BC410

Programming User Dialogs with Classical Screens (Dynpros)

Date _____

Training Center _____

Instructors _____

_____

Education Website _____

Participant Handbook

Course Version: 63
Course Duration: 3 Day(s)
Material Number: 50085480

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

Internal Use SAP Partner Only

Copyright

Trademarks

Disclaimer

About This Handbook

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

Typographic Conventions

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

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

Internal Use SAP Partner Only

Icons in Body Text

The following icons are used in this handbook.

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

Contents

Internal Use SAP Partner Only

Course Overview

This course prepares you to program dynamic screen processing and user dialogs using the different screen elements in the SAP System.

Target Audience

This course is intended for the following audiences:

- Programmers
- Consultants

Course Prerequisites

Required Knowledge

- SAPTEC Basis Technology
- BC400 ABAP Workbench Concepts and Tools

Recommended Knowledge

- BC430 ABAP Dictionary
- Programming experience with ABAP

Course Goals

This course will prepare you to:

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

Course Objectives

After completing this course, you will be able to:

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

SAP Software Component Information

The information in this course pertains to the following SAP Software Components and releases: SAP NetWeaver 7.0.

Unit 1

Introduction to Screen Programming

Unit Overview

This unit describes the principles of screen programming. You will learn to create screens and add ABAP Dictionary screen elements. The unit also presents details on how to modify attributes dynamically, initialize SCREEN system table, and insert screen sequences for complex transactions.

Unit Objectives

After completing this unit, you will be able to:

- Describe the Single-transaction programming model
- organize your program source code with includes
- Create and process screens
- Add ABAP Dictionary screen elements
- Explain PBO and PAI processing
- Make dynamic screen modifications
- Insert screen sequences

Unit Contents

Lesson: General aspects

Lesson Overview

In this lesson you will hear about the term User Dialog, you will learn that the current SAP programming model is the single-screen transaction, you will learn what kind of program types there are and how to organize your porogram with includes.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the Single-transaction programming model
- organize your program source code with includes

Business Example

You want to program user dialogs for the SAP GUI.

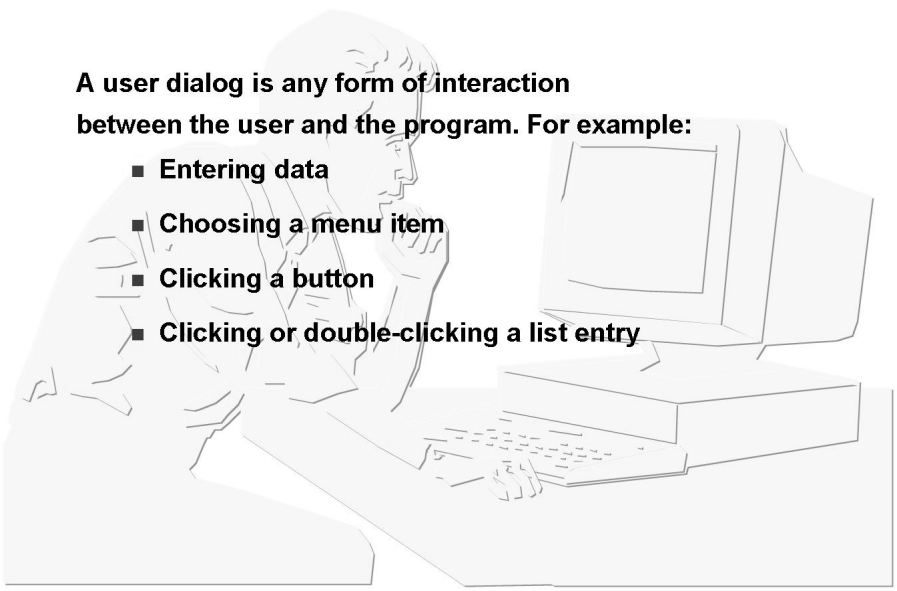


Figure 1: User Dialogs

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

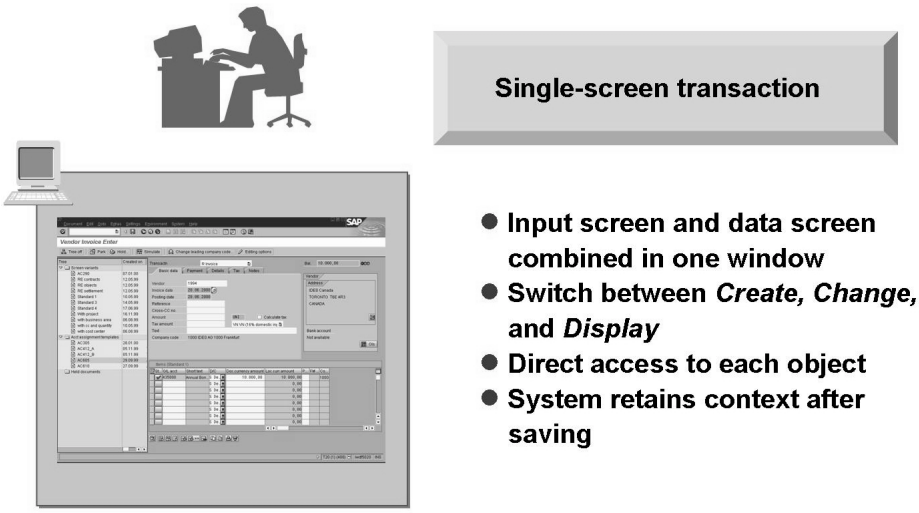


Figure 2: SAP Programming Model

The aim of the current 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 SAP 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.

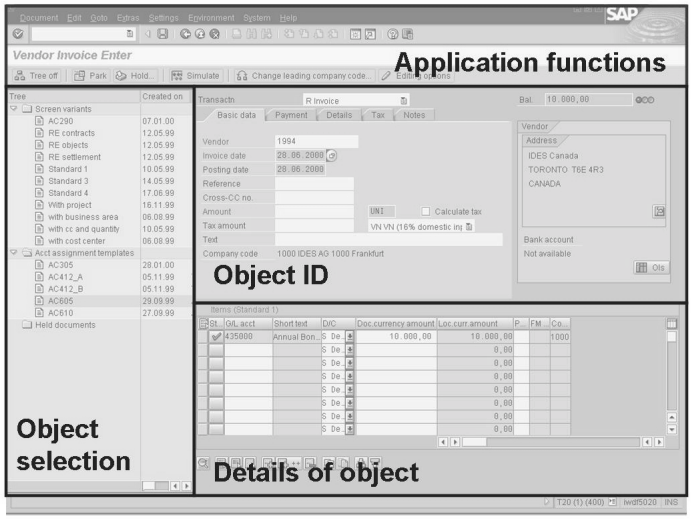


Figure 3: Example Screen Layout: Transaction FB60

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.

In the course of this class, you will create a program according to the single-screen transacion paradigm. In particular, you will make use of elements like radiobuttons and tabstrip pages.

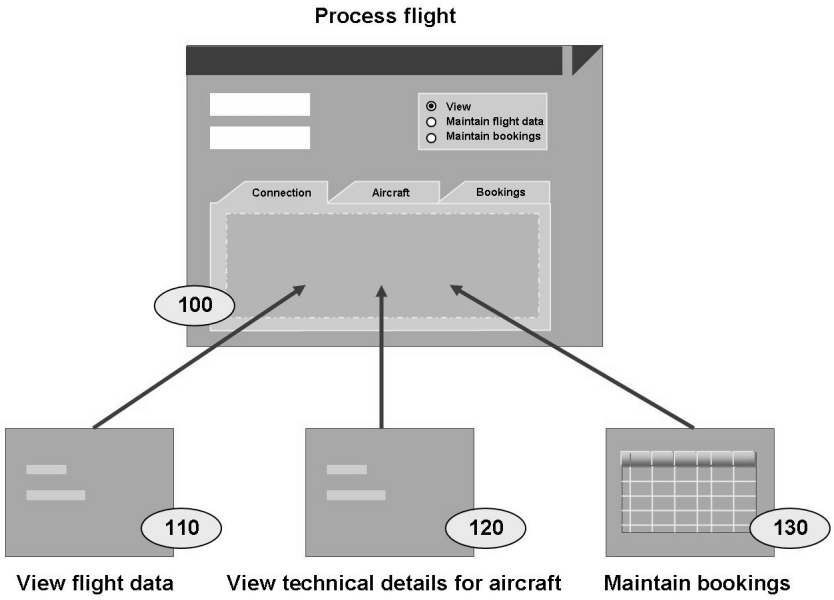


Figure 4: Exercise: Screen Programming

ABAP has several program types. Two of them represent programs that are complete: Types 1 and M can be executed. The other types are incomplete, i.e .they can be used within other programs, but they cannot be executed as such.

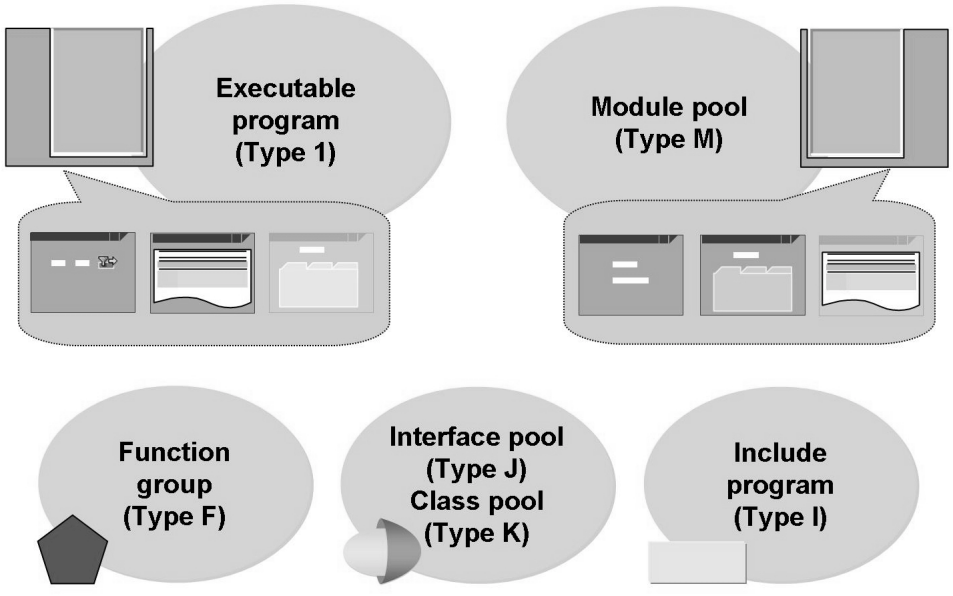


Figure 5: ABAP Program Types

Executable program (type 1)

Executable programs can be run via *System* → *Services* → *Reporting*or directly from the ABAP Editor. A set of processing blocks is processed in a predefined order. You can use a standard selection screen. Traditionally, type 1 programs create and display lists. In modern reports, however, the SAP List Viewer (ALV) is used.

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 modularization units, 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.

