box* for the input/output field **sdyn\_conn-carrid**. Make sure that the program attribute *Value list* is set to '' (from ABAP Dictionary).
- 1-4 Create a radio button group to allow the user to choose one of a range of program modes.
- 1-4-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410AINPS\_HELP\_FOR\_INPUT**). Use the model solution **SAPMBC410AINPS\_RADIOBUTTON** for orientation.
- 1-4-2 On screen 100, create a radio button group with the buttons *view*, *maintain flights*, and *maintain bookings*. Make sure that the function code **MODE** (with type ') is triggered when the user chooses a different mode. Create a group box around the radio button group called **frame** and assign it the text *Mode*. Declare the relevant data fields in your top include.
- 1-4-3 Program the *Maintain flight data* mode. In this mode, the input/output field **sdyn\_conn-planetype** should be ready for input and required. Create a module **modify\_screen** to make the corresponding dynamic screen modification when the field is populated with data.
- If the user enters a new aircraft type only, check whether the number of seats booked is greater than the maximum number of seats. To do this, update the field **sdyn\_conn-seatsmax** from table **SAPLANE** in a new PAI module. If an error occurs, display message **109** from class **BC410** as an error message and transport the maximum number of seats back to the screen.
- 1-4-4 Create a new PAI module **trans\_from\_dynp** to populate a new Global variable **wa\_sflight** of ABAP Dictionary type **SFLIGHT** with the corresponding fields from **sdyn\_conn**. The new work area will be used next to update the **SFLIGHT** table with the new aircraft type and the new maximum number of seats.
- Assign the function code **SAVE** (function type ') to the standard key Ctrl-S (*Save*). If the user chooses this function, save the new data record in the database. In subroutine **update\_sflight**, use a direct database update in the form:
- ```

UPDATE sflight FROM wa_sflight.
IF sy-subrc NE 0.
MESSAGE a008.
ENDIF.
MESSAGE s009.

```
- (This process would normally use a suitable SAP lock, but we have omitted it here for simplicity.)



## Unit: Screen Elements for Input/Output

### Topic: Input/output fields on screens, input checks, input help, and mode selection using a radio button group

## 5-1 Model solution SAPBC410AINPS\_INPUT\_FIELD

Add the coding in bold type to your program. Create the new subroutines using forward navigation.

### Flow logic screen 100

PROCESS BEFORE OUTPUT.

MODULE status\_0100.

MODULE clear\_ok\_code.

\*

PROCESS AFTER INPUT.

**MODULE read\_sflight.**

MODULE user\_command\_0100.

### Top include

PROGRAM sapmbc410ainps\_input\_field **MESSAGE-ID bc410.**

**TABLES** sdyn\_conn.

DATA ok\_code TYPE sy-ucomm.

### PBO module include

MODULE status\_0100 OUTPUT.

SET PF-STATUS 'STATUS\_100'.

SET TITLEBAR 'TITLE\_100'.

ENDMODULE. " STATUS\_0100 OUTPUT

MODULE clear\_ok\_code OUTPUT.

CLEAR ok\_code.

ENDMODULE. " clear\_ok\_code OUTPUT

## PAI module include

```
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.                                " USER_COMMAND_0100  INPUT

MODULE read_sflight INPUT.
  SELECT SINGLE * FROM sflight INTO CORRESPONDING FIELDS OF sdyn_conn
    WHERE carrid = sdyn_conn-carrid
    AND   connid = sdyn_conn-connid
    AND   fldate = sdyn_conn-fldate.
  IF sy-subrc NE 0.
    CLEAR: sdyn_conn.
    MESSAGE i038.
  ENDIF.
ENDMODULE.                                " read_sflight  INPUT
```

## 5-2 Model solution SAPMBC410AINPS\_CHECK\_INPUT

Add the coding in bold type to your program. Create the new modules using forward navigation.

### Top include

No changes are necessary.

### Flow logic screen 100

```
PROCESS BEFORE OUTPUT.
```

```
MODULE status_0100.
```

```
MODULE clear_ok_code.
```

```
*
```

```
PROCESS AFTER INPUT.
```

```
MODULE exit AT EXIT-COMMAND.
```

```
CHAIN.
  FIELD: sdyn_conn-carrid,
        sdyn_conn-connid,
        sdyn_conn-fldate  MODULE read_sflight ON CHAIN-REQUEST.
ENDCHAIN.
```

```
MODULE user_command_0100.
```

## PBO module include

No changes are necessary.

## PAI module include

```
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.                                " USER_COMMAND_0100  INPUT
```

```
MODULE exit INPUT.
  CASE ok_code.
    WHEN 'EXIT'.
      LEAVE PROGRAM.
    WHEN 'CANCEL'.
      CLEAR: sdyn_conn.
      SET PARAMETER ID: 'CAR' FIELD space,
                        'CON' FIELD space,
                        'DAY' FIELD space.
      LEAVE TO SCREEN 100.
  ENDCASE.
ENDMODULE.                                " exit  INPUT
```

```
MODULE read_sflight INPUT.
  SELECT SINGLE * FROM sflight INTO CORRESPONDING FIELDS OF sdyn_conn
    WHERE carrid = sdyn_conn-carrid
    AND   connid = sdyn_conn-connid
    AND   fldate = sdyn_conn-fldate.
  IF sy-subrc NE 0.
    MESSAGE e038.
  ENDIF.
ENDMODULE.                                " check_sflight  INPUT
```

## 5-4 Model solution SAPMBC410AINPS\_RADIOBUTTON

Add the coding in bold type to your program. Create the new modules using forward navigation.

### Main Program

```
INCLUDE mbc410ainps_radiobuttontop.  
INCLUDE mbc410ainps_radiobuttonono01.  
INCLUDE mbc410ainps_radiobuttoni01.  
INCLUDE mbc410ainps_radiobuttonf01.
```

### Top include

```
PROGRAM sapmbc410ainps_radiobutton MESSAGE-ID bc410 .  
  
TABLES sdyn_conn.  
  
DATA: BEGIN OF mode,  
      view VALUE 'X',           "selected  
      maintain_flights,  
      maintain_bookings,  
      END OF mode.  
  
DATA ok_code TYPE sy-ucomm.  
  
DATA wa_sflight TYPE sflight.
```

### Subroutine include

Insert the following subroutine in the include:

```
FORM update_sflight.  
  UPDATE sflight FROM wa_sflight.  
  IF sy-subrc NE 0.  
    MESSAGE a008.  
  ENDIF.  
  MESSAGE s009.  
ENDFORM.           " update_sflight
```

## Flow logic screen 100

PROCESS BEFORE OUTPUT.

MODULE status\_0100.

**MODULE modify\_screen.**

MODULE clear\_ok\_code.

\*

PROCESS AFTER INPUT.

MODULE exit AT EXIT-COMMAND.

CHAIN.

FIELD: sdyn\_conn-carrid,

sdyn\_conn-connid,

sdyn\_conn-fldate MODULE check\_sflight ON CHAIN-REQUEST.

ENDCHAIN.

**CHAIN.**

**FIELD: sdyn\_conn-planetype,**

**sdyn\_conn-seatsmax MODULE check\_planetype ON CHAIN-REQUEST.**

**ENDCHAIN.**

**MODULE trans\_from\_dynp.**

**MODULE user\_command\_0100.**

## PBO module include

MODULE status\_0100 OUTPUT.

SET PF-STATUS 'STATUS\_100'.

SET TITLEBAR 'TITLE\_100'.

ENDMODULE. " STATUS\_0100 OUTPUT

MODULE clear\_ok\_code OUTPUT.

CLEAR ok\_code.

ENDMODULE. " clear\_ok\_code OUTPUT

**MODULE modify\_screen OUTPUT.**

**CHECK NOT mode-maintain\_flights IS INITIAL.**

```

CHECK NOT sdyn_conn-planetype IS INITIAL.
LOOP AT SCREEN.
  IF screen-name = 'SDYN_CONN-PLANETYPE'.
    screen-input = 1.
    screen-required = 1.
    MODIFY SCREEN.
  ENDIF.
ENDLOOP.
ENDMODULE.

```

## PAI module include

```

MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'SAVE'.
      PERFORM update_sflight.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.                                " USER_COMMAND_0100  INPUT

MODULE trans_from_dynp INPUT.
  MOVE-CORRESPONDING sdyn_conn TO wa_sflight.
ENDMODULE.                                " trans_from_dynp  INPUT

MODULE exit INPUT.
  CASE ok_code.
    WHEN 'EXIT'.
      LEAVE PROGRAM.
    WHEN 'CANCEL'.
      CLEAR: sdyn_conn.
      SET PARAMETER ID: 'CAR' FIELD space,
                        'CON' FIELD space,
                        'DAY' FIELD space.
      LEAVE TO SCREEN 100.
  ENDCASE.
ENDMODULE.                                " exit  INPUT

```

```

MODULE check_planetype INPUT.
  SELECT SINGLE seatsmax INTO sdyn_conn-seatsmax FROM saplane
    WHERE planetype = sdyn_conn-planetype.
  CHECK sdyn_conn-seatsmax < sdyn_conn-seatsocc.
  MESSAGE e109.
ENDMODULE.                                " check_planetype  INPUT

```

```

MODULE read_sflight INPUT.
  SELECT SINGLE * INTO CORRESPONDING FIELDS OF sdyn_conn FROM sflight
    WHERE carrid = sdyn_conn-carrid
      AND connid = sdyn_conn-connid
      AND fldate = sdyn_conn-fldate.
  IF sy-subrc NE 0.
    MESSAGE e038.
  ENDIF.
ENDMODULE.                                " check_sflight  INPUT

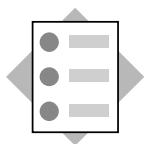
```

### Contents:

- Subscreens
- Tabstrip controls

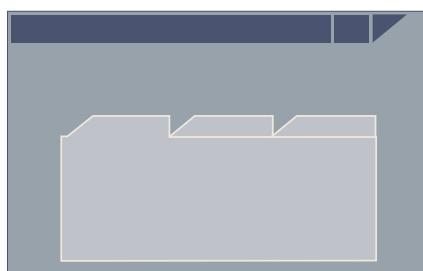


© SAP AG 1999

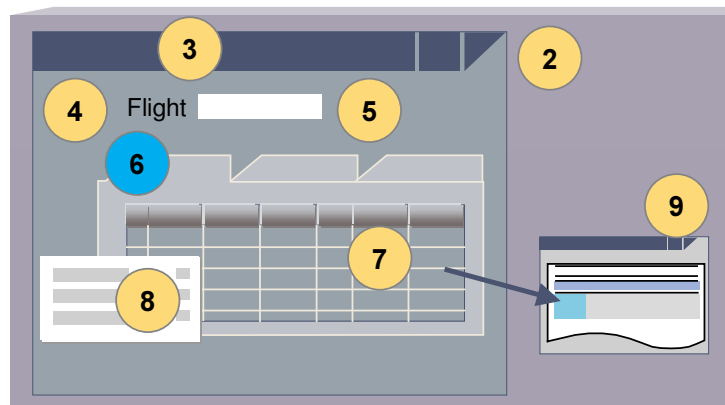


**At the conclusion of this unit, you will be able to:**

- **Use subscreens and tabstrip controls on screens and selection screens in your programs.**



© SAP AG 2002



© SAP AG 2002

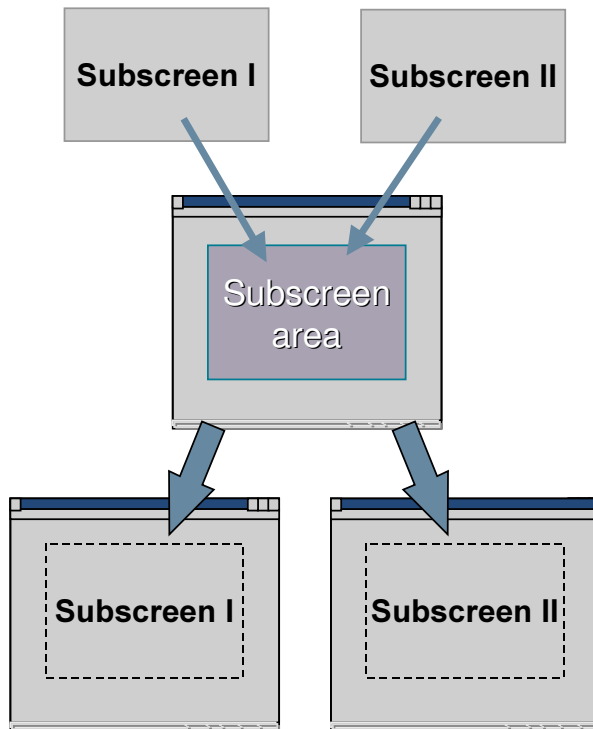
- Unit 1 Course Overview
- Unit 2 Introduction to Screen Programming
- Unit 3 The Program Interface
- Unit 4 Screen Elements for Output
- Unit 5 Screen Elements for Input/Output
- Unit 6 **Screen Elements: Subscreens and Tabstrip Controls**
- Unit 7 Screen Elements: Table Controls
- Unit 8 Context Menus
- Unit 9 Lists in Screen Programming



**Subscreens**

**Tabstrip controls**

© SAP AG 1999

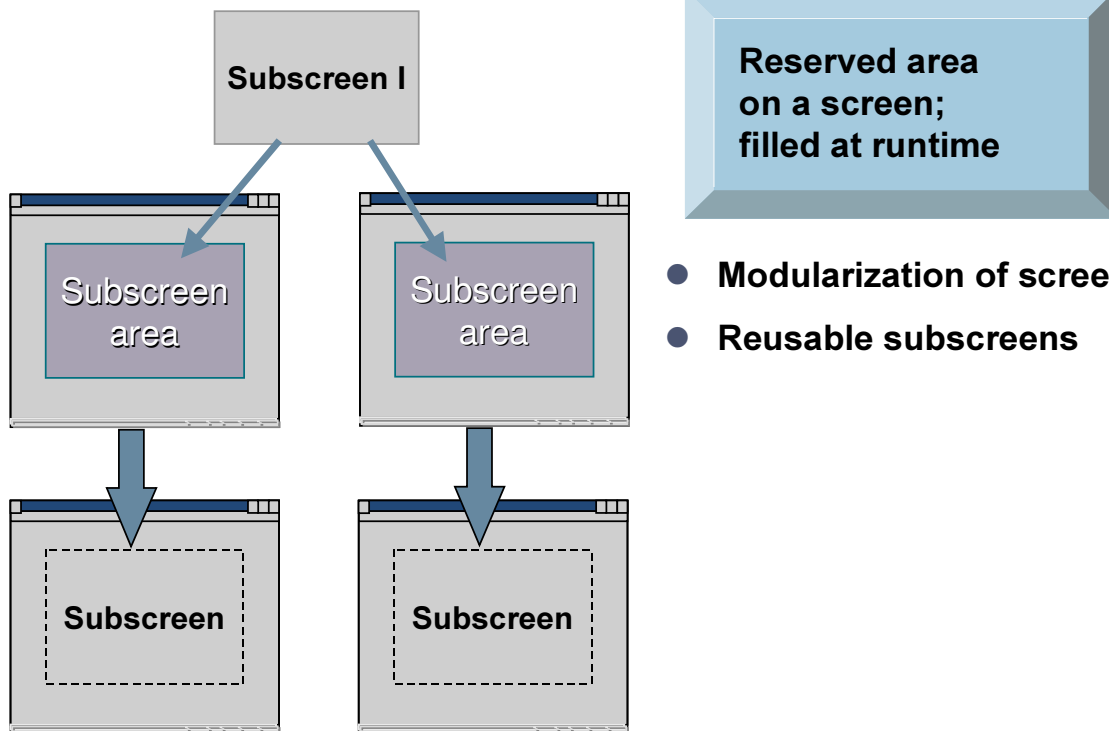


**Reserved area  
on a screen;  
filled at runtime**

- **Modularization of screens**
- **Dynamic screen modifications**

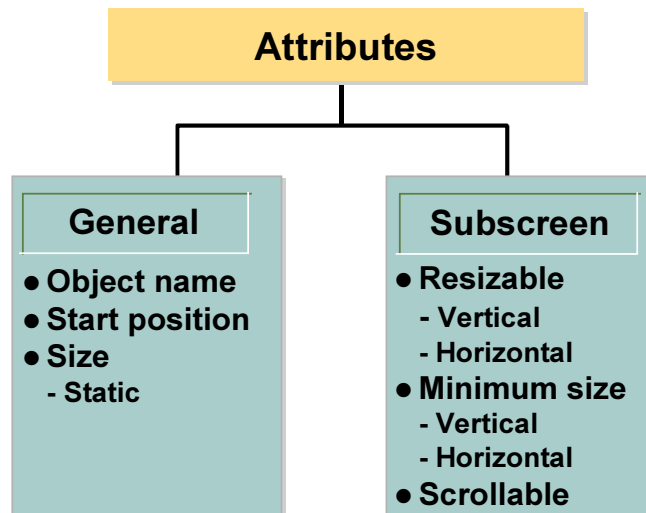
© SAP AG 2002

- A subscreen area is a reserved rectangular area on a screen, in which you place another screen at run time. Subscreen areas may not contain any other screen elements. To use a subscreen, you create a second screen (with the type subscreen) and display it in the subscreen area you defined on the main screen.
- A subscreen is an independent screen which you display within another screen. You may want to use a subscreen as a way of displaying a group of objects from the main screen in certain circumstances, but not in others. You can use this technique to display or hide extra fields on the main screen, depending on the entries the user has made.



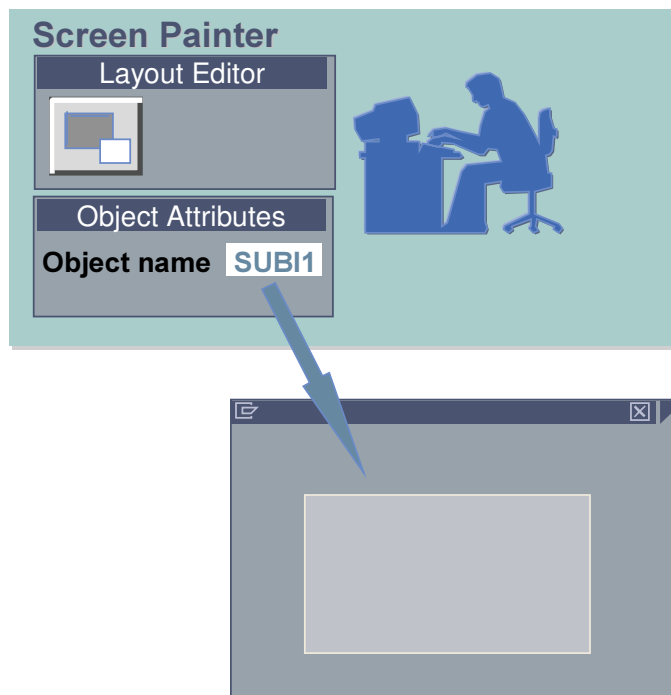
© SAP AG 2002

- A second use for subscreens is that different programs can use the same subscreens. To set this up, you must execute other screen programs within your main program.
- You can include more than one subscreen on a single main screen. You can also determine the subscreens dynamically at run time.
- You can use subscreens in the following circumstances:
  - In screen enhancements (screen exits)
  - Within other screen objects (tabstrip controls)
  - In the Modification Assistant
  - In Web transactions



© SAP AG 2002

- If the subscreen is larger than the subscreen area in which it is called, the system displays only what will fit on the screen, starting at the upper-left corner. However, you can use the *Scrollable* attribute to ensure that, if the screen is too big, the system will display scrollbars.
- The resizing attributes control whether the size of a subscreen area can be changed vertically and horizontally. You should set these attributes if you want the size of the subscreen area to change with the size of the whole window. You can use the minimum size attribute to set a lower limit beyond which the subscreen area cannot be resized.
- The *Context menu* attribute allows you to assign a context-sensitive menu to the output fields on the subscreen.
- The following restrictions apply to subscreens:
  - CALL SUBSCREEN is not allowed between LOOP and ENDLOOP or between CHAIN and ENDCHAIN.
  - A subscreen may not have a named OK\_CODE field.
  - Object names must be unique within the set of all subscreens called in a single main screen.
  - Subscreens may not contain a module with the AT EXIT-COMMAND addition.
  - You cannot use the SET TITLEBAR, SET PF-STATUS, SET SCREEN, or LEAVE SCREEN statements in the modules of a subscreen.

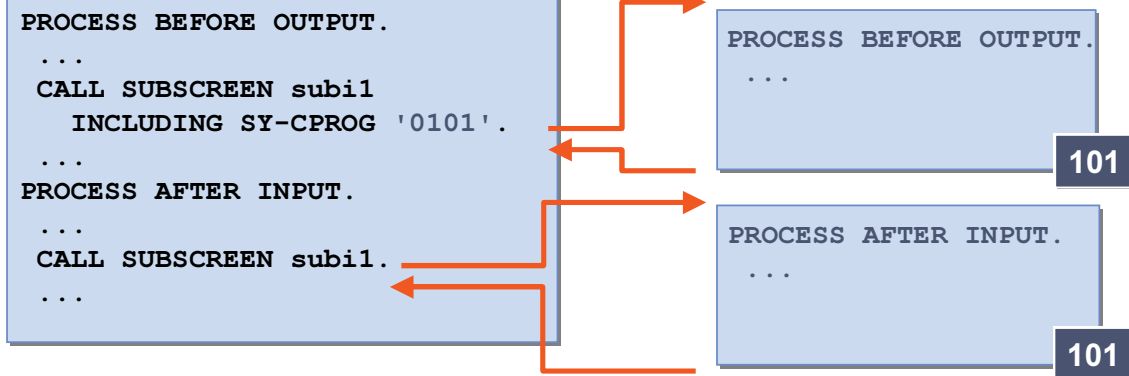


© SAP AG 2002

- To create a subscreen area, choose subscreen from the object list in the Screen Painter and place it on the screen. Fix the top-left corner of the table control area and then drag the object to the required size.
- In the *Object text* field, enter a name for the subscreen area. You need this to identify the area when you call the subscreen.

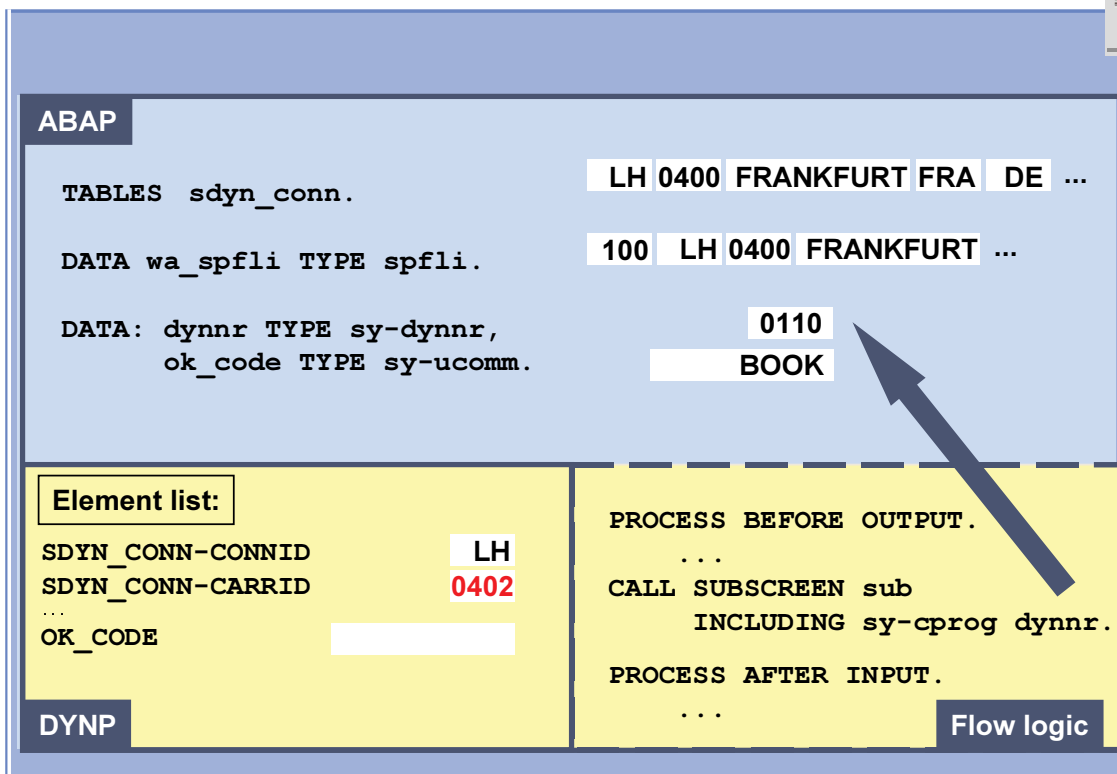
```
PROCESS BEFORE OUTPUT.
  CALL SUBSCREEN <subarea>
    INCLUDING <program_name> <dynpro_number>.
PROCESS AFTER INPUT.
  CALL SUBSCREEN <subarea>.
```

Subscreen in same program



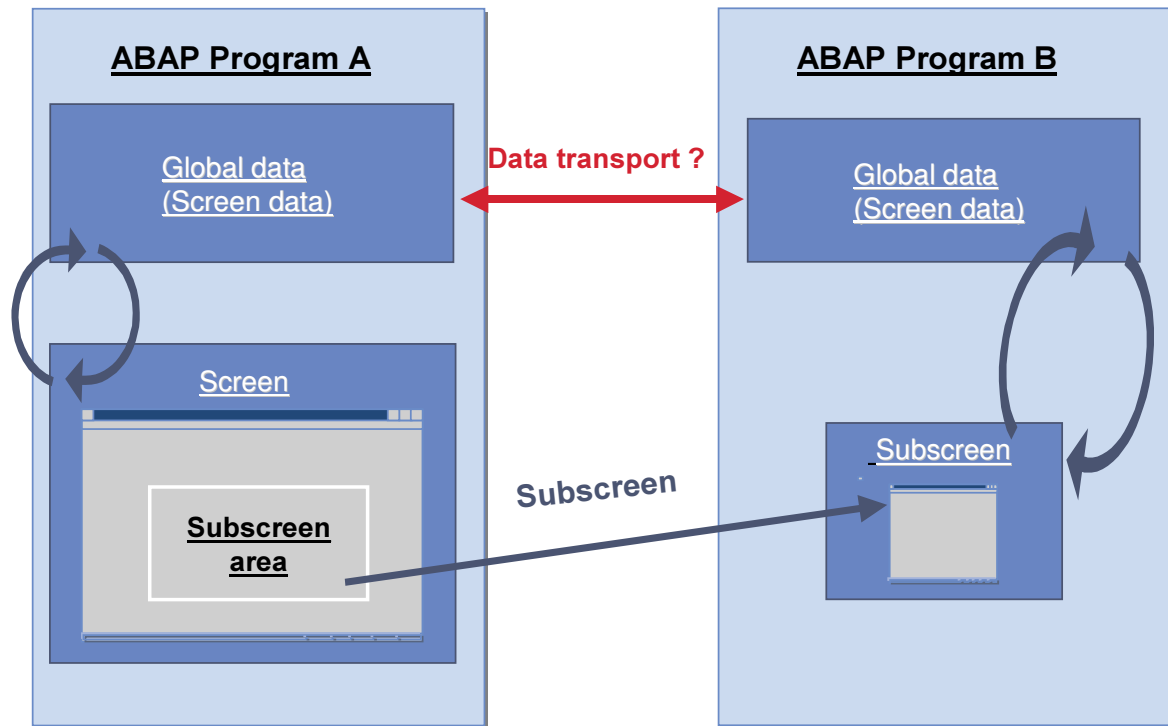
© SAP AG 1999

- To use a subscreen, you must call it in both the PBO and PAI sections of the flow logic of the main screen. The CALL SUBSCREEN <subarea> statement tells the system to execute the PBO and PAI processing blocks for the subscreen as components of the PBO and PAI of the main screen. You program the ABAP modules for subscreens in the same way as for a normal screen (apart from the restrictions already mentioned).



© SAP AG 2002

- The fields that you use within the flow logic are global fields of your ABAP program. These fields must be declared in the TOP include of your program.

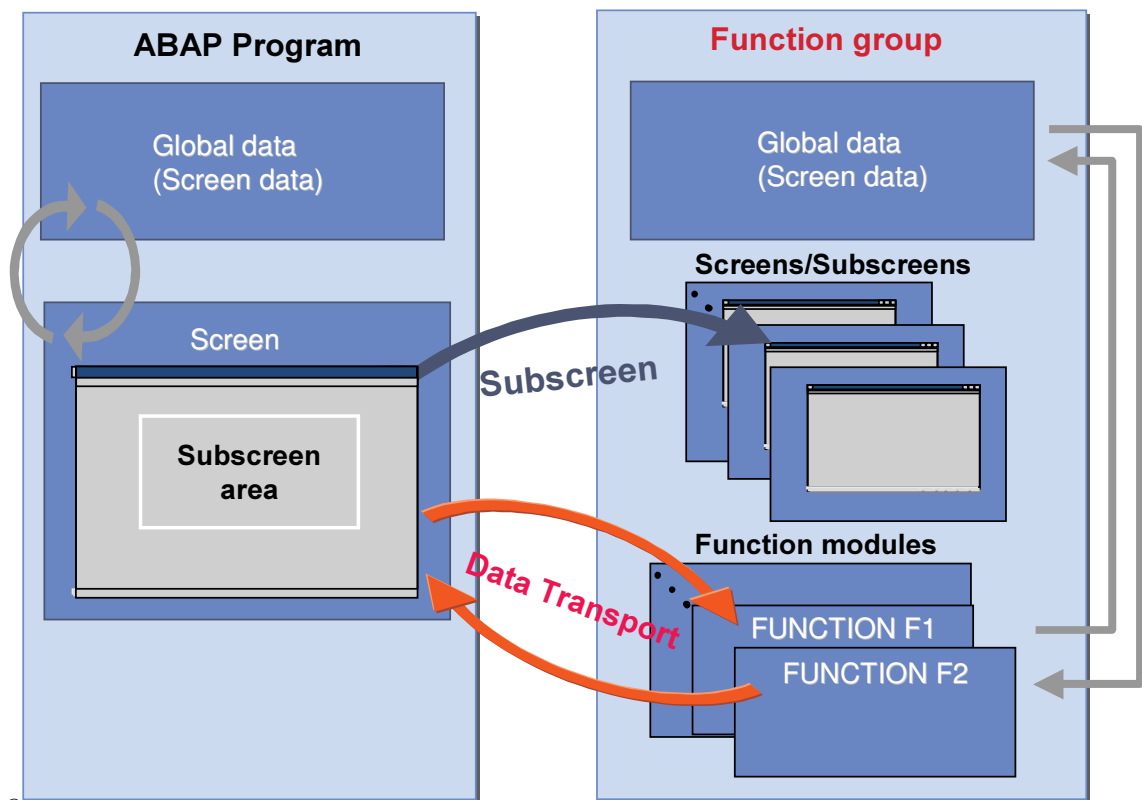


© SAP AG 2002

- If the subscreen is not in the same module pool as the main program, the global data of the main program is not available to the subscreen and the data from the screen will not be transferred back to the program. You must program the data transfer yourself (for example, using a function module that exports and imports data, with an appropriate MOVE statement in the subscreen coding).

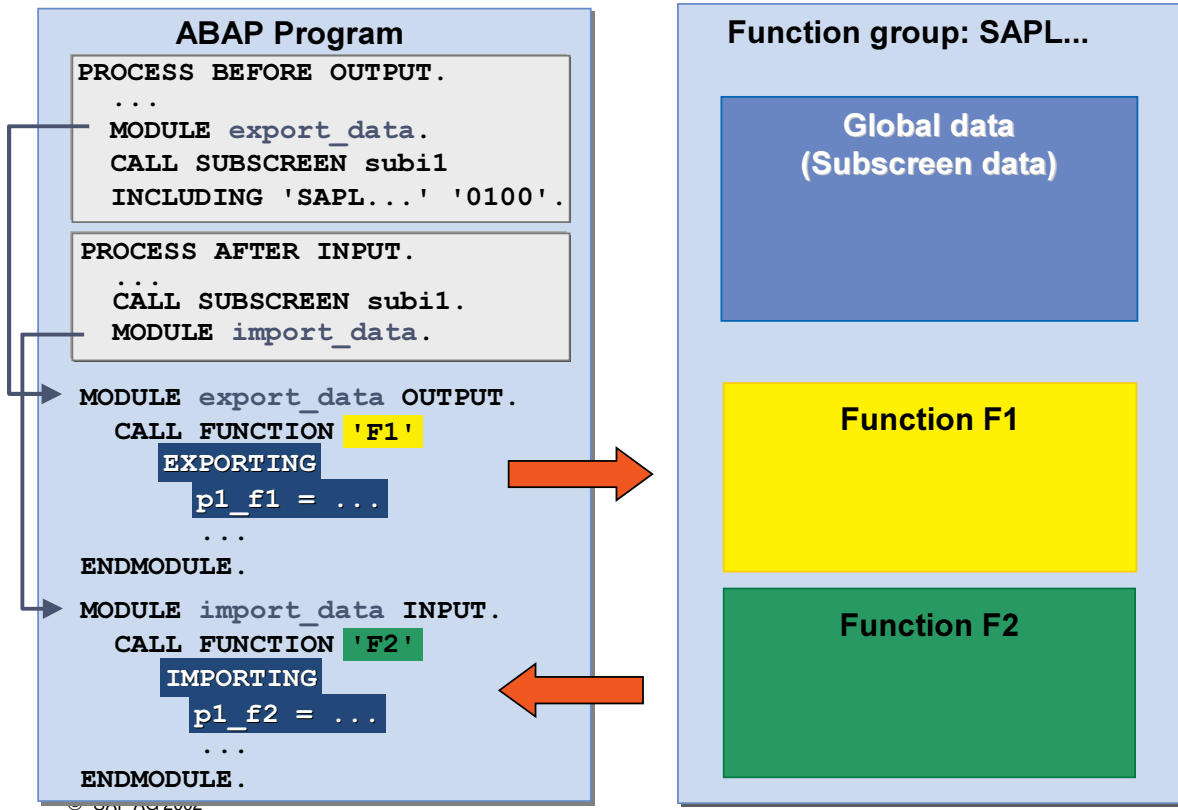
## Subscreens: Encapsulation in Function Groups

SAP

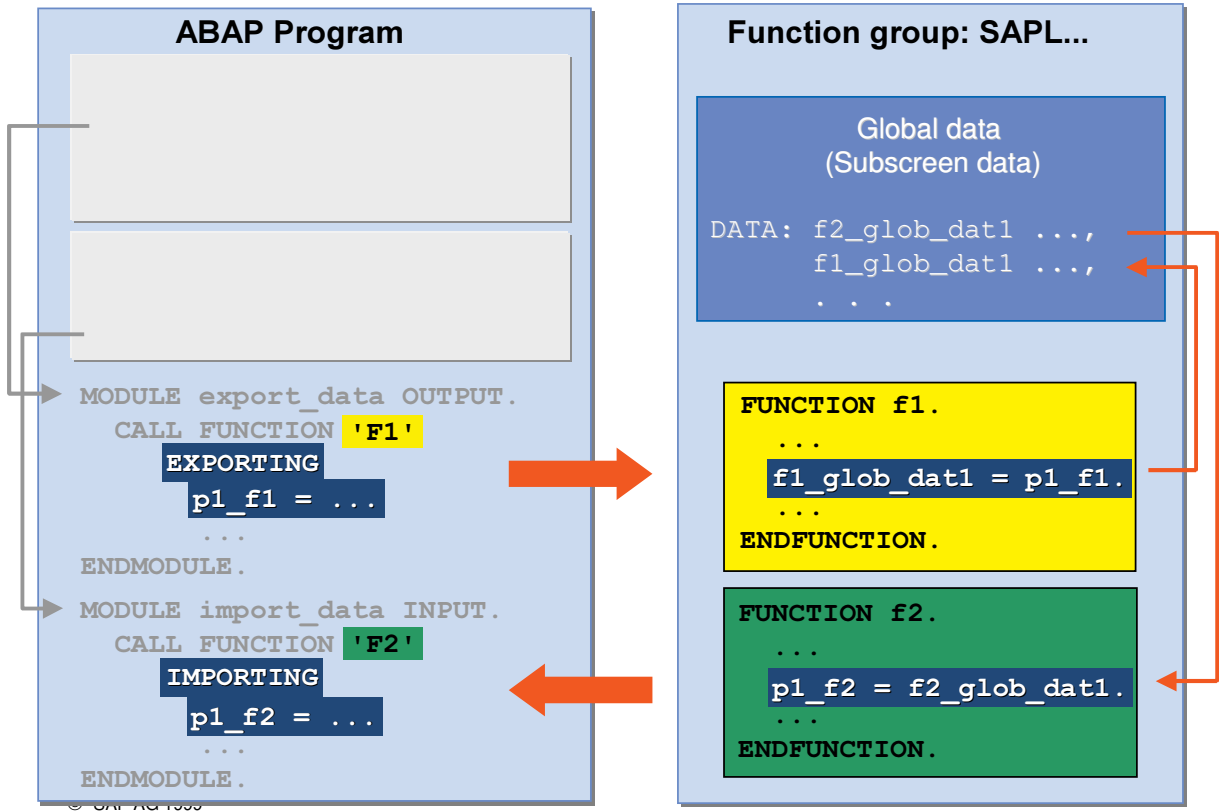


© SAP AG 2002

- If you want to use subscreens in the screens of several different programs, you encapsulate the subscreens in a function group and use function modules to transport data between the program in which you want to use the subscreen and the function group.
- You pass data between the calling program and the function group using the interfaces of the function modules.
- This is the technique used for customer subscreens (screen enhancements).



- You use function modules to transport data between the calling program and the function group.
- To declare the data from the calling program to the subscreen from the function group, use a module before the subscreen call. This calls a function module whose interface you can use to pass the required data to the function group.
- The function module call must occur before the subscreen call. This ensures that the data is known in the function group before the PROCESS BEFORE OUTPUT processing block of the subscreen is called.
- The sequence is reversed in the PAI module of the calling screen. You call the PROCESS AFTER INPUT processing block of the subscreen before you call a function module to pass the data from the function group back to the calling program.



- For the data from the calling program to be available globally in the function group, you must transfer the interface parameters from the function module into global data fields of the function group.
- The function module that you use to transfer the data from the calling program into the function group must copy its interface parameters into the global data in the function group.
- The function module that you use to transfer data from the function group to the calling program must copy the corresponding data from the global data of the function group into its interface parameters.

Subscreens



Tabstrip controls

© SAP AG 1999

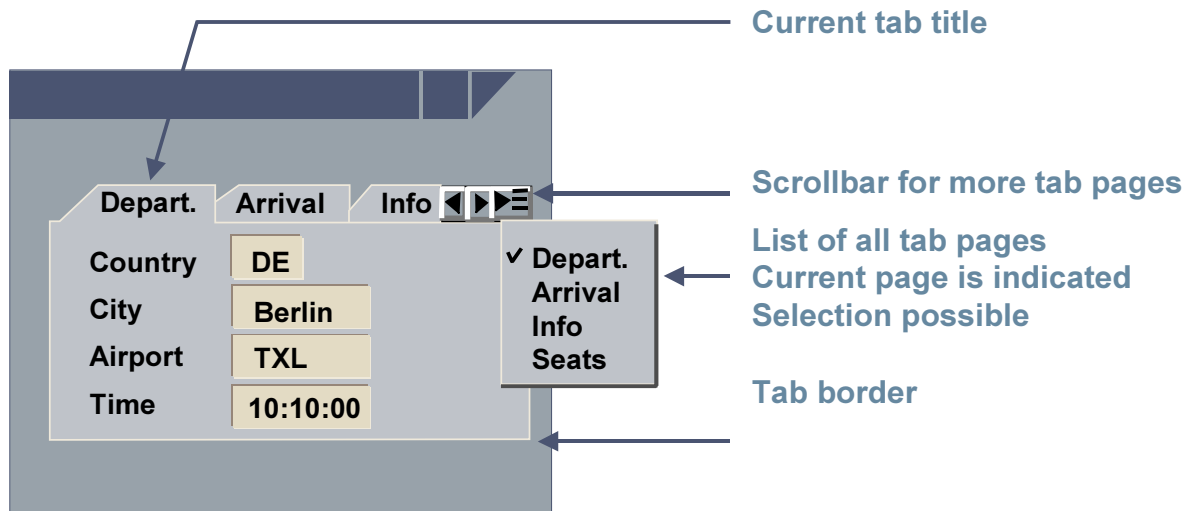


An easy way to present information that belongs together logically

- Displays various components of an application on a single screen and allows the user to navigate between the components
- Container for other screen objects

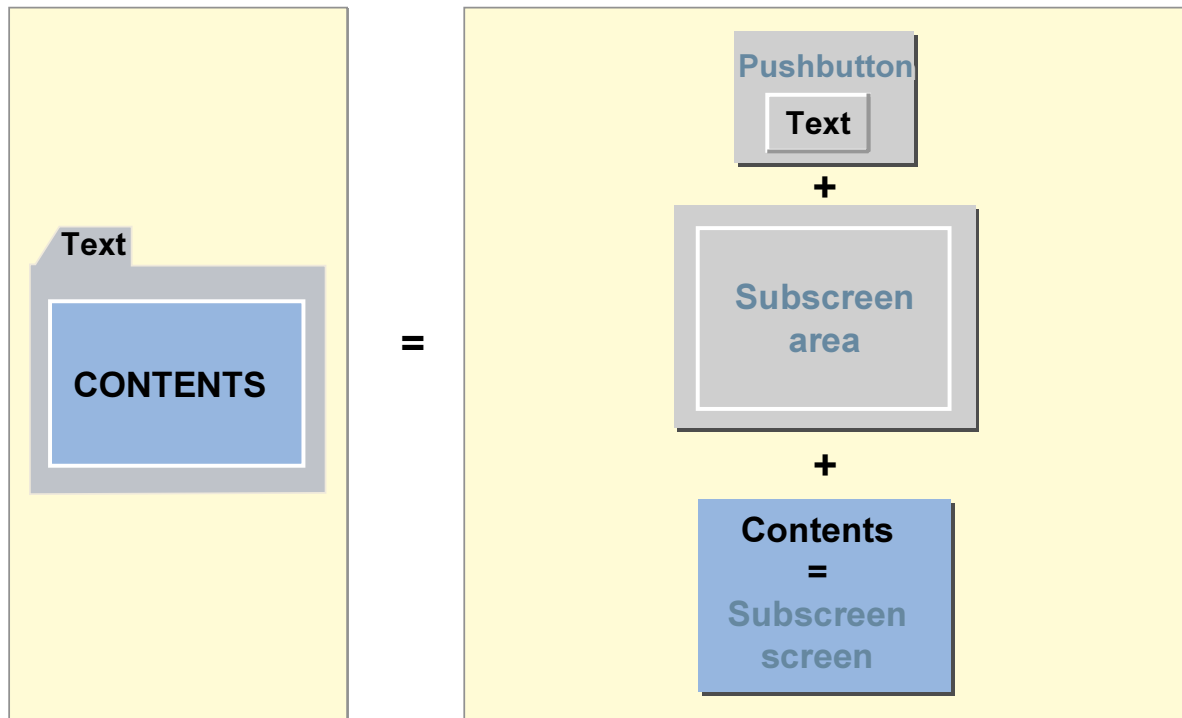
© SAP AG 2002

- Tabstrip controls provide you with an easy, user-friendly way of displaying different components of an application on a single screen and allowing the user to navigate between them. Their intuitive design makes navigation much easier for end users.
- Tabstrip controls are a useful way of simplifying complex applications. You can use tabstrip controls wherever you have different components of an application that form a logical unit. For example, you might have a set of header data that remains constant, while underneath it you want to display various other sets of data.
- You should **not** use tabstrip controls if:
  - You need to change the screen environment (menus, pushbuttons, header data, and so on) while processing the application components. The screen surrounding the tabstrip must remain constant.
  - The components must be processed in a certain order. Tabstrip controls are designed to allow users to navigate freely between components.
  - The components are processed dynamically, that is, user input on one tab page causes other tab pages to suddenly appear.
- Tabstrip controls are compatible with batch input processing.



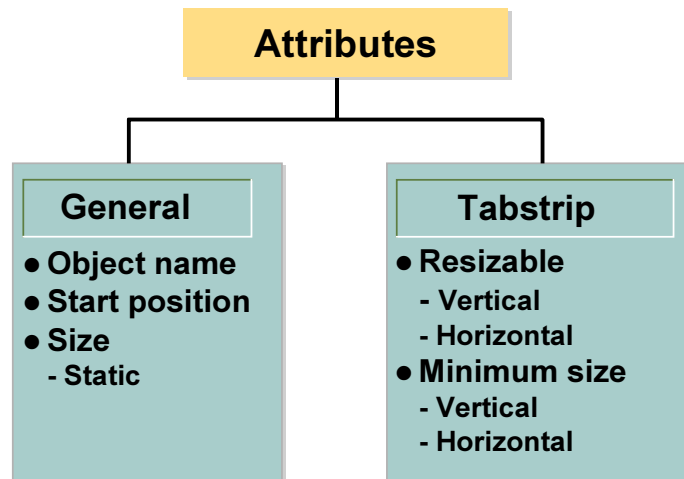
© SAP AG 2002

- A tabstrip control consists of individual pages with a tab page and tab title.
- The tabstrip control may have only one row of tab titles.
- If the tabstrip control contains too many pages, it is not possible for all of the tab titles to be displayed at once. In this case, a scrollbar allows you to scroll through the remaining tab pages. In the upper-right corner of the tab is a pushbutton. If the user selects this pushbutton, a list of all of the tab titles is displayed. The active tab title is marked with a checkmark.



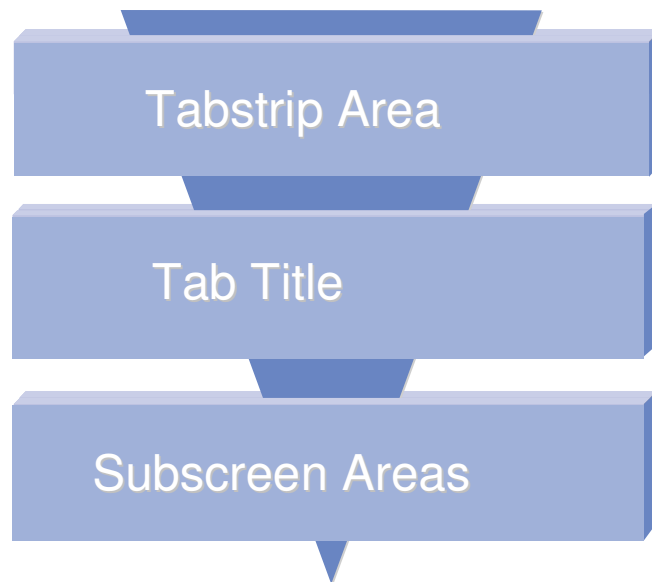
© SAP AG 2002

- A page element consists of a tab title, a subscreen area, and a subscreen.
- From a technical point of view, the system handles tab titles like pushbuttons.
- The contents of page elements are displayed using the subscreen technique. You assign a subscreen area to each page element for which you can then call a subscreen.



© SAP AG 2002

- In addition to the general attributes *Object name*, *Starting position*, and static size, tabstrip controls also have special attributes.
- For details of these special attributes, see the section in this unit on *subscreen attributes*.

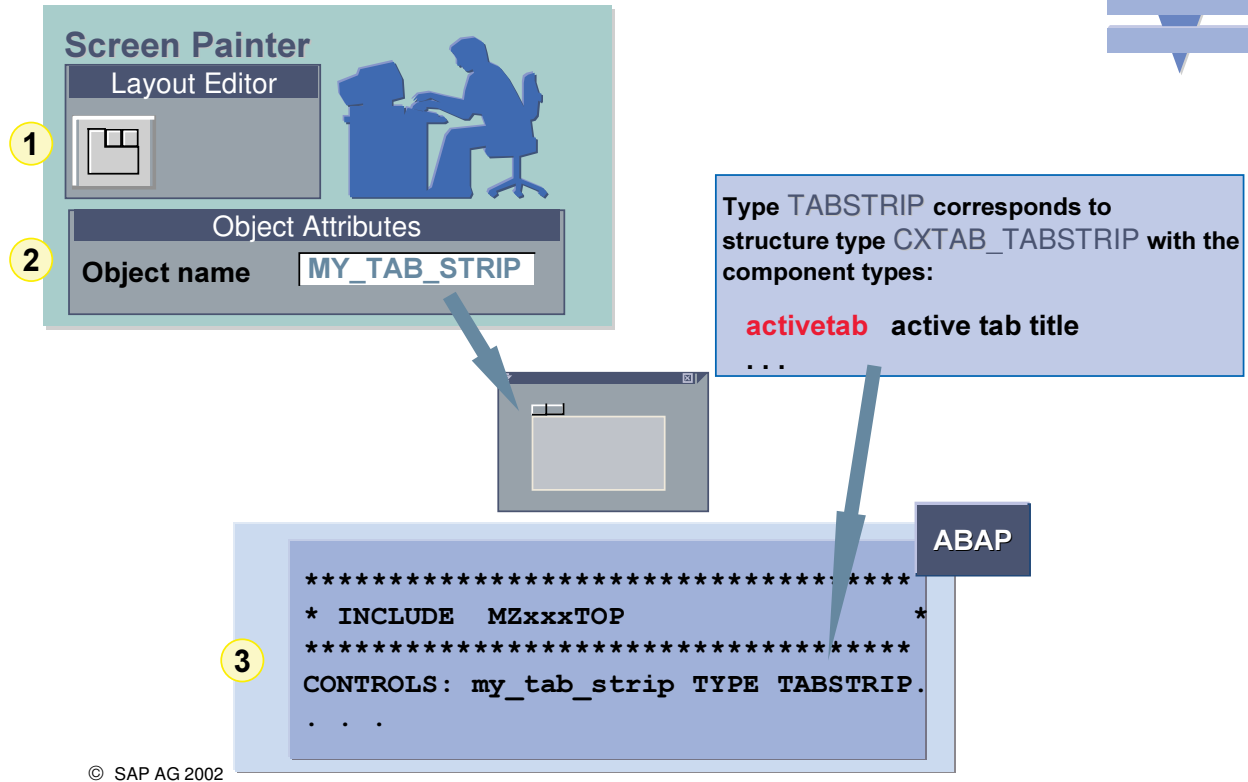


© SAP AG 2002

- You create a tabstrip control by carrying out the following three steps:
  - Define the tab area.
  - Define the tab titles and, if necessary, add further tab titles.
  - Assign a subscreen area to each page element.

## Creating a Tabstrip Control: Tabstrip Area

SAP



© SAP AG 2002

- To create a tabstrip control area, choose *Tabstrip control* from the object list in the Screen Painter and place it on the screen. Fix the upper-left corner of the table control area and then drag the object to the required size.
- Assign a name to the tabstrip control in the *Object name* attribute. You need this name to identify your tabstrip control.
- In your ABAP program, use the **CONTROLS** statement to declare an object with the same name. Use **TABSTRIP** as the type.
- The type **TABSTRIP** is defined in the type pool **CXTAB**. The **ACTIVETAB** field contains the function code of the tab title of the currently active tabstrip. The other fields are reserved for internal use.
- The default number of tab pages for a tabstrip control is two.

## Creating a Tabstrip Control: Tab Title

SAP

**Screen Painter**

Layout Editor

1

Object Attributes

Name:

Text:

2 FctCode:  FctType:  3

Object List

| Name    | Type | FctCode | FctType |
|---------|------|---------|---------|
| but1    |      | FC1     |         |
| but2    |      | FC2     |         |
| but3    |      | FC3     |         |
| ok_code | OK   |         |         |

4

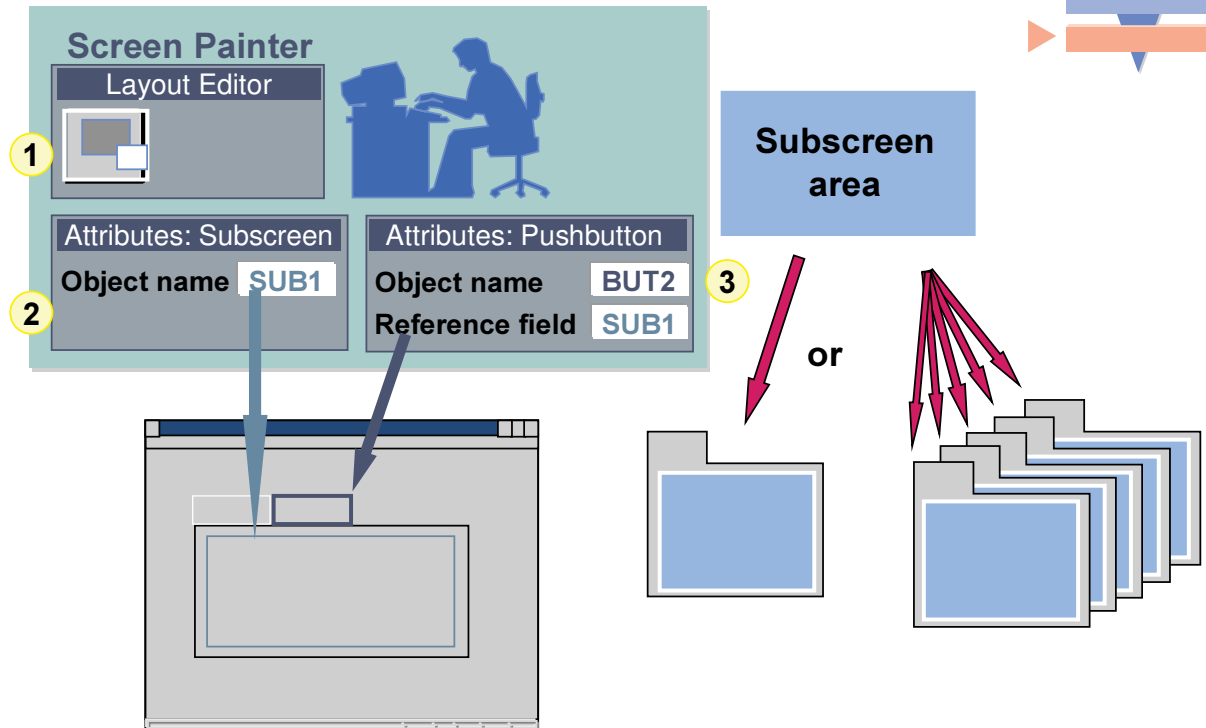
| FctType | Meaning                   |
|---------|---------------------------|
| P       | Local GUI function normal |

© SAP AG 2002

- Technically, tab titles are treated in the same way as pushbuttons. They have a name, a text, a function code, and a function type. You enter these in the *Name*, *Text*, *FctCode*, and *FctType* fields of the object attributes.
- A tab title can have the function type '' (space) or P. If the function type is '' (space), the PAI processing block is triggered when the user chooses that tab and the function code of the tab title is placed in the command field. If the function type is P, the user can scroll between different tab pages of the same type without triggering the PAI processing block. If you want your tabstrip control to have more than two pages, you must create further tab titles. To do this, choose *Pushbutton* from the object list in the Screen Painter and place it in the tab title area.

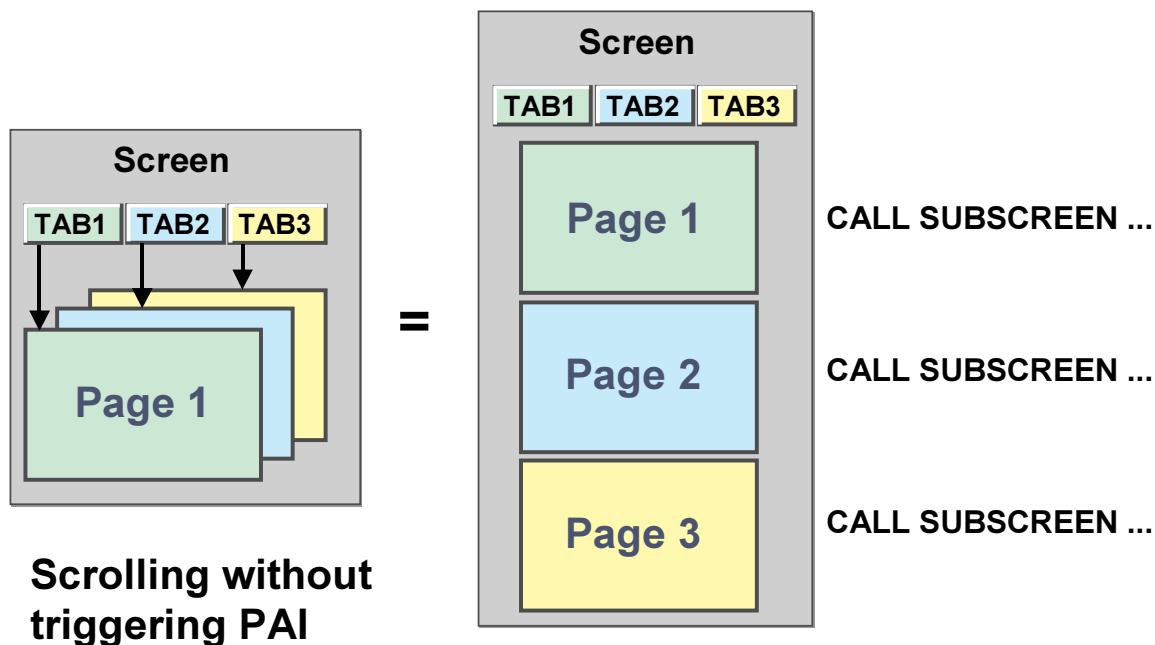
## Creating Tabstrip Control: Tabstrip Subscreens

SAP



© SAP AG 1999

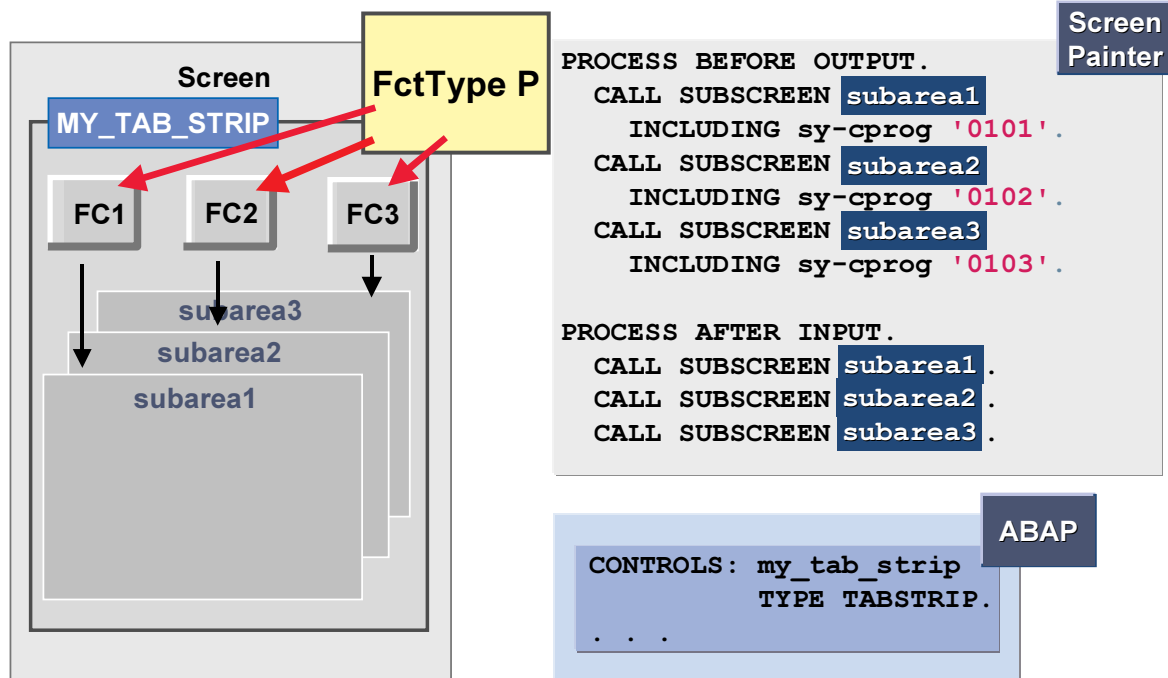
- You must assign a subscreen area to each tab page.
- The subscreen area assigned to a tab page is automatically entered as the *Reference object* (in the *Dictionary* attributes) for the **tab title** of that page.
- To assign a subscreen area to one or more tab pages, choose the relevant tab title in the fullscreen editor, choose the *Subscreen* object, and place it on the tab page.
- Alternatively, you can assign a single subscreen area to several tab pages by entering the name of the subscreen area directly in the *Reference object* field of the attributes of the relevant tab pages.



**Scrolling without triggering PAI**

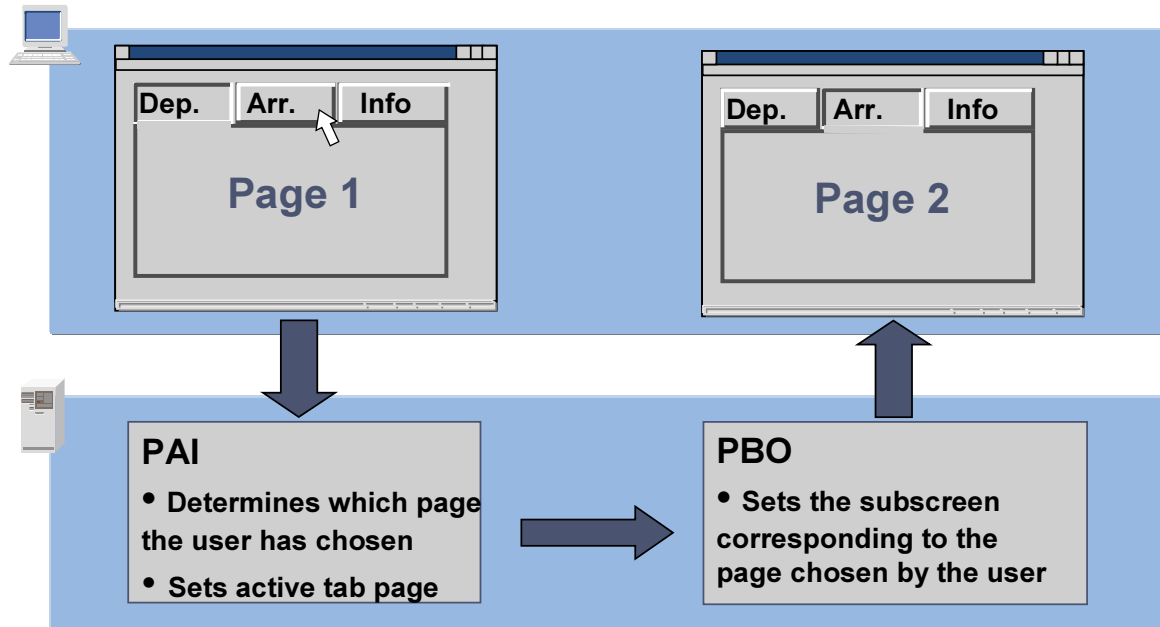
© SAP AG 2002

- If you have assigned a different subscreen area to each page element in a tabstrip control, you can scroll between the pages locally at the front end.
- To do this, you must send all of the subscreens to the front end when you send the main screen itself. All of the tab titles in the tabstrip control must also have function type P.
- When you scroll between the different page elements, there is no communication between the presentation server and the application server.
- When the user chooses a function on the screen that triggers PAI processing, the system processes the PAI blocks of **all** of the subscreens as well. This means that **all of the field checks** are run. In this respect, you could regard the tabstrip control as behaving like a single large screen.
- Local scrolling in tabstrip controls is more appropriate for display transactions.



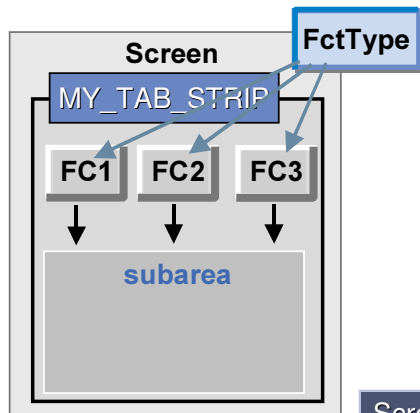
© SAP AG 2002

- To program a tabstrip control to scroll locally at the front end, you must:
  - Assign a separate subscreen area to each tab page. A subscreen will be sent to each of these when the screen is processed.
  - Call all of the subscreens from the flow logic.
  - Assign function type P to all of the tab titles.
- The system hides any page element whose subscreen contains no elements that can be displayed.
- If there are no page elements containing elements that can be displayed, the system hides the entire tabstrip control.
- For more information about tabstrip controls, refer to the online documentation, **SUB-1**.



© SAP AG 2002

- If all of the page elements share a single subscreen area, the program analyzes the function code of the chosen tab title to determine which screen is displayed.
- There are two steps in this process:
  - In the PAI processing block, the program determines which page element needs to be active, based on the tab title chosen by the user.
  - When the PBO processing block is processed again, the program displays the corresponding screen.
- During this process, the system checks only the fields of the displayed subscreen.



Screen Painter

```
PROCESS BEFORE OUTPUT.
  MODULE fill_dynnr.
  ...
  CALL SUBSCREEN subarea
    INCLUDING SY-CPROG dynnr.

PROCESS AFTER INPUT.
  CALL SUBSCREEN subarea .
  ...
  MODULE user_command.
```

```
CONTROLS: my_tab_strip TYPE TABSTRIP.
DATA      : ok_code      TYPE sy-ucomm,
            dynnr         TYPE sy-dynnr.

MODULE fill_dynnr OUTPUT.
  CASE my_tab_strip-activetab.
    WHEN 'FC1'.
      dynnr = '0101'.
    WHEN 'FC2'.
      dynnr = '0102'.
    WHEN 'FC3'.
      dynnr = '0103'.
    WHEN OTHERS.
      dynnr = '0101'.
      my_tab_strip-activetab = 'FC1'.
  ENDCASE.
ENDMODULE.

MODULE user_command INPUT.
  CASE ok_code.
    WHEN 'FC1' OR 'FC2' OR 'FC3'.
      my_tab_strip-activetab = ok_code.
  ENDCASE.
ENDMODULE.
```

ABAP

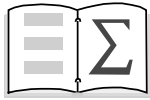
© SAP AG 2002

- If you want the application program to process scrolling in a tabstrip control, the following requirements must be met:
  - All of the tab pages must share a common subscreen area.
  - All of the tab titles must have the function code type '' (space).
  - In the flow logic, you must use a variable to call the screen that is to be displayed in the subscreen area.
- In the PAI block, you must call a module in which the function code of the active tab title is placed in the ACTIVETAB field of the structure you created in your program with type TABSTRIP. In the example in the graphic, this is MY\_TAB\_STRIP.
- The PBO processing block must contain a module before the subscreen is called, in which you place the name of the subscreen in the corresponding variable. You must assign an initial value to this field so that the screen is processed the first time (before the user has had a chance to choose a tab title).
- You can hide a tab page at run time by setting the corresponding tab title to inactive using the system table SCREEN (SCREEN-ACTIVE = 0). You should do this before processing the tabstrip control for the first time to ensure that the screen environment remains constant.



© SAP AG 2002

- You can use the *Tabstrip Control Wizard* to help you create tabstrip controls and insert them on screens in a program. The Wizard guides you through the process. You can return to previous settings at any time. Program objects are created upon the final screen only on completion of the process. The Wizard creates not only the tabstrip control but also the corresponding statements in the flow logic, together with the relevant modules, subroutines, and necessary data definitions.
- In addition to the tabstrip control on the screen and the corresponding flow logic, the following objects are created if they do not already exist:
  - The main program and the screen for the tabstrip control together with its flow logic.
  - Empty subscreens for the individual tabstrip control pages.
  - Includes for data definition, PBO modules, PAI modules, and INCLUDE statements for these includes.
- All objects are placed in the inactive object list.



**You are now able to:**

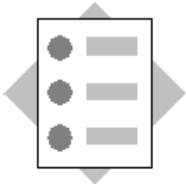
- **Use subscreens and tabstrip controls on screens and selection screens in your programs**

# Exercises



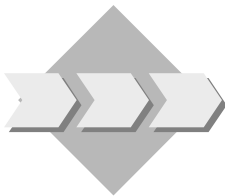
## Unit: Subscreens and Tabstrip Controls

### Topic: Creating subscreens and tabstrip controls



At the conclusion of these exercises, you will be able to:

- Use subscreens and tabstrip controls on screens and selection screens in your programs.



Display additional information on your screen, depending on the mode in which the user is working.

Extend the display to allow users to switch between the additional information using a tabstrip control.

1-1 Extend the *Maintenance screen* (100) to display flight information and the aircraft type. Use a subscreen to do this.

1-1-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410AINPS\_RADIOBUTTON**). You can use the model solution **SAPMBC410ASUBS\_SUBSCREEN** for orientation.

1-1-2 On the maintenance screen (100), create a subscreen area with the following attributes:

|           |     |                                                               |
|-----------|-----|---------------------------------------------------------------|
| Subscreen | SUB | <b>Attributes:</b><br>Vertical and horizontal<br>Resizing: ON |
|-----------|-----|---------------------------------------------------------------|

- 1-1-3 Create three screens 110, 120, and 130, each with the type subscreen and the following attributes:

|            |                                                                                                                                                                          |                                                                                                       |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| Screen 110 | <b>I/O fields, text fields:</b><br><br>SDYN_CONN<br>- COUNTRYFR<br>- COUNTRYTO<br>- CITYFROM<br>- CITYTO<br>- AIRPFROM<br>- AIRPTO<br>- DEPTIME<br>- ARRTIME             | <b>For each field:</b><br><br><b>Input:</b> OFF<br><b>Output:</b> ON                                  |
| Screen 120 | <b>I/O Fields, Text fields:</b><br>SAPLANE<br>- PLANETYPE<br>- PRODUCER<br>- SEATSMAX<br>- TANKCAP<br>- CAP_UNIT<br>- WEIGHT<br>- WEI_UNIT<br>- OP_SPEED<br>- SPEED_UNIT | <b>Attributes for each field:</b><br><b>Input:</b> OFF<br><b>Output:</b> ON<br><b>Output only:</b> ON |
| Screen 130 | <b>empty</b> (Provided for the bookings table)                                                                                                                           |                                                                                                       |

- 1-1-4 In your TOP include, create a field **DYNNR** that you can use in the flow logic to determine which subscreen should appear in the subscreen area.
- 1-1-5 Call the subscreen screens in the flow logic of screen 100. Before the call, write a PBO module to determine which of the subscreens will appear. If the user is in *Display* mode, call subscreen screen 110 with the flight information. If the user is in *Maintain flight data* mode, call subscreen screen 120 with the aircraft information. If the user chooses *Maintain bookings* mode, then empty screen 130 appears.
- 1-1-6 In the flow logic of screen 110, read the flight information from table SPFLI using the key field values.
- 1-1-7 In the flow logic for screen 120, read the information for the aircraft information from table SAPLANE using the value you have for the aircraft type.