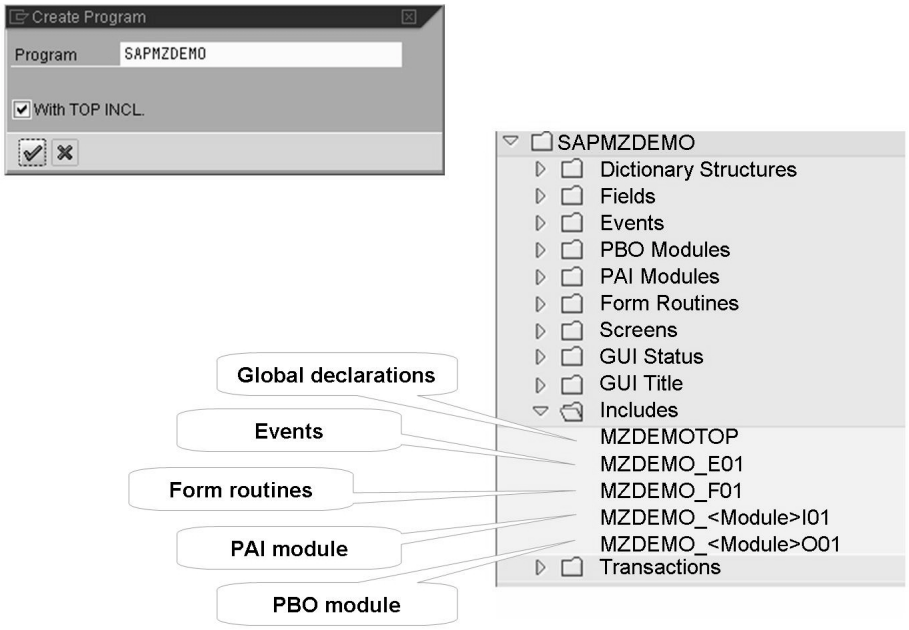


Figure 6: Program Organization

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.

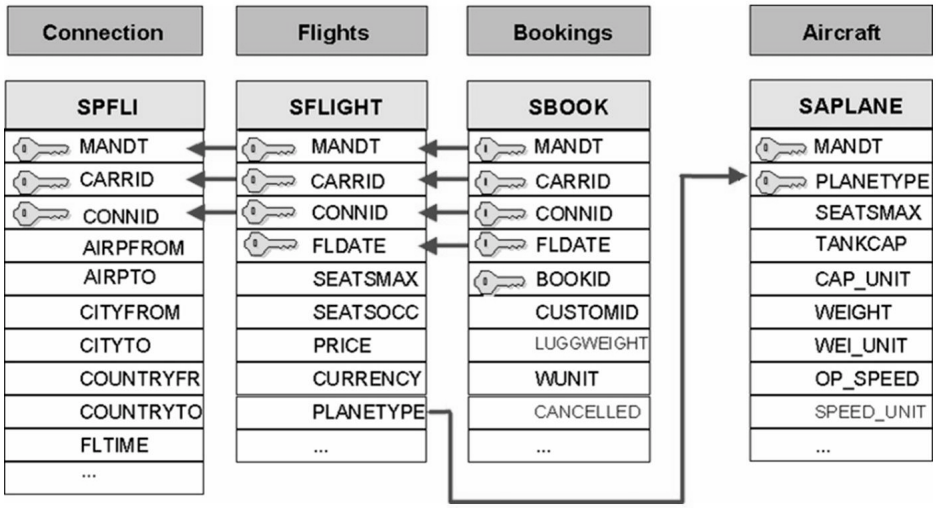


Figure 7: Tables of the Flight Data Model (BC410)

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

SPFLI

CARRID	Airline
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 (Economy class)
PRICE	Basic flight price
CURRENCY	Currency

SBOOK

CARRID, CONNID, FLDATE	See SFLIGHT
BOOKID	Booking number
CUSTOMID	Customer number

SAPLANE

PLANETYPE	Plane type
SEATSMAX	Maximum capacity

Internal Use SAP Partner Only

Lesson Summary

You should now be able to:

- Describe the Single-transaction programming model
- organize your program source code with includes

Internal Use SAP Partner Only

Lesson: Principles of Screen Programming

Lesson Overview

This lesson will help you understand the principles of screen programming. You will also learn how to create screens and to add ABAP Dictionary screen elements. Finally, you will learn about PBO and PAI processing.

Lesson Objectives

After completing this lesson, you will be able to:

- Create and process screens
- Add ABAP Dictionary screen elements
- Explain PBO and PAI processing

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. Assume that you are a manager in the travel agency. You need to perform these tasks.

Introducing Screens

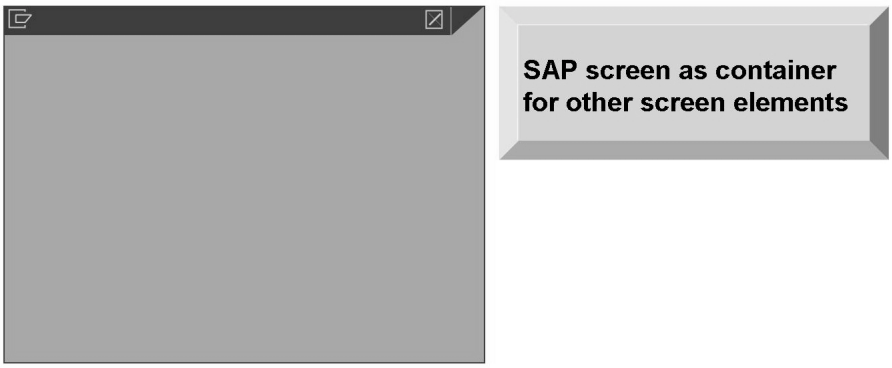


Figure 8: Screens

Screens are freely definable objects that you can use to display or enter information through input and output fields, lists, and so on.

Screens are a form of dialog between the user and the ABAP program.

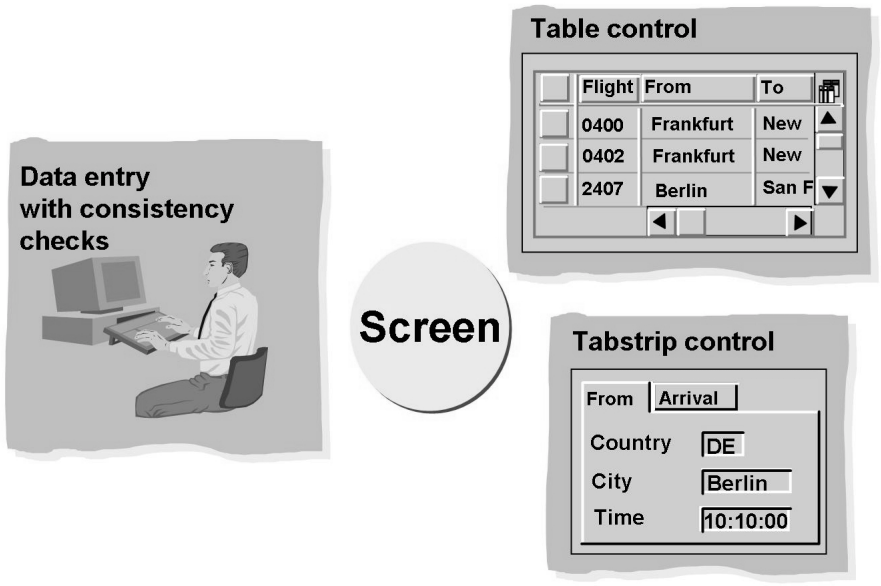


Figure 9: Strengths of Screens

Screens allow you to enter and display data.

You can use screens in combination 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.

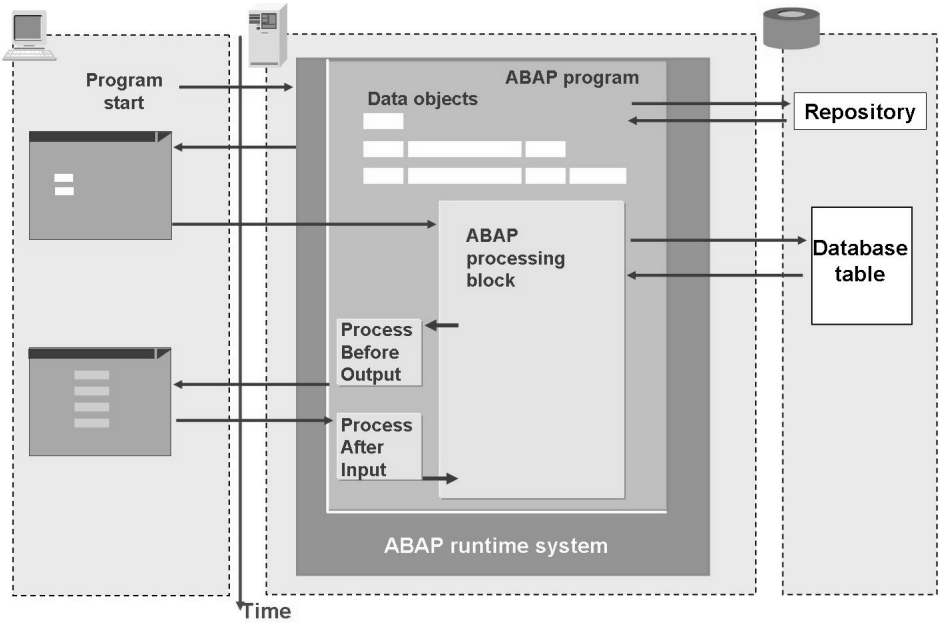


Figure 10: Screens in Dialog Programs

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 push-button, returns control to the runtime system. The screen fields are then transported into the structure that serves as the interface between screen and program. 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.

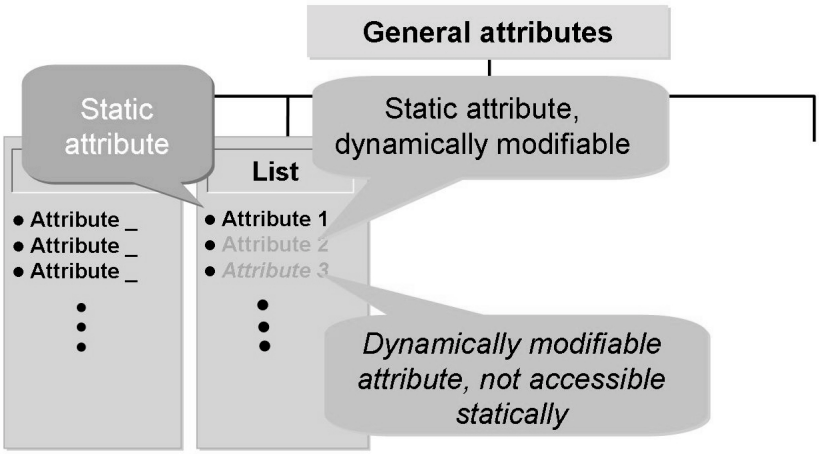


Figure 11: Attributes of Screen Elements (Key)

The screen elements such as *text field*, *input/output field*, *status icon*, *group box*, *radio button*, *checkbox*, and *pushbutton* all have general, Dictionary, program, 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 be set only dynamically**

For complete documentation of the attributes of screen elements, see the online documentation (*SAP Library → SAP NetWeaver Components → ABAP Workbench → ABAP Workbench Tools → Screen Painter → Defining the Element Attributes*).