- 1-2 Create a tabstrip control on screen 100 for displaying extra flight information and details of the aircraft type.

1-2-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410ASUBS\_SUBSCREEN**). You can use the model solution **SAPMBC410ASUBS\_TABSTRIP** for orientation.

- 1-2-2 **Creating a tabstrip:** Remove the subscreen area on screen 100 and create a tabstrip control with the following attributes:

|                             |                             |                                                                                                                                                                     |
|-----------------------------|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tabstrip controls           | <b>Name:</b><br>MY_TABSTRIP | <b>Attributes:</b><br>Vertical and horizontal<br>Resizing: ON                                                                                                       |
| Pushbutton<br>(Tab title 1) | <b>Name:</b><br>P1          | <b>Attributes:</b><br><b>Text:</b> View flight data<br><b>Function code:</b> FC1<br><b>Function type:</b> <blank><br><b>Reference field:</b> SUB                    |
| Pushbutton<br>(Tab title 2) | <b>Name:</b><br>P2          | <b>Attributes:</b><br><b>Text:</b> View technical data<br>for aircraft<br><b>Function code:</b> FC2<br><b>Function type:</b> <blank><br><b>Reference field:</b> SUB |
| Pushbutton<br>(Tab title 3) | <b>Name:</b><br>P3          | <b>Attributes:</b><br><b>Text:</b> Maintain booking<br><b>Function code:</b> FC3<br><b>Function type:</b> <blank><br><b>Reference field:</b> SUB                    |

In the TOP include of your program, create a data object for the tabstrip control using the following statement:

**CONTROLS MY\_TABSTRIP . . .**

- 1-2-3 In the flow logic of screen 100, implement the call for the subscreen screen in the tabstrip control. The subscreen number will be in the field DYNNR created in the previous exercise.
- 1-2-4 Before calling the subscreen, write a PBO module in which you determine which of the subscreens is to be called (regardless of the mode in which the user is working). Additionally, determine which subscreen screen you want to set the first time the screen is displayed and assign the corresponding function code to the field **MY\_TABSTRIP-ACTIVETAB**.
- 1-2-5 Extend your command field processing for screen 100 with the scroll logic for the tabstrip control. Do this by assigning the relevant value to **MY\_TABSTRIP-ACTIVETAB**.



## Unit: Subscreens and Tabstrip Controls

### Topic: Creating subscreens and tabstrip controls

#### 1-1 Model solution SAPMBC410ASUBS\_SUBSCREEN

Add the coding in bold type, and create new modules where appropriate using forward navigation.

##### Top include

```
PROGRAM  sapmbc410asubs_subscreen MESSAGE-ID bc410 .

DATA  dynnr TYPE sy-dynnr.                                "#EC NEEDED

TABLES saplane.

TABLES  sdyn_conn.

DATA: BEGIN OF mode,
      view VALUE 'X',                                "selected
      maintain_flights,
      maintain_bookings,
END OF mode.

DATA ok_code TYPE sy-ucomm.

DATA wa_sflight TYPE sflight.
```

##### Subroutine include

No changes are necessary.

## Flow logic screen 100

PROCESS BEFORE OUTPUT.

MODULE status\_0100.

MODULE modify\_screen.

**MODULE fill\_dynnr.**

**CALL SUBSCREEN sub INCLUDING sy-cprog dynnr.**

MODULE clear\_ok\_code.

\*

PROCESS AFTER INPUT.

MODULE exit AT EXIT-COMMAND.

\* **CALL SUBSCREEN sub.**

CHAIN.

FIELD: sdyn\_conn-carrid,  
sdyn\_conn-connid,  
sdyn\_conn-fldate MODULE check\_sflight ON CHAIN-REQUEST.

ENDCHAIN.

CHAIN.

FIELD: sdyn\_conn-planetype,  
sdyn\_conn-seatsmax MODULE check\_planetype ON CHAIN-REQUEST.

ENDCHAIN.

MODULE trans\_from\_dynp.

MODULE user\_command\_0100.

## Flow logic screen 110

PROCESS BEFORE OUTPUT.

**MODULE** get\_spfli.

PROCESS AFTER INPUT.

## Flow logic screen 120

PROCESS BEFORE OUTPUT.

**MODULE** get\_saplane.

PROCESS AFTER INPUT.

## Flow logic screen 130

No changes are necessary.

## PBO module include

MODULE status\_0100 OUTPUT.

SET PF-STATUS 'STATUS\_100'.

SET TITLEBAR 'TITLE\_100'.

ENDMODULE. " STATUS\_0100 OUTPUT

MODULE clear\_ok\_code OUTPUT.

CLEAR ok\_code.

ENDMODULE. " clear\_ok\_code OUTPUT

MODULE modify\_screen OUTPUT.

CHECK NOT mode-maintain\_flights IS INITIAL.

LOOP AT SCREEN.

IF screen-name = 'SDYN\_CONN-PLANETYPE'.

screen-input = 1.

screen-required = 1.

```

        MODIFY SCREEN.
    ENDIF.
ENDLOOP.
ENDMODULE.                                " modify_screen  OUTPUT

```

```

MODULE get_spfli OUTPUT.
    ON CHANGE OF wa_sflight-carrid
        OR wa_sflight-connid.
        SELECT SINGLE * INTO CORRESPONDING FIELDS OF sdyn_conn FROM spfli
            WHERE carrid = wa_sflight-carrid
            AND connid = wa_sflight-connid.
    ENDON.
ENDMODULE.                                " GET_SPFLI  OUTPUT

```

```

MODULE get_saplane OUTPUT.
    ON CHANGE OF wa_sflight-planetype.
        SELECT SINGLE * FROM saplane
            WHERE planetype = wa_sflight-planetype.
    ENDON.
ENDMODULE.                                " GET_SAPLANE  OUTPUT

```

```

MODULE fill_dynnr OUTPUT.
    CASE 'X'.
        WHEN mode-view.
            dynnr = 110.
        WHEN mode-maintain_flights.
            dynnr = 120.
        WHEN mode-maintain_bookings.
            dynnr = 130.
    ENDCASE.
ENDMODULE.                                " set_dynnr  OUTPUT

```

## PAI module include

```

MODULE user_command_0100 INPUT.
    CASE ok_code.
        WHEN 'SAVE'.
            PERFORM update_sflight.
        WHEN 'BACK'.

```

```

        LEAVE TO SCREEN 0.
    ENDCASE.
ENDMODULE.                                " USER_COMMAND_0100  INPUT

MODULE trans_from_dynp INPUT.
    MOVE-CORRESPONDING sdyn_conn TO wa_sflight.
ENDMODULE.                                " trans_from_dynp  INPUT

MODULE read_sflight INPUT.
    SELECT SINGLE * INTO CORRESPONDING FIELDS OF sdyn_conn FROM sflight
        WHERE carrid = sdyn_conn-carrid
            AND connid = sdyn_conn-connid
            AND fldate = sdyn_conn-fldate.
    IF sy-subrc NE 0.
        MESSAGE e038.
    ENDIF.
ENDMODULE.                                " read_sflight  INPUT

MODULE exit INPUT.
    CASE ok_code.
        WHEN 'EXIT'.
            LEAVE PROGRAM.
        WHEN 'CANCEL'.
            CLEAR: sdyn_conn, saplane, wa_sflight.
            SET PARAMETER ID: 'CAR' FIELD space,
                                'CON' FIELD space,
                                'DAY' FIELD space.
            LEAVE TO SCREEN 100.
        ENDCASE.
ENDMODULE.                                " exit  INPUT

MODULE check_planetype INPUT.
    SELECT SINGLE seatsmax INTO sdyn_conn-seatsmax FROM saplane
        WHERE planetype = sdyn_conn-planetype.
    CHECK sdyn_conn-seatsmax < sdyn_conn-seatsocc.
    MESSAGE e109.
ENDMODULE.                                " check_planetype  INPUT

```

## 1-2 Model solution **SAPMBC410ASUBS\_TABSTRIP**

Add the coding in bold type, and create new modules where appropriate using forward navigation.

### Top include

```
CONTROLS my_tabstrip TYPE TABSTRIP.
```

### Subroutine include

No changes are necessary.

### Flow logic screen 100

No changes are necessary.

### Flow logic screen 110

No changes are necessary.

### Flow logic screen 120

No changes are necessary.

### Flow logic screen 130

No changes are necessary.

### PBO module include

```
MODULE status_0100 OUTPUT.  
    SET PF-STATUS 'STATUS_100'.  
    SET TITLEBAR 'TITLE_100'.  
ENDMODULE.                                " STATUS_0100  OUTPUT  
  
MODULE clear_ok_code OUTPUT.  
    CLEAR ok_code.  
ENDMODULE.                                " clear_ok_code  OUTPUT  
  
MODULE modify_screen OUTPUT.  
    CHECK NOT mode-maintain_flights IS INITIAL.  
    LOOP AT SCREEN.
```

```

    IF screen-name = 'SDYN_CONN-PLANETYPE'.
        screen-input = 1.
        screen-required = 1.
        MODIFY SCREEN.
    ENDIF.
ENDLOOP.
ENDMODULE.                                " modify_screen  OUTPUT

MODULE get_spfli OUTPUT.
    ON CHANGE OF wa_sflight-carrid OR wa_sflight-connid.
        SELECT SINGLE * INTO CORRESPONDING FIELDS OF sdyn_conn FROM spfli
            WHERE carrid = wa_sflight-carrid
            AND connid = wa_sflight-connid.
    ENDON.
ENDMODULE.                                " GET_SPFLI  OUTPUT

MODULE get_saplane OUTPUT.
    ON CHANGE OF wa_sflight-planetype.
        SELECT SINGLE * FROM saplane
            WHERE planetype = wa_sflight-planetype.
    ENDON.
ENDMODULE.                                " GET_SAPLANE  OUTPUT

MODULE fill_dynnr OUTPUT.
    CASE my_tabstrip-activetab.
        WHEN 'FC1'.
            dynnr = 110.
        WHEN 'FC2'.
            dynnr = 120.
        WHEN 'FC3'.
            dynnr = 130.
        WHEN OTHERS.
            my_tabstrip-activetab = 'FC1'.
            dynnr = 110.
    ENDCASE.

```

## PAI module include

```
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'FC1' or 'FC2' or 'FC3'.
      my_tabstrip-activetab = ok_code.
    WHEN 'SAVE'.
      PERFORM update_sflight.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.                                " USER_COMMAND_0100  INPUT

MODULE trans_from_dynp INPUT.
  MOVE-CORRESPONDING sdyn_conn TO wa_sflight.
ENDMODULE.                                " trans_from_dynp  INPUT

MODULE read_sflight INPUT.
  SELECT SINGLE * INTO CORRESPONDING FIELDS OF sdyn_conn FROM sflight
    WHERE carrid = sdyn_conn-carrid
      AND connid = sdyn_conn-connid
      AND fldate = sdyn_conn-fldate.
  IF sy-subrc NE 0.
    MESSAGE e038.
  ENDIF.
ENDMODULE.                                " read_sflight  INPUT

MODULE exit INPUT.
  CASE ok_code.
    WHEN 'EXIT'.
      LEAVE PROGRAM.
    WHEN 'CANCEL'.
      CLEAR: sdyn_conn, saplane, wa_sflight.
      SET PARAMETER ID: 'CAR' FIELD space,
                        'CON' FIELD space,
                        'DAY' FIELD space.
      LEAVE TO SCREEN 100.
    ENDCASE.
ENDMODULE.                                " exit  INPUT
```

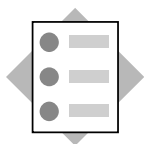
```
MODULE check_planetype INPUT.  
  SELECT SINGLE seatsmax INTO sdyn_conn-seatsmax FROM saplane  
    WHERE planetype = sdyn_conn-planetype.  
  CHECK sdyn_conn-seatsmax < sdyn_conn-seatsocc.  
  MESSAGE e109.  
ENDMODULE.                                " check_planetype  INPUT
```

### Contents:

- Table controls: overview
- Creating a table control
- Processing a table control
- Further techniques

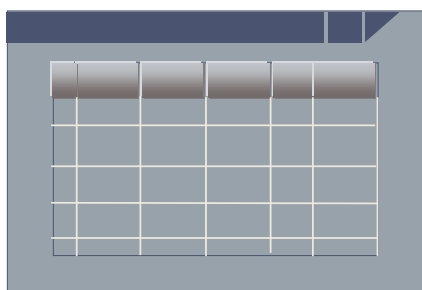


© SAP AG 2002

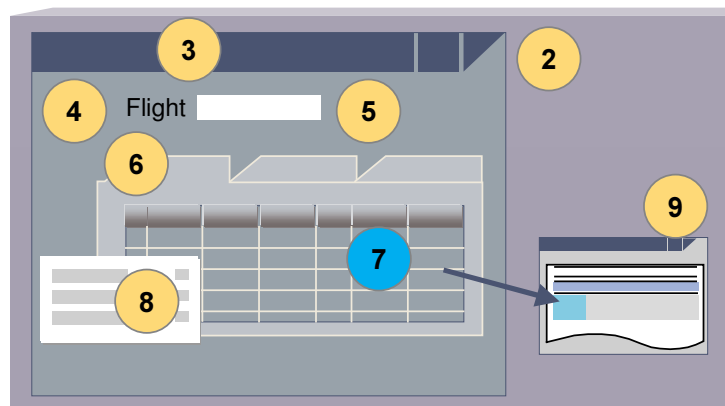


**At the conclusion of this unit, you will be able to:**

- **Display the contents of an internal table in a table control and implement special table control functions.**



© SAP AG 2002



© SAP AG 2002

- |          |                                                   |
|----------|---------------------------------------------------|
| ■ Unit 1 | Course Overview                                   |
| ■ Unit 2 | Introduction to Screen Programming                |
| ■ Unit 3 | The Program Interface                             |
| ■ Unit 4 | Screen Elements for Output                        |
| ■ Unit 5 | Screen Elements for Input/Output                  |
| ■ Unit 6 | Screen Elements: Subscreens and Tabstrip Controls |
| ■ Unit 7 | <b>Screen Elements: Table Controls</b>            |
| ■ Unit 8 | Context Menus                                     |
| ■ Unit 9 | Lists in Screen Programming                       |



Table controls: overview

Creating table control

Processing table control

Further techniques

Airline

| Flight | Dep.      | Arrival |
|--------|-----------|---------|
| 0400   | Frankfurt | New Yo  |
| 0402   | Frankfurt | New Yo  |
| 2407   | Berlin    | San Fra |
|        |           |         |

### Displaying large amounts of data in a table

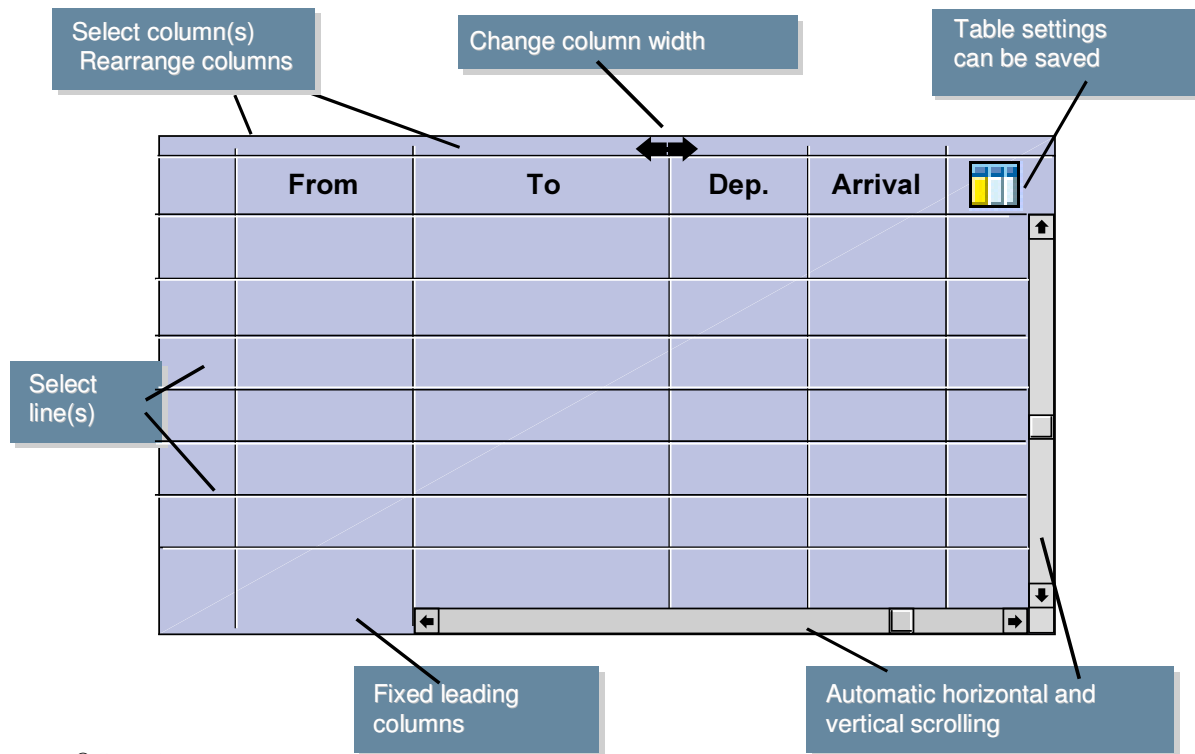
- Can be configured by the user
- Allows you to change several lines of a table at the same time

© SAP AG 1999

- A table control is an area on the screen in which the system displays data in tabular form. It is processed using a loop. The top line of a table control is the header line, which is distinguished by a gray separator.
- Within a table control, you can use table elements, key words, templates, checkboxes, radio buttons, radio button groups, and pushbuttons. A line may have up to 255 columns; each column may have a title.

## Table Controls: Features

SAP



© SAP AG 2002

- You can display or enter single structured lines of data using a table control.
- Features:
  - Resizable table for displaying and editing data
  - The user or program can change the column width and position, save the changes, and reload them later
  - Check column for marking lines; marked lines are highlighted in a different color
  - Line selection: single lines, multiple lines, all lines, and deselection
  - Column headings double as pushbuttons for marking columns
  - Scrollbars for horizontal and vertical scrolling
  - Any number of key (leading) columns can be set
  - Cell attributes are variable at run time

**Table Settings**

**Select variants**

|                 |               |
|-----------------|---------------|
| Current setting | Basic setting |
| Default setting | Basic setting |

**Variant administration**

**Variants**

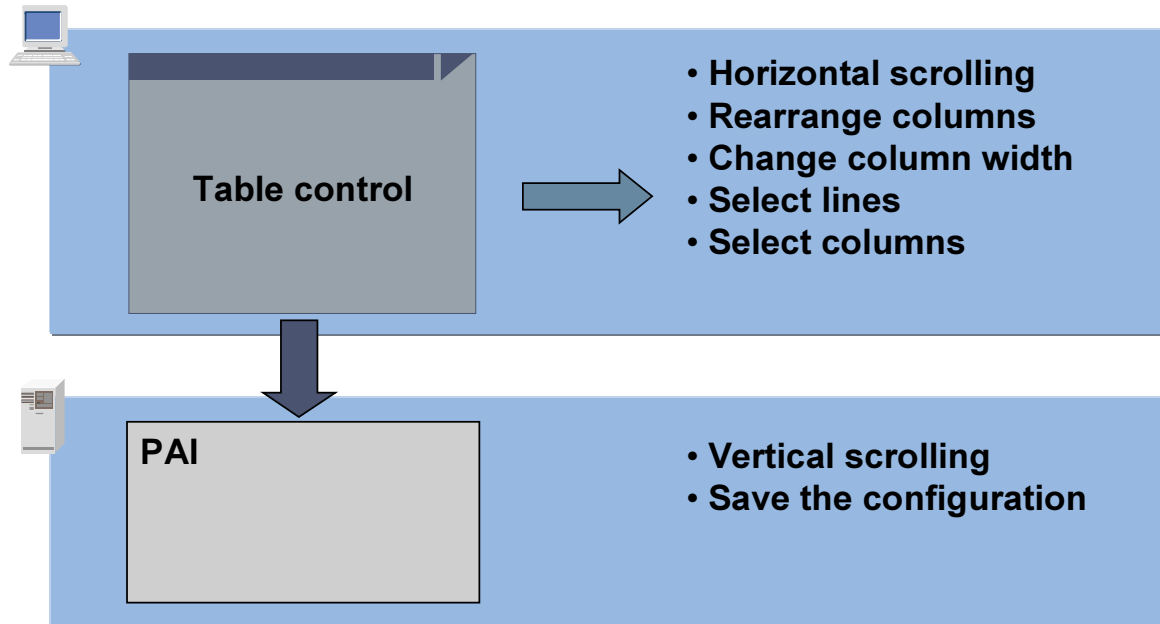
☒ Use as default setting

Create Delete

OK Administrator i X

© SAP AG 2002

- Users can save display variants for table controls. Users can save these variants along with the basic setting, as the current display setting or as the default display setting.



© SAP AG 2002

- The table control contains a series of actions that are controlled entirely at the presentation server:
  - Horizontal scrolling using the scrollbar in the table control
  - Swapping columns
  - Changing column widths
  - Selecting columns
  - Selecting lines
- The PAI processing block is triggered when you scroll vertically in the table control or save the user configuration.

Table controls: overview

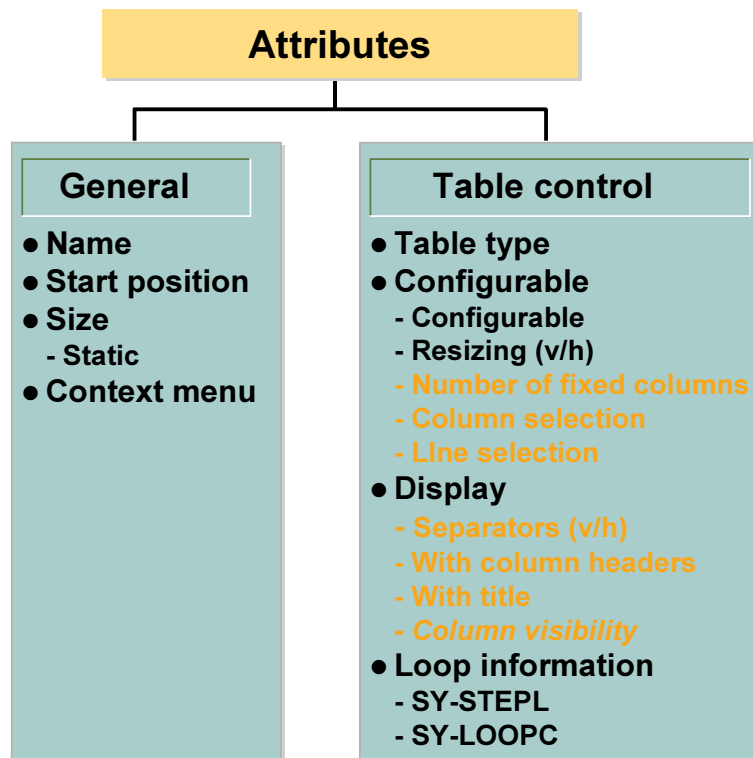


Creating table control

Processing table control

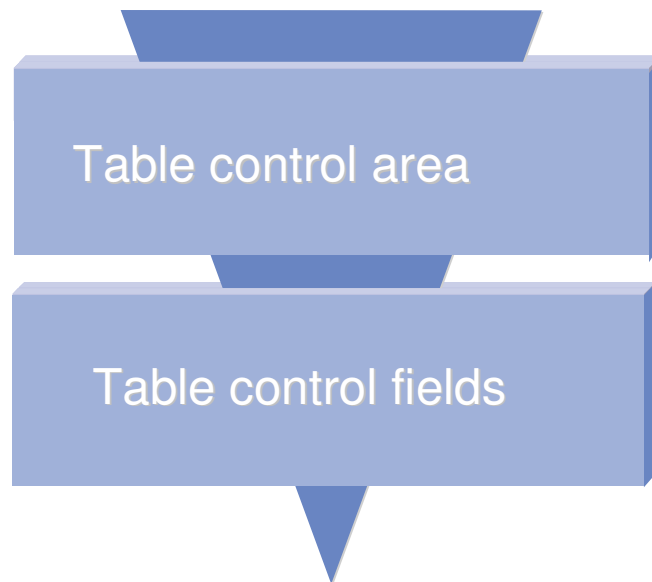
Further techniques

© SAP AG 2002



© SAP AG 2002

- In addition to the normal *Name*, *Start position on screen*, and *Static size* attributes, table controls also have special attributes.
- The special attributes determine the table type and display options for a table control, as well as whether the table control can be configured by the user. The *stepl* and *loopc* fields of *structure syst* contain information about the loop processing used with table controls (see the following graphics).
- Use *Entry table* as the table type if you are creating the table to enter data, if the table has at least one field ready for input. Use *Selection table* if the table is only for selecting and transferring entries, or if the table exists only in display mode.
- For more information about selection screens, refer to the online documentation (reference **TAB-1**).

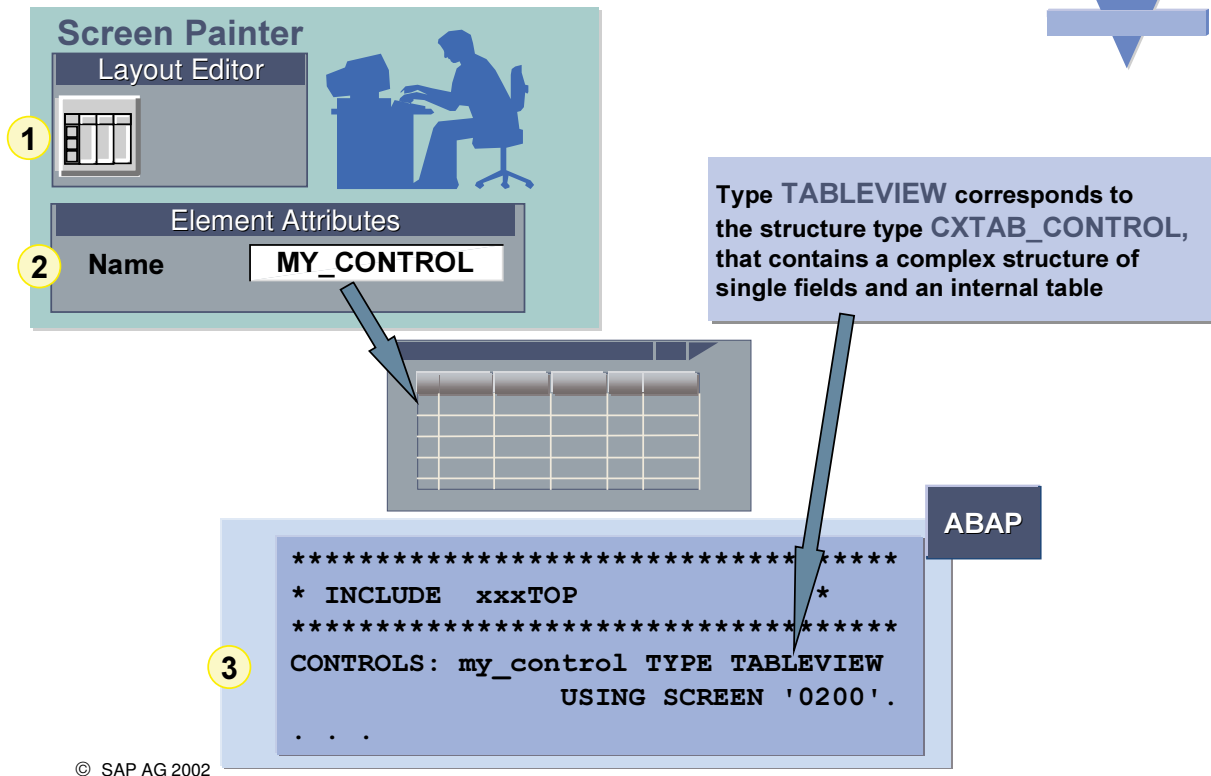


© SAP AG 1999

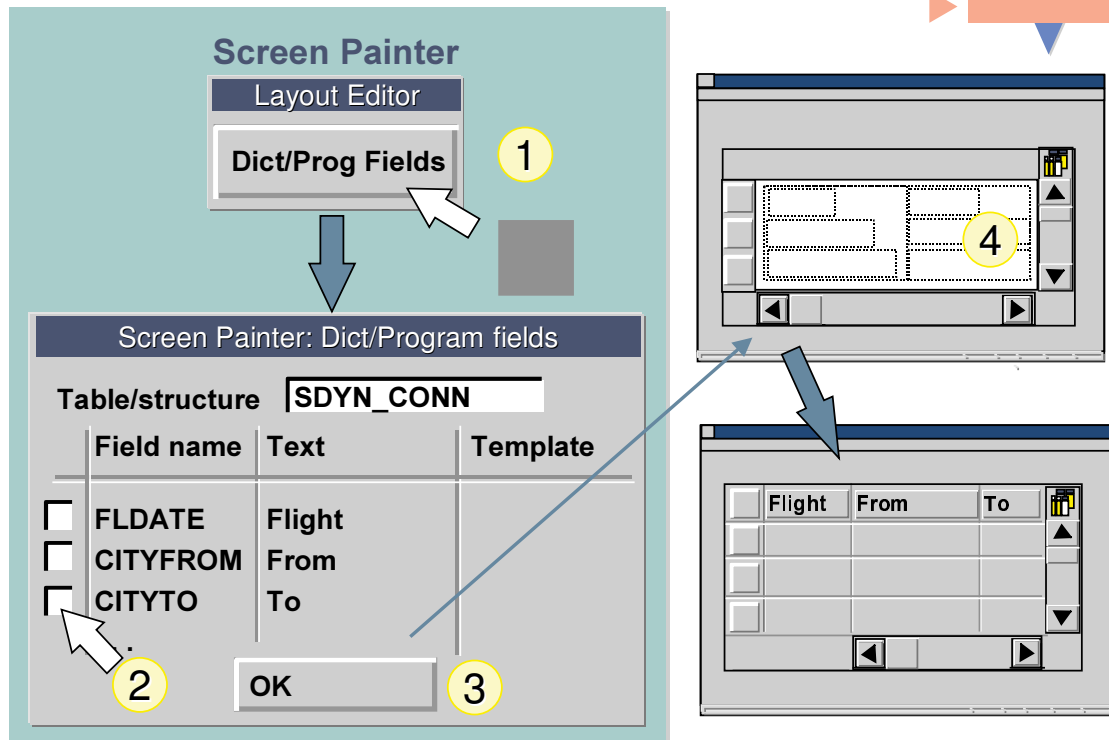
- When you create a table control, you must create:
  - A table control area
  - Table control fields

## Creating a Table Control: Table Control Area

SAP



- To create a table control area, choose the table control object from the object list in the Screen Painter and place it in the screen work area. Fix the upper-left corner of the table control area and then drag the object to the required size.
- In the *Name* attribute, assign a name to your table control. In the ABAP program, declare a structure with the same name, containing the dynamically changeable attributes of the table control.
- The **CONTROLS** statement declares a complex data object with the type **TABLEVIEW** (corresponding to the type **CXTAB\_CONTROL**, declared in type group **CXTAB** in the ABAP Dictionary). At run time, the data object (*my\_control*) contains the static attributes of the table control.
- You maintain the initial values (static attributes) in the Screen Painter. The **USING SCREEN** addition in the **CONTROLS** statement determines the screen whose initial values are to be used for the table control.
- You can reset a table control to its initial attributes at any time using the statement **REFRESH CONTROL <ctrl> FROM SCREEN <scr>** whereby <scr> does not have to be the same as the initial screen of the table control.



© SAP AG 2002

- You create fields in a table control using the *Dict./Program fields* function. This involves the following steps:
  - Enter the name of the structure whose fields you want to use in the table control and select *Enter*.
  - In the field list, choose the fields that you want to use and choose *OK*.
  - Click in the table control area. The system places all of the selected fields in the table control. If the fields have data element texts, the system uses these as column headings.
- Alternatively, you can position individual input/output fields in the table control area. Each field generates a single column.

## Creating Table Controls: Selection Column

SAP



**Screen Painter**

Table Control Attributes

☒ Sel. Col.

**Dictionary Structure: Display Fields**

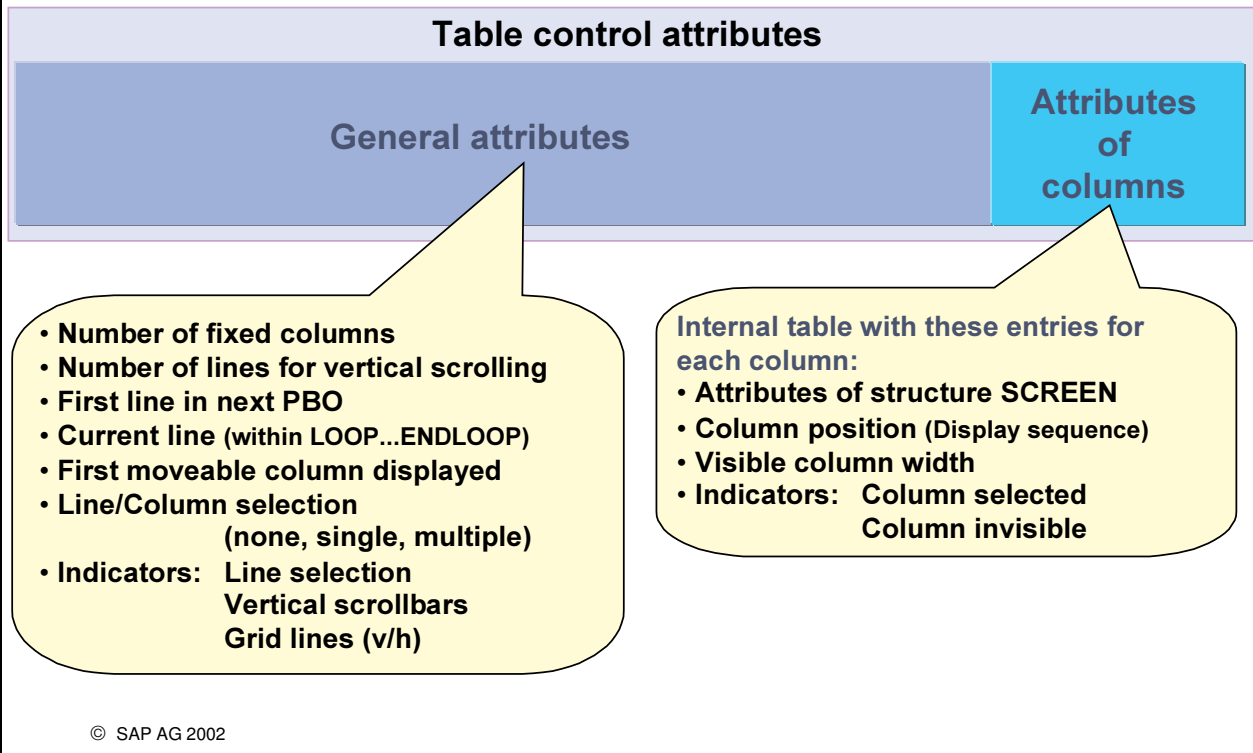
Structure

| Field name | Type | Length | Short desc. |
|------------|------|--------|-------------|
| MARK       | CHAR | 1      | Sel. column |

The diagram illustrates the configuration of a selection column in SAP. It shows the 'Screen Painter' interface with 'Table Control Attributes' where 'Sel. Col.' is checked and 'SDYN\_CONN-MARK' is entered. Below, the 'Dictionary Structure: Display Fields' shows the structure 'SDYN\_CONN' with a field 'MARK' of type 'CHAR' and length '1', described as 'Sel. column'. An arrow points from the 'Sel. Col.' attribute to the selection column in a table control example on the right, which has columns 'Flight', 'From', 'To', and a selection column with checkboxes.