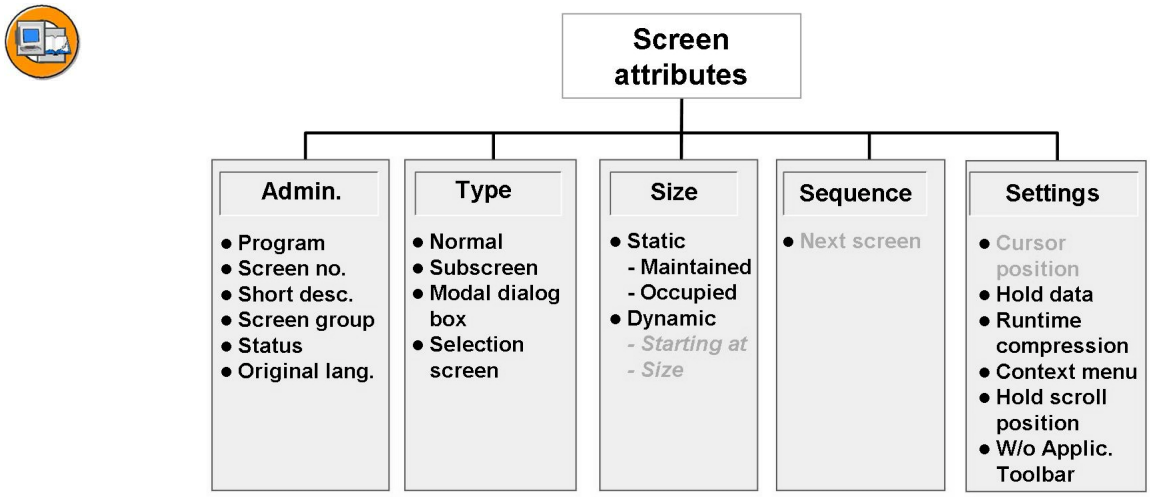


Figure 12: Screens: Attributes

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 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 the 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 the screen type.

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 (*SAP Library → SAP NetWeaver Components → ABAP Workbench → ABAP Workbench Tools → Screen Painter → Creating Screens*).

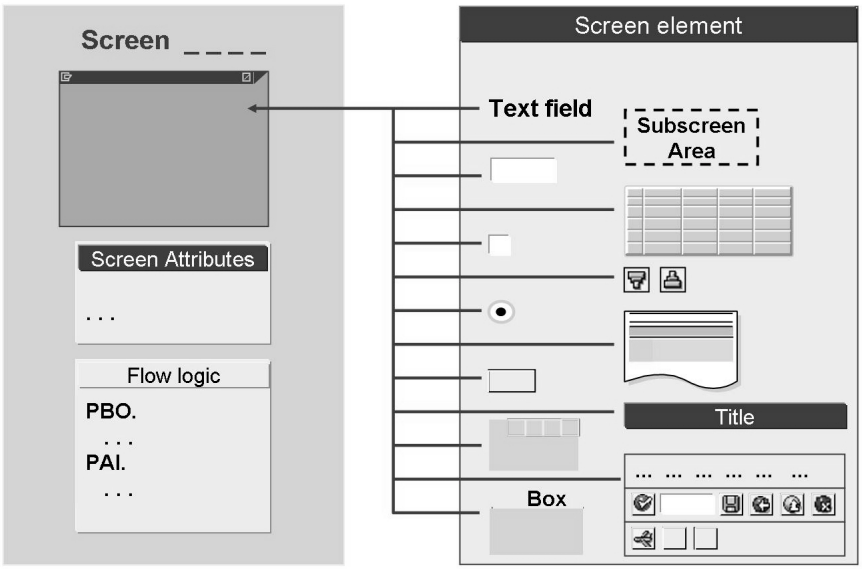


Figure 13: Defining and Managing Screens

A screen consists of a screen image and its flow logic. Flow logic is a program that controls the way the screen image is processed. For 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.

Creating Screens

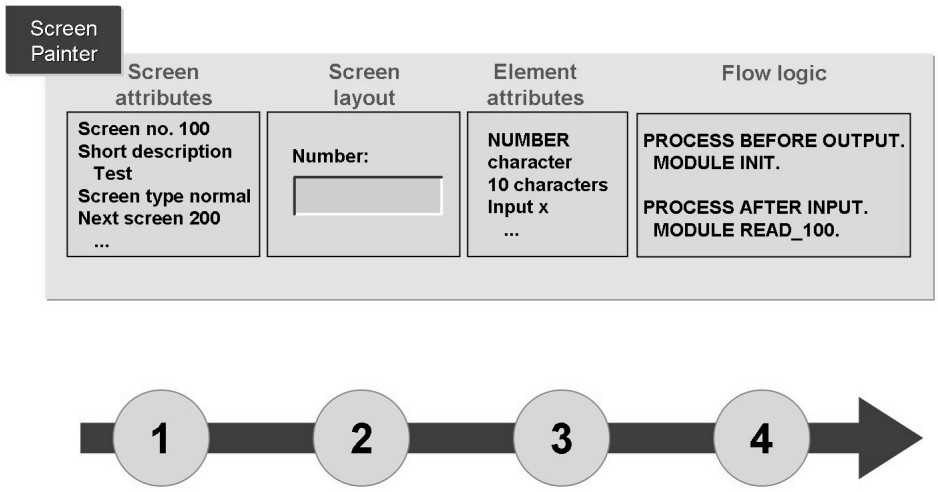


Figure 14: Creating Screens

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.

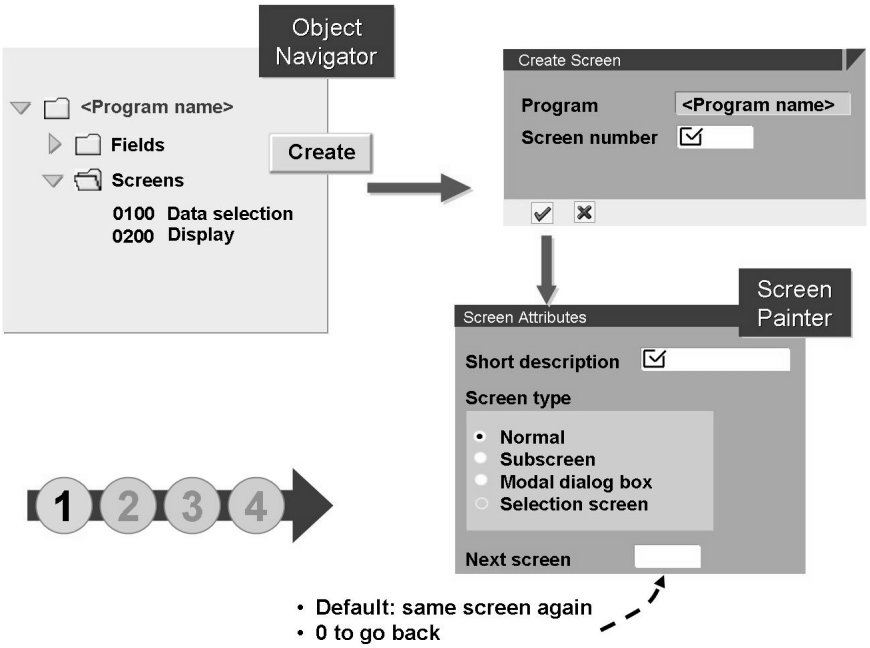


Figure 15: Creating Screens: Screen Attributes

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 statement in the ABAP Editor and then double-clicking the screen number.

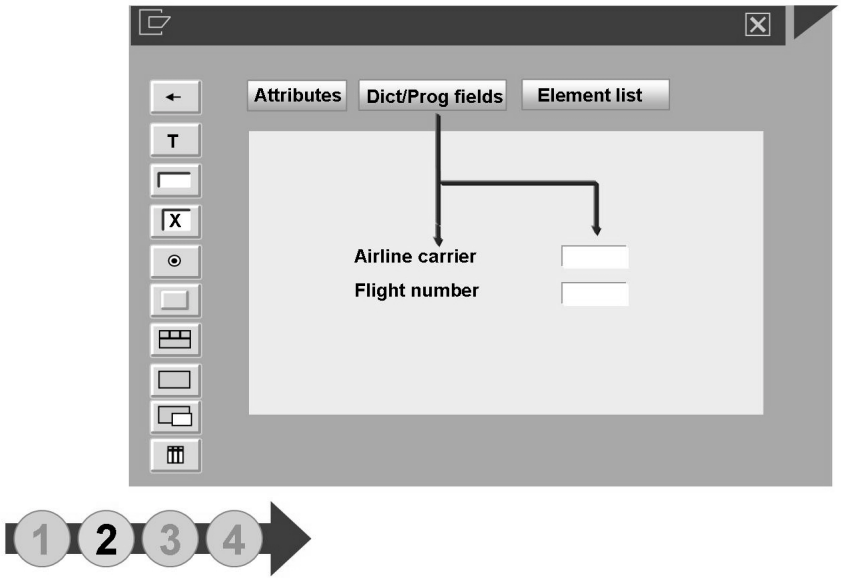


Figure 16: Creating Screens: Layout

You 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 UNIX and Windows NT, Windows 95 or a higher Windows version.

If you use a different operating system, you must use the alphanumeric Screen Painter.

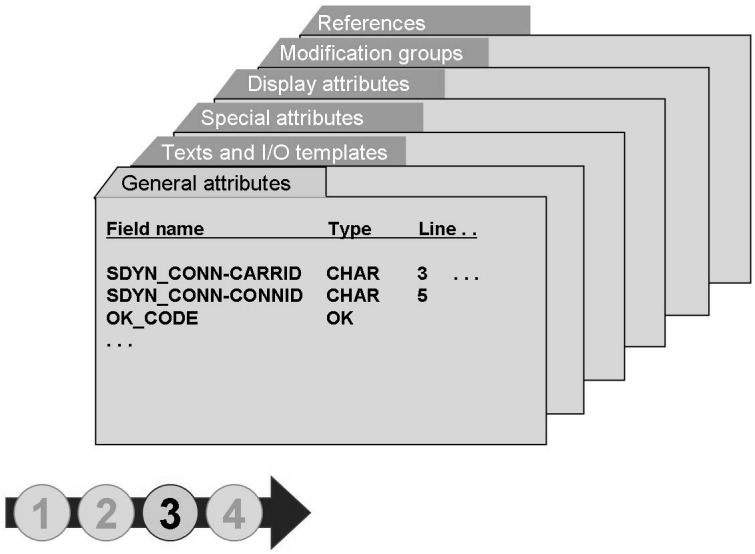


Figure 17: Creating Screens: The Element List

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 or 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 statement**SELECT - Assigment rules for individual columns**. For example:

ABAP Dictionary Data Type	ABAP Data Type
CHAR	C
NUMC	N

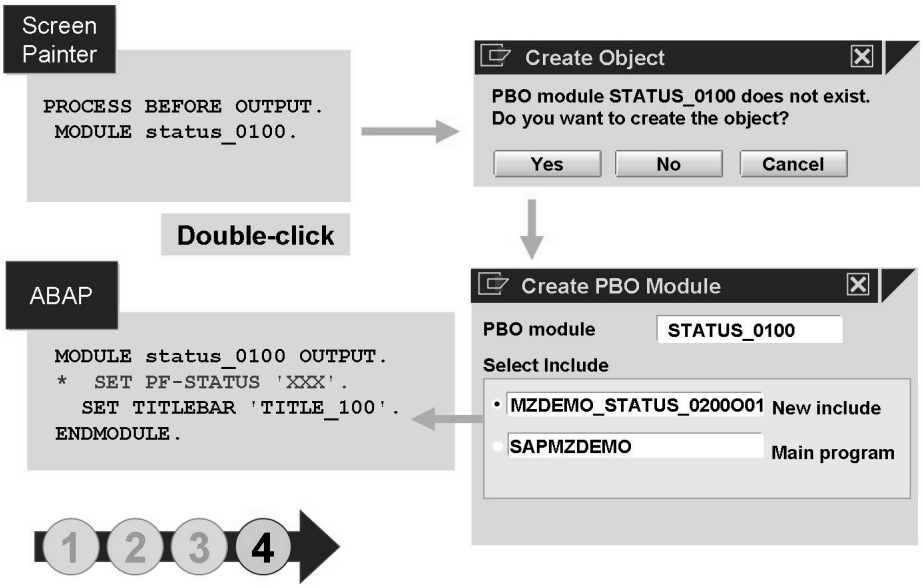


Figure 18: Creating Screens: Flow Logic

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. From within the Screen Painter, start help (**Ctrl-F8**, then choose *Keyword overview*.