© SAP AG 2002

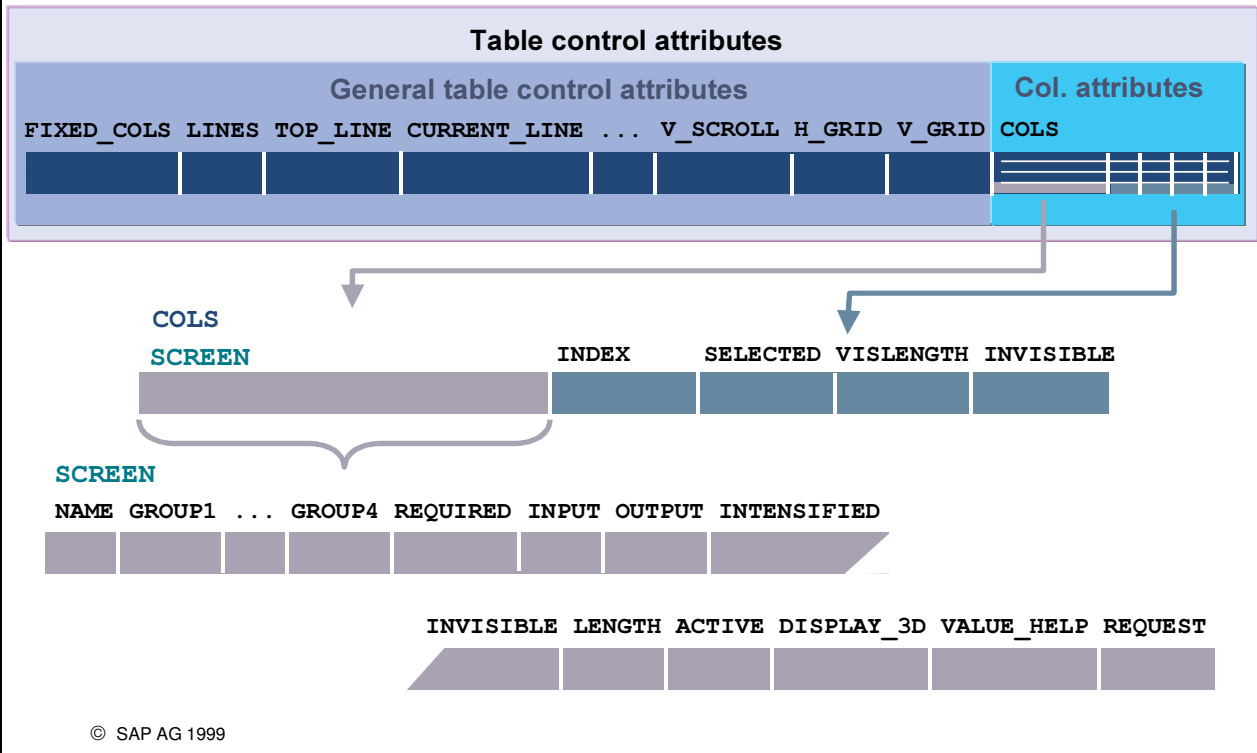
- When you create a table control, the system automatically proposes one with a selection column.
- The selection column behaves like a checkbox. It must, be a field with length one and data type CHAR. You must enter the field name in the attributes of the table control.
- The selection column is a field of the structure used for transport between the screen and the ABAP program.



- The table control attributes, saved at run time in the structure that you declared in the CONTROLS statement, can be divided into **general attributes** and **column attributes**.
- The **general attributes** contain information about the properties of the entire table control, such as the *number of fixed columns*.
- The column attributes are saved in an internal table (one entry for each column). Each column consists of the attributes from the SCREEN structure, along with the column position, selection indicator, visibility indicator, and visible width of the column.
- For further details about the names of the attributes and their precise meanings, see the ABAP keyword documentation for the CONTROLS statement (choose *Table control*, then *CXTAB\_CONTROL*), or the online documentation **TAB-1**.

## Table Control Attributes (Structure)

SAP



- You can change a table control dynamically by modifying the contents of the fields in the table control structure declared in your program.
- The fields of the table control structure also provide information about user interaction with the table control. For example, you can use the selected field to determine whether the user has selected a particular column.

Table controls: overview

Creating table control



Processing table control

Further techniques

© SAP AG 2002

## Processing a Table Control (Principle)

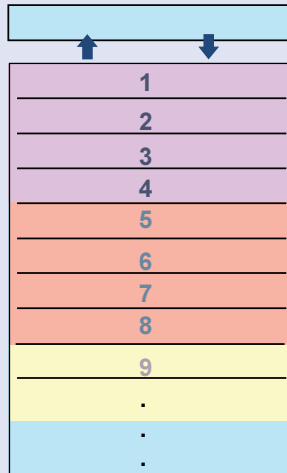
SAP

Database table



ABAP program

Internal table  
(buffer)



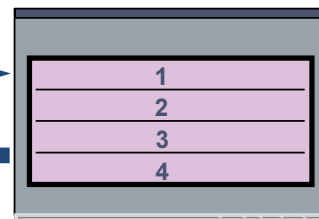
PBO



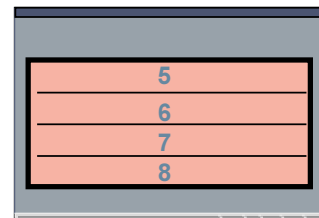
PAI



Screen



Next page



© SAP AG 2002

- For performance reasons, read the data for the table control once from the database and store it in an internal table.
- The system fills the table control lines from this internal table.

## ABAP

```
MODULE FILL_ITAB OUTPUT.
  IF wa_spfli-carrid NE key_scarr-carrid.

    MOVE-CORRESPONDING wa_spfli TO key_scarr.
    SELECT ... INTO TABLE itab_spfli
      WHERE ...

    DESCRIBE TABLE itab_spfli
      LINES my_control-lines.
  ENDIF.
ENDMODULE.
```

## Screen (with table control)

### Screen Painter

#### PBO

```
MODULE fill_itab.
  LOOP ...
```

Read line by line into  
the internal table

```
ENDLOOP.
```

#### PAI

```
LOOP ...
```

Update the internal table  
line by line

```
ENDLOOP.
```

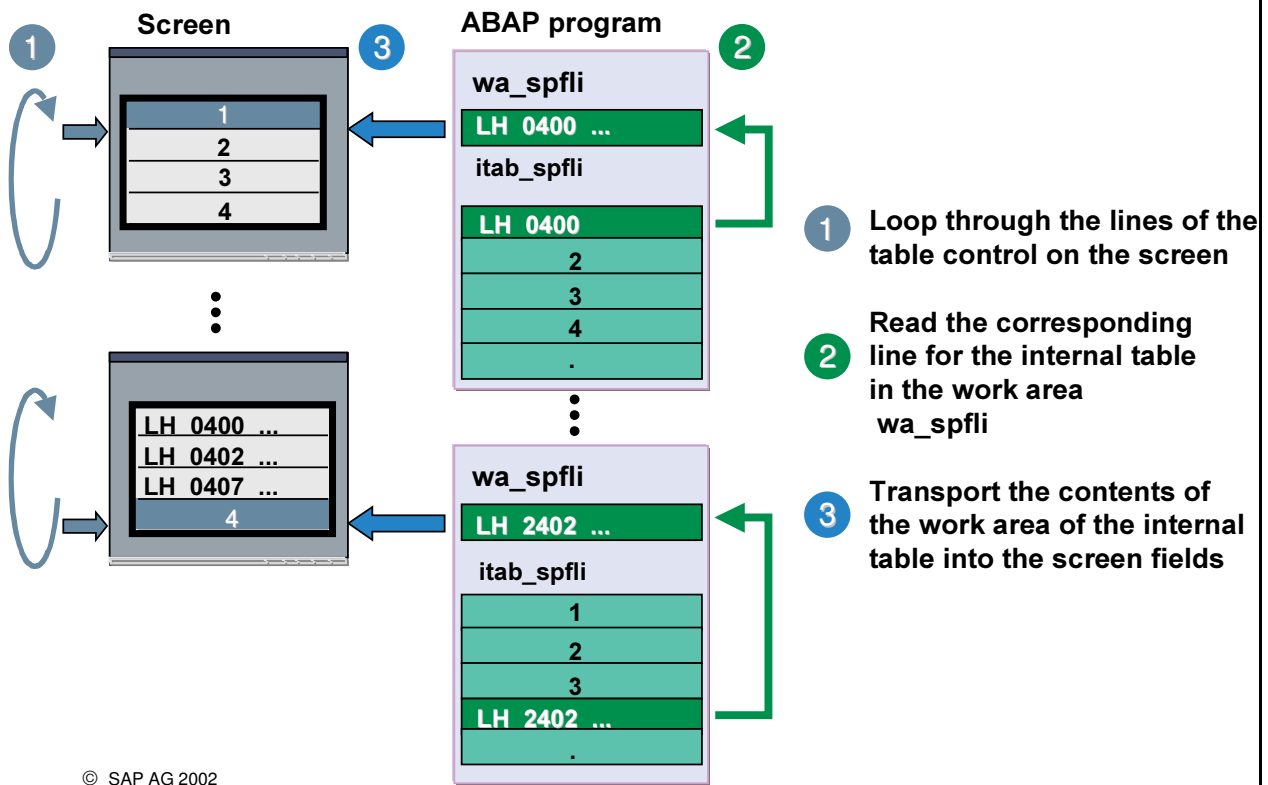
**Scroll page by page**  
**Change database table**

© SAP AG 2002

- Before you can display data from an internal table in a table control, you must first fill the table. Make sure that you do not fill the internal table in every PBO event, but only when the key fields change (in the above example, airline and flight number).
- In order to process the table control, the system needs to know how far the user can scroll vertically (the size of the internal table). You should, use the DESCRIBE TABLE statement to find out the number of entries in the internal table and save this number in the lines field of the table control.
- There is only one work area for processing lines in the table control. For this reason, you need a LOOP ... ENDLOOP structure in both the PBO and PAI events for each table control.
- In the PBO processing block, you must fill one line of the table control with the corresponding line from the internal table in each loop pass.
- Similarly, in the PAI processing block, you must pass the changes made in the table control back to the correct line of the internal table.
- When you process functions, you must distinguish between those that should apply only to individual lines of a table control and those that should apply to the entire screen.

## Filling a Table Control

SAP



- There are three steps involved in displaying buffered data from the internal table in the table control:
  - The system loops through the lines of the table control on the screen. The lines of the screen table are processed one by one.
    - For each line, the system places the current line of the internal table in the work area of the internal table.
    - For each line, the system copies the data from the work area of the internal table to the relevant line of the table control.

## Code: Filling a Table Control

SAP

PROCESS BEFORE OUTPUT.

```
LOOP AT itab_spfli INTO wa_spfli
    WITH CONTROL my_control.
    MODULE move_to_tc.
ENDLOOP.
```

Screen  
Painter

1 + 2

Automatically carried out by loop in the flow logic

MODULE move\_to\_tc OUTPUT.

```
MOVE-CORRESPONDING wa_spfli
    TO sdyn_conn.
ENDMODULE.
```

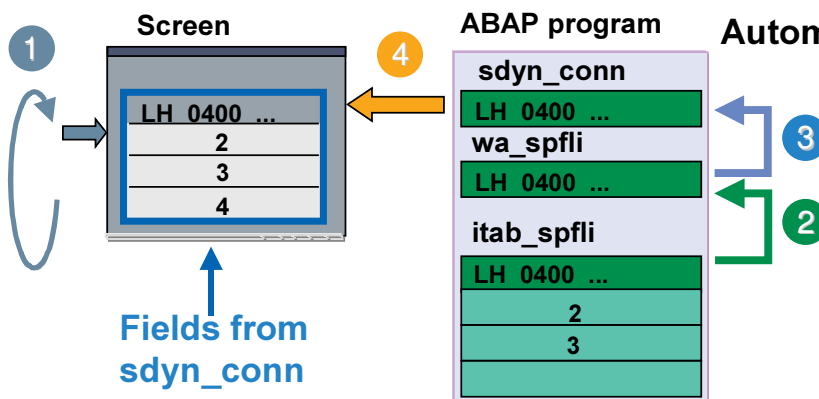
ABAP

3

Programmed in ABAP

4

Automatic transport



© SAP AG 1999

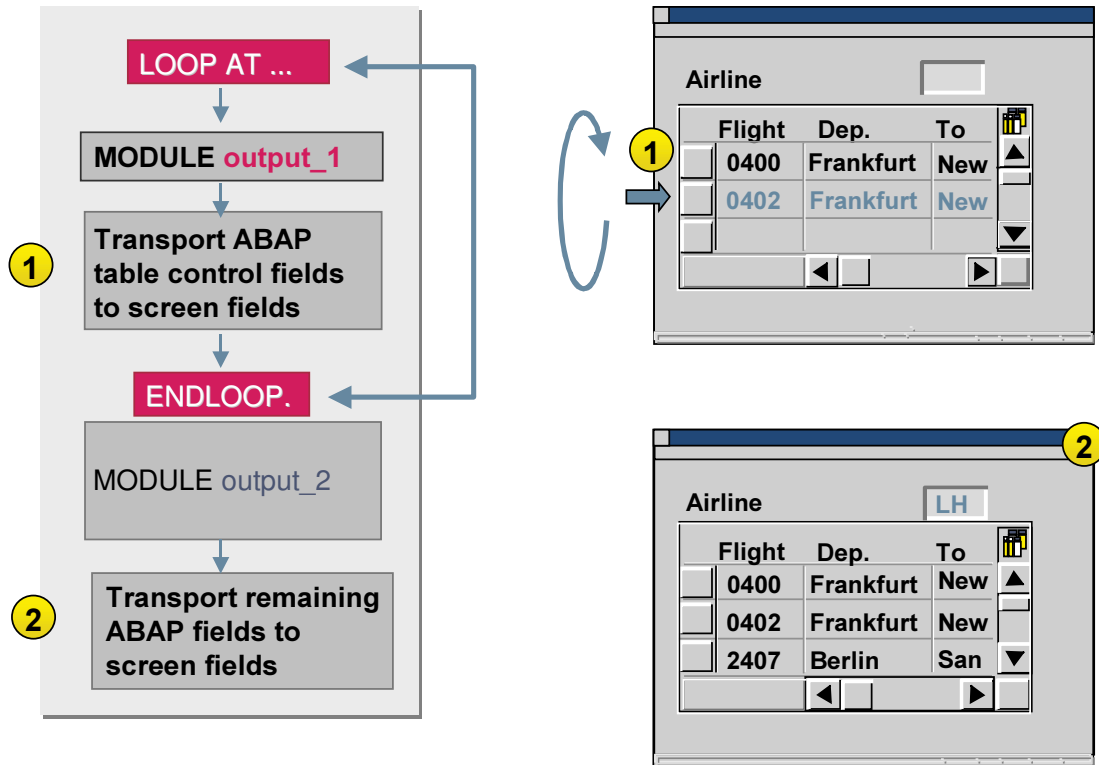
- In the flow logic, the loop statement

LOOP AT <itab> INTO <wa\_itab> WITH CONTROL <tc\_name>

starts a loop through the screen table, and reads the line of the internal table corresponding to the current line of the screen table, placing it in <wa\_itab>.

<itab> is the name of the internal table containing the data, <wa\_itab> is the name of the work area for the internal table, and <tc\_name> is the name of the table control on the screen.

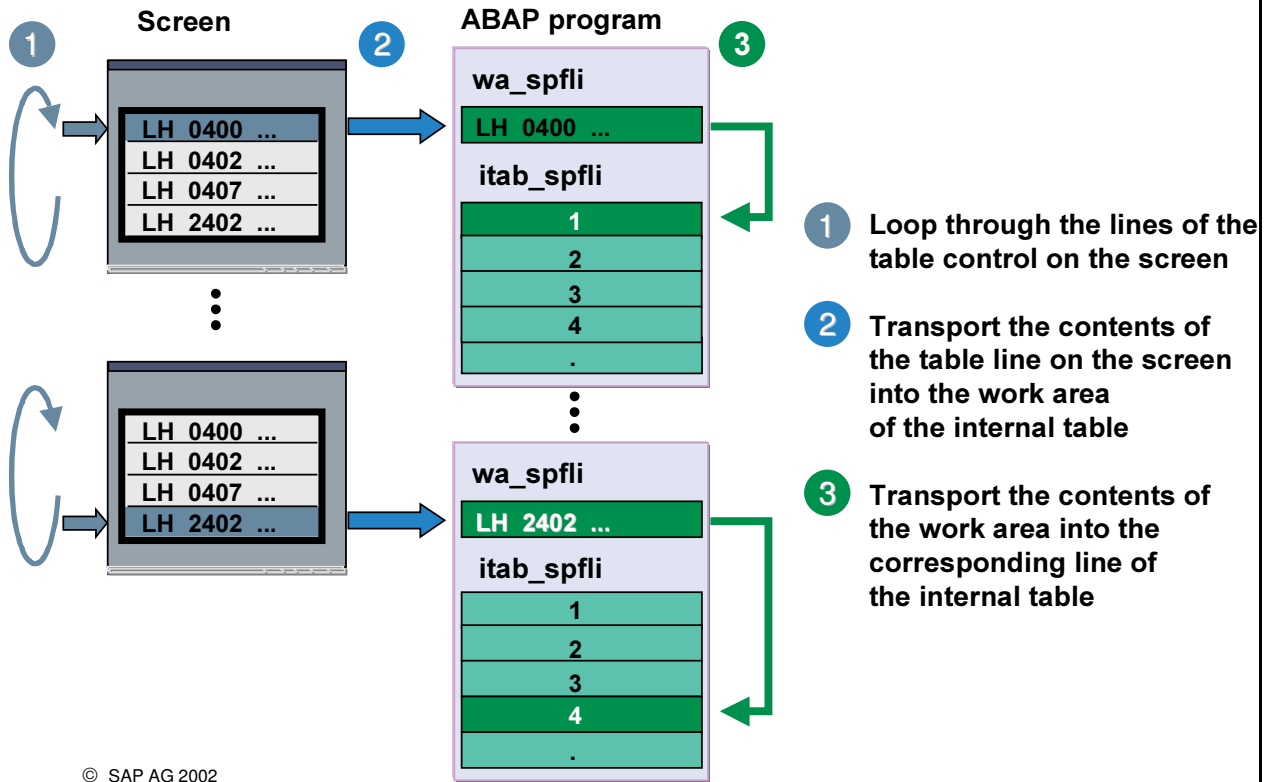
- If the fields in your table control have the same structure and name as those in the work area <wa\_itab>, the system can transport data between the ABAP program and the screen automatically (step 3).
- If you are not using the same structure for the table control fields and the work area of the internal table, you must call a module between LOOP and ENDLOOP that moves the data from the work area <wa\_itab> into the screen fields (MOVE-CORRESPONDING <wa\_itab> TO ...).
- The system calculates the value of <ctrl>-TOP\_LINE when you scroll, but not when you scroll a page at a time outside the table control.



- When you use table controls on a screen, the automatic field transport sequence changes.
- In the PBO processing block, data is transferred from the ABAP program to the screen **after each** loop pass in the flow logic. The rest of the screen fields are filled, as normal, at the end of the PBO.

## Changing the Contents of a Table Control

SAP



- To transfer changed values from the table control back to the internal table, the following three steps must be carried out:
  - The system loops through the lines of the table control.
  - For each line, the system copies the data from the current line of the table control to the fields with identical names in the work area of the internal table.
  - Transport of the contents of the work area into the corresponding line of the internal table must be programmed.

## Code: Changing the Contents of Table Control

SAP

```
PROCESS AFTER INPUT.
  LOOP AT itab_spfli.
    FIELD sdyn_conn-mark
      MODULE modify_itab ON REQUEST.
  ENDLOOP.
```

Screen  
Painter

1 + 2

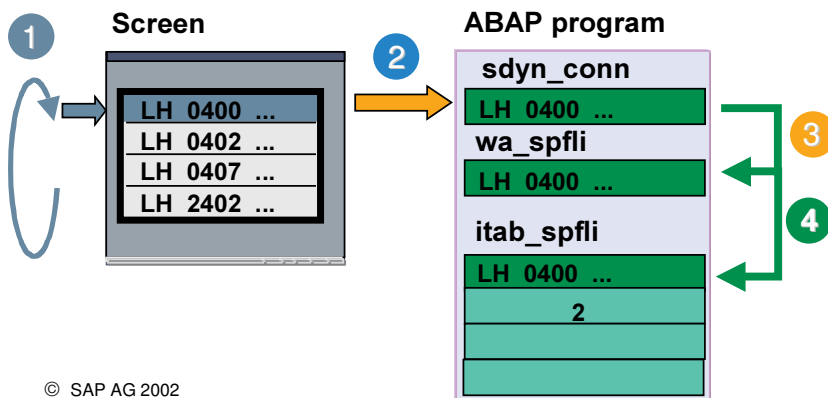
Automatically carried  
out by loop in  
the flow logic

```
MODULE modify_itab INPUT.
  MOVE-CORRESPONDING sdyn_conn
    TO wa_spfli.
  MODIFY itab_spfli FROM wa_spfli
    INDEX my_control-current_line.
ENDMODULE.
```

ABAP

3 + 4

Programmed in ABAP

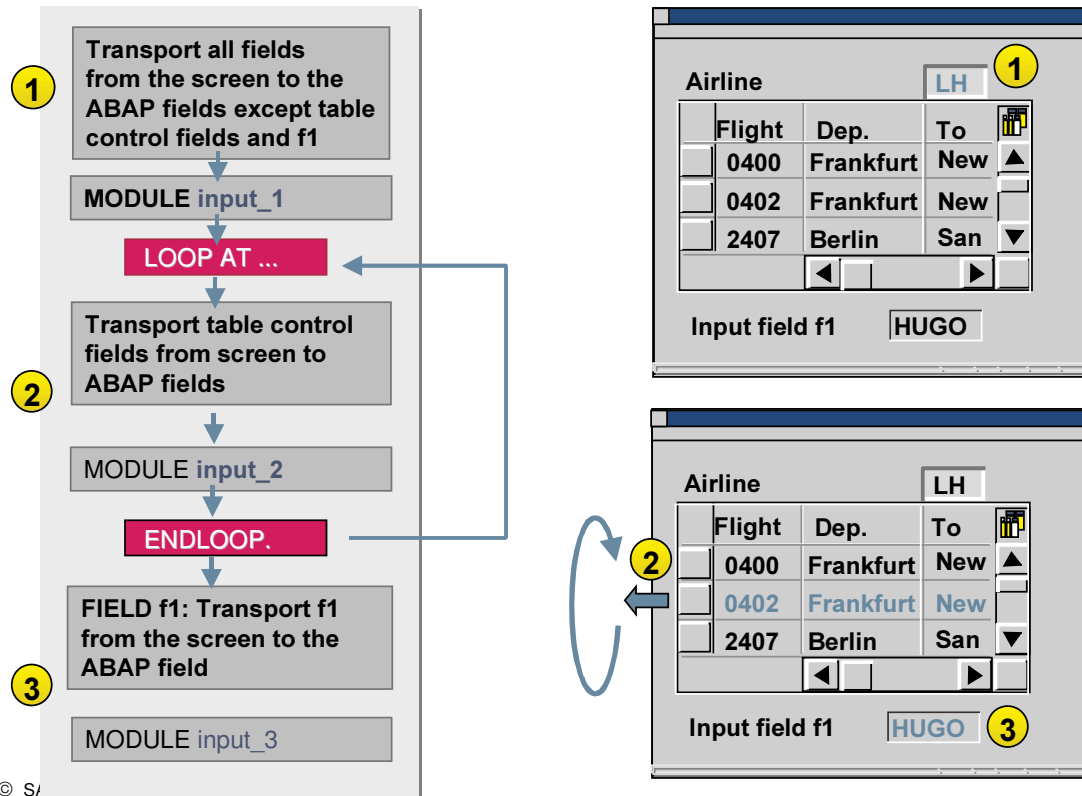


© SAP AG 2002

- The LOOP AT <itab>. ... ENDLOOP block processes a loop through the lines of the table on the screen.
- If the fields on your screen have the same names as the fields in the internal table, you must return the data from the work area of the internal table to the body of the table itself. You do this using the field <control>-current\_line.
- If the fields on your screen do not have the same names as the fields in the internal table, you must first copy the data into the work area of the internal table. You can then copy the data back to the internal table itself. You can also use the <control>-current\_line field to do this.

## Table Controls: Field Transport in the PAI

SAP



- In the PAI processing block, all screen fields that do not belong to a table control and that are not listed in a FIELD statement are transported back to the work fields in the ABAP program first.
- The contents of the table control are transported line by line to the corresponding work area in the ABAP program in the appropriate loop.
- As usual, the fields that occur in FIELD statements are transported directly before that statement.

## Field Transports for Formatted Fields

SAP

Dictionary: Display Structure: Currency/Quantity Fields

**SDYN\_CONN** Structure

| Field name | DType | Reference table | Reference field |
|------------|-------|-----------------|-----------------|
| PRICE      | CURR  | SDYN_CONN       | CURRENCY        |

1

Automatic data transport

2

Delayed data transport for the quantity

**SDYN\_CONN**

CURRENCY JPY PRICE 1,000,000

1

2

CURRENCY JPY PRICE 1,000,000

TABLES sdyn\_conn.

Screen Painter

PROCESS AFTER INPUT.

1

LOOP...

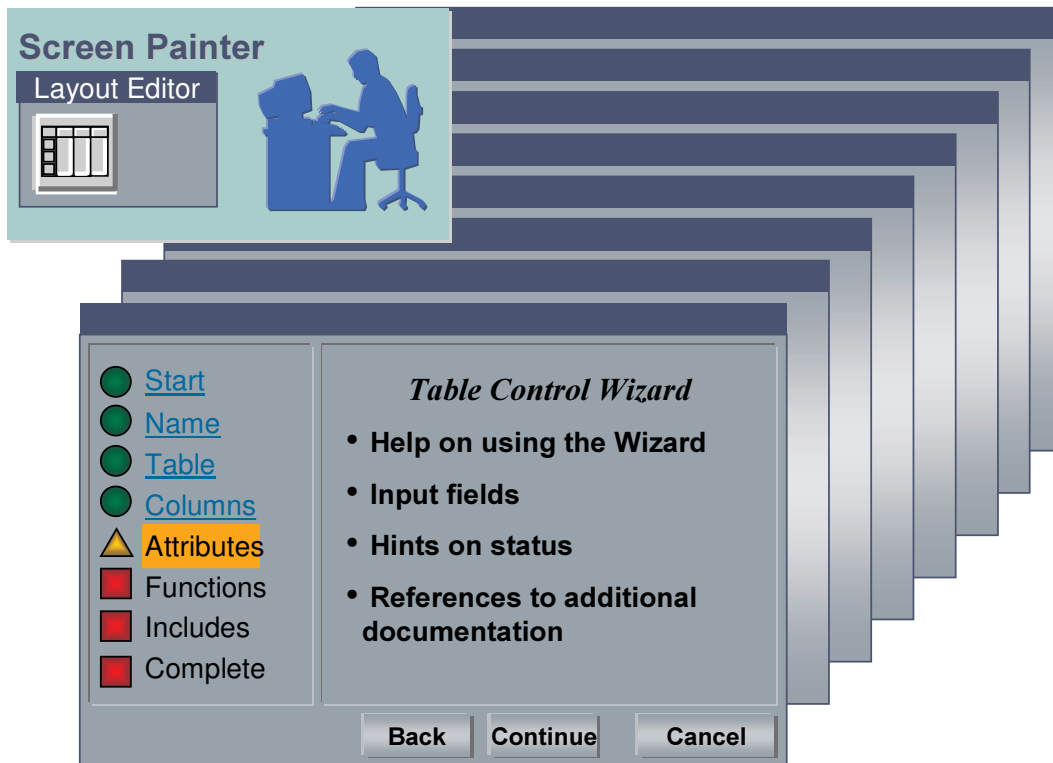
2

FIELD sdyn\_conn-price.

ENDLOOP.

© SAP AG 2002

- A system program for currency amounts executes additional formatting for each currency during the automatic data transport from the screen to ABAP. The program refers to the contents of the corresponding reference field. The program also refers to the contents of the **ABAP field**. Errors occur when the reference field does not contain the respective currency.
- The sequence of the data transport is dependent on the position of the fields on the screen. It is not necessary to consider the exclusively technical aspects of the data transport, since the screen layout is extremely user-friendly.
- You must ensure that in the flow logic programming, the fields are transported in the correct order. that (the reference field's followed by the amounts). To do this, delay the data transport of the quantity fields using a corresponding FIELD statement.



© SAP AG 2002

- You can use the *Table Control Wizard* to help you create table controls and insert them on screens in a program. The Wizard guides you through the process. You can return to previous settings at any time. Program objects are created on the final screen only on completion of the process. The Wizard creates not only the table control but also the corresponding statements in the screen flow logic, together with the relevant modules and subroutines and necessary data definitions. If necessary, programs or includes are generated and concatenated using INCLUDE statements.
- You can also implement the scrolling function for the table control.
- All objects are placed in the inactive object list.

Table controls: overview

Creating table control

Processing table control



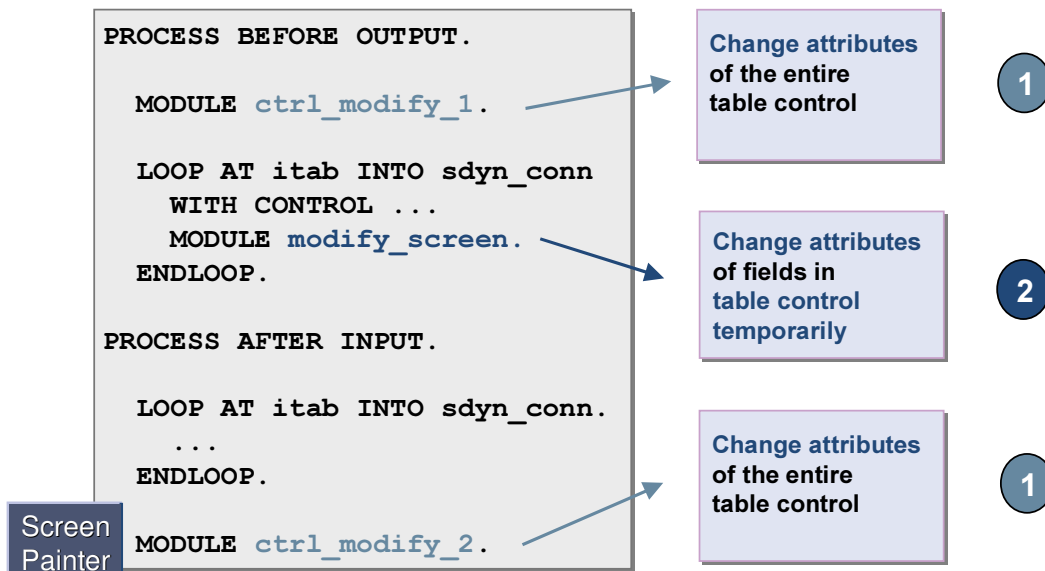
Further techniques

© SAP AG 2002

## Changing Table Control

SAP

- 1 **Permanent changes:** You can carry out changes to the table control structure at any point in the program flow.
- 2 **Temporary changes:** You must carry out changes between `LOOP` and `ENDLOOP` in the flow logic of the subscreen container.



© SAP AG 2002

- You can modify the attributes of a table control by overwriting the field contents of the structure created in the `CONTROLS` statement.
- To change the attributes of individual cells **temporarily**, change the `SCREEN` table in a PBO module that you process between `LOOP` and `ENDLOOP` in the flow logic (`LOOP AT SCREEN`, `MODIFY SCREEN`).
- In the `LOOP`, the runtime system initializes the attributes set statically for the table control in the Screen Painter. You can change these attributes only in a module called from a `LOOP` through the table control.

FIXED\_COLS ...

## Screen Painter

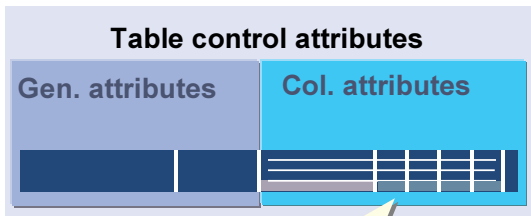
```
MODULE change_table_control_1 ...
    my_control-fixed_cols
        = dyn_col_num.
    ...
ENDMODULE.
```



- 7-30

## Changing the Attributes of Table Control (2)

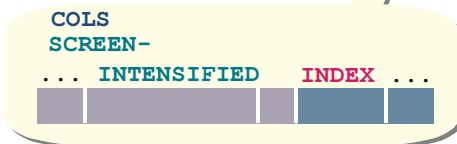
SAP



PBO or PAI

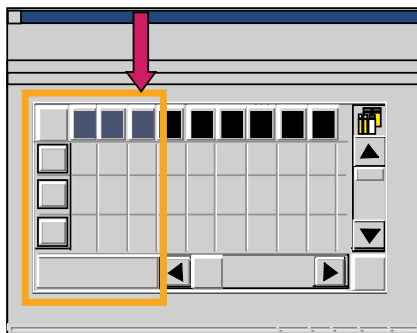
MODULE change\_table\_control\_2.

Screen  
Painter



ABAP

```
DATA wa LIKE LINE OF my_control-cols.
...
MODULE change_table_control_2 ...
  LOOP AT my_control-cols INTO wa.
    IF wa-index BETWEEN 1 AND 3.
      wa-screen-intensified = 1.
    ELSE.
      wa-screen-intensified = 0.
    ENDIF.
    MODIFY my_control-cols FROM wa.
  ENDLOOP.
ENDMODULE.
```



© SAP AG 2002

- If you want to change the attributes of the columns in a table control, you must change the respective entries in the <control>-cols table. Note that the table has no header lines. This means that you have to create an explicit work area.
- The fields of the table control structure also provide information about user interaction with the table control. For example, you can use the selected field to determine whether the user has selected a particular column.



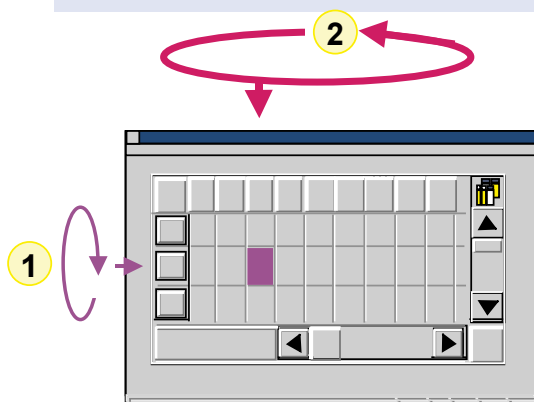
- 
- 7-32

## Table Controls: Changing Field Attributes Temporarily

SAP

Screen table for line: **2**

| Name | Input | Output | Intensified | ... |
|------|-------|--------|-------------|-----|
| Col1 |       |        |             |     |
| Col2 |       |        |             |     |
| Col3 |       |        |             |     |
| ...  |       |        |             |     |



© SAP AG 2002

Screen Painter

**1** `LOOP AT itab INTO sdyn_conn.`  
`MODULE modify_screen.`  
`ENDLOOP.`

ABAP

`MODULE modify_screen OUTPUT.`  
`CHECK wa_spfli-mark = 'X'.`  
**2** `LOOP AT SCREEN.`  
`IF screen-group1 = 'SEL'.`  
`screen-intensified = 1.`  
`ENDIF.`  
`MODIFY SCREEN.`  
`ENDLOOP.`  
`ENDMODULE.`

- It is possible to change the attributes of table control fields temporarily. These changes are effective only while the current screen is being processed.
- To change attributes temporarily, you call a module from within the table control loop in the flow logic, in which you change the attributes of the current line.
- To change the attributes of the fields of a line in the table control, use a `LOOP AT SCREEN. ... ENDLOOP.` block to loop through the fields of the current line. Within this loop, you can change the attributes of the fields of the current line of the table control.

## Sorting Table Controls: Example

SAP

ABAP

```
CONTROLS my_control TYPE TABLEVIEW
  USING SCREEN 200.
DATA wa LIKE LINE OF my_control-cols.
...
MODULE user_command_0200 INPUT.
  CASE ok_code.
    WHEN 'SRTU'.
      READ TABLE my_control-cols
        WITH KEY selected = 'X' INTO wa.
      IF sy-subrc = 0.
        SORT itab BY (wa-screen-name+10).
      ELSE.
        MESSAGE i055(bc410).
      ENDIF.
    ...
  ENDCASE.
ENDMODULE.
```

Find out  
sort field

Sort  
ascending

Column  
chosen?

© SAP AG 2002

- You can easily sort the table control display by a particular column using the table control attributes, <wa\_cols>-selected and <wa\_cols>-screen-name.
- You can define the sort criteria using string processing. Ensure that the <wa\_cols>-screen-name field contains the name of the screen field and not the column name of the internal table.

## Syntax

### GET CURSOR

FIELD f  
VALUE v  
LINE l  
OFFSET o.

### SET CURSOR

FIELD f  
LINE l  
OFFSET o.

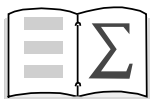
**Determining the cursor position:**  
Which line of the internal table corresponds to a line in the table control?

ABAP

```
DATA: selline TYPE sy-stepl,  
      tabix TYPE sy-tabix.  
...  
GET CURSOR LINE selline.  
tabix = my_control-TOP_LINE  
        + selline - 1.  
READ TABLE spfli_itab  
INDEX tabix.
```

© SAP AG 2002

- The LINE parameter in the GET or SET statement refers to the sy-stepl system field, the special loop index in the flow logic.
- You calculate the internal table line that corresponds to the selected table control line as follows:  
line = <ctrl>-top\_line + cursor position - 1.
- The GET CURSOR statement sets the return code as follows: sy-subrc = 0: cursor was on a field, sy-subrc = 4: cursor was not on a field.
- If you use a table control on your screen, you can place the cursor on a particular element within the table control. To do this, use the LINE parameter and enter the line on which the cursor should be positioned:  
SET CURSOR FIELD <field\_name> LINE <line>.
- You can also use the OFFSET and LINE parameters together.



**You are now able to:**

- **Display the contents of an internal table in a table control and implement special table control functions.**

# Exercises



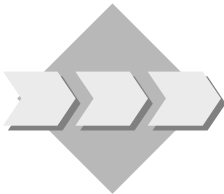
## Unit: Table Controls

### Topic: Creating a table control



At the conclusion of these exercises, you will be able to:

- Use a table control and its processing logic in your program .



On the third page of your tabstrip control, create a table control in which you can maintain bookings for your flight.

7-1 On the third page of your tabstrip control, create a table control containing the booking information for a flight.

7-1-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410ASUBS\_TABSTRIP**). You can use the model solution **SAPMBC410ATABS\_TABLE\_CONTROL1** for orientation.

#### 7-1-2 Table Control Area:

On subscreen screen 130, create a table control with the following attributes:

|               |                                         |                                                                                                                                                                                                                                                                                                                                |
|---------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Table control | <b>Name:</b><br><b>MY_TABLE_CONTROL</b> | <b>Attributes:</b><br><b>Vertical and horizontal Resizing:</b> ON<br><b>Vertical and horizontal Separators:</b> ON<br><b>Column selection:</b> SINGLE<br><b>Line selection:</b> MULTIPLE<br><b>Column title:</b> ON<br><b>Configurable:</b> ON<br><b>Selection column:</b><br>SDYN_BOOK-MARK<br><b>No. of fixed columns:</b> 2 |
|---------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- 7-1-3 **Note:** You cannot set the *number of fixed columns* attribute for the table control until you have created all of its columns. In the TOP include, create a complex data object for the attributes of your table control: **CONTROLS :**  
**MY\_TABLE\_CON. . .**

**Create table control columns:** In your table control, create the following structure fields as columns:

|                                                        |                                                                                                                                                                                                                               |                                                              |
|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| Input/output field<br>Text field<br>(in table control) | <b>SDYN_BOOK</b><br>- BOOKID<br>- CUSTOMID<br>- CUSTTYPE<br>- SMOKER<br>- LUGGWEIGHT<br>- WUNIT<br>- INVOICE<br>- CLASS<br>- FORCURAM<br>- FORCURKEY<br>- LOCCURAM<br>- LOCCURKEY<br>- ORDER_DATE<br>- COUNTER<br>- AGENCYNUM | <b>Attributes:</b><br><b>Input:</b> OFF<br><b>Output:</b> ON |
|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|

Now enter the *number of fixed columns* in the table control attributes.

- 7-1-4 **Declaration for an internal table:** In the TOP include of your program, create an internal table **IT\_SDYN\_BOOK**. This will buffer the bookings that you are going to display in the table control. Create the internal table with type **STANDARD** and no header line. Declare a suitable work area for the internal table. Use the line type **SDYN\_BOOK** to declare both the internal table and the work area.
- 7-1-5 **Data retrieval:** In the flow logic of screen 130, create a PBO module in which you read all of the bookings for the flight selected that have not been canceled. Read SBOOK table **only** if the user enters different flight data on screen 100. The **LINES** field in your table control requires the number of lines in your internal table. To find out the value, use the **DESCRIBE TABLE . . .** statement.

- 7-1-6 **Implement a table control:** Program a loop for the table control in both the PBO and PAI events of screen 130 (read the entries in the internal table in the flow logic): **LOOP . . . . ENDLOOP**. In the PBO loop, call a module to copy data from the work area of the internal table into the screen fields. In the PAI loop, call a module to copy the data from the screen into the internal table. The module should be called only for lines that have been selected on the screen (**FIELD . . . MODULE . . . ON REQUEST.**).

**Note:** You can test whether your changes (selected lines) are properly adopted by selecting a line and then scrolling down and back up within the table control. If the selected entry is still selected after you have scrolled, your changes have been copied correctly from the internal table to the table control.

- 7-2 Implement functions for canceling a booking, selecting all unmarked table control lines, and deselecting all marked table control lines.

- 7-2-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410ATABS\_TABLE\_CONTROL1**). You can use the model solution **SAPMBC410ATABS\_TABLE\_CONTROL2** for orientation.

- 7-2-2 **Create pushbuttons:** Create the following pushbuttons on screen 130:

|            |                              |                                                                                                                                                                |
|------------|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pushbutton | <b>Name:</b><br>SELECT_ALL   | <b>Function code:</b> SELE<br><b>Function type:</b> <blank><br><b>Icon:</b> ICON_SELECT_ALL                                                                    |
| Pushbutton | <b>Name:</b><br>DESELECT_ALL | <b>Function code:</b> DSELE<br><b>Function type:</b> <blank><br><b>Icon:</b><br>ICON_DESELECT_ALL                                                              |
| Pushbutton | <b>Name:</b><br>P_DELETE     | <b>Function code:</b> DELE<br><b>Function type:</b> <blank><br><b>Icon:</b> ICON_DELETE_ROW<br><b>Input:</b> OFF<br><b>Output:</b> OFF<br><b>Invisible:</b> ON |

- 7-2-3 **Implement the function code:** Extend the **OK\_CODE** processing for screen 130 to implement a cancellation function. The pushbutton for canceling a booking should appear only if the user is in *Maintain bookings* mode. To do this, create a PBO module for screen 130 in which you change the attributes of pushbutton **P\_DELETE** at runtime (**LOOP AT SCREEN . . .**). Ensure that the function code for the *Maintain bookings* checkbox is set to trigger PAI. Encapsulate the cancellations of bookings in a subroutine. Create a new internal table **IT\_SBOOK\_UPD** of type **STANDARD** and a corresponding work area locally in the subroutine. Use the line type **SBOOK** to declare both the internal table and the work area. Copy the selected data records from internal table **IT\_SDYN\_BOOK** to internal table **IT\_SBOOK\_UPD**. Set the flag to X. Pass both internal tables to the function module **BC\_GLOBAL\_UPDATE\_BOOK**. The function module makes the database changes for table **SBOOK** and the resulting changes in table **SFLIGHT**. Display the screen again once the flight has been canceled. Ensure that the bookings have changed, and that the internal table **IT\_SDYN\_BOOK** is read from the database again before it is displayed.
- 7-2-4 Extend the command field processing of screen 130 to include the *Select all* and *Deselect all* functions.
- 7-3 In the table control, implement the function *Sort*.
- 7-3-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410ATABS\_TABLE\_CONTROL2**). You can use the model solution **SAPMBC410ATABS\_TABLE\_CONTROL3** for orientation.
- 7-3-2 **Sort table control:** To implement sorting data in a table control according to a selected column you must carry out the following:

**Create pushbuttons:** On screen 130, create the following pushbuttons:

|            |                        |                                                                                            |
|------------|------------------------|--------------------------------------------------------------------------------------------|
| Pushbutton | <b>Name:</b><br>P_SRTU | <b>Function code:</b> SRTU<br><b>Function type:</b> <blank><br><b>Icon:</b> ICON_SORT_UP   |
| Pushbutton | <b>Name:</b><br>P_SRTD | <b>Function code:</b> SRTD<br><b>Function type:</b> <blank><br><b>Icon:</b> ICON_SORT_DOWN |

- 7-3-3 **Implement functions:** Extend the command field processing for screen 130 to implement the two sort functions. Use the table control structure **MY\_TABLE\_CONTROL** to find out the column in the table control selected by the user. You will need to write a loop for the internal table **MY\_TABLE\_CONTROL-COLS**. This means that you will also need a work area for **MY\_TABLE\_CONTROL-COLS**. Create this in the TOP include of your program (suggested name: **WA\_COLS**) . Use the fields **MY\_TABLE\_CONTROL-COLS-SELECTED** and **WA\_COLS-SCREEN-NAME** to find out the name of the column selected by the user. Since the field **WA\_COLS-SCREEN-NAME** contains the name of the screen field (**SDYN\_BOOK-<fname>**), you will need to find out the field name using an offset specification. Sort the internal table containing the data for the table control by the selected field in the chosen direction.



## Unit: Table Controls

### Topic: Creating a table control

#### 1-1 **Model solution** SAPMBC410ATABS\_TABLE\_CONTROL1

Extend the sections in bold and create corresponding new modules (using forward navigation).

##### Top include

```
PROGRAM  sapmbc410tabs_table_control1 MESSAGE-ID bc410 .

CONTROLS my_table_control TYPE TABLEVIEW USING SCREEN 130.
TABLES  sdyn_book.

DATA: wa_sdyn_book TYPE sdyn_book,
      it_sdyn_book LIKE TABLE OF wa_sdyn_book.

CONTROLS my_tabstrip TYPE TABSTRIP.

DATA  dynnr TYPE sy-dynnr.                                     "#EC NEEDED

TABLES saplane.

TABLES sdyn_conn.

DATA: BEGIN OF mode,
      view VALUE 'X',                                     "selected
      maintain_flights,
      maintain_bookings,
      END OF mode.

DATA ok_code TYPE sy-ucomm.
```

DATA wa\_sflight TYPE sflight.

DATA: wa\_sbook TYPE sbook,  
      it\_sbook LIKE TABLE OF wa\_sbook.

**CONSTANTS not\_cancelled VALUE space.**

## Subroutine include

```
FORM update_sflight.  
  UPDATE sflight FROM wa_sflight.  
  IF sy-subrc NE 0.  
    MESSAGE a008.  
  ENDIF.  
  MESSAGE s009.  
ENDFORM.                                " update_sflight
```

## Flow logic screen 100

```
PROCESS BEFORE OUTPUT.  
  MODULE status_0100.  
  MODULE modify_screen.  
  MODULE fill_dynnr.  
  CALL SUBSCREEN sub INCLUDING sy-cprog dynnr.  
  MODULE clear_ok_code.  
*  
PROCESS AFTER INPUT.  
  MODULE exit AT EXIT-COMMAND.  
  CALL SUBSCREEN sub.  
  CHAIN.  
    FIELD: sdyn_conn-carrid,  
           sdyn_conn-connid,  
           sdyn_conn-fldate  MODULE read_sflight ON CHAIN-REQUEST.  
  ENDCHAIN.  
  CHAIN.  
    FIELD: sdyn_conn-planetype,  
           sdyn_conn-seatsmax MODULE check_planetype ON CHAIN-REQUEST.  
  ENDCHAIN.
```

```
MODULE trans_from_dynp.  
MODULE user_command_0100.
```

## Flow logic screen 110

No changes are necessary.

## Flow logic screen 120

No changes are necessary.

## Flow logic screen 130

### PROCESS BEFORE OUTPUT.

```
MODULE get_sbook.  
LOOP AT it_sdyn_book INTO wa_sdyn_book WITH CONTROL my_table_control.  
    MODULE trans_to_tc.  
ENDLOOP.
```

### PROCESS AFTER INPUT.

```
LOOP AT it_sdyn_book.  
    FIELD sydyn_book-mark  
    MODULE trans_from_tc ON REQUEST.  
ENDLOOP.
```

## PBO module include

```
MODULE status_0100 OUTPUT.  
    SET PF-STATUS 'STATUS_100'.  
    SET TITLEBAR 'TITLE_100'.  
ENDMODULE.                                " STATUS_0100  OUTPUT
```

```
MODULE clear_ok_code OUTPUT.  
    CLEAR ok_code.  
ENDMODULE.                                " clear_ok_code  OUTPUT
```

```

MODULE modify_screen OUTPUT.
  CHECK NOT mode-maintain_flights IS INITIAL.
  LOOP AT SCREEN.
    IF screen-name = 'SDYN_CONN-PLANETYPE'.
      screen-input = 1.
      screen-required = 1.
      MODIFY SCREEN.
    ENDIF.
  ENDLOOP.
ENDMODULE.                                " modify_screen  OUTPUT

MODULE get_spfli OUTPUT.
  ON CHANGE OF wa_sflight-carrid OR wa_sflight-connid.
    SELECT SINGLE * INTO CORRESPONDING FIELDS OF sdyn_conn FROM spfli
      WHERE carrid = wa_sflight-carrid
        AND connid = wa_sflight-connid.
  ENDON.
ENDMODULE.                                " GET_SPFLI   OUTPUT

MODULE get_saplane OUTPUT.
  ON CHANGE OF wa_sflight-planetype.
    SELECT SINGLE * FROM saplane WHERE planetype = wa_sflight-planetype.
  ENDON.
ENDMODULE.                                " GET_SAPLANE  OUTPUT

MODULE fill_dynnr OUTPUT.
  CASE my_tabstrip-activetab.
    WHEN 'FC1'.
      dynnr = 110.
    WHEN 'FC2'.
      dynnr = 120.
    WHEN 'FC3'.
      dynnr = 130.
    WHEN OTHERS.
      my_tabstrip-activetab = 'FC1'.
      dynnr = 110.
  ENDCASE.
ENDMODULE.                                " set_dynnr   OUTPUT

```

```

MODULE get_sbook OUTPUT.
  IF wa_sflight-carrid <> sdyn_book-carrid OR
    wa_sflight-connid <> sdyn_book-connid OR
    wa_sflight-fldate <> sdyn_book-fldate.
    SELECT * FROM sbook INTO CORRESPONDING FIELDS OF TABLE it_sdyn_book
      WHERE carrid = wa_sflight-carrid
        AND connid = wa_sflight-connid
        AND fldate = wa_sflight-fldate
        AND cancelled = not_cancelled.

    DESCRIBE TABLE it_sdyn_book LINES my_table_control-lines.
  ENDIF.
ENDMODULE.                                " get_sbook  OUTPUT

```

```

MODULE trans_to_tc OUTPUT.
  MOVE wa_sdyn_book TO sdyn_book.
ENDMODULE.                                " trans_to_dynp  OUTPUT

```

## PAI module include

```

MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'FC1' OR 'FC2' OR 'FC3'.
      my_tabstrip-activetab = ok_code.
    WHEN 'SAVE'.
      PERFORM update_sflight.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.                                " USER_COMMAND_0100  INPUT

MODULE trans_from_dynp INPUT.
  MOVE-CORRESPONDING sdyn_conn TO wa_sflight.
ENDMODULE.                                " trans_from_dynp  INPUT

```

```

MODULE read_sflight INPUT.
  SELECT SINGLE * INTO CORRESPONDING FIELDS OF sdyn_conn FROM sflight
    WHERE carrid = sdyn_conn-carrid
      AND connid = sdyn_conn-connid
      AND fldate = sdyn_conn-fldate.
  IF sy-subrc NE 0.
    MESSAGE e038.
  ENDIF.
ENDMODULE.                                " check_sflight  INPUT

MODULE exit INPUT.
  CASE ok_code.
    WHEN 'EXIT'.
      LEAVE PROGRAM.
    WHEN 'CANCEL'.
      CLEAR: sdyn_conn, saplane, wa_sflight.
      SET PARAMETER ID: 'CAR' FIELD space,
                        'CON' FIELD space,
                        'DAY' FIELD space.
      LEAVE TO SCREEN 100.
    ENDCASE.
ENDMODULE.                                " exit  INPUT

MODULE check_planetype INPUT.
  SELECT SINGLE seatsmax INTO sdyn_conn-seatsmax FROM saplane
    WHERE planetype = sdyn_conn-planetype.
  CHECK sdyn_conn-seatsmax < sdyn_conn-seatsock.
  MESSAGE e109.
ENDMODULE.                                " check_planetype  INPUT

MODULE trans_from_tc INPUT.
  MOVE sdyn_book-mark TO wa_sdyn_book-mark.
  MODIFY it_sdyn_book INDEX my_table_control-current_line
    FROM wa_sdyn_book TRANSPORTING mark.
ENDMODULE.                                " trans_from_tc  INPUT

```

## 1-2 Model solution SAPMBC410ATABS\_TABLE\_CONTROL2

Extend the sections in bold and create corresponding new modules (using forward navigation).

### Top include

Insert the following code

```
DATA bookings_changed.
```

### Subroutine include

Insert the following subroutine in your include.

```
FORM cancel_bookings.
  DATA: wa_sbook_upd TYPE sbook,
         it_sbook_upd LIKE TABLE OF wa_sbook_upd.
  CONSTANTS cancelled VALUE 'X'.

  LOOP AT it_sdyn_book INTO wa_sdyn_book WHERE mark = 'X'.
    MOVE-CORRESPONDING wa_sdyn_book TO wa_sbook_upd.
    MOVE cancelled TO wa_sbook_upd-cancelled.
    APPEND wa_sbook_upd TO it_sbook_upd.
  ENDLOOP.

  CALL FUNCTION 'BC_GLOBAL_UPDATE_BOOK'
    TABLES
      booking_tab      = it_sdyn_book
      booking_tab_upd  = it_sbook_upd.
  MESSAGE s009.
  bookings_changed = 'X'.
ENDFORM.
```

### Flow logic screen 100

No changes are necessary.

### Flow logic screen 110

No changes are necessary.

## Flow logic screen 120

No changes are necessary.

## Flow logic screen 130

PROCESS BEFORE OUTPUT.

MODULE get\_sbook.

**MODULE modify\_screen\_130.**

LOOP AT it\_sdyn\_book INTO wa\_sdyn\_book WITH CONTROL my\_table\_control.

MODULE trans\_to\_tc.

ENDLOOP.

PROCESS AFTER INPUT.

LOOP AT it\_sdyn\_book.

FIELD sdyn\_book-mark

MODULE trans\_from\_tc ON REQUEST.

ENDLOOP.

**MODULE user\_command\_0130.**

## PBO module include

Insert the following code in your include.

MODULE get\_sbook OUTPUT.

IF wa\_sflight-carrid <> sdyn\_book-carrid OR

wa\_sflight-connid <> sdyn\_book-connid OR

wa\_sflight-fldate <> sdyn\_book-fldate **OR**

**bookings\_changed = 'X'.**

**CLEAR bookings\_changed.**

SELECT \* FROM sbook INTO CORRESPONDING FIELDS OF TABLE it\_sdyn\_book

WHERE carrid = wa\_sflight-carrid

AND connid = wa\_sflight-connid

AND fldate = wa\_sflight-fldate

AND cancelled = not\_cancelled.

DESCRIBE TABLE it\_sdyn\_book LINES my\_table\_control-lines.

ENDIF.

ENDMODULE.

" get\_sbook OUTPUT

```

MODULE modify_screen_130 OUTPUT.
  CHECK NOT mode-maintain_bookings IS INITIAL.
  LOOP AT SCREEN.
    IF screen-name = 'P_DELETE'.
      screen-invisible = 0.
      MODIFY SCREEN.
    ENDIF.
  ENDLOOP.
ENDMODULE.                                " modify_screen_130  OUTPUT

```

## PAI module include

Implement the command field processing for screen 130.

```

MODULE user_command_0130 INPUT.
  CASE ok_code.
    WHEN 'DELE'.
      READ TABLE it_sdyn_book INTO wa_sdyn_book WITH KEY mark = 'X'
        TRANSPORTING NO FIELDS.
      IF sy-subrc = 0.
        PERFORM cancel_bookings.
      ENDIF.
    WHEN 'SELE'.
      LOOP AT it_sdyn_book INTO wa_sdyn_book WHERE mark = space.
        wa_sdyn_book-mark = 'X'.
        MODIFY it_sdyn_book FROM wa_sdyn_book TRANSPORTING mark.
      ENDLOOP.
    WHEN 'DSELE'.
      LOOP AT it_sdyn_book INTO wa_sdyn_book WHERE mark = 'X'.
        wa_sdyn_book-mark = space.
        MODIFY it_sdyn_book FROM wa_sdyn_book TRANSPORTING mark.
      ENDLOOP.
  ENDCASE.
ENDMODULE.                                " user_command_0130  INPUT

```

## 1-3 **Model solution** SAPMBC410ATABS\_TABLE\_CONTROL3

Extend the sections in bold and create corresponding new modules (using forward navigation).

### **Subroutine include**

No changes are necessary.

### **Flow logic screen 100**

No changes are necessary.

### **Flow logic screen 110**

No changes are necessary.

### **Flow logic screen 120**

No changes are necessary.

### **Flow logic screen 130**

No changes are necessary.

### **PBO module include**

No changes are necessary.

### **PAI module include**

Implement the table control functions in the module user\_command\_130.

```
MODULE user_command_0130 INPUT.  
  CASE ok_code.  
    WHEN 'SRTU'.  
      READ TABLE my_table_control-cols INTO wa_cols  
        WITH KEY selected = 'X'.  
  
      IF sy-subrc = 0.  
        SORT it_sdyn_book BY (wa_cols-screen-name+10) ASCENDING.  
      ENDIF.  
    WHEN 'SRTD'.  
      READ TABLE my_table_control-cols INTO wa_cols  
        WITH KEY selected = 'X'.  
  
      IF sy-subrc = 0.
```

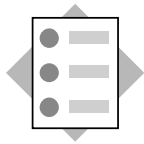
```
      SORT it_sdyn_book BY (wa_cols-screen-name+10) DESCENDING.
    ENDIF.
  WHEN 'DELE'.
    READ TABLE it_sdyn_book INTO wa_sdyn_book WITH KEY mark = 'X'.
    IF sy-subrc = 0.
      PERFORM cancel_bookings.
    ENDIF.
  WHEN 'SELE'.
    LOOP AT it_sdyn_book INTO wa_sdyn_book WHERE mark = space.
      wa_sdyn_book-mark = 'X'.
      MODIFY it_sdyn_book FROM wa_sdyn_book TRANSPORTING mark.
    ENDLOOP.
  WHEN 'DSELE'.
    LOOP AT it_sdyn_book INTO wa_sdyn_book WHERE mark = 'X'.
      wa_sdyn_book-mark = space.
      MODIFY it_sdyn_book FROM wa_sdyn_book TRANSPORTING mark.
    ENDLOOP.
  ENDCASE.
ENDMODULE.                                " user_command_0130  INPUT
```

### Contents:

- Creating, using, and modifying context menus

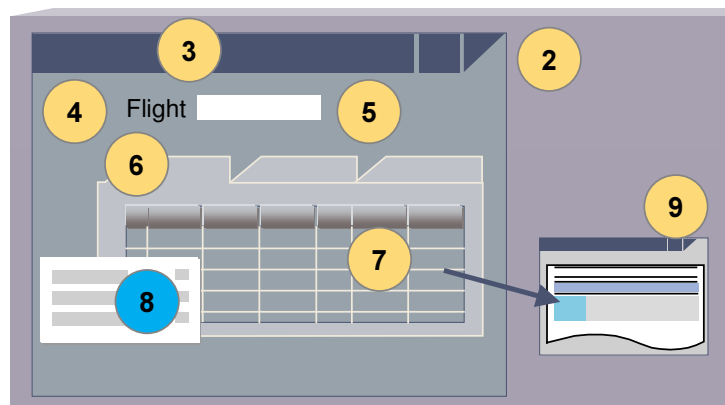


© SAP AG 1999



**At the conclusion of this unit, you will be able to:**

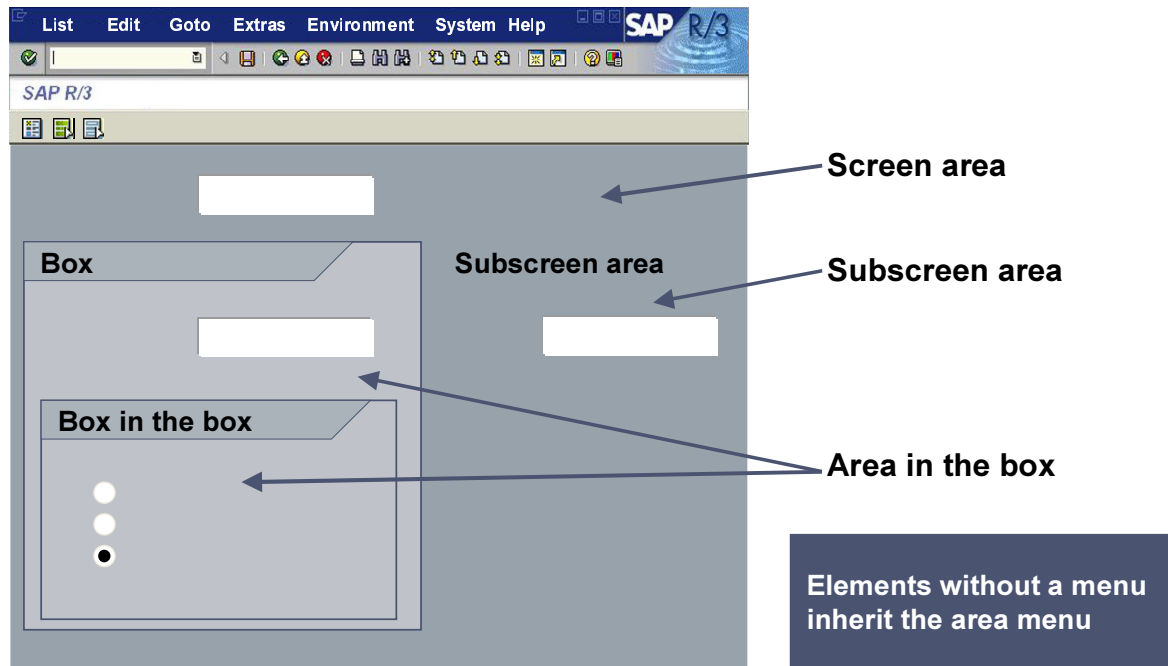
- **Use context menus in your programs.**



© SAP AG 2002

- |          |                                                   |
|----------|---------------------------------------------------|
| ■ Unit 1 | Course Overview                                   |
| ■ Unit 2 | Introduction to Screen Programming                |
| ■ Unit 3 | The Program Interface                             |
| ■ Unit 4 | Screen Elements for Output                        |
| ■ Unit 5 | Screen Elements for Input/Output                  |
| ■ Unit 6 | Screen Elements: Subscreens and Tabstrip Controls |
| ■ Unit 7 | Screen Elements: Table Controls                   |
| ■ Unit 8 | <b>Context Menus</b>                              |
| ■ Unit 9 | Lists in Screen Programming                       |

## Screen: Area of encapsulation of a context menu

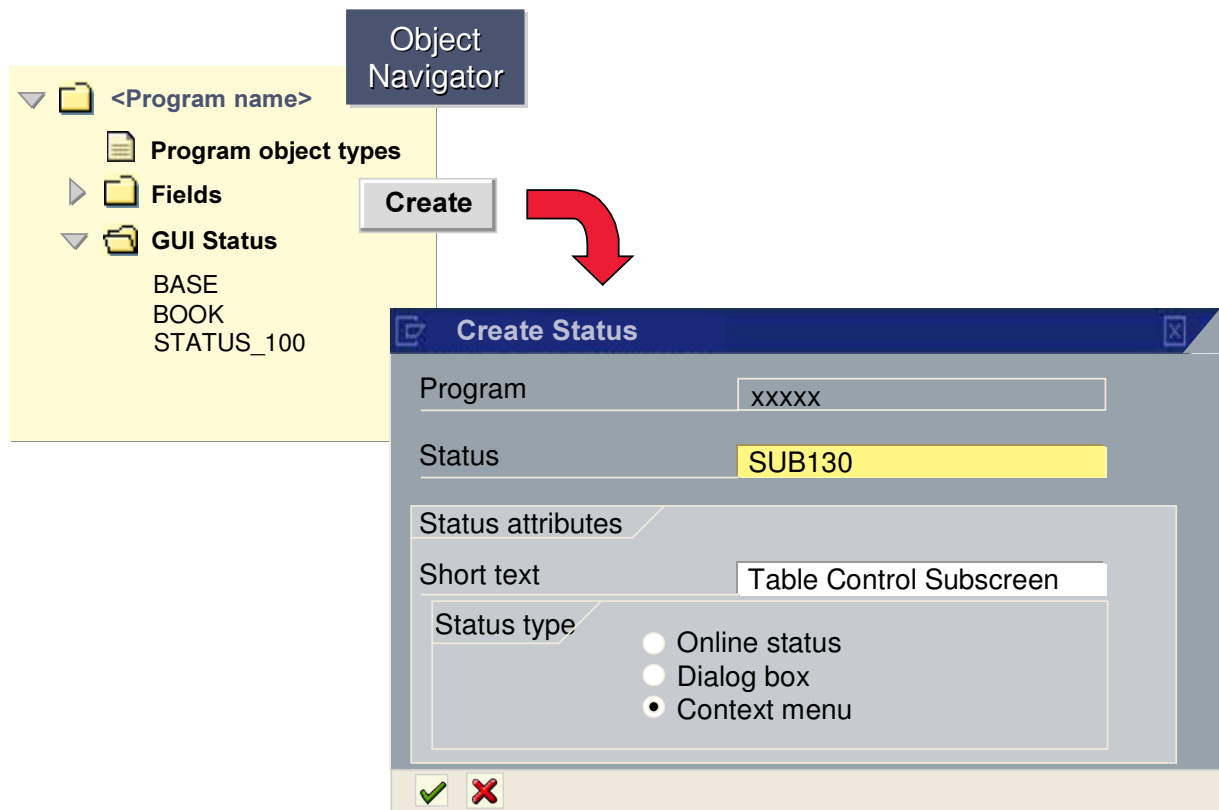


© SAP AG 2002

- Context menus (available with right mouse button or SHIFT-F10) are shortcuts for functions that are frequently used.
- They can be used to display context-sensitive menus. The context is defined by the position (cursor for SHIFT-F10, mouse location for right mouse button) where the user called the context menu. The user can select functions that are relevant for the current context using the context menu.
- You define if a context menu should be offered when you create a screen object (screens, input fields, table controls, boxes, and so on). When the user selects a context menu on an object, an event mechanism (as understood by ABAP objects) calls a certain subroutine in the application program. This delivers a menu reference to the subroutine. The program uses this menu reference to build the display menu. Here you can use menus defined with the Menu Painter and dynamic menus.
- After the user executes a menu function, the application program regains control and can react to the user input.
- Context menus are assigned to output fields. When you assign a context menu to a box, table control or screen (normal or subscreen), all the subordinate output fields that do not have a context menu inherit that one.

## Creating a Context Menu

SAP

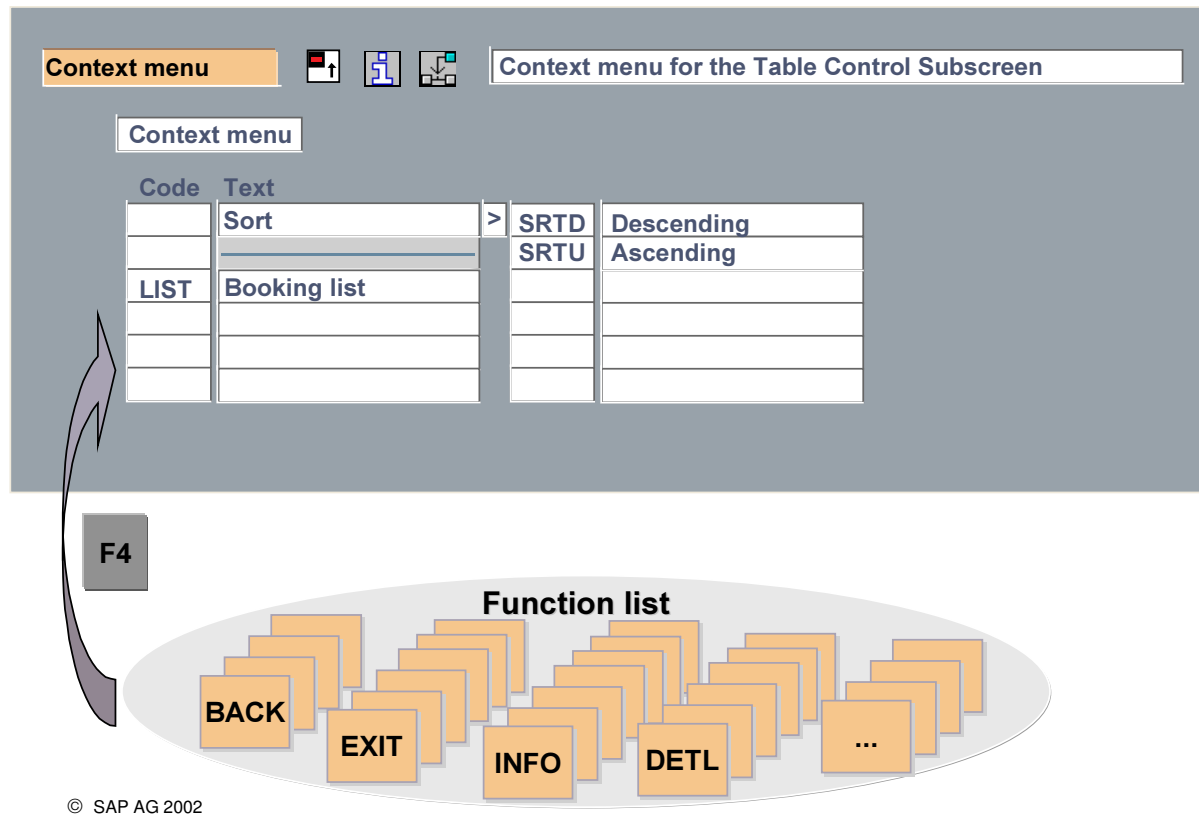


© SAP AG 2002

- You can create a context menu from within the object list of the Object Navigator. Position the cursor on *GUI status* and right-click. The Object Navigator automatically opens the Menu Painter.
- You can also create a context menu directly in the Menu Painter.
- A context menu is a special GUI status. Assign it a name, a descriptive text, and status type *Context menu*.