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.

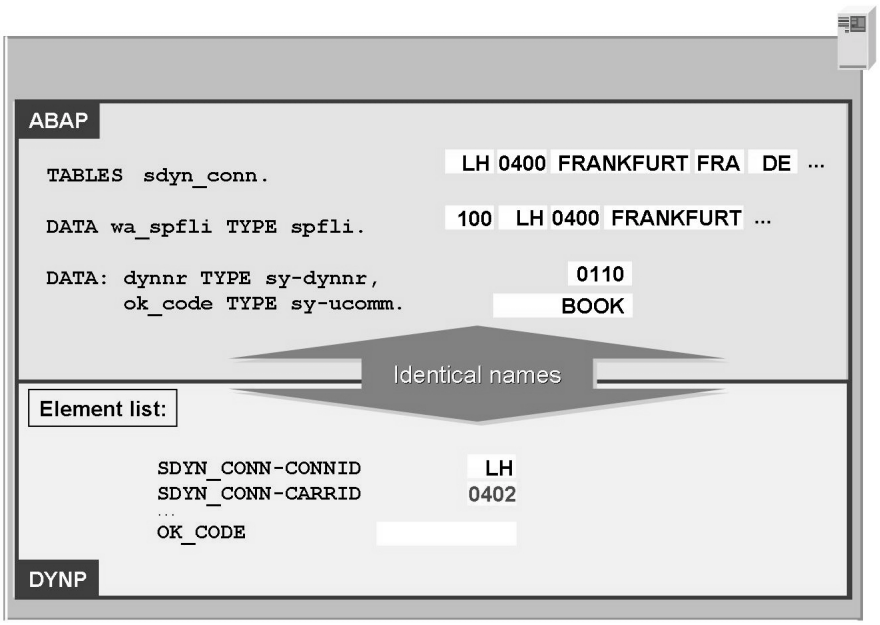


Figure 19: Data Visibility

There are two 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, two 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.

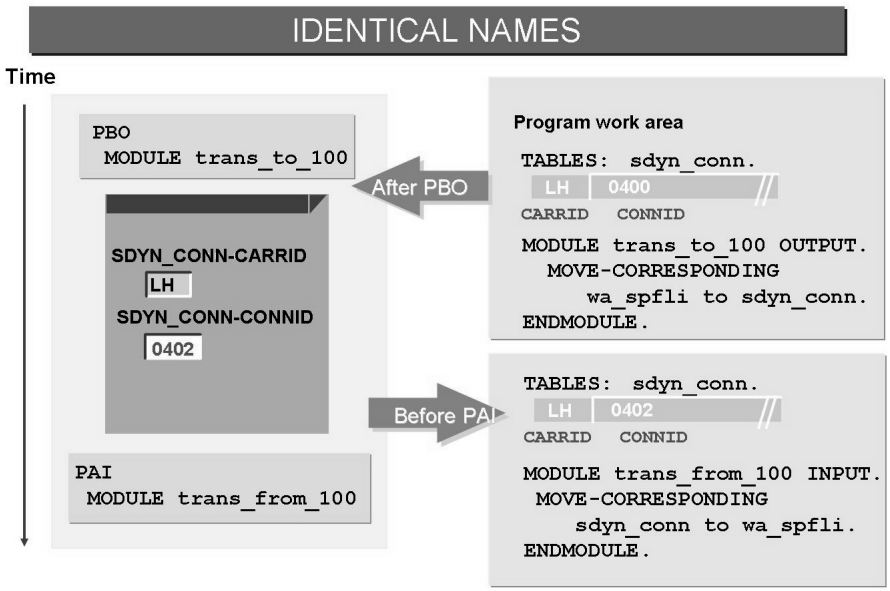


Figure 20: Data Exchange: Screens - ABAP Programs

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 the system 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 the system 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.

Lesson Summary

You should now be able to:

- Create and process screens
- Add ABAP Dictionary screen elements
- Explain PBO and PAI processing

Lesson: Screen Modification and Sequence

Lesson Overview

In this lesson, you will learn how to modify attributes dynamically. You will learn how to initialize SCREEN system table. In addition, you will learn to insert screen sequences for complex transactions where you need to use multiple screens.

Lesson Objectives

After completing this lesson, you will be able to:

- Make dynamic screen modifications
- Insert screen sequences

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. These screens should enable you to modify attributes dynamically. For complex transactions where you need to use multiple screens, such as display flight bookings and view flight data, you need to know how to insert screen sequences.

Dynamic Screen Modifications

Dynamic changes to the attributes of screen elements are **temporary**.

Using this technique to modify the attributes of a screen element, you can replace long sequences of separate screens, which are more costly in terms of both programming time and runtime. For example, using this technique, you can change the attribute of a screen element to decide whether an input/output field is ready for input.

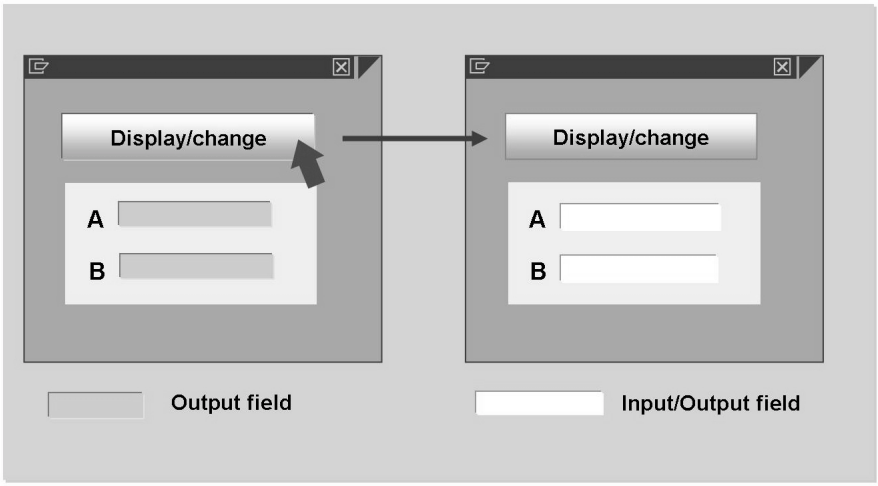


Figure 21: Modifying Attributes Dynamically: Example

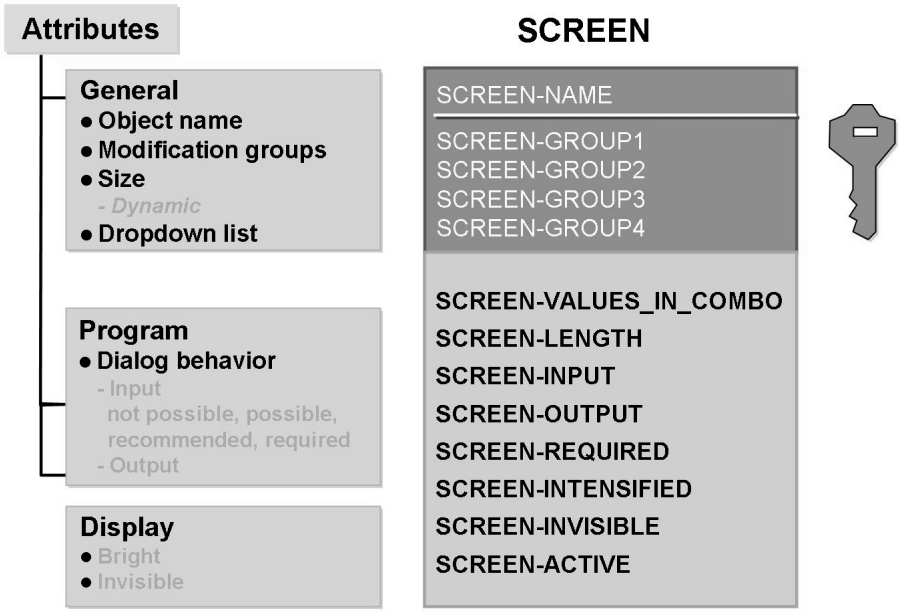


Figure 22: Dynamically Modifiable Static Attributes

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.

SCREEN-NAME is the unique identifier for a screen element on one screen. There is an exception to this rule: a text element may be called like the input element it describes.

SCREEN-REQUIRED may have three values: 0 means not required, 1 means required, and 2 (which was introduced in SAP Web Application Server 6.10) means recommended.

For a complete definition of the SCREEN structure, refer to the *Help on* documentation on *LOOP AT SCREEN* in the ABAP Editor.

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.

NAME	GROUP1	GROUP2	GROUP3	GROUP4	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
FIELDB	SEL				15	1	1	0	0	0	1

Figure 23: The SCREEN System Table

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.

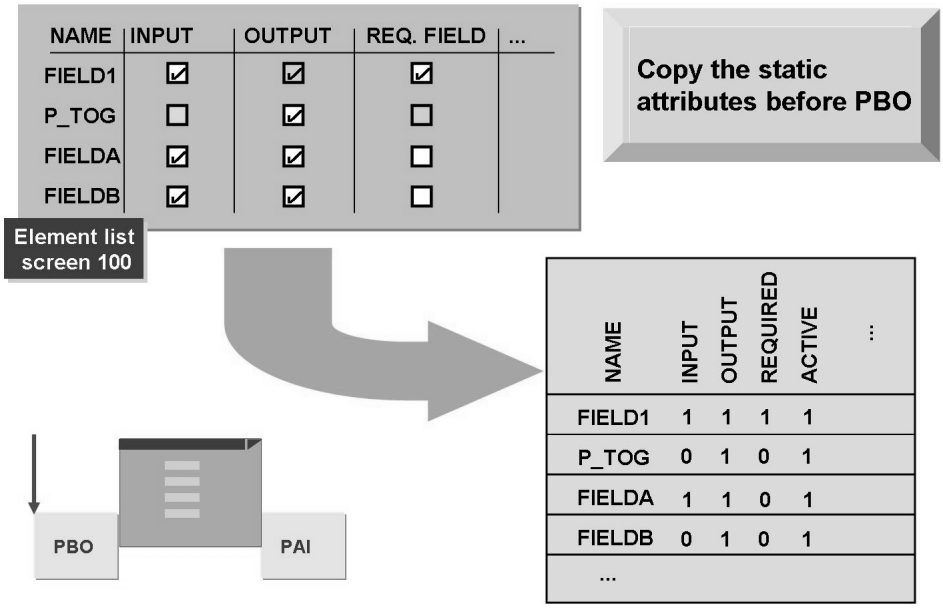


Figure 24: Initializing the SCREEN System Table

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' and 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.

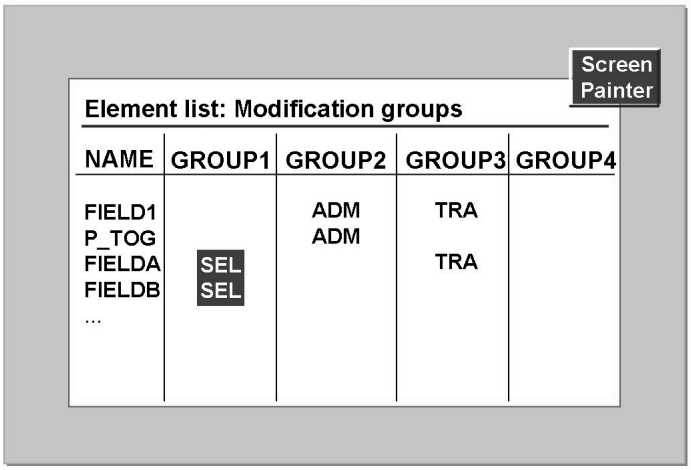


Figure 25: The Modification Group Attribute

You can change the attributes of several screen elements simultaneously at runtime, 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 maximum 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.