## Creating a Context Menu: Assigning Functions

SAP



- In a context menu, you can link any function codes and function texts. In particular, you can take advantage of your screen pushbuttons. The functions already provided in the interface can be used as an F4 input help.
- The link technique ensures consistent context menus in large applications.
- You should observe the following rules when designing context menus.
  - Do not use any functions that cannot be found elsewhere in the system. (pushbuttons or interface)
  - Avoid using more than two hierarchy levels in context menus.
  - Do not use more than 10 entries, but map all the available pushbuttons.
  - Use separators to structure the context menu optically.
  - Place object-specific statements at the beginning of the menu.

## Screen Painter



### Element Attributes

#### Context Menu Form

ON\_CTMENU\_ RBGFRAME

### Screen Attributes

#### General Attributes

#### Context Menu Form

ON\_CTMENU\_ SUB130

```
*****
***  INCLUDE xxxxF01          *
*****
```

...

```
FORM on_ctmenu_rbgframe
  USING p_menu TYPE REF TO cl_ctmenu.
...
ENDFORM.
```

...

```
FORM on_ctmenu_sub130
  USING p_menu TYPE REF TO cl_ctmenu.
...
ENDFORM.
```

ABAP

© SAP AG 2002

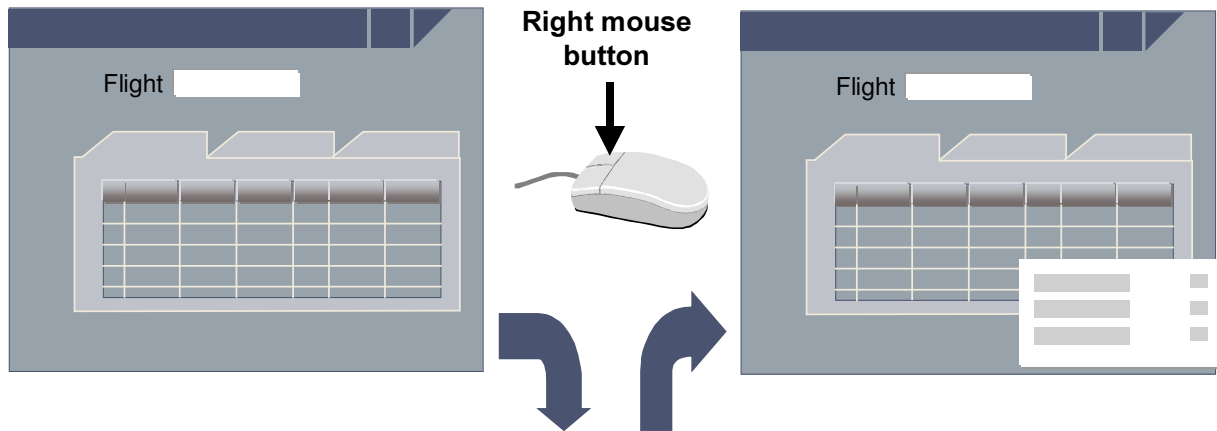
- Right-clicking triggers a callback routine in your program. You must create the callback routine in your application program. It is named ON\_CTMENU\_<name>. You determine which callback routine is called in the Screen Painter, either in the screen attributes or in the general attributes of a screen element.
- You can directly assign a callback routine to input/output fields, text fields, and status icons. Checkboxes, radio buttons, and pushbuttons do not have their own callback routines. However, these fields can inherit context menus from boxes or screens.
- If you assign a callback routine to a table control, it is triggered for all the fields of the table control that do not have their own callback routine.
- The callback routine takes the following form:

```
FORM ON_CTMENU_<name> USING p_menu TYPE REF TO cl_ctmenu.
  <definition of the context menu>.
ENDFORM.
```

- You create the structure of the context menu by loading a statically defined menu or by passing an instance of the class cl\_ctmenu with a method call.

## Using the Context Menu

SAP



```
FORM on_ctmenu_sub130
  USING p_menu TYPE REF TO cl_ctmenu.

  CALL METHOD cl_ctmenu=>load_gui_status
    EXPORTING program = sy-cprog
              status  = 'SUB130'
              menu    = p_menu.

ENDFORM.
```

ABAP

© SAP AG 2002

- Right-clicking an output field triggers the corresponding callback routine.
- You can now use the static method `load_gui_status` of class `cl_ctmenu` to load a context menu that was predefined in the Menu Painter. Using other methods of class `cl_ctmenu` (see next graphic), you can also completely rebuild the context menu or modify a loaded menu.
- If the user triggers a function in the context menu, the corresponding function code is placed in the command field and triggered depending on function type PAI of the screen.

## Modifying Context Menus Dynamically

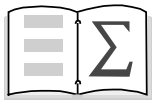


```
CALL METHOD <instance>-><name> EXPORTING ... .
```

| <i><b>Method</b></i>     | <i><b>Meaning</b></i>    |
|--------------------------|--------------------------|
| <b>ADD_FUNCTION</b>      | <b>Add a function</b>    |
| <b>ADD_SEPARATOR</b>     | <b>Add a separator</b>   |
| <b>HIDE_FUNCTIONS</b>    | <b>Hide functions</b>    |
| <b>SHOW_FUNCTIONS</b>    | <b>Show functions</b>    |
| <b>DISABLE_FUNCTIONS</b> | <b>Disable functions</b> |
| <b>ENABLE_FUNCTIONS</b>  | <b>Enable functions</b>  |

© SAP AG 2002

- Class `cl_ctmenu` provides a number of other methods in addition to the static method `load_gui_status`. You can use them to adjust the context menu at run time (for example, using the values in data fields).
- The corresponding methods are called within the callback routine.
- You can find further information in the documentation for class `cl_ctmenu` in the Class Builder.



**You are now able to:**

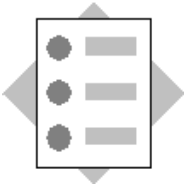
- **Use context menus in your programs.**

# Exercises



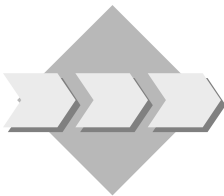
## Unit: Context Menus

### Topic: Creating and using a context menu - Optional



At the conclusion of these exercises, you will be able to:

- Use context menus in your programs.



Make the functions for your table control available in a context menu.

- 1-1 Create a GUI status with type context menu and use it for the output fields on screen 130

Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410ATABS\_TABLE\_CONTROL3**). You can use the model solution **SAPMBC410ACONS\_CONTEXTMENU** for orientation.

- 1-2 Create the GUI status **sub130** with type *context menu* and the short description *Table control subscreen*. Assign the following functions to the menu:

| Context menu | Function code: | Function text:        |
|--------------|----------------|-----------------------|
|              | DELE           | Delete bookings       |
|              | SELE           | Select all            |
|              | DSELE          | Deselect all          |
|              | Separator      |                       |
|              | SRTU           | Sort in Ascending...  |
|              | SRTD           | Sort in Descending... |

Assign function type ' ' (space) to all of the functions. Deactivate the function DELE.

- 1-3 In the screen attributes of 130, declare that you want to use subroutine **on\_ctmenu\_sub130** to create the context menu.

1-4 Write the subroutine to create the context menu.

1-5 Optional:

Activate the DELE function at run time if the user is in booking maintenance mode.  
Note that you must pass the function code to the method in a table with type  
**ui\_functions**.



## Unit: Context Menus

### Topic: Creating and using a context menu

#### 1-1 Model solution SAPMBC410CONS\_CONTEXTMENU

Add the coding in bold type to your program and create the subroutine.

##### Subroutine include

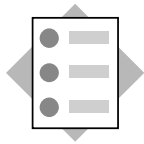
```
FORM on_ctmenu_sub130 USING p_menu TYPE REF TO cl_ctmenu.  
  DATA fcodes TYPE ui_functions.  
  
  CALL METHOD cl_ctmenu=>load_gui_status  
    EXPORTING program = sy-cprog  
              status  = 'SUB130'  
              menu    = p_menu.  
  
  CHECK NOT mode-maintain_bookings IS INITIAL.  
  APPEND 'DELE' TO fcodes.  
  CALL METHOD p_menu->enable_functions EXPORTING fcodes = fcodes.  
  
ENDFORM.                                " ON_CTMENU_SUB130
```

### Contents:

- Lists on screens

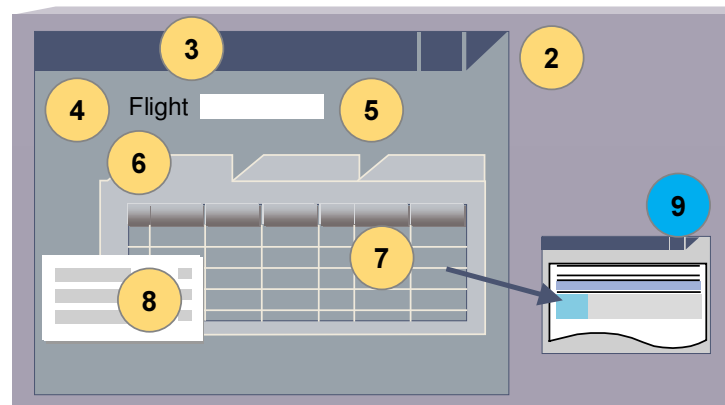


© SAP AG 1999



**At the conclusion of this unit, you will be able to:**

- **Use lists on screens in your programs.**



© SAP AG 2002

- Unit 1 Course Overview
- Unit 2 Introduction to Screen Programming
- Unit 3 The Program Interface
- Unit 4 Screen Elements for Output
- Unit 5 Screen Elements for Input/Output
- Unit 6 Screen Elements: Subscreens and Tabstrip Controls
- Unit 7 Screen Elements: Table Controls
- Unit 8 Context Menus
- Unit 9 **Lists in Screen Programming**

## Creating a List Buffer

SAP

Screen  
Painter

```
PROCESS BEFORE OUTPUT.  
  :  
  MODULE write_PBO.  
  :
```

Screen 100

```
PROCESS AFTER INPUT.  
  :  
  MODULE write_PAI.  
  :
```

Screen  
Painter

ABAP

```
MODULE write_....  
  ...  
  LOOP AT it_book  
    INTO wa_book.  
    WRITE ...  
  ENDLOOP.  
  ...  
ENDMODULE.
```



List buffer

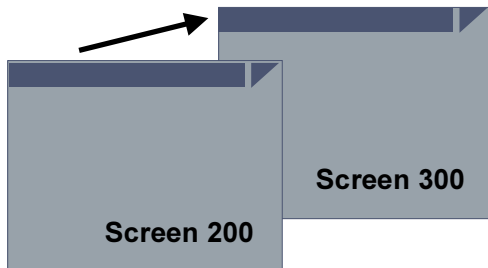
© SAP AG 2002

- Fill the corresponding basic list buffer with WRITE statements at PBO or PAI. You can create your own list and column headers by programming an event TOP-OF-PAGE. This event will be triggered whenever a new page is created in the list buffer with NEW-PAGE.
- You can send the output directly to the spool with the NEW-PAGE PRINT ON statement.

## List Display at the Front End

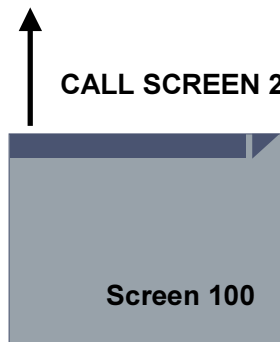
SAP

SET SCREEN 300.

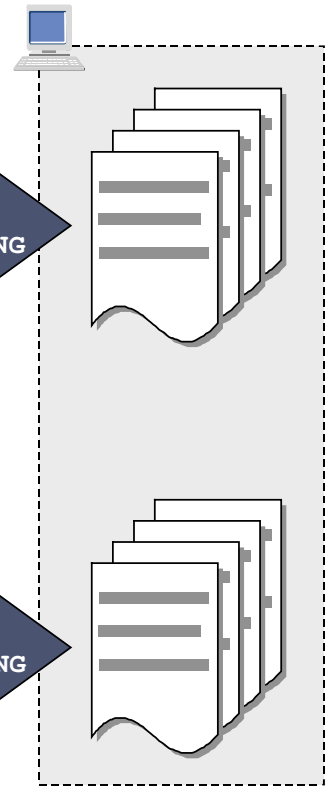


PBO/PAI (Screen 300)  
LEAVE TO LIST-PROCESSING

CALL SCREEN 200.



PBO/PAI (Screen 100)  
LEAVE TO LIST-PROCESSING



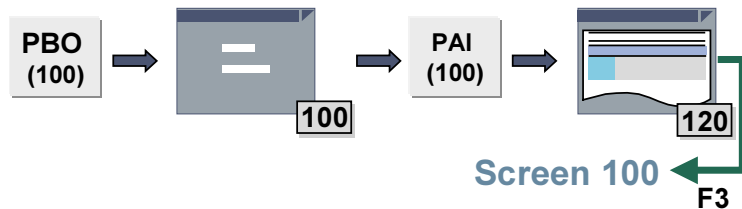
© SAP AG 1999

- There is no common list buffer outside of a CALL level.
- The list display is processed at the end of the screen in which LEAVE TO LIST-PROCESSING was programmed at PBO or PAI.
- To direct the output to the spool, use the NEW-PAGE PRINT ON statement, but **not** LEAVE TO LIST-PROCESSING.

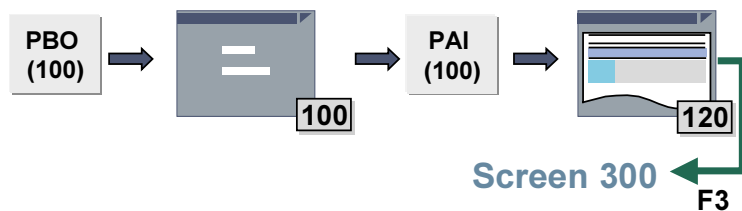
## Displaying List on a Screen

SAP

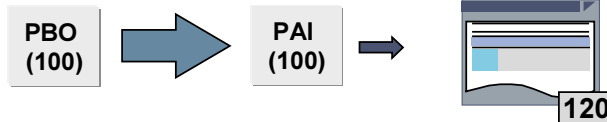
```
MODULE <Module_name>.  
  LEAVE TO LIST-PROCESSING.  
  SET PF-STATUS SPACE.  
  ...  
  WRITE...SKIP...ULINE...  
ENDMODULE.
```



```
MODULE <Module_name>.  
  LEAVE TO LIST-PROCESSING  
  AND RETURN TO SCREEN 300.  
  SET PF-STATUS SPACE.  
  ...  
ENDMODULE.
```

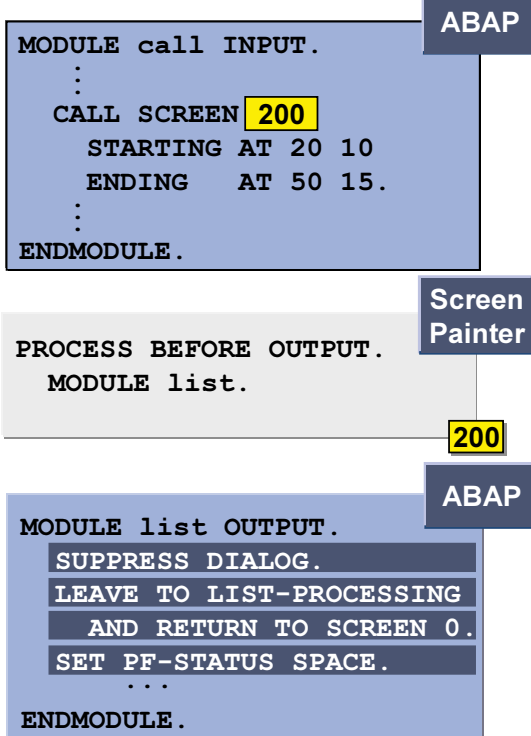


```
MODULE <Module_name> OUTPUT.  
  SUPPRESS DIALOG.  
  LEAVE TO LIST-PROCESSING.  
  SET PF-STATUS SPACE.  
  ...  
ENDMODULE.
```



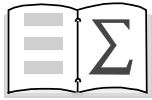
© SAP AG 2002

- To create a list that is displayed on a screen, use the ABAP statement **LEAVE TO LIST-PROCESSING**. This sets a switch that ensures that the contents of the list buffer are output once the current screen has been processed. The **SET PF-STATUS SPACE** statement ensures that the list is displayed with the standard GUI status for lists.
- Once the screen has been fully processed and **LEAVE TO LIST-PROCESSING** is executed, the list is displayed on list screen 120 (screen for a basis program).
- You can also use the following format: **LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0**. **SET PF-STATUS SPACE**. **WRITE ... LEAVE SCREEN**.
- When the system exits list processing (user selects F3 or ABAP statement **LEAVE LIST-PROCESSING**), the system carries on processing the PBO of the calling screen from which the list processing was started. You can override this by using the **AND RETURN TO SCREEN <scr>** addition in the **LEAVE TO LIST-PROCESSING** statement.
- If you include the ABAP statement **SUPPRESS DIALOG** in a PBO module, the current screen is not displayed.



© SAP AG 2002

- If you want to display a list in a dialog box within a transaction, you must call a screen, but include the SUPPRESS DIALOG statement in its PBO processing block.
- In order to return to the calling screen when the user leaves the list, use the statement LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.



**You are now able to:**

- **Use lists on screens in your programs**

# Exercises



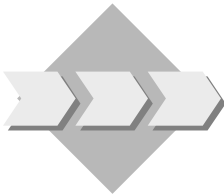
## Unit: Lists in Screen Programming

### Topic: Displaying a list on a screen



At the conclusion of these exercises, you will be able to:

- Use lists on screens in your programs.



Extend your flight maintenance screen to display a booking list. Allow the user to display the bookings by choosing a pushbutton or menu entry.

Program the booking list.

- 1-1 Extend your program **SAPMZ##BC410\_SOLUTION** from the previous exercise (or copy the model solution **SAPMBC410ACONS\_CONTEXTMENU**). Use the model solution **SAPMBC410ALISS\_LISTS\_ON\_DYNPROS** for orientation.
- 1-2 Write a subroutine **display\_bookings** to display the data from the internal table **it\_sbook**. Display the following data:
  - Booking number (bookid),*
  - Customer number (customid)*
  - Customer type (custtype)*
  - Luggage weight (luggweight)*
  - Unit of weight (wunit)*
  - Class (class)*
  - Booking date (order\_date).*
- 1-3 Assign the function **BOOK** in status **STATUS\_100** to function key **F5**, and also to a pushbutton and a menu entry.
- 1-4 Call screen **200** (**STARTING AT...**) if the user chooses the **BOOK** function. Create the screen (type: Modal dialog box). In the **PBO** event of the screen, call a module in which you create the list. Screen **200** is only a container. You should not actually display it. Create a GUI title (**T\_BOOK**) for the booking list. Set the GUI status **SPACE** and GUI title **T\_BOOK** and start list processing.



## Unit: Lists in Screen Programming

### Topic: Displaying a list on a screen

## Model solution SAPMBC410ALISS\_LISTS\_ON\_DYNPROS

Add the coding in bold type to your program and create the module.

### Flow logic screen 200

PROCESS BEFORE OUTPUT.

**MODULE list.**

\*

PROCESS AFTER INPUT.

### PBO module include

Add the coding in bold type to your include

MODULE status\_0100 OUTPUT.

SET PF-STATUS 'STATUS\_100'.

SET TITLEBAR 'TITLE\_100'.

ENDMODULE.

" STATUS\_0100 OUTPUT

MODULE clear\_ok\_code OUTPUT.

CLEAR ok\_code.

ENDMODULE.

" clear\_ok\_code OUTPUT

MODULE modify\_screen OUTPUT.

CHECK NOT mode-maintain\_flights IS INITIAL.

LOOP AT SCREEN.

IF screen-name = 'SDYN\_CONN-PLANETYPE'.

screen-input = 1.

screen-required = 1.

MODIFY SCREEN.

```

ENDIF.
ENDLOOP.
ENDMODULE.                                " modify_screen  OUTPUT

MODULE get_spfli OUTPUT.
  ON CHANGE OF wa_sflight-carrid OR wa_sflight-connid.
    SELECT SINGLE * INTO CORRESPONDING FIELDS OF sdyn_conn FROM spfli
      WHERE carrid = wa_sflight-carrid
        AND connid = wa_sflight-connid.
  ENDON.
ENDMODULE.                                " GET_SPFLI  OUTPUT

MODULE get_saplane OUTPUT.
  ON CHANGE OF wa_sflight-planetype.
    SELECT SINGLE * FROM saplane WHERE planetype = wa_sflight-planetype.
  ENDON.
ENDMODULE.                                " GET_SAPLANE  OUTPUT

MODULE fill_dynnr OUTPUT.
  CASE my_tabstrip-activetab.
    WHEN 'FC1'.
      dynnr = 110.
    WHEN 'FC2'.
      dynnr = 120.
    WHEN 'FC3'.
      dynnr = 130.
    WHEN OTHERS.
      my_tabstrip-activetab = 'FC1'.
      dynnr = 110.
  ENDCASE.
ENDMODULE.                                " set_dynnr  OUTPUT

MODULE get_sbook OUTPUT.
  IF wa_sflight-carrid <> sdyn_book-carrid OR
    wa_sflight-connid <> sdyn_book-connid OR
    wa_sflight-fldate <> sdyn_book-fldate OR
    bookings_changed = 'X'.
    CLEAR bookings_changed.

    SELECT * FROM sbook INTO CORRESPONDING FIELDS OF TABLE it_sdyn_book
      WHERE carrid = wa_sflight-carrid
        AND connid = wa_sflight-connid
        AND fldate = wa_sflight-fldate
        AND cancelled = not_cancelled.

    DESCRIBE TABLE it_sdyn_book LINES my_table_control-lines.
  ENDIF.

```

```

ENDMODULE.                                " get_sbook  OUTPUT

MODULE trans_to_tc OUTPUT.
  MOVE wa_sdyn_book TO sdyn_book.
ENDMODULE.                                " trans_to_dynp  OUTPUT

MODULE modify_screen_130 OUTPUT.
  CHECK NOT mode-maintain_bookings IS INITIAL.
  LOOP AT SCREEN.
    IF screen-name = 'P_DELETE'.
      screen-invisible = 0.
      MODIFY SCREEN.
    ENDIF.
  ENDLOOP.
ENDMODULE.                                " modify_screen_130  OUTPUT

MODULE list OUTPUT.
  SELECT * FROM sbook
    INTO TABLE it_sbook
    WHERE carrid = wa_sflight-carrid
      AND connid = wa_sflight-connid
      AND fldate = wa_sflight-fldate
      AND cancelled = space.
  IF sy-dbcnt = 0.
    MESSAGE i186.
  ELSE.
    PERFORM display_bookings.
    SET PF-STATUS space.
    SET TITLEBAR 'T_BOOK'.
  ENDIF.
  LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.
  SUPPRESS DIALOG.
ENDMODULE.                                " LIST  OUTPUT

```

## Subroutines include

Add the coding in bold type to your include

```

FORM update_sflight.
  UPDATE sflight FROM wa_sflight.
  IF sy-subrc NE 0.
    MESSAGE a008.
  *   Fehler beim Ändern
  ENDIF.
  MESSAGE s009.
ENDFORM.                                " update_sflight

```

```

FORM cancel_bookings.
  DATA: wa_sbook_upd TYPE sbook,
         it_sbook_upd LIKE TABLE OF wa_sbook_upd.
  CONSTANTS cancelled VALUE 'X'.

  LOOP AT it_sdyn_book INTO wa_sdyn_book WHERE mark = 'X'.
    MOVE-CORRESPONDING wa_sdyn_book TO wa_sbook_upd.
    MOVE cancelled TO wa_sbook_upd-cancelled.
    APPEND wa_sbook_upd TO it_sbook_upd.
  ENDLOOP.
  CALL FUNCTION 'BC_GLOBAL_UPDATE_BOOK'
    TABLES
      booking_tab      = it_sdyn_book
      booking_tab_upd  = it_sbook_upd.

  MESSAGE s009.

ENDFORM.                                " cancel_bookings

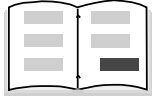
FORM on_ctmenu_sub130 USING p_menu TYPE REF TO cl_ctmenu.
  DATA fcodes TYPE ui_functions.

  CALL METHOD cl_ctmenu=>load_gui_status
    EXPORTING program = sy-cprog
              status = 'S_SUB130'
              menu = p_menu.

  CHECK NOT mode-maintain_bookings IS INITIAL.
  APPEND 'DELE' TO fcodes.
  CALL METHOD p_menu->enable_functions EXPORTING fcodes = fcodes.
ENDFORM.                                " on_ctmenu_sub130

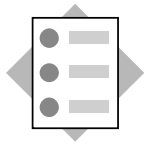
FORM display_bookings.
  LOOP AT it_sbook INTO wa_sbook.
    WRITE: / wa_sbook-bookid,
            wa_sbook-customid,
            wa_sbook-custtype,
            wa_sbook-luggweight UNIT wa_sbook-wunit,
            wa_sbook-wunit,
            wa_sbook-class,
            wa_sbook-order_date,
            wa_sbook-cancelled.
  ENDLOOP.
ENDFORM.                                " DISPLAY_BOOKINGS

```



- **This section contains supplementary material to be used for reference.**
- **This material is not part of the standard course.**
- **The instructor may not cover this material during the course presentation.**

© SAP AG 2002



**At the conclusion of this topic, you will be able to:**

- **Create transaction variants that you can assign to users using the above methods.**

© SAP AG 2002

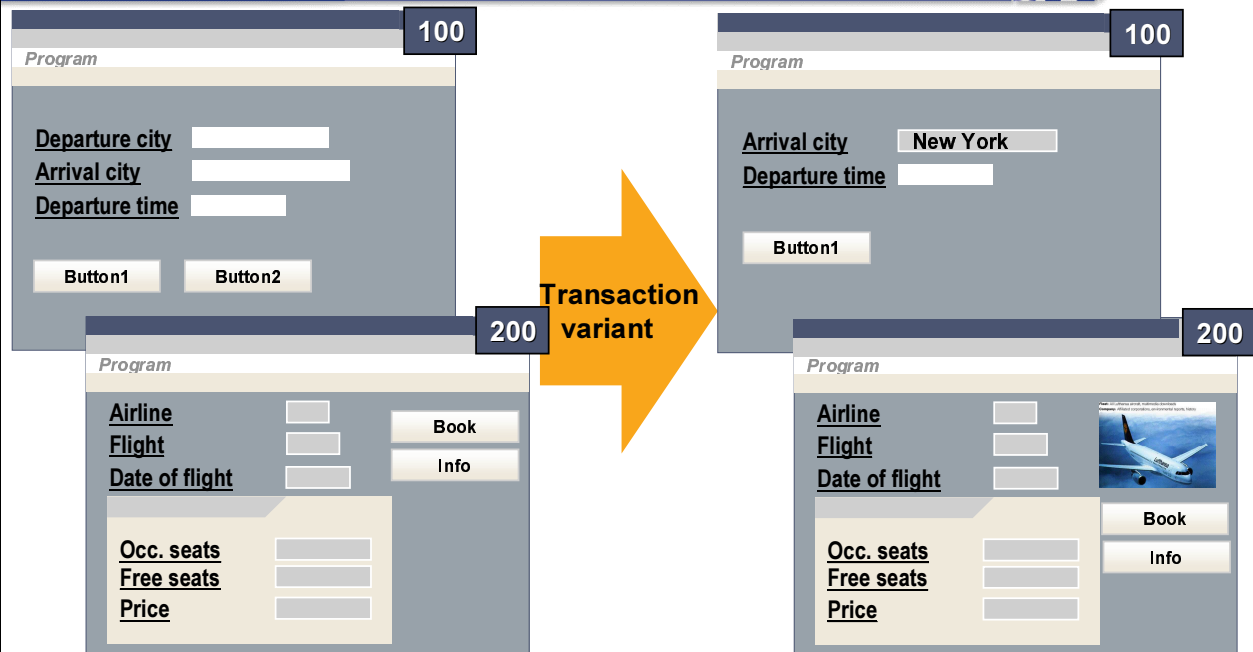
- There are ways to set single transactions to the needs of your enterprise or individual user groups. In this unit, you will see how a transaction can be simplified without being modified.

- **Simplify system by suppressing functions that are not required:**
  - **Predefine fields**
  - **Revoke ready for input status**
  - **Suppress screen elements that are not needed (fields, subscreens, and screens)**
- **Different scope:**
  - **System: Global fixed values**
  - **Transaction: Transaction variants**
- **Standard variants or individual variants**
- **WYSIWYG maintenance with special recording function**

© SAP AG 2002

## Transaction Variants: Example

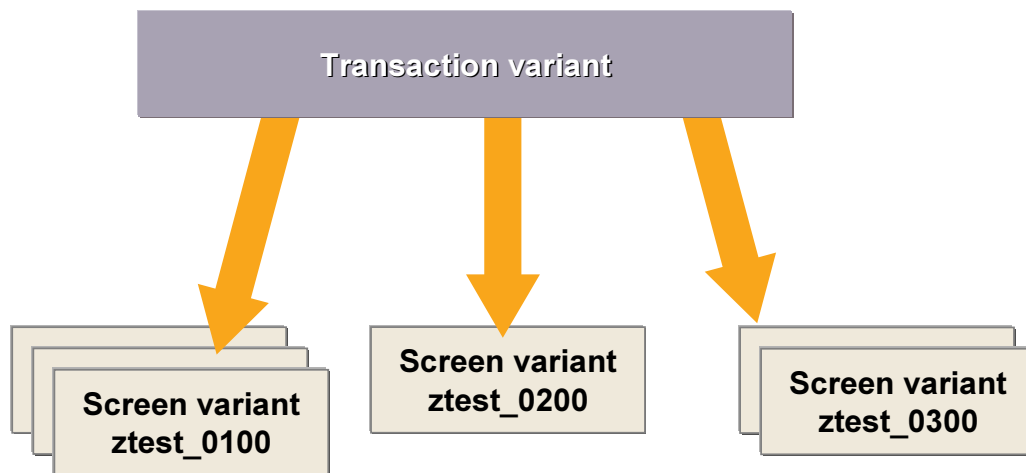
SAP



- Predefine fields with values
- Suppress fields
- Hide screens

© SAP AG 2002

- In this example, you see two screens of an SAP transaction that should be redesigned using a transaction variant.
- Screen 100 is changed as follows: fields are hidden, field attributes are changed, and buttons are hidden.
- Screen 200 shows the following changes: buttons moved and screen inserted (with GuiXT). The use of GuiXT will be discussed in more detail later.

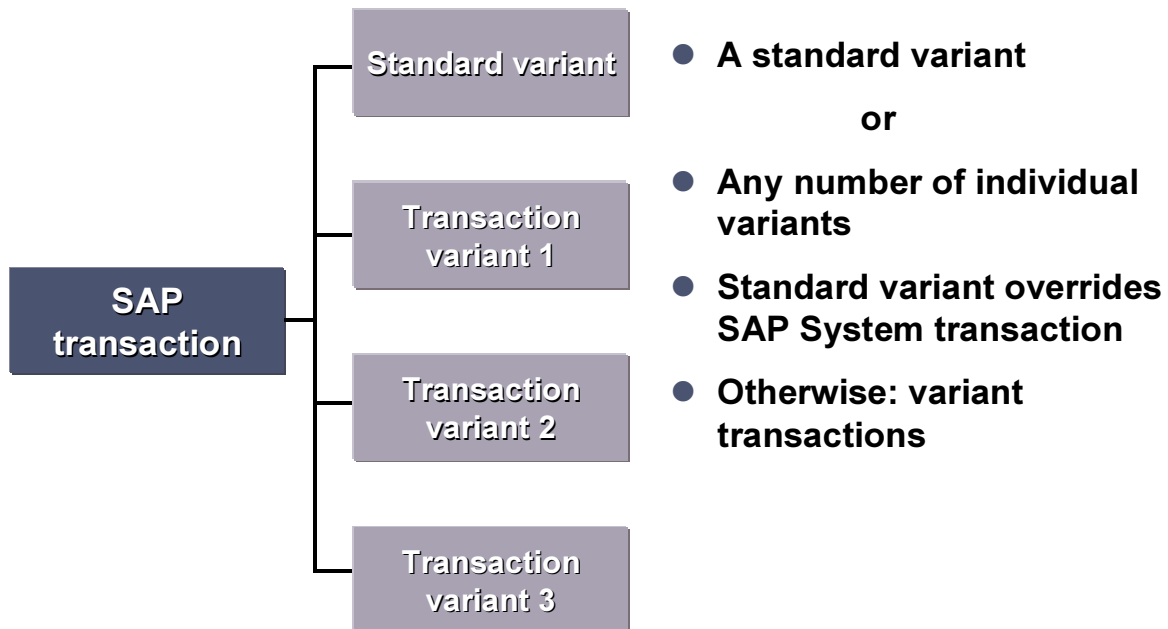


© SAP AG 2002

- A **transaction variant** is a reference to a set of **screen variants**.
- You can create any number of screen variants for a screen. The transaction variant consists of these screen variants.

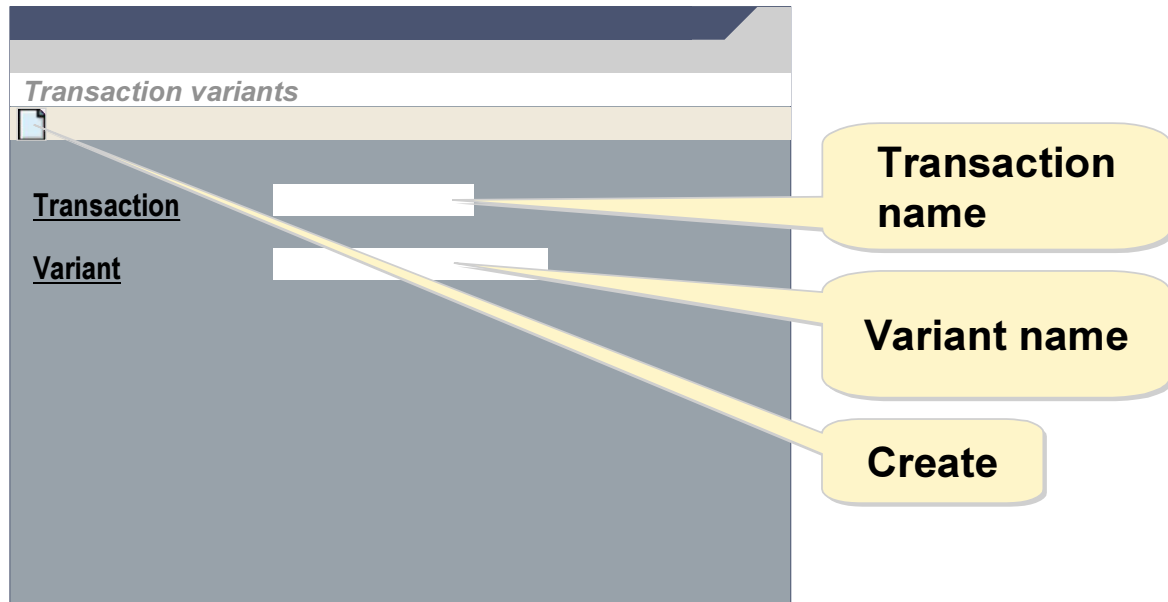
## Transaction Variants: Options:

SAP



© SAP AG 2002

- You can create different kinds of transaction variants for an SAP System transaction:
  - A standard variant
  - Any number of normal transaction variants
- The standard variant is executed at runtime instead of the SAP Systemdelivered transaction. No new transaction code is required.
- A normal transaction variant will be called with its own transaction code of type *variant transaction*.



- **Tools** ➤ **AcceleratedSAP**
- **Personalization** ➤ **Transaction variants**

© SAP AG 2002

- To create transaction variants, choose *AcceleratedSAP* → *Personalization* → *Transaction variant*. This takes you to the transaction for maintaining transaction variants.
- Enter the name of the transaction for which you want to create a variant. The name of the variant must be unique in the system and be in the customer namespace.
- With the menu option *Goto*, choose whether you want to create a client-specific or a cross-client transaction variant.
- To create the variant, choose the appropriate button in the application toolbar.

## Transaction Variants: Evaluating Fields

SAP

100

Program

Departure city

Arrival city

Departure time

Button1 Button2

Departure city Frankfurt

Arrival city New York

Departure time

Button1

Button2

Finish and save Menu functions GuiXT

© SAP AG 2002

- Select *Screen entries* to start the transaction in CALL mode.
- Triggering a dialog also triggers Process After Input (PAI) of the current screen. The system sends another screen in which you can evaluate the fields of the screen.
- Online documentation provides further information about transaction variants.
- The screen that was evaluated is stored as a screen variant when you continue.

The screenshot shows the SAP Screen Variant Manager interface. It includes a header bar with the title 'Screen Variants' and the SAP logo. Below the header, there is a table with columns for field names and checkboxes for various attributes. The table contains the following data:

| Field Name     | Value     | Attribute 1              | Attribute 2              | Attribute 3              | Attribute 4              | Attribute 5              |
|----------------|-----------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| Departure city | Frankfurt | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Arrival city   | New York  | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Departure time |           | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Button1        |           | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Button2        |           | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

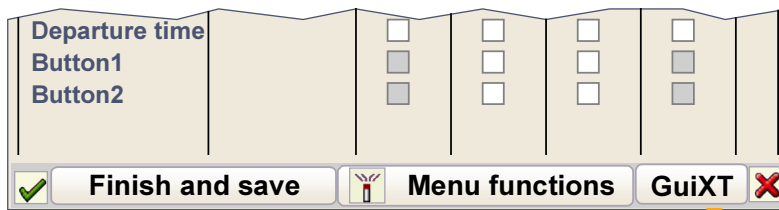
Callouts point to the following elements:

- Screen variant name:** Points to the text input field at the top of the table.
- Description:** Points to the text input field below the name field.
- Set field attributes:** Points to the checkboxes in the table.
- Deactivate menu functions:** Points to the 'Menu functions' button at the bottom.

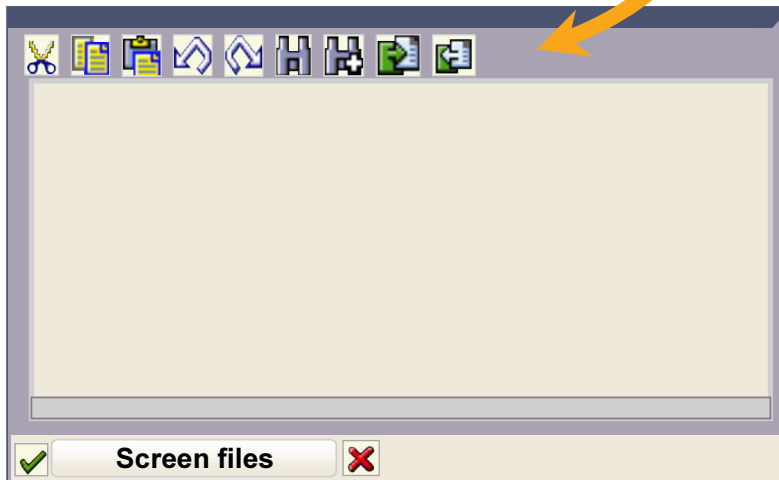
At the bottom of the interface, there are three buttons: 'Finish and save' (with a green checkmark icon), 'Menu functions' (with a red 'X' icon), and 'GuiXT' (with a red 'X' icon).

© SAP AG 2002

- A screen variant is an independent Repository object, which has a unique name in the system. The name is constructed as follows:
  - Variant name
  - Client (only for client-specific transaction variants)
  - Screen number
- Specify whether or not field contents should be copied to the screen variant. You can set various attributes for the individual fields:
  - You can undo or hide the input status of a field
  - You can find a detailed list of options in the online documentation about transaction variants



Screen variant  
maintenance screen



- Script editor
- Script is stored as text file
- Scripts can be imported

© SAP AG 2002

- The GuiXT tool allows you to design the individual screens in a more flexible manner. GuiXT uses a script language to:
  - Position objects on the screen
  - Set attributes
  - Include new objects
- If you select **GuiX**, an editor window appears where you can enter the script. You can also choose GuiXT files stored on your local machine.
- You can also import scripts created on the local machine and export them there.

```
// Version: 19990921151118
```

**Comment**

```
IMAGE (1,1) "C:\sap.jpg"
```

**Insert screen**

```
BOX (10,20) (16,44) "Frame"
```

**Insert frame**

```
POS [Element] [Element]+(10,0)
```

**Move  
element**

```
POS [Area] [Area]+(10,0)
```

```
Pushbutton (10,50) "Text" "SCMP"
```

**Pushbutton  
with text and  
function code**

© SAP AG 2002

- You can change the layout of a screen with the script language used by GuiXT. You can:
  - Move objects
  - Insert screens
  - Insert pushbuttons
  - Insert value helps
  - Change the input attributes of fields
  - Delete screen elements
- You are provided with complete documentation of GuiXT with the installation. You can find more information on the homepage of the GuiXT vendor, <http://www.synactive.com>.

### Options for starting transaction variants

- Test environment
- Transaction code
- From user menu

© SAP AG 2002

- You have the following options for starting a transaction variant:
  - Test environment
  - Transaction code of type *variant transaction*
  - User menu
- You can test the process flow of the transaction in the test environment of transaction variant maintenance. This is intended primarily for developers who are creating transaction variants.
- To hang a variant transaction in a user menu or role, you must create a transaction code of type *variant transaction*.

## Creating Variant Transactions

SAP

### Transaction Maintenance

Transaction code

Transaction code name



Display



Change



Create

Create

### Create transaction

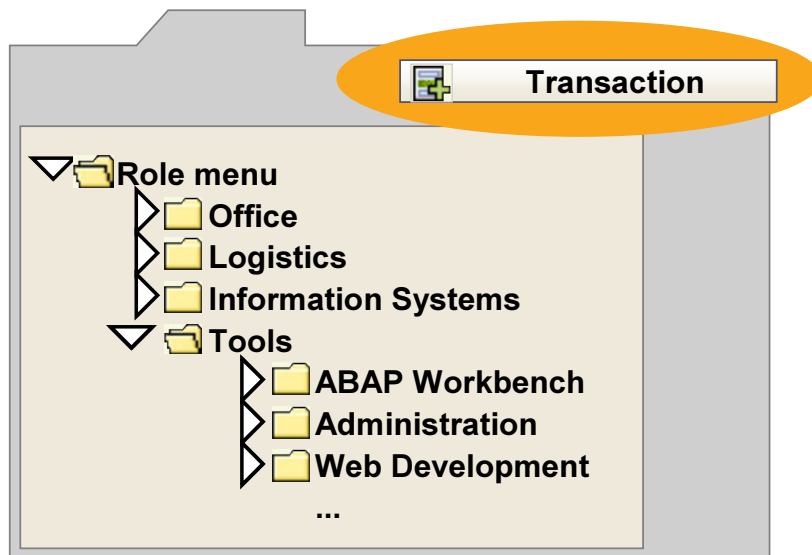
Transaction code

#### Transaction attributes

- ☐ Dialog transaction
  - ☐ Report transaction
  - ☐ OO Transaction
  - ☒ **Variant transaction**
- Parameter transaction

© SAP AG 2002

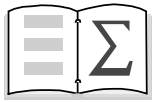
- To start a transaction variant from a menu, you must create a transaction code of type *variant transaction*. You can navigate there directly from the maintenance screen for the transaction variants. Alternatively, you can start the corresponding transaction from the ABAP Workbench.



- Role
- Area menu

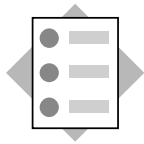
© SAP AG 2002

- You can insert the transaction in a menu by maintaining:
  - A role
  - An area menu.
- The user can immediately see the changes made in this way.



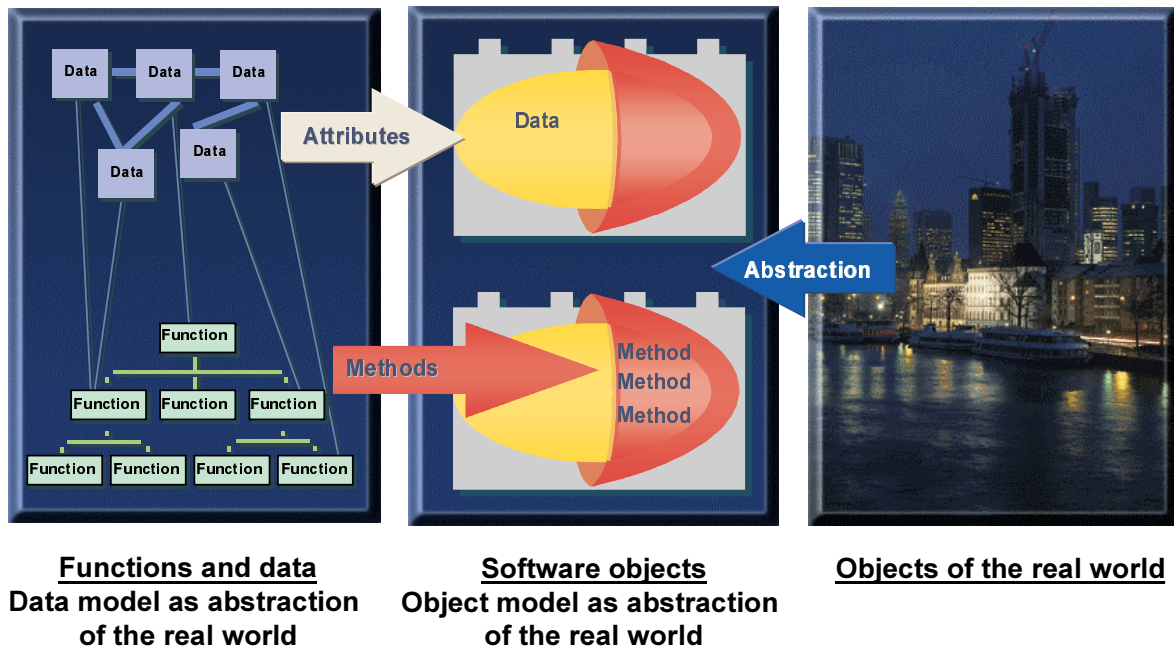
**You are now able to:**

- **Create transaction variants that you can assign to users using the above methods**



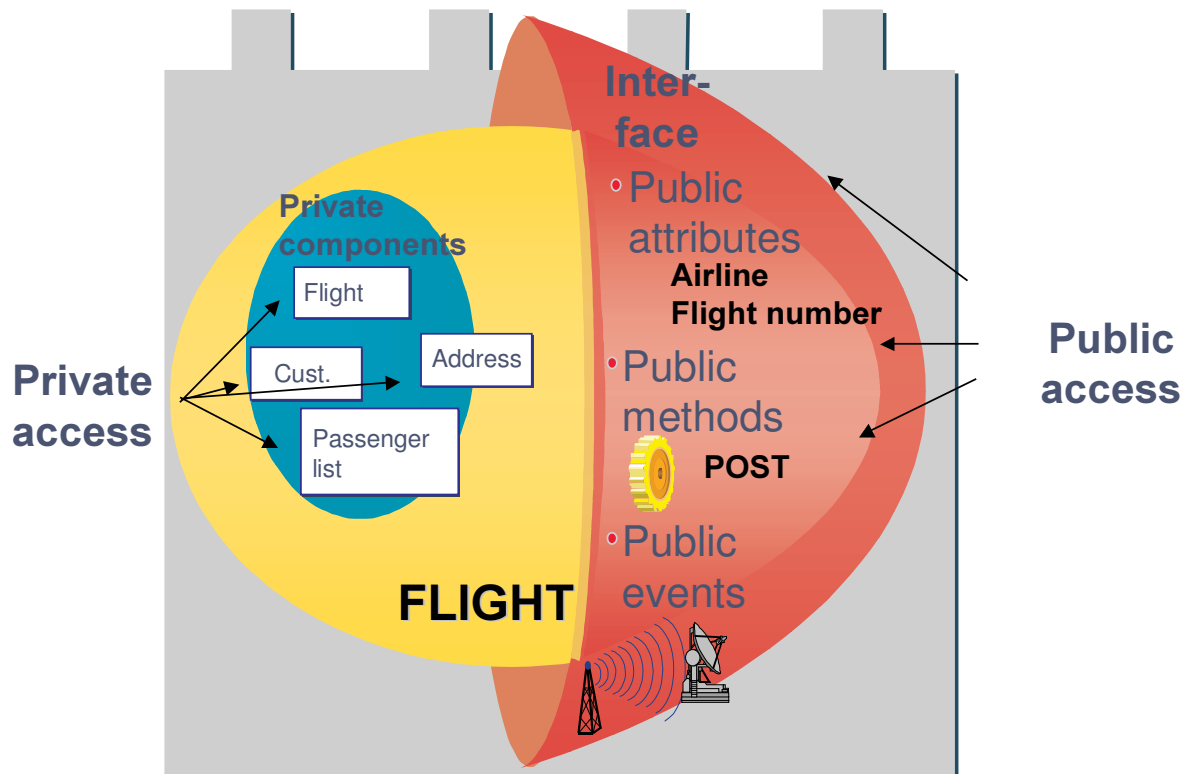
**At the conclusion of this unit, you will be able to:**

- **Explain the principles for using controls when developing user dialogs.**
- **Describe the basic conditions for using controls.**
- **Identify sources for obtaining more information.**



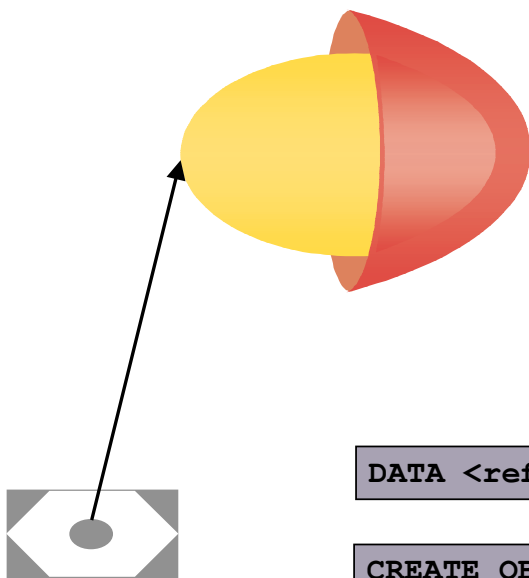
© SAP AG 1999

- In the past, information systems were primarily defined by their functions. Data and functions were kept separately and were linked with one another using input-output relationships.
- Object-oriented programming is based on abstract or concrete items that represent the real world. The state and characteristics of the objects are mapped by their inner structure and attributes (data). The behavior of the objects is described with methods (functions).
- Objects form a capsule connecting the state with the corresponding behavior. Objects should permit a one-to-one representation of the model of a problem area and a proposed solution.
- EnjoySAP Controls are special pairs of objects. They consist of a GUI object that is implemented at the front end as an ActiveX control or Java Bean and a (proxy) object at the back end. The latter is an object in your application and is created and edited using ABAP objects. The ABAP Workbench supports you here with global classes.



© SAP AG 2002

- An object basically has two layers - inside and outside:
  - **Public components:** The object components that are visible from outside, such as attributes, methods, and events. The public components can be used directly by all users. The public components of an object make up the **interface** of this object.
  - **Private components:** These components are visible only within the object. They can also be attributes, methods and events.
- The aim is to have an object itself ensure that it is consistent. For this reason, the data is normally internal, that is, it has private attributes. The internal (private) attributes of an object can be manipulated only with methods of this object (encapsulation). In general, only methods that manipulate the data and ensure that the object is consistent are offered as public components.
- An object also has a unique identification to distinguish it from other objects with the same attributes and methods.



```

CLASS <cl_name> DEFINITION.
  PUBLIC SECTION.
    . . . .
  PROTECTED SECTION.
    . . . .
  PRIVATE SECTION.
    . . . .
ENDCLASS.

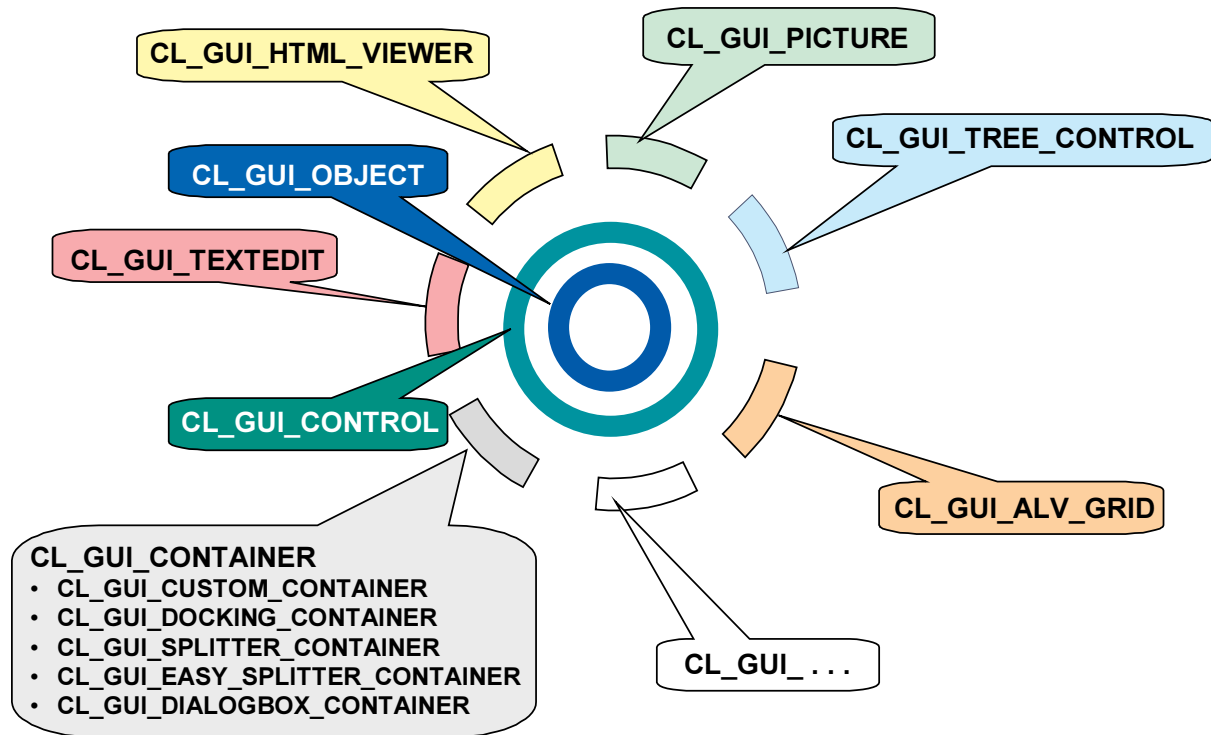
CLASS <name> IMPLEMENTATION.
  . . . .
ENDCLASS.
    
```

```
DATA <reference> TYPE REF TO <cl_name>.
```

```
CREATE OBJECT <reference> EXPORTING . . . .
```

© SAP AG 2002

- Classes describe the attributes and methods of a set of objects. There are two parts to this description: the attributes and methods are declared and then the methods are implemented.
- Each object belongs to a class. To create an object of a class, you must first declare an object reference variable (... TYPE REF TO <cl\_name>). You can then create (instantiate) an object of this class using the ABAP object keyword CREATE OBJECT <reference>.
- An ABAP program can work with any number of runtime instances of the same class. The individual runtime instances represent objects that can be uniquely identified and are addressed with the corresponding object references.
- You can call a method of an object with CALL METHOD. You must specify the name of the method and the object with whose attributes the method is executed. The syntax for this is: CALL METHOD <reference>-><method>, where <reference> is a reference variable pointing to an object and <method> is a method of the class of this object. The operator -> is called the object component selector.
- You can also call methods dynamically using parentheses in the same way as in other ABAP statements (Dynamic Invoke). In contrast to dynamic subroutine and function module calls, you can also pass the parameters and handle exceptions dynamically (see documentation on CALL METHOD).



© SAP AG 2002

- To use the Control Framework as described here, you need an R/3 System with at least Release 4.6A and a locally installed SAP GUI with Release 4.6A.
- The Control Framework provides you with a number of global classes, for example:
  - CL\_GUI\_TEXTEDIT      Text editor
  - CL\_GUI\_HTML\_VIEWER      Web browser
  - CL\_GUI\_PICTURE      Display of pictures
  - CL\_GUI\_TREE\_CONTROL      Hierarchy display in tree form
  - CL\_GUI\_ALV\_GRID      List display
  - CL\_GUI\_CFW      Central services for communicating with the GUI controls
  - CL\_GUI\_CONTAINER      Special controls used to hold additional controls



Basic terminology

Implementation of controls

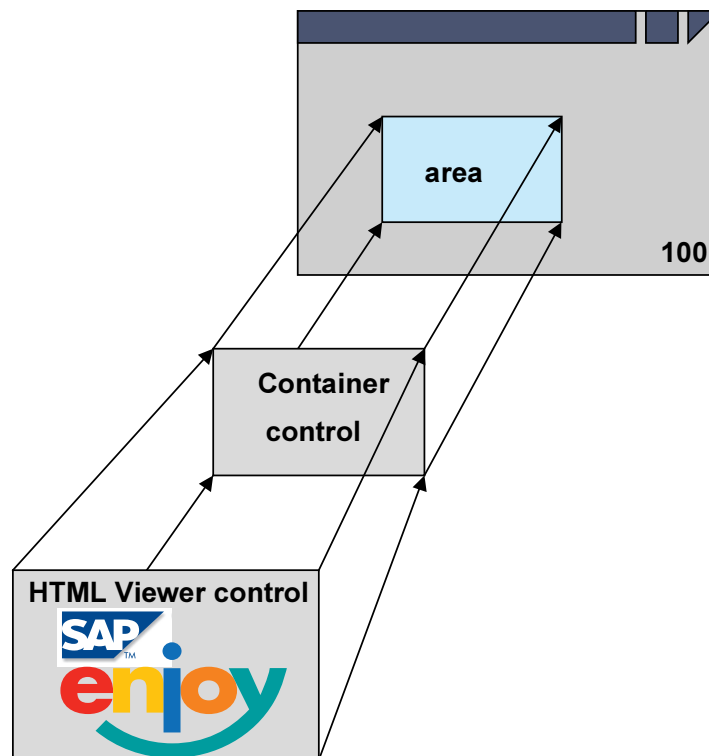
Communication with controls

Sources of information

### EnjoySAP Control: Life cycle

- **Generating the control and integrating it in a screen**
- **Using method calls to pass information between the application server and the presentation server**
- **Event handling: the program's reaction to a change in status of the control**
- **Releasing memory used by the control**

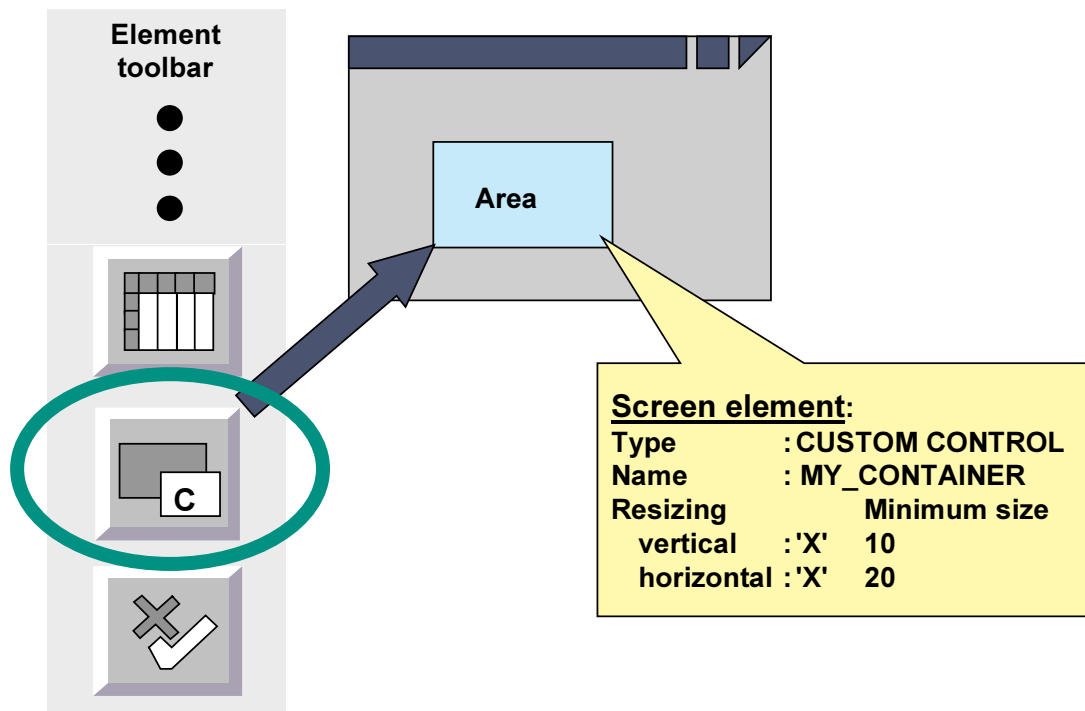
© SAP AG 2002



© SAP AG 2002

- Each EnjoySAP control resides in a container control. The container control provides space for displaying other controls on the screen. Container controls are themselves EnjoySAP controls and can therefore, be nested.
- There are different kinds of container controls. This training course will discuss only the SAP Custom Container.
- You define a customer control area on the screen for the SAP Custom Container. The SAP Custom Container is stored here at run time and then reserves space for your application control.

## Screen Painter: Layout Editor



© SAP AG 2002

- You define the custom control area on your screen with the Layout Editor in the Screen Painter.
- You assign the Custom Container area a name and maintain the corresponding static attributes. You define the size and resize parameters for the area.

```
* data declarations
```

```
DATA: container TYPE REF TO cl_gui_custom_container.
```

```
...
```

```
* PBO of screen containing MY_CONTAINER
```

```
MODULE create_objects OUTPUT.
```

```
...
```

```
* create objects only one time
```

```
CHECK container IS INITIAL.
```

```
* create container object
```

```
CREATE OBJECT container
```

```
EXPORTING container_name = 'MY_CONTAINER'.
```

```
...
```

```
ENDMODULE.
```

© SAP AG 2002

- In your ABAP program, you first define a reference variable referring to the global class `cl_gui_custom_container`.
- Next you instantiate an appropriate object for the PBO of your screen. You pass the name of the Custom Container area to the SAP Custom Container.
- Make sure that only one instance of the SAP Custom Container is created on your screen. The runtime system simply places further instances in the area and the previous control can no longer be used. All the methods applied are thus lost. In this case, only the corresponding ABAP object on the back end is removed by the Garbage Collector.

## Example HTML Viewer I



```
...
DATA: container TYPE REF TO cl_gui_custom_container,
      html_viewer TYPE REF TO cl_gui_html_viewer,
      url(255) value 'http://www.sap.com' .
```

START-OF-SELECTION.

CALL SCREEN '0100'.

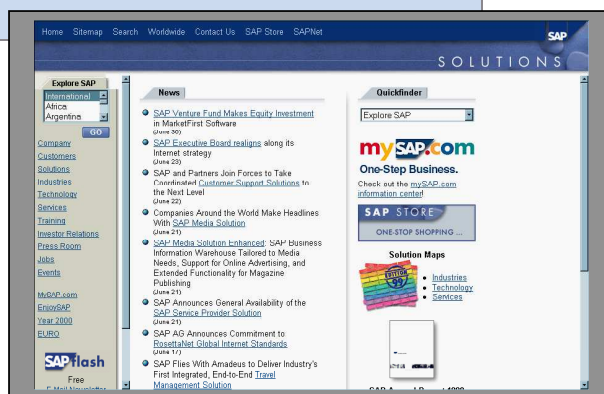
PROCESS BEFORE OUTPUT.

MODULE create\_objects.

MODULE set\_url.

PROCESS AFTER INPUT.

MODULE exit AT EXIT-COMMAND.



© SAP AG 2002

- The browser installed on the front end is used for an HTML Viewer.
- In the ABAP program, you declare a reference variable for global class `cl_gui_html_viewer`. In a PBO module, you create an appropriate instance and declare the container control (in which the SAP HTML Viewer resides) as a parameter.
- You can define a Uniform Resource Locator (URL) by calling a method in a module in the PBO event. The corresponding resource is then displayed on the screen.
- Make sure that the objects at the front end and back end release the corresponding storage area before leaving the program. Otherwise unwanted processes might remain active at the front end.
- You can use the following module, for example:

```
MODULE exit INPUT.
CALL METHOD html_viewer->free.    " destroy the GUI object
CALL METHOD container->free.
FREE html_viewer.               " destroy the ABAP object
FREE container.
LEAVE PROGRAM.
ENDMODULE.                      " EXIT INPUT
```

```

MODULE create_objects OUTPUT.
  CHECK container IS INITIAL.
  * create the container
  CREATE OBJECT container
    EXPORTING container_name = 'MY_CONTAINER'.
  CHECK html_viewer IS INITIAL.
  * create the control
  CREATE OBJECT html_viewer
    EXPORTING parent = container.
    . . . .
ENDMODULE.

```

```

MODULE set_url OUTPUT.
  * call method for setting url
  CALL METHOD html_viewer->show_url
    EXPORTING url = url.
ENDMODULE.

```

© SAP AG 2002

- You need to generate only a single instance of the HTML Viewer at run time. Previous instances will be hidden and the corresponding methods will be lost.
- You can insert method calls into your program code using the corresponding statement template or simply by using Drag&Drop.

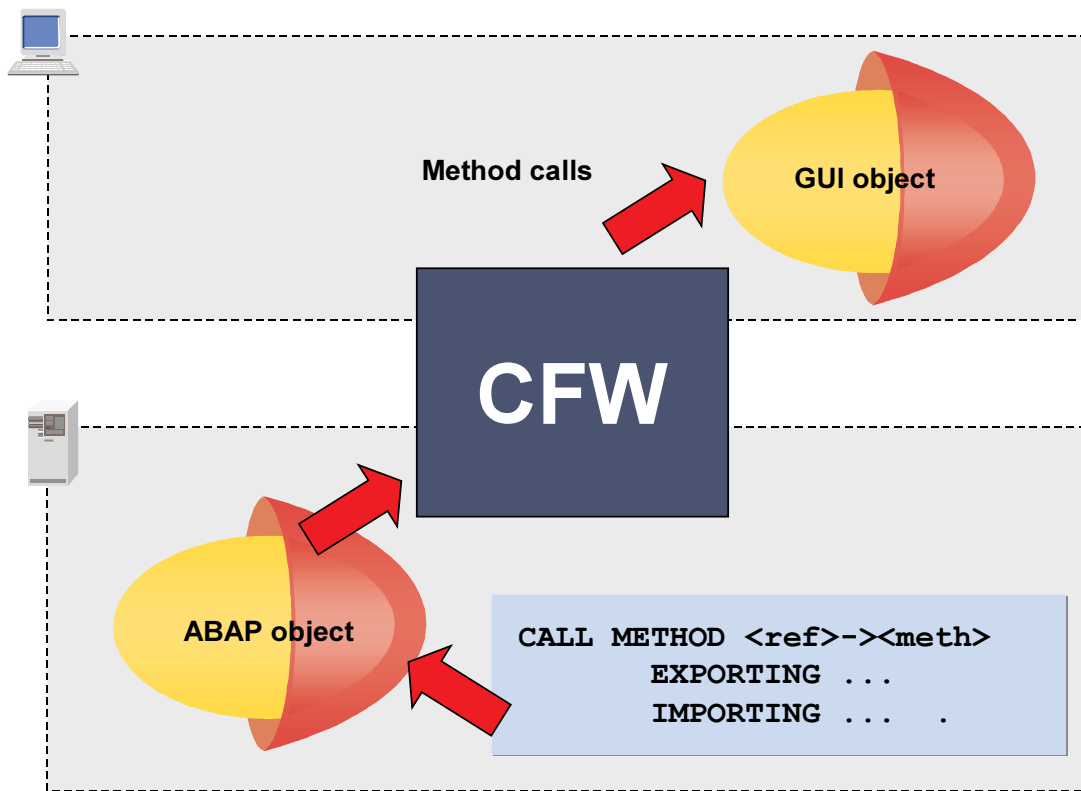
Basic terminology

Implementation of controls



Communication with controls

Sources of Information

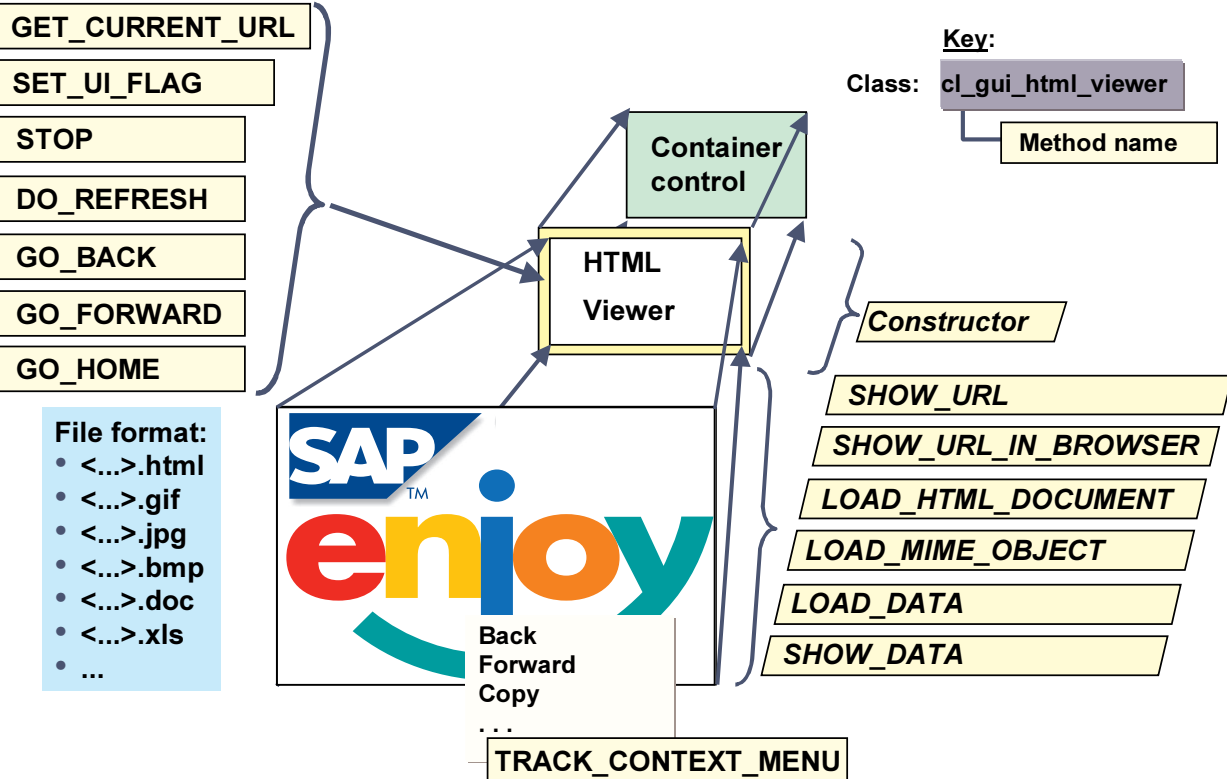


© SAP AG 1999

- If you call a method of an EnjoySAP control in your ABAP program, the corresponding ABAP object passes the call to the Control Framework.
- It in turn calls a method of the corresponding GUI object at the front end.
- The data is transported between the back end and the front end.