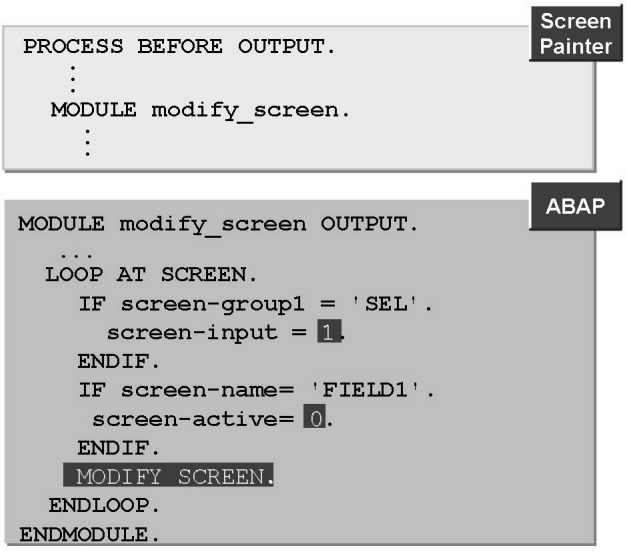


Figure 26: Modifying Attributes Dynamically: Program

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.

Screen Sequence

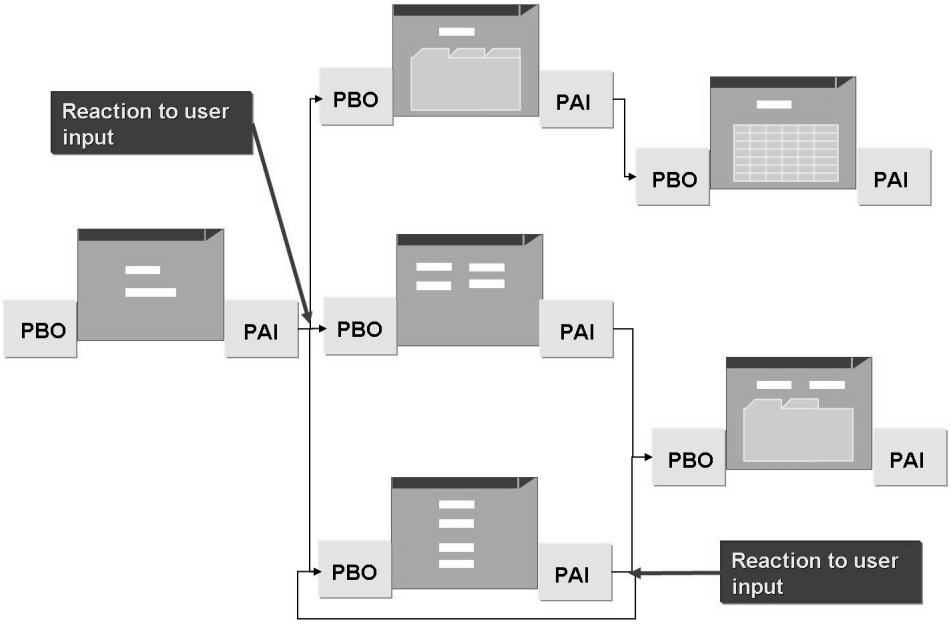


Figure 27: Determining the Next Screen

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 runtime, you can temporarily override the static next screen using the SET SCREEN statement.

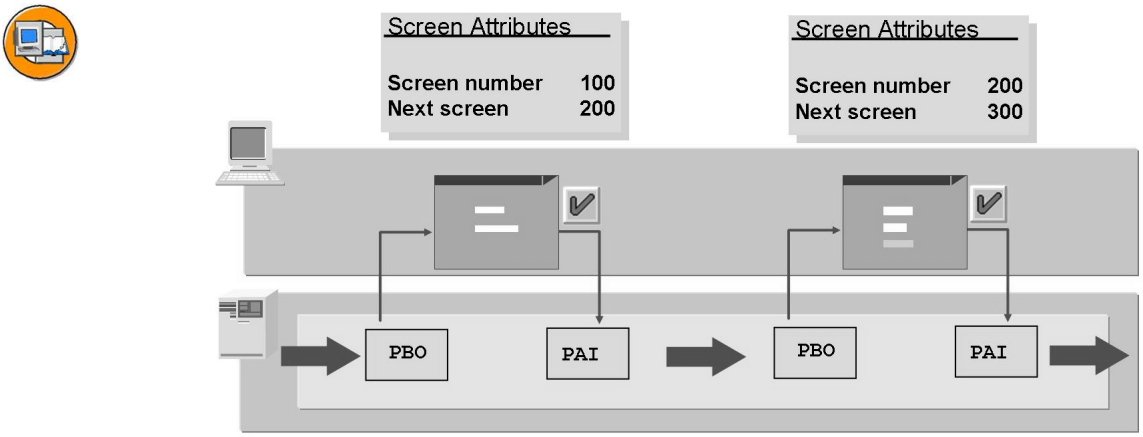


Figure 28: Static Screen Sequences

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.

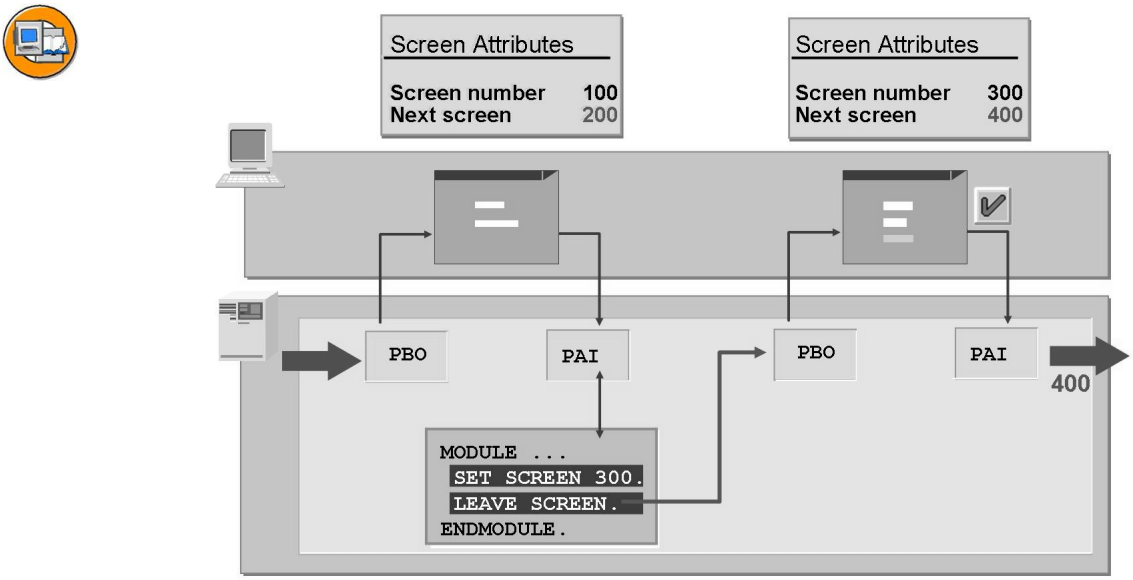


Figure 29: Setting the Next Screen Dynamically

The SET SCREEN statement **temporarily** overwrites the *Next screen* attribute.

The screen 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 statement.

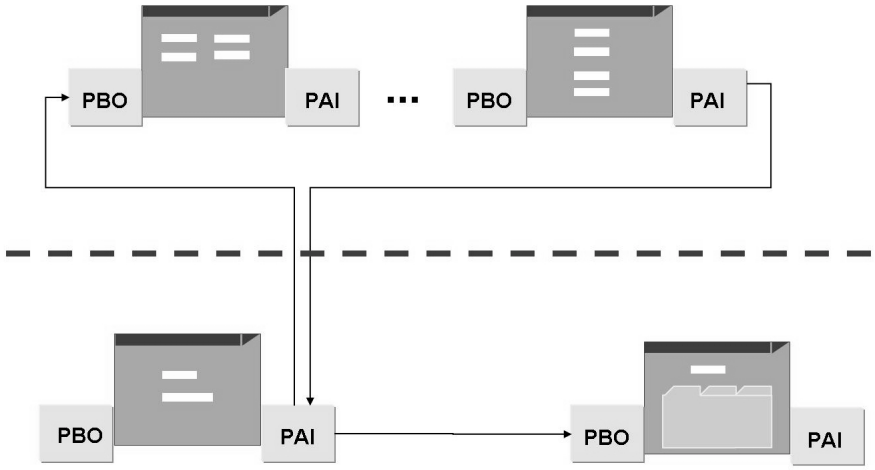


Figure 30: Inserting Screen Sequences

You can insert a screen sequence. This adds another layer to a stack.

You insert a screen sequence using the CALL SCREEN 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.

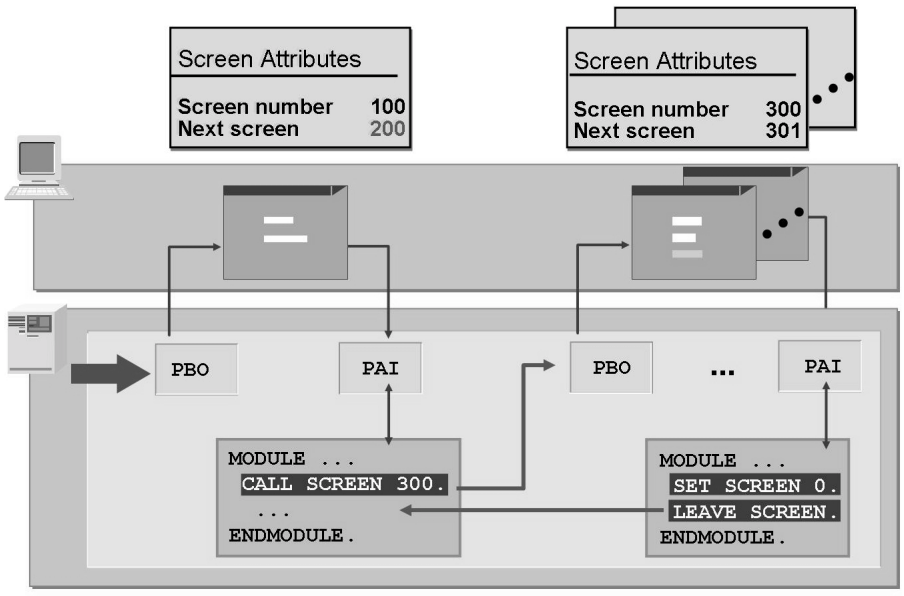


Figure 31: Inserting a Sequence of Screens Dynamically

To interrupt processing of the current screen and branch to a new screen or sequence of screens, use the CALL SCREEN statement. The screen must belong to the same program.

In the program, the system constructs a stack.

To return to the statement following the CALL SCREEN statement, you can use either SET SCREEN 0. LEAVE SCREEN. or LEAVE TO SCREEN 0. The screen that called the other screen is then processed further.

If you use the above statements outside a call chain, the program terminates, and control returns to the point from which it was called. You can also terminate a program using the ABAP statement LEAVE PROGRAM.