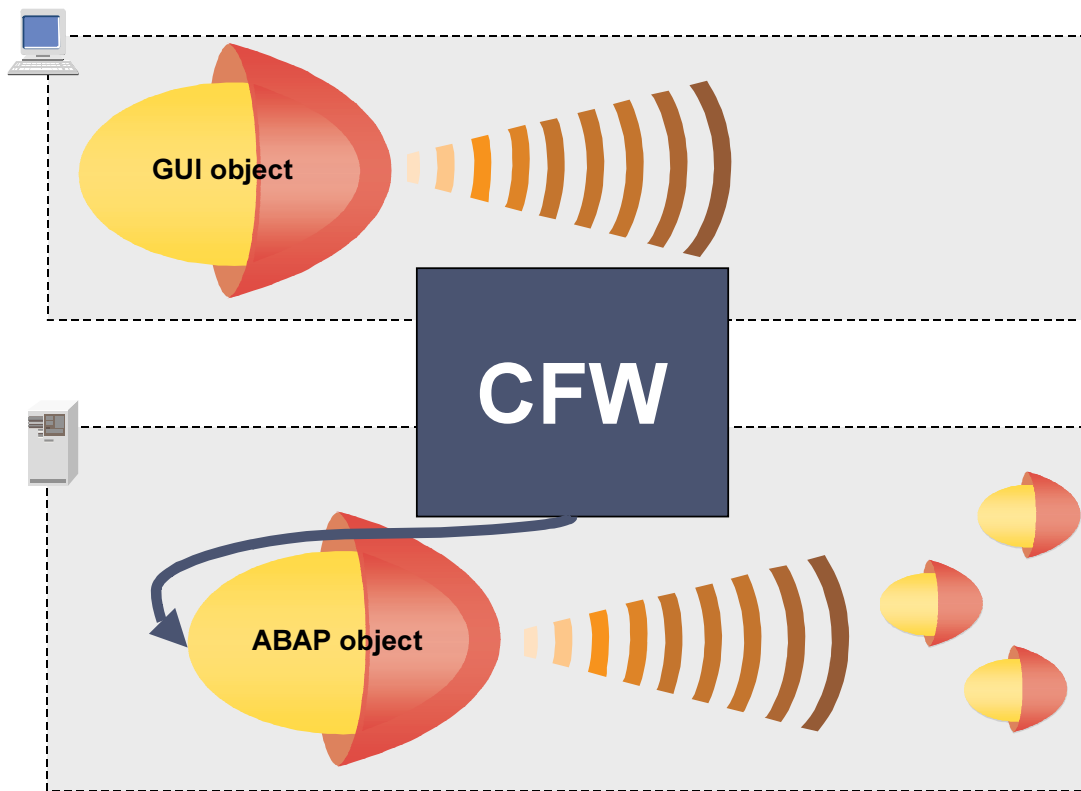
## SAP HTML Viewer: Features

SAP



© SAP AG 1999

- The SAP HTML Viewer has the following methods:
  - Service functions:
    - Initialize: constructor
    - Configure: set\_ui\_flag
    - Info: get\_current\_url
    - Context menu: track\_context\_menu
  - Navigation: stop, do\_refresh, go\_back, go\_forward, go\_home
  - Data sources:
    - External: show\_url, show\_url\_in\_browser
    - SAP Web Repository: load\_html\_document, load\_mime\_object
    - SAP Data Provider: load\_data, show\_data
- You can find further information in the online documentation about global class `cl_gui_html_viewer`.



© SAP AG 2002

- An object can declare that its state has changed by triggering an event.
- Other objects can contain handling methods that are executed when the event occurs.
- In contrast to normal method calls in which the calling program has control and knows the method called, the program triggering an event does not know what will handle this event. This is true for both the time when the event is defined and the time when the event occurs at run time.
- A class can thus define static events and an object can trigger events at run time without having to know whether and how they are used.
- If a GUI object of an EnjoySAP control triggers an event at the front end, the Control Framework makes sure that the corresponding ABAP object triggers an event that the instances of other ABAP objects can react to with their handling methods at the back end.

# Creating a Model Solution Using the EnjoySAP Controls

SAP

The screenshot displays the SAP 'Flugdaten (Pflege Flugdaten)' transaction. The left pane shows a tree view of flight dates from 18.03.2000 to 09.03.2001, with '29.04.2000' selected. The main area displays flight details for 'Qantas Airways' on '29.04.2000', including flight number '6', flight type 'A319', and a total amount of '311.152,37'. The bottom pane shows a table of 'Pflege Buchungsdaten' with columns for Buchung, Kunden, G, R, Gepäckgew, M, R, and Betrag. The table lists various bookings and their associated costs.

© SAP AG 2002

- You could have created the flight maintenance transaction using a new programming template in EnjoySAP Controls.
- An SAP Tree Control is used to display the selection of flights. The booking data is displayed in a list using an ALV Grid Control.

Basic terminology

Implementation of controls

Communication with controls



Sources of information

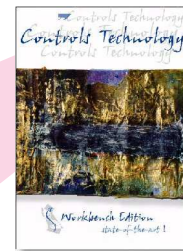
© SAP AG 2001

Online documentation

Object Navigator  
Environment → Control examples

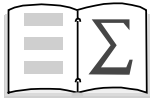
**BC412**  
**Dialog Programming**  
**Using EnjoySAP Controls**

**Workbench Edition:**  
**Controls Technology**



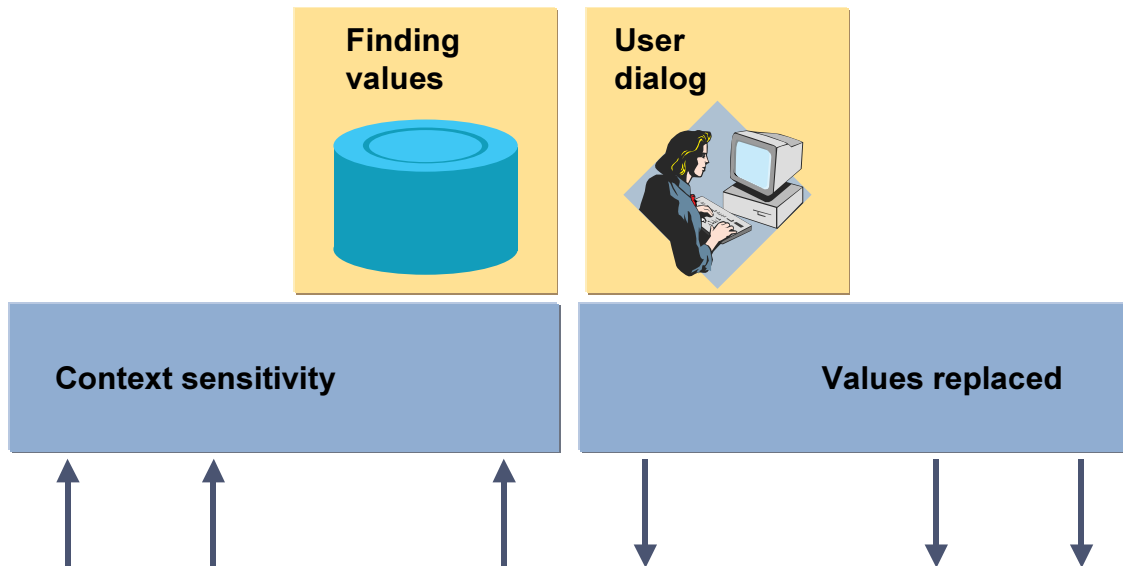
© SAP AG 2002

- Further information on developing user dialogs using EnjoySAP Controls can be found in the Object Navigator under *Environment* → *Controls examples* in the online documentation and in the SAPNet Help Portal, <http://help.sap.com/>.
- You can find a detailed description of how to implement and use the various EnjoySAP Controls in the book **ABAP Workbench Edition: Controls Technology 4.6**. The book is available from the SAPShop (order number **50032529**).



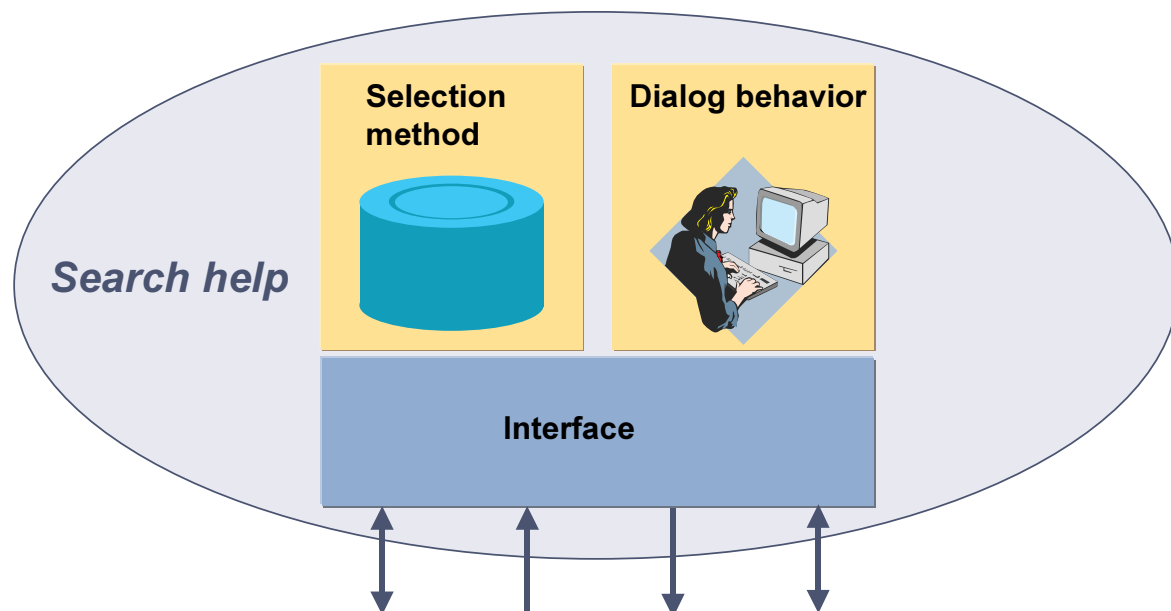
**You are now able to:**

- **Explain the principles for using controls when developing user dialogs.**
- **Describe the basic conditions for using controls.**
- **Identify sources for obtaining more information.**



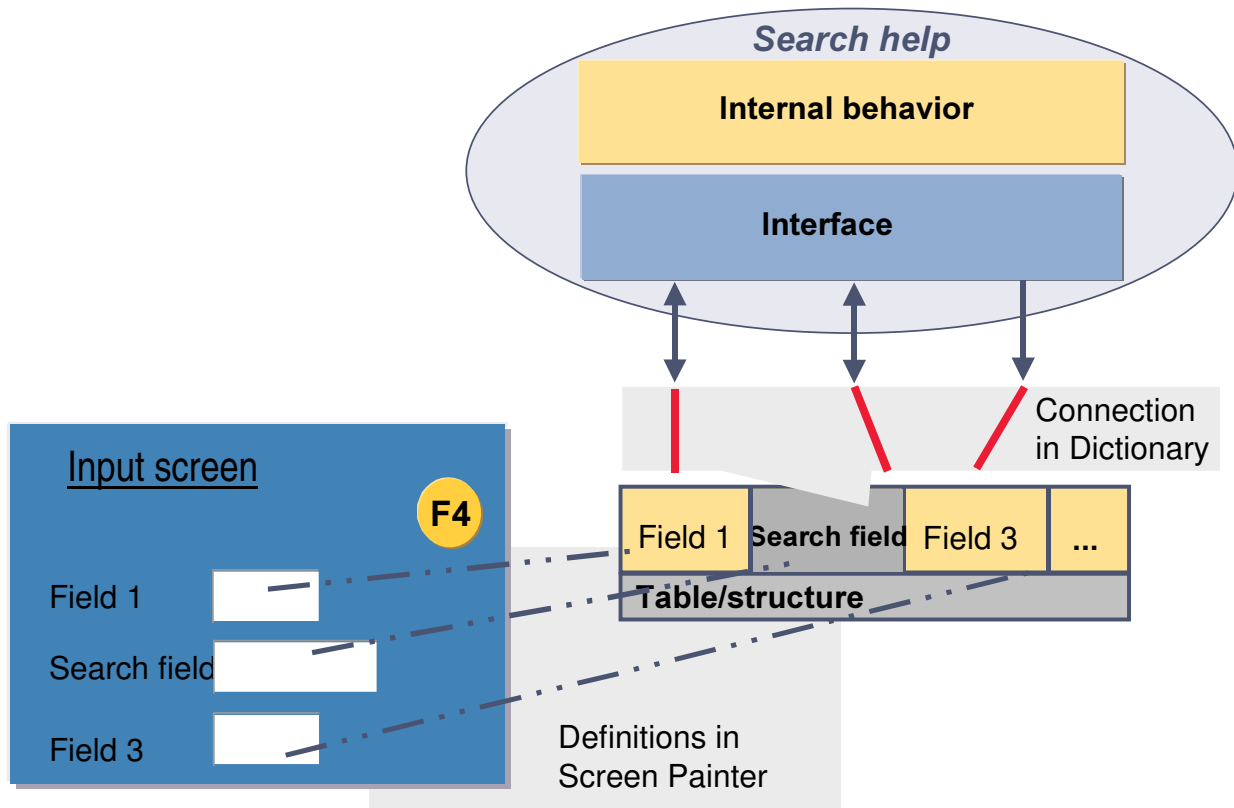
© SAP AG 2002

- Various things are required of input help for a screen field (the **search field**) as described in the following text.
- The input help must take into account information that the system already knows (the context). This includes both information that the user has entered on the current screen, and information from previous dialog steps. The input help normally uses the context to restrict the set of possible values.
- The input help must find out the values that it then presents to the user for selection. It must also determine the data that will be displayed as additional information in the list of possible values. In determining the possible values, it must take into account restrictions that arise from the context, as well as those entered by the user as specific search conditions.
- The input help must conduct a user dialog. This involves (at least) displaying the possible values with additional information, and allowing the user to choose a value from it. In many cases, the input help will also contain an input screen on which the user can specify conditions to restrict the number of possible entries displayed.
- When the user selects a value, the input help must place it in the search field. In many cases, there are extra fields on the input screen (often only output fields), containing extra information about the search field. The input help should also update the contents of these fields.



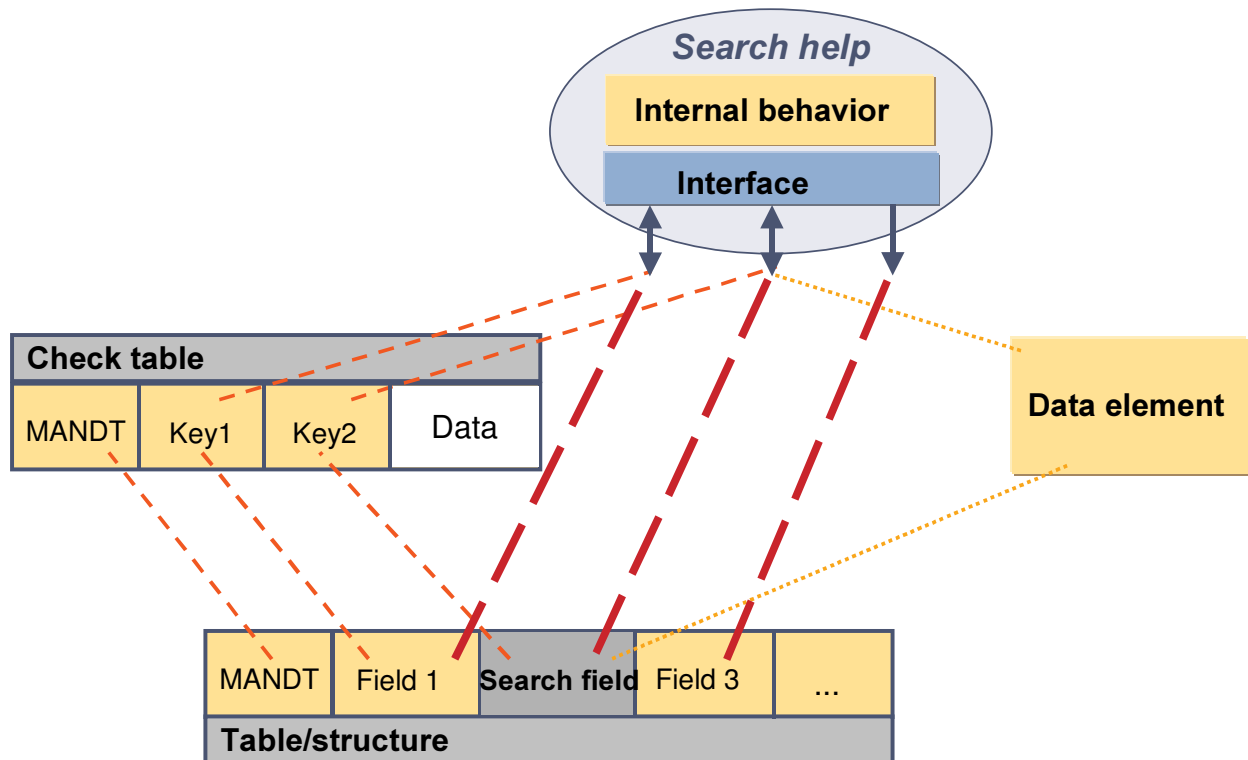
© SAP AG 2002

- The ABAP Dictionary object **search help** is a description of an input help. Its definition contains the information that the system requires to meet the user's needs.
- The **interface** of the search help controls the data that is passed between the input screen and the F4 help. The interface determines the context data that is required and the data that can be placed back on the input screen when the user chooses a value.
- The **internal behavior** of the search help describes the actual F4 process. This contains the **selection method**, which retrieves the values for display, and the **dialog behavior**, which describes the interaction with the user.
- Similarly to function modules, search helps have an interface, which defines their capacity to exchange data with other software components, and an internal behavior (which, in the case of a function module, is its source code).
- It is only worth defining a search help if there is a mechanism that allows you to address it from a screen. This mechanism is called a **search help connection**.
- Like the function module editor, the search help editor also allows you to test your objects. This allows you to check how a search help behaves before you assign it to a screen field.



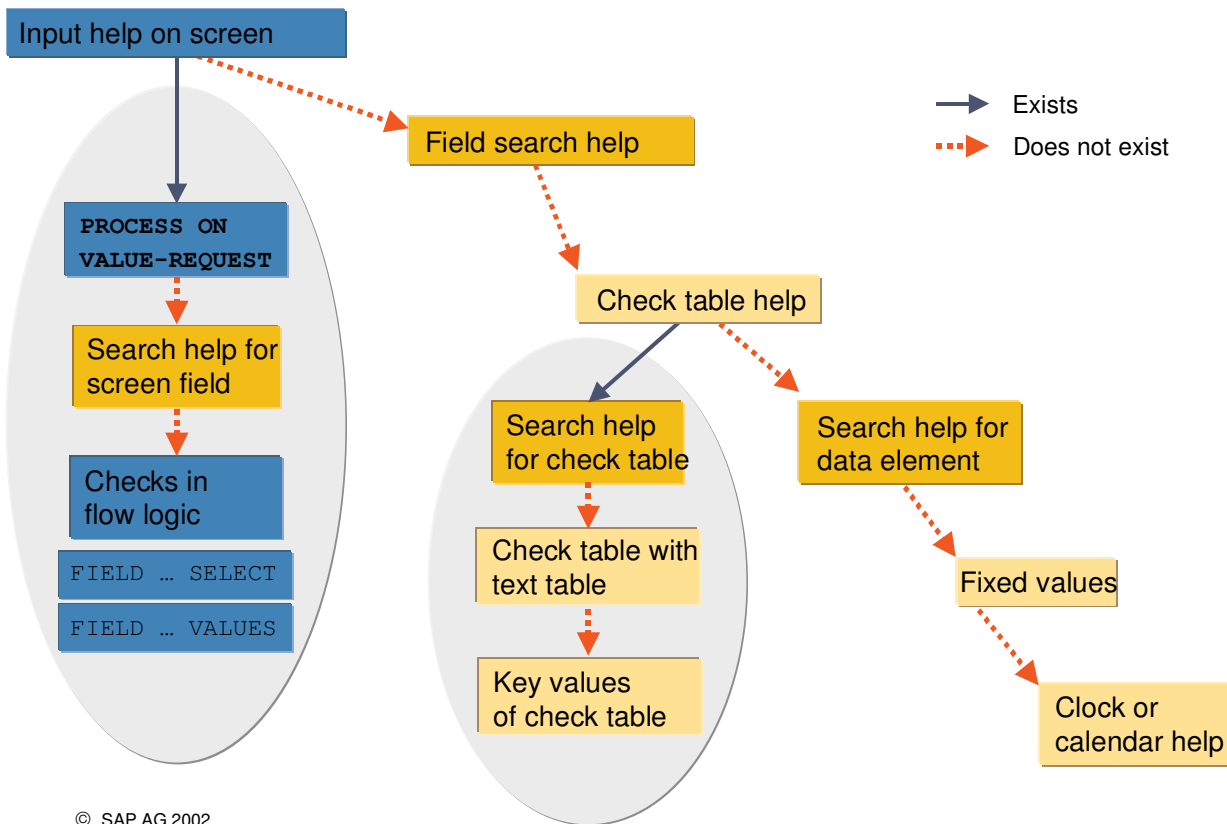
© SAP AG 2002

- A search help describes the process of an input help. In order for it to work, you need a mechanism that assigns the search help to the field. This is called the **search help connection**.
- Connecting a search help to a field affects its behavior. It is, regarded as part of the field definition.
- The semantic and technical attributes of a screen field (type, length, and F1 help) are normally not directly defined directly when you define the screen. Normally, you use a reference in the Screen Painter to an existing field in the ABAP Dictionary. The screen field then inherits the attributes of the ABAP Dictionary field.  
The same principle applies when you define input help for a screen field. The link between the search help and the search field is established using the ABAP Dictionary field, not the screen field.
- When you assign a search help, its interface parameters are assigned to the screen fields that are filled by the search help or that pass information to it from the screen. The search field must be assigned to an **EXPORT** parameter of the search help. You should also make the search field an **IMPORT** parameter so that the search help can take into account a search pattern already entered in the field by the user.
- A field can have input help even if it does not have a search help; there are other mechanisms for F4 help (for example, fixed values for a domain).



© SAP AG 2002

- You can link a search help to a field in the ABAP Dictionary in three ways: by assigning it directly to a field, by assigning it to a check table, or by assigning it to a data element.
- It can be assigned directly to a field of a structure or table. You define this link in very much the same way as you would define a foreign key. You should define the assignment here (between the interface parameters of the search help and the structure field). The system generates a proposal.
- If the field has a check table, its contents are automatically proposed as possible values in the input help. The key fields of the check table are displayed. If the check table has a text table, the first non-key character field is also displayed. If the default display is insufficient for your requirements, you can attach a search help to the check table. This search help is then used for all fields that have that check table. When you link the search help, you must define the assignment between the search help's interface and the key of the check table.
- The semantics of a field and its possible values are defined by its data element. You can therefore also link a search help to a data element. The search help is then used by all fields that are based on that data element. When you link the search help, you must specify a single EXPORT parameter, which will be used to transfer the data.
- Attaching a search help to a check table (or data element) increases its reusability, however, it does restrict your options for passing extra values to the search help interface.



- To allow as many fields as possible to carry useful input help, the R/3 System contains a wide range of mechanisms with which you can define input help. If it is possible to use more than one of these mechanisms for a particular field, the one highest in the hierarchy is used.
- In addition to defining the input help for a field in the ABAP Dictionary (as you have already seen), you can also define it in the screen field. An advantage of this method is that you cannot reuse it automatically.
- The screen event POV (PROCESS ON VALUE-REQUEST) allows you to program input help for a field yourself. You can make this help appear in standard form by using the function modules F4IF\_FIELD\_VALUE\_REQUEST or F4IF\_INT\_TABLE\_VALUE\_REQUEST. However, you should first check to see you cannot program your own input help better using a search help exit (see appendix).
- You can also attach a search help to a screen field in the Screen Painter. However, the functional scope of this technique is more restricted than attaching a search help in the ABAP Dictionary.
- You should no longer use input checks programmed directly in the flow logic (and from which input help can be derived).
- The context menu (available with right-click) for the hit list contains a *Technical info* function. This tells you which mechanism is being used in a particular case.

## Defining Tabstrip Controls on the Selection Screen

SAP

ABAP

```
SELECTION-SCREEN BEGIN OF SCREEN 101 AS SUBSCREEN.
```

```
...
```

```
SELECTION-SCREEN END OF SCREEN 101.
```

```
SELECTION-SCREEN BEGIN OF SCREEN 102 AS SUBSCREEN.
```

```
...
```

```
SELECTION-SCREEN END OF SCREEN 102.
```

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

```
SELECTION-SCREEN TAB (length) tabname1 USER-COMMAND ucomm1 DEFAULT SCREEN 101.
```

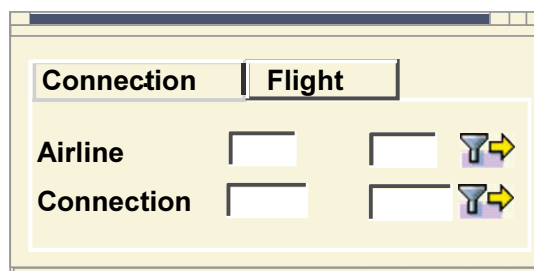
```
SELECTION-SCREEN TAB (length) tabname2 USER-COMMAND ucomm2 DEFAULT SCREEN 102.
```

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

```
INITIALIZATION.
```

```
  tabname1 = TEXT-001.          "TEXT-001 EN: Connection
```

```
  tabname2 = TEXT-002.          "TEXT-002 EN: Flight
```



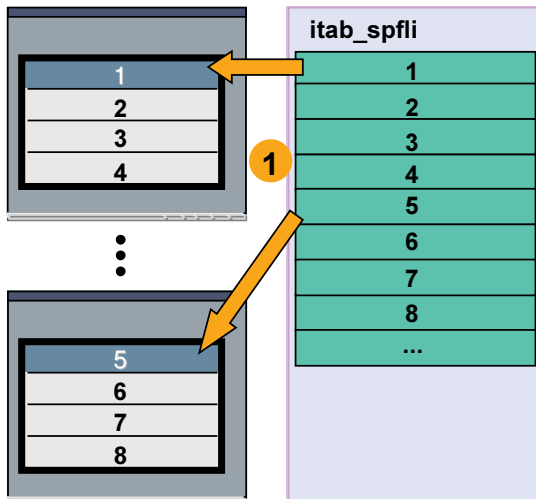
© SAP AG 2002

- You define a subscreen for a tabstrip control on a selection screen as follows:  
SELECTION-SCREEN BEGIN OF TABBED BLOCK <blockname> FOR <n> LINES.  
SELECTION-SCREEN END OF BLOCK <blockname>.  
The size of the subscreen area in lines is defined by <n>.
- The system automatically generates a CONTROLS statement: CONTROLS:  
TABSTRIP\_BLOCKNAME TYPE TABSTRIP. Do not write your own CONTROLS statement. If you try to do so, a syntax error results.
- You define the individual tab pages as follows:  
SELECTION-SCREEN TAB (length) <name> USER-COMMAND <ucomm> [DEFAULT  
[PROGRAM <prog>/SCREEN <dynnr>]].  
Optional additions:  
[DEFAULT [PROGRAM <prog>/SCREEN <dynnr>]].  
If you use the DEFAULT addition, you must also use the SCREEN addition. The PROGRAM addition is optional. You need it only if the screen comes from another program.
- You can delay specifying the link between the tab title and the selection screen until run time. You can also change an existing assignment at run time. To do this, fill the blockname structure. This is created automatically for every tabstrip block. The structure has the same name as the tabstrip block and contains the PROG, DYNNR, and ACTIVETAB fields. For further information, refer to the online documentation **SUB-2**.

## Table Controls: Scrolling Page by Page (Example)

SAP

1 my\_control-top\_line



PROCESS BEFORE OUTPUT.

```
LOOP...
  MODULE get_looplevels.
ENDLOOP.
```

Screen  
Painter

DATA: looplines LIKE sy-loopc.

```
...
MODULE get_looplevels OUTPUT.
  looplines = sy-loopc.
ENDMODULE.
```

```
MODULE user_command_0200 INPUT.
CASE ok_code.
```

```
...
WHEN 'P++' OR 'P+' OR 'P-' OR 'P--'.
  CALL FUNCTION 'SCROLLING_IN_TABLE'
```

EXPORTING

```
  entry_act = my_control-top_line
  entry_to  = my_control-lines
  loops    = looplines
  ok_code   = ok_code
```

IMPORTING

```
  entry_new = my_control-top_line.
```

```
...
ENDCASE.
ENDMODULE.
```

ABAP

© SAP AG 2002

- You can scroll a page at a time in a table control using the table control attribute, <ctrl>-top\_line.
- In the PAI processing block, you need to know the current number of lines in the corresponding table control.
- The sy-loopc system field contains the number of table control lines in the PBO processing block. However, in the PAI, it contains the number of filled lines.
- SY-LOOPC is filled only between LOOP and ENDLOOP, since it always refers to the current loop.
- Use the SCROLLING\_IN\_TABLE function module for scrolling.



## Documentation References

| Ref.         | Path in Documentation                                                                                                                                                                      |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GUI-1</b> | In Menu Painter: <i>Goto</i> → <i>Interface objects</i> ;<br><i>Function key settings</i> → <name> → <i>Pushbutton settings</i> ;<br><i>Interface</i> → <i>Subobject</i> → <i>Create</i>   |
| <b>GUI-2</b> | <i>SAP Library</i> → <i>Basis</i> → <i>ABAP Workbench</i> → <i>BC ABAP Workbench Tools</i> → <i>ABAP Workbench: Tools</i> → <i>Menu Painter</i> → <i>Functions</i>                         |
| <b>GUI-3</b> | In Menu Painter: <i>Utilities</i> → <i>Help texts</i> → <i>Internal key number</i>                                                                                                         |
| <b>GUI-4</b> | In Menu Painter: <i>Utilities</i> → <i>Help texts</i> → <i>Standards/proposals</i>                                                                                                         |
| <b>ILS-1</b> | In ABAP Editor: <i>Utilities</i> → <i>Help on...</i> ; <i>ABAP term: READ</i>                                                                                                              |
| <b>ILS-2</b> | In ABAP Editor: <i>Utilities</i> → <i>Help on...</i> ; <i>ABAP term: MODIFY</i>                                                                                                            |
| <b>ILS-3</b> | In ABAP Editor: <i>Utilities</i> → <i>Help on...</i> ; <i>ABAP term: GET CURSOR</i>                                                                                                        |
| <b>DIA-1</b> | <i>SAP Library</i> → <i>Getting Started with the SAP System</i> → <i>Layout Menu</i>                                                                                                       |
| <b>DIA-2</b> | <i>SAP Library</i> → <i>Basis</i> → <i>ABAP Workbench</i> → <i>BC ABAP Workbench Tools</i> → <i>ABAP Workbench: Tools</i> → <i>Screen Painter</i> → <i>Defining the Element Attributes</i> |
| <b>DIA-3</b> | <i>SAP Library</i> → <i>Basis</i> → <i>ABAP Workbench</i> → <i>BC ABAP Workbench Tools</i> → <i>ABAP Workbench: Tools</i> → <i>Screen Painter</i> → <i>Creating Screens</i>                |

|              |                                                                                                                                                                                                                                              |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>OUT-1</b> | In Screen Painter: <i>Goto</i> → <i>Translation</i>                                                                                                                                                                                          |
| <b>OUT-2</b> | <i>SAP Library</i> → <i>Basis</i> → <i>ABAP Workbench</i> → <i>BC - SAP Style Guide</i> → <i>R/3 Icons and symbols</i> → <i>Icons</i> → <i>Icons as status displays</i> .                                                                    |
| <b>INP-1</b> | <i>SAP Library</i> → <i>Basis</i> → <i>ABAP Workbench</i> → <i>BC – SAP Style Guide</i> → <i>Interface elements</i> → <i>Input/output fields</i>                                                                                             |
| <b>INP-2</b> | <i>SAP Library</i> → <i>Basis</i> → <i>ABAP Workbench</i> → <i>BC ABAP Workbench Tools</i> → <i>ABAP Workbench: Tools</i> → <i>Screen Painter</i> → <i>Defining the Element Attributes</i> → <i>Choosing Field Formats</i>                   |
| <b>INP-3</b> | <i>SAP Library</i> → <i>Basis</i> → <i>ABAP Workbench</i> → <i>BC - SAP Style Guide</i> → <i>Functions – General Guidelines and Overview</i> → <i>Navigation Functions - Overview</i> → <i>Overview of Navigation Options</i>                |
| <b>INP-4</b> | <i>SAP Library</i> → <i>Basis</i> → <i>ABAP Workbench</i> → <i>BC - SAP Style Guide</i> → <i>Functions – General guidelines</i> → <i>Navigation Functions – Overview</i> → <i>Comparison of Exit, Back, and Cancel</i> .                     |
| <b>SUB-1</b> | <i>SAP Library</i> → <i>Basis</i> → <i>BC – ABAP Programming and Runtime Environment</i> → <i>BC - ABAP Programming</i> → <i>ABAP User Dialogs</i> → <i>Screens</i> → <i>Complex Screen Elements</i> → <i>Tabstrip Controls</i>              |
| <b>SUB-2</b> | <i>SAP Library</i> → <i>Basis</i> → <i>BC – ABAP Programming and Runtime Environment</i> → <i>BC - ABAP Programming</i> → <i>ABAP User Dialogs</i> → <i>Selection Screens</i> → <i>Subscreens and Tabstrip Controls on Selection Screens</i> |
| <b>TAB-1</b> | <i>SAP Library</i> → <i>Basis</i> → <i>BC – ABAP Programming and Runtime Environment</i> → <i>BC - ABAP Programming</i> → <i>ABAP User Dialogs</i> → <i>Screens</i> → <i>Complex Screen Elements</i> → <i>Table Controls</i>                 |