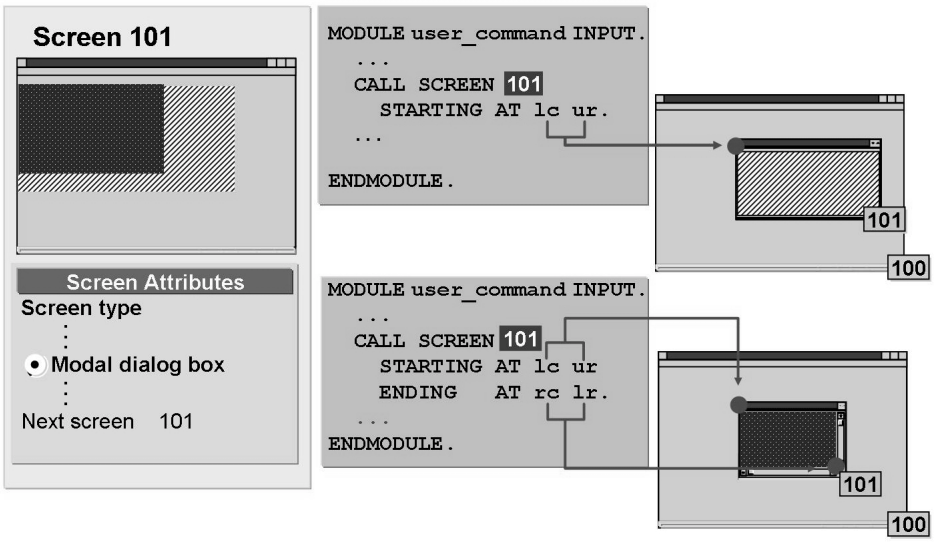


Figure 32: Calling a Dialog Box Dynamically

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.

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

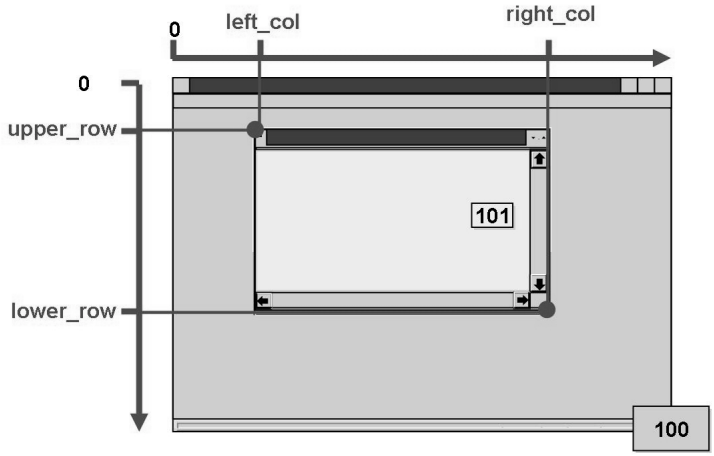


Figure 33: Window Coordinates

The starting position or 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 SAP screen from which you display the second screen with *CALL SCREEN*, screen 100 in the example as shown in the figure.

CALL SCREEN <nnnn>

STARTING AT left_col upper_row

ENDING AT right_col lower_row.

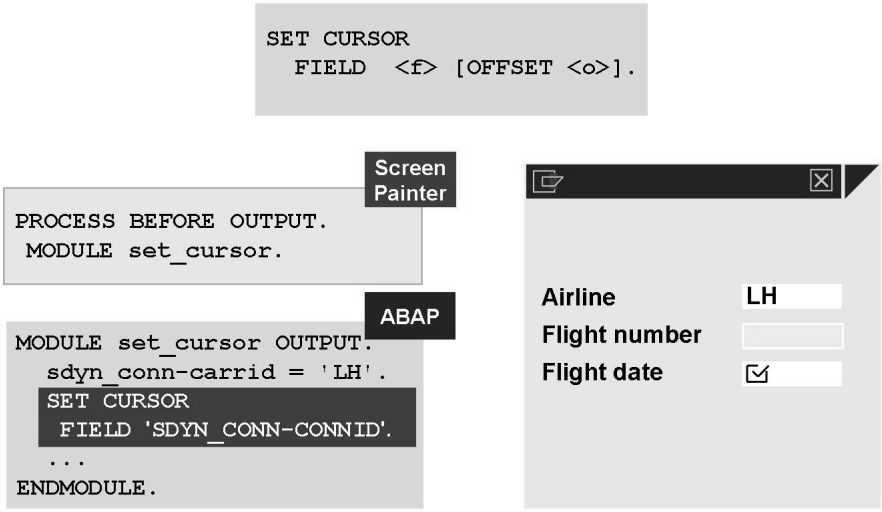


Figure 34: Setting the Cursor Position Dynamically

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.

Exercise 1: Creating Screens

Exercise Objectives

After completing this exercise, you will be able to:

- Create screens and use them in your applications

Business Example

Create a program with a screen including input and output fields, and read data from the database. Create an appropriate transaction for your program.

Task:

Create a program with a screen including input and output fields, and read data from the database. Create an appropriate transaction for your program.

1. Create a program.

   Create package **ZBC410_##** (where ## is your group number) and assign it to the change request provided by your instructor.

   In this package, create the program **SAPMZ##BC410_SOLUTION** with a TOP include (type: module pool). Accept the name proposed by the system for the TOP include. You can use the model solution **SAPMBC410ADIAS_DYNPRO** for orientation.

2. Create the following program object:

Screen	0100	**Description**: Maintenance screen **Type**: Normal **Next screen**: 0100

3. In your package, create a transaction (type: dialog transaction) **Z##SOLUTION** for your program and test it. Note: Since you have not yet programmed any navigation for your program, testing it will temporarily generate an endless loop, which means that you need to restart the Workbench.

4. Create the following fields on the screen. Use the facility for using fields from the ABAP Dictionary.

Continued on next page

Screen 100	**I/O Fields, Text Fields** SDYN_CONN • CARRID • CONNID • FLDATE	**Attributes for Each Field:** **Input:** on **Output:** on **Required Entry:** on
	I/O Fields, Text Fields SDYN_CONN • PRICE • CURRENCY • PLANETYPE • SEATSMAX • SEATSOCC • PAYMENTSUM	**Attributes for Each Field:** **Input:** off **Output:** on

5. In the PAI event for screen 100, create a module CHECK_SFLIGHT. Use the forward navigation to do this, to create the module in a new include. Accept the name proposed by the system. Read the fields from the database table SFLIGHT where the data corresponds for airline, flight number, and flight date.

   You have two options:

   You can either read directly into the structure SDYN_CONN. Or you use a matching internal structure, wa_sflight, which you create in the TOP include, to read from the database. In this case, you need to ensure that the help structure's fields are copied to the screen's fields. To do this, you require a new PBO module. The best way to create this is to use the forward navigation.

   If no data can be read for the entries on the screen, display message 007 of message class BC410 as an information message and initialize the screen's fields. (Note: The fields CARRID and CONNID are automatically checked. You only check the combination of CARRID, CONNID, and FLDATE.)

6. We have not yet dealt with normal navigation (with a Back button, and so on). However, use your existing knowledge to create a simple way of exiting the program (this will be more conveniently controlled in the next unit):

   Create a single-character character field on the screen. If the user enters X in this field, he or she should return to the point from which the program was called.

Continued on next page

Internal Use SAP Partner Only

Program an appropriate query in a new PAI module, *USER_COMMAND_0100*.

Hint: The simplest way to do this is to first create the single-character character field in ABAP (TOP include) and activate your program. You can then easily include the field in the screen in Screen Painter using the orange-colored button.

7. **Optional:** Display the time in a dialog box.

   If the user enters T in the field, a dialog box is to appear, showing the time and the command field is to be reinitialized.

   To do this, create a new screen and call it in PAI module *USER_COMMAND_0100*:

Screen	0150	**Description**: Time **Type**: Modal dialog box **Next screen**: 0

   Display the field *UZEIT* of the Dictionary structure *SYST* on the screen.

Solution 1: Creating Screens

Task:

Create a program with a screen including input and output fields, and read data from the database. Create an appropriate transaction for your program.

1. Create a program.

   Create package **ZBC410_##** (where ## is your group number) and assign it to the change request provided by your instructor.

   In this package, create the program **SAPMZ##BC410_SOLUTION** with a TOP include (type: module pool). Accept the name proposed by the system for the TOP include. You can use the model solution **SAPMBC410ADIAS_DYNPRO** for orientation.

   a) See sample solution.

2. Create the following program object:

Screen	0100	**Description**: Maintenance screen **Type**: Normal **Next screen**: 0100

   a) See sample solution.

3. In your package, create a transaction (type: dialog transaction) **Z##SOLUTION** for your program and test it. Note: Since you have not yet programmed any navigation for your program, testing it will temporarily generate an endless loop, which means that you need to restart the Workbench.

   a) See sample solution.

4. Create the following fields on the screen. Use the facility for using fields from the ABAP Dictionary.

Continued on next page

Screen 100	**I/O Fields, Text Fields** SDYN_CONN • CARRID • CONNID • FLDATE	**Attributes for Each Field:** **Input:** on **Output:** on **Required Entry:** on
	I/O Fields, Text Fields SDYN_CONN • PRICE • CURRENCY • PLANETYPE • SEATSMAX • SEATSOCC • PAYMENTSUM	**Attributes for Each Field:** **Input:** off **Output:** on

a) See sample solution.

5. In the PAI event for screen 100, create a module CHECK_SFLIGHT. Use the forward navigation to do this, to create the module in a new include. Accept the name proposed by the system. Read the fields from the database table SFLIGHT where the data corresponds for airline, flight number, and flight date.

   You have two options:

   You can either read directly into the structure SDYN_CONN. Or you use a matching internal structure, wa_sflight, which you create in the TOP include, to read from the database. In this case, you need to ensure that the help structure's fields are copied to the screen's fields. To do this, you require a new PBO module. The best way to create this is to use the forward navigation.

   If no data can be read for the entries on the screen, display message 007 of message class BC410 as an information message and initialize the screen's fields. (Note: The fields CARRID and CONNID are automatically checked. You only check the combination of CARRID, CONNID, and FLDATE.)

   a) See sample solution.

6. We have not yet dealt with normal navigation (with a Back button, and so on). However, use your existing knowledge to create a simple way of exiting the program (this will be more conveniently controlled in the next unit):

Continued on next page

Create a single-character character field on the screen. If the user enters X in this field, he or she should return to the point from which the program was called.

Program an appropriate query in a new PAI module, *USER_COMMAND_0100*.

Hint: The simplest way to do this is to first create the single-character character field in ABAP (TOP include) and activate your program. You can then easily include the field in the screen in Screen Painter using the orange-colored button.

a) See sample solution.

7. **Optional:** Display the time in a dialog box.

If the user enters T in the field, a dialog box is to appear, showing the time and the command field is to be reinitialized.

To do this, create a new screen and call it in PAI module *USER_COMMAND_0100*:

Screen	0150	**Description**: Time **Type**: Modal dialog box **Next screen**: 0

Display the field *UZEIT* of the Dictionary structure *SYST* on the screen.

a) See sample solution.

Result

Model Solution SAPMBC410ADIAS_DYNPRO

Main program

```
INCLUDE MBC410ADIAS_DYNPROTOP.

INCLUDE MBC410ADIAS_DYNPROI01.

INCLUDE MBC410ADIAS_DYNPROOI01.
```

Flow logic screen 100

```
PROCESS BEFORE OUTPUT.
* MODULE STATUS_0100.
*
```

Continued on next page

Internal Use SAP Partner Only

```
  MODULE move_to_dynp.

PROCESS AFTER INPUT.
  MODULE user_command_0100.
  MODULE check_sflight.
```

Top include

```
PROGRAM  sapmbc410adias_dynpro.
TABLES: sdyn_conn.

DATA:
  wa_sflight TYPE sflight,
  io_command.
```

PBO module include

```
MODULE move_to_dynp OUTPUT.
  MOVE-CORRESPONDING wa_sflight TO sdyn_conn.
ENDMODULE.                 " move_to_dynp  OUTPUT
```

PAI module include

```
MODULE check_sflight INPUT.
  SELECT SINGLE *
    FROM sflight
* INTO CORRESPONDING FIELDS OF sdyn_conn  " direct read
    INTO wa_sflight         " Read into internal structure
    WHERE carrid = sdyn_conn-carrid AND
          connid = sdyn_conn-connid AND
          fldate = sdyn_conn-fldate.
  CHECK sy-subrc <> 0.
  CLEAR wa_sflight.
  MESSAGE i007(bc410).

ENDMODULE.                 " check_sflight  INPUT

MODULE user_command_0100 INPUT.
  CASE io_command.
    WHEN 'X'.
      LEAVE TO SCREEN 0.

    WHEN 'T'.
      CALL SCREEN 150
        STARTING AT 10 10
        ENDING   AT 50 20.
      CLEAR io_command.
```

Continued on next page

```
  ENDCASE.
ENDMODULE.                 " user_command_0100  INPUT
```

Lesson Summary

You should now be able to:

- Make dynamic screen modifications
- Insert screen sequences

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Describe the Single-transaction programming model
- organize your program source code with includes
- Create and process screens
- Add ABAP Dictionary screen elements
- Explain PBO and PAI processing
- Make dynamic screen modifications
- Insert screen sequences

Test Your Knowledge

1. When you create a screen, you must:

   a. Design the screen layout in the layout editor.

   b. Set the general screen attributes on the attribute screen.

   c. Write the flow logic in the flow logic editor.

   d. Set the field attributes in the field list.

   Select the correct order:

   Choose the correct answer(s).

   □ A a, b, c, d
   □ B c, d, b, a
   □ C b, a, d, c
   □ D d, b, a, c

2. After the processing of _____, the data is transferred to the screen and displayed.

   Fill in the blanks to complete the sentence.

3. To allow you to set the attributes of all screen elements, the Screen Painter contains an element list with four views.

   Determine whether this statement is true or false.

   □ True
   □ False

4. To interrupt processing of the current screen and branch to a new screen, or sequence of screens, you use the following statement:

   Choose the correct answer(s).

   □ A CALL SCREEN <nnnn>
   □ B SET SCREEN <nnnn>
   □ C LEAVE SCREEN
   □ D LEAVE TO SCREEN <nnnn>

5. You must program your screen modifications in a module that is processed during the _____ processing block.

   Fill in the blanks to complete the sentence.

Answers

1. When you create a screen, you must:

   a. Design the screen layout in the layout editor.
   b. Set the general screen attributes on the attribute screen.
   c. Write the flow logic in the flow logic editor.
   d. Set the field attributes in the field list.

   Select the correct order:

   Answer: C

   The correct order is to 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, and write the flow logic in the flow logic editor.

2. After the processing of PBO, the data is transferred to the screen and displayed.

   Answer: PBO

3. To allow you to set the attributes of all screen elements, the Screen Painter contains an element list with four views.

   Answer: False

   To allow you to set the attributes of all screen elements, the Screen Painter contains an element list with six views.

4. To interrupt processing of the current screen and branch to a new screen, or sequence of screens, you use the following statement:

   Answer: A

   To interrupt processing of the current screen and branch to a new screen, or sequence of screens, you use the CALL SCREEN <nnnn> statement.

5. You must program your screen modifications in a module that is processed during the PROCESS BEFORE OUTPUT processing block.

   Answer: PROCESS BEFORE OUTPUT

Unit 2

The Program Interface

Internal Use SAP Partner Only

Unit Overview

This unit explains how GUI title and status are part of the user interface. It describes the procedure to create a GUI title and GUI status. The need to reuse existing menu bars, application toolbars, and key settings is discussed. Finally, how to use GUI status is discussed.

Unit Objectives

After completing this unit, you will be able to:

- Create a GUI title
- Identify function keys
- Create a GUI status
- Process the Function code

Unit Contents

Lesson: User Interfaces

Lesson Overview

This lesson will help you understand how GUI title and status are part of the user interface. You will learn how to create a GUI title in three ways: from the object list in the Object Navigator, from the Menu Painter, or by forward navigation from the ABAP Editor. In addition, you will understand from the technical point of view that a status is a reference to a menu bar, certain key assignments, and an application toolbar. Finally, function key settings, menus, and menu bars are explained.

Lesson Objectives

After completing this lesson, you will be able to:

- Create a GUI title
- Identify function keys

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. For these screens, you need to create a user interface. This interface should include the GUI title and status.

Overview: Interface

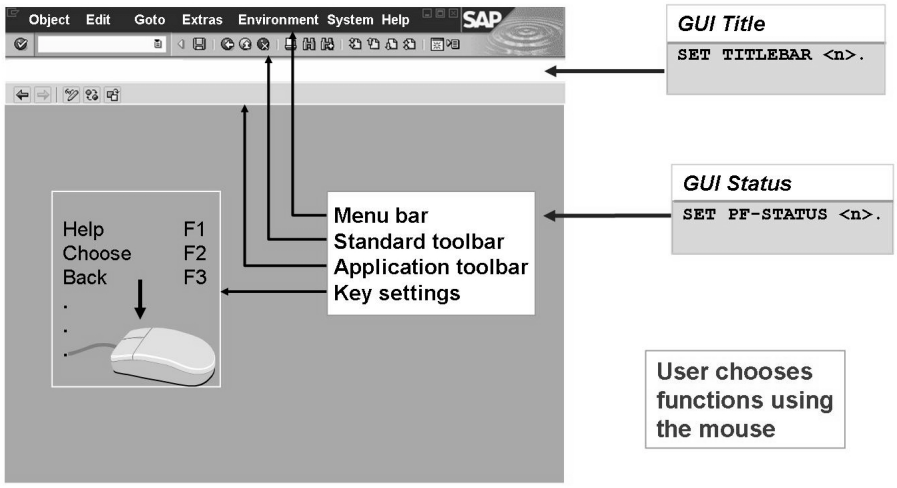


Figure 35: Overview: Interface

A GUI status consists of a menu bar, a standard toolbar, an application toolbar, and 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 SAP 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 SAP 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.

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.

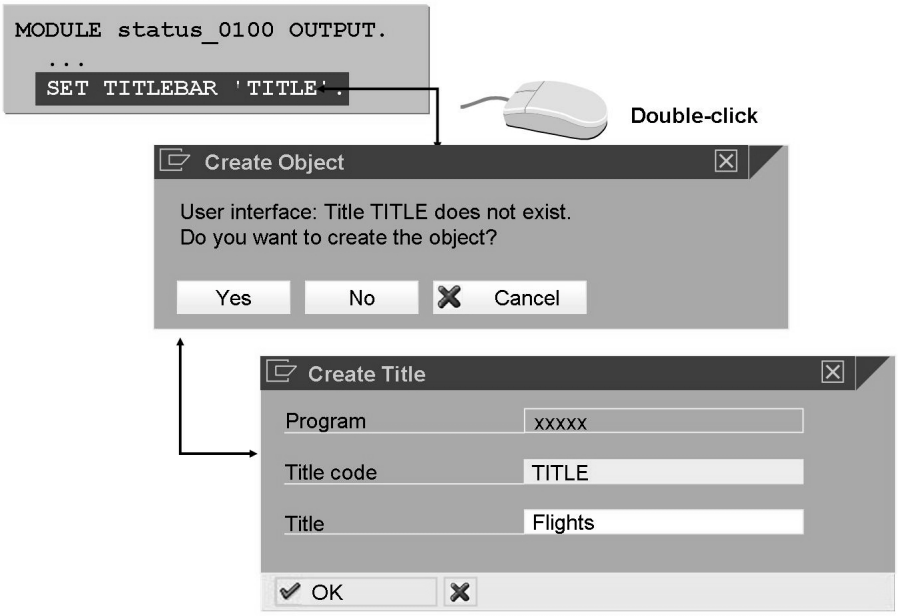


Figure 36: GUI Title

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 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.

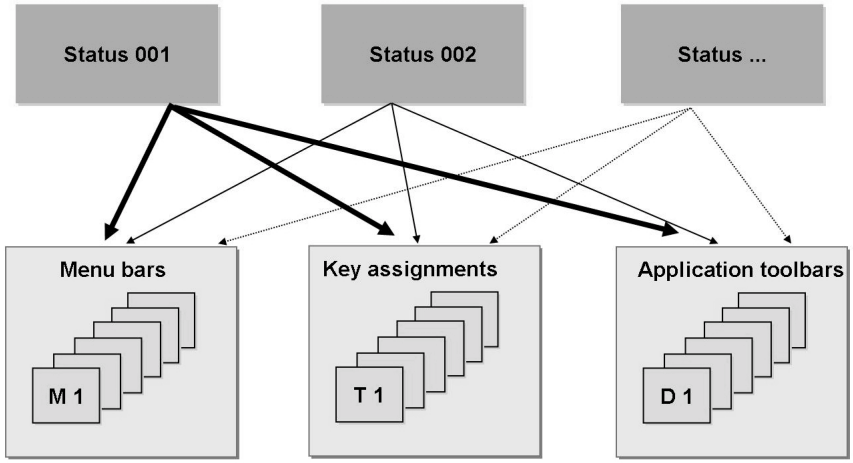


Figure 37: Status: Technical View (1)

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.

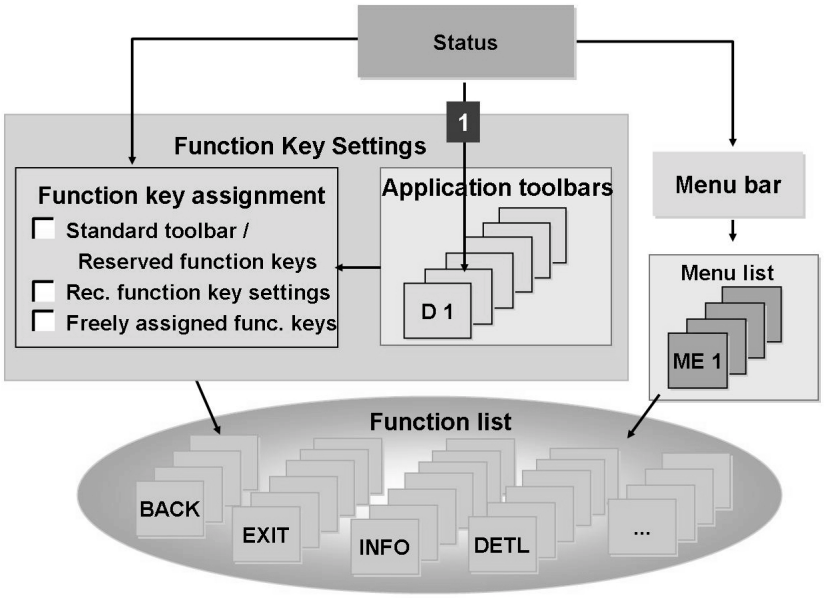


Figure 38: Status: Technical View (2)

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 *Goto → Interface objects; Function key settings → <name> → Pushbutton settings; User interface → Subobject → Create* in the Menu Painter. 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.

Functions

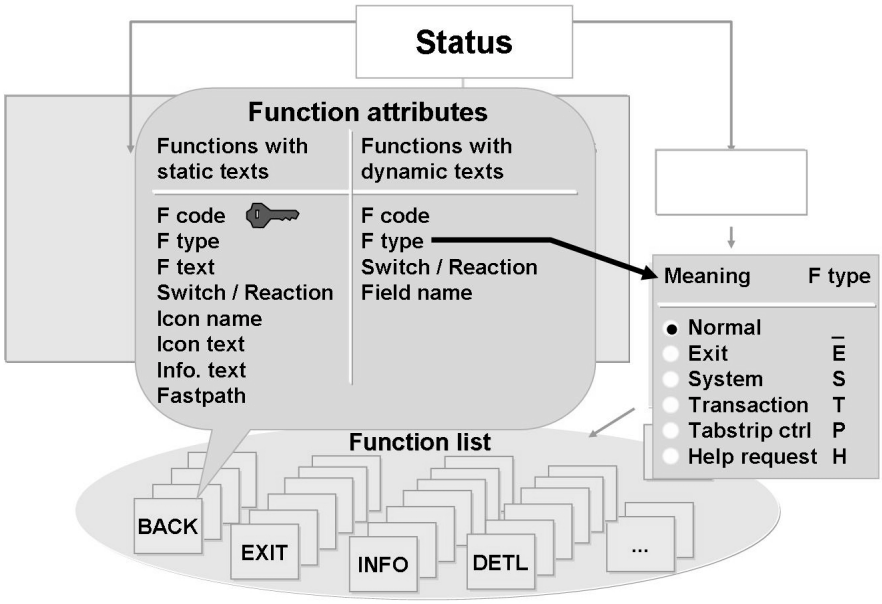


Figure 39: Functions

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. 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.

A function can be connected to a switch. If the switch is switched on, you can define a reaction. The function then is either displayed where it when it was not displayed before or hidden when it was displayed before.

For further information about functions, refer to the online documentation path *SAP Library → SAP NetWeaver Components → ABAP Workbench → ABAP Workbench Tools → Menu Painter → Functions*.

For further documentation about switches, refer to *ABAP Keyword documentation → ABAP Changes by Release → Changes in Release 7.0 → Switch Framework*.

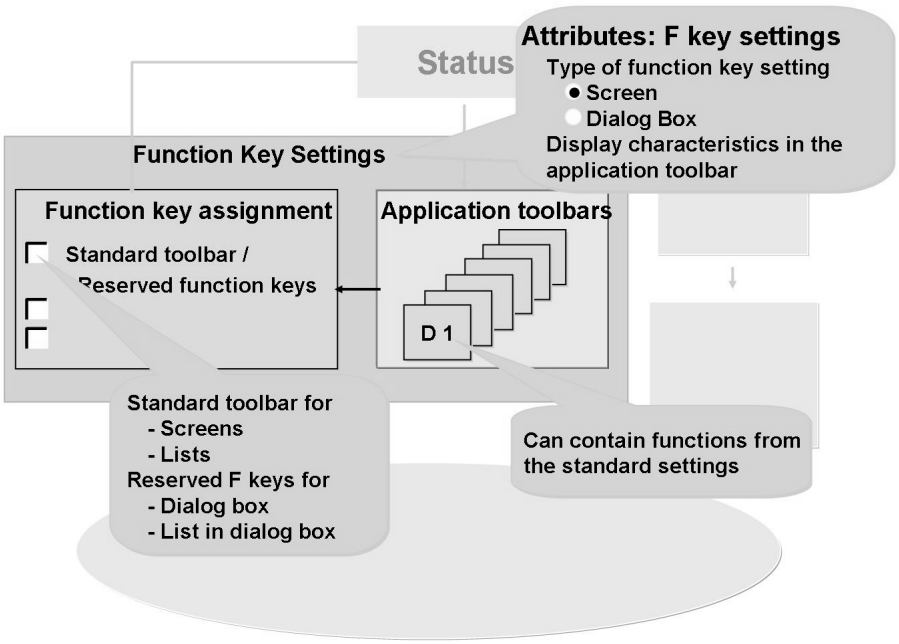


Figure 40: Function Key Settings

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.

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*.

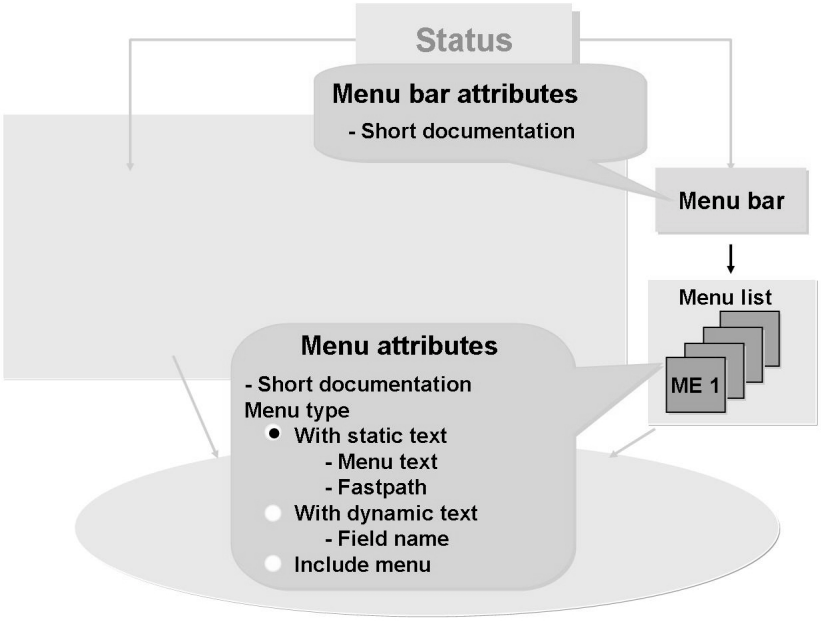


Figure 41: Menus and Menu Bars

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 menus can be freely assigned. The system automatically adds both the *System* menu and the *Help* menu to every menu bar.

Lesson Summary

You should now be able to:

- Create a GUI title
- Identify function keys

Internal Use SAP Partner Only

Lesson: GUI Status

Lesson Overview

In this lesson, you will learn how to create a GUI status using function key assignments, an application toolbar, and a menu bar. In addition, you will understand the need to reuse existing menu bars, application toolbars, and key settings. Finally, you will understand how to use a GUI status.

Lesson Objectives

After completing this lesson, you will be able to:

- Create a GUI status
- Process the Function code

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. On a screen, you need to create a GUI status that is a reference to a menu bar, to certain key assignments, and to an application toolbar. You should be able to create the GUI status either by creating links to existing components or by creating a blank status.

Creating a GUI Status

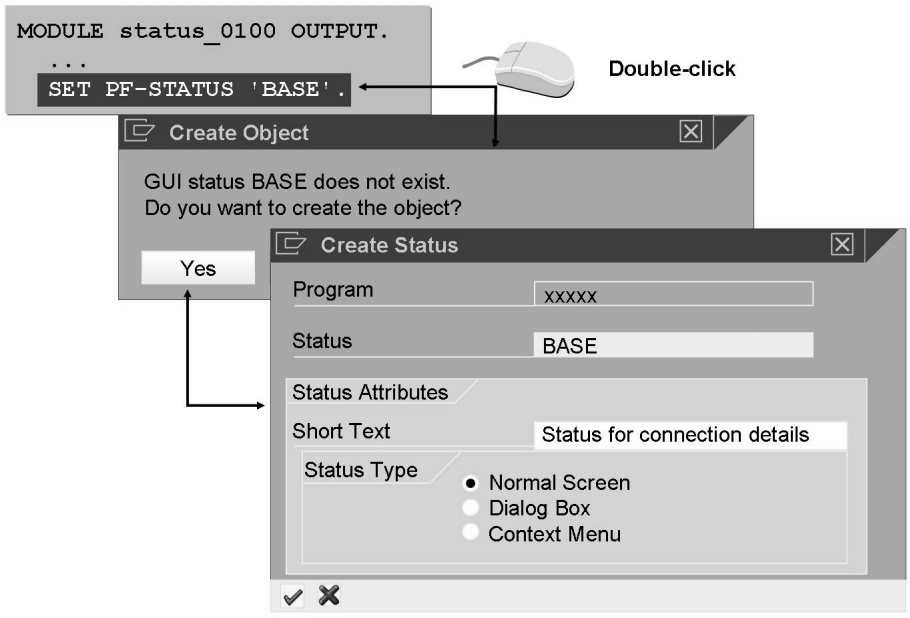


Figure 42: Creating a GUI Status

The status type indicates the technical attributes of the status. You can choose between a status for a normal screen (full screen), a dialog box status (for use with modal dialog boxes), and a context menu statu. Context menus are special collections of functions that can be displayed with a right-click.

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 status referring to them.

The reference technique is particularly effective for ensuring consistency in very large applications that use several status. The links ensure that the user can access functions in the same way whatever status is set.

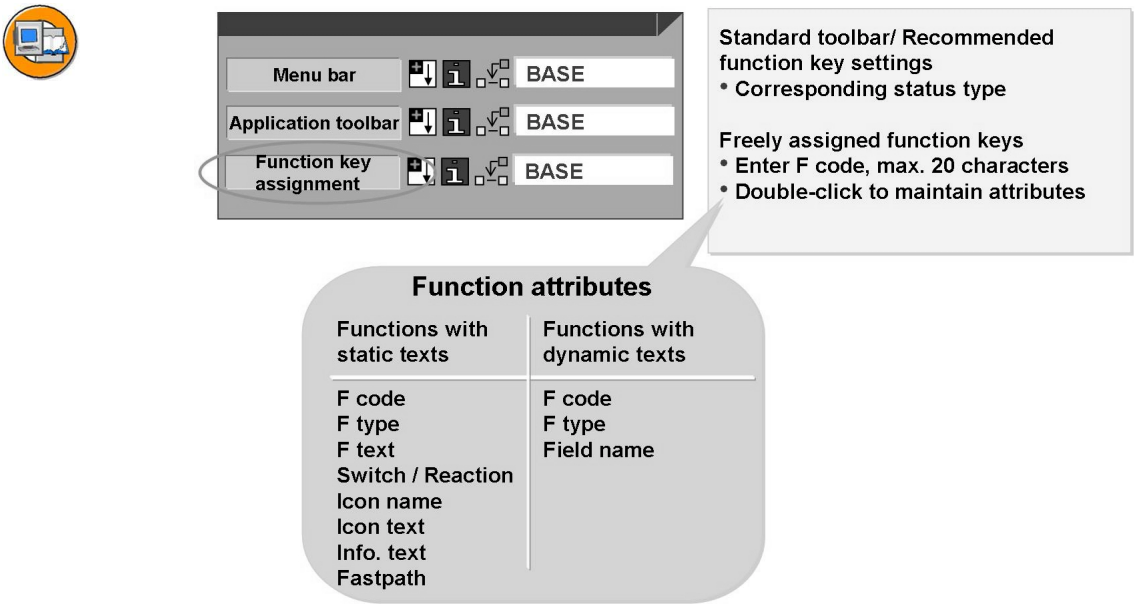


Figure 43: Creating a GUI Status: Function Key Assignment

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 be of 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 ergonomic standards, which can be found in the *Environment* menu in the Menu Painter.

Reserved function keys appear in the standard toolbar of the SAP GUI.

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.

Icon	Function key	Meaning
	Enter	
	Ctrl-S or F11	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

Figure 44: Standard Toolbar: Automatic Assignments

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 *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 the menu path *Utilities* → *Help texts* → *Internal key numbers* in the Menu Painter.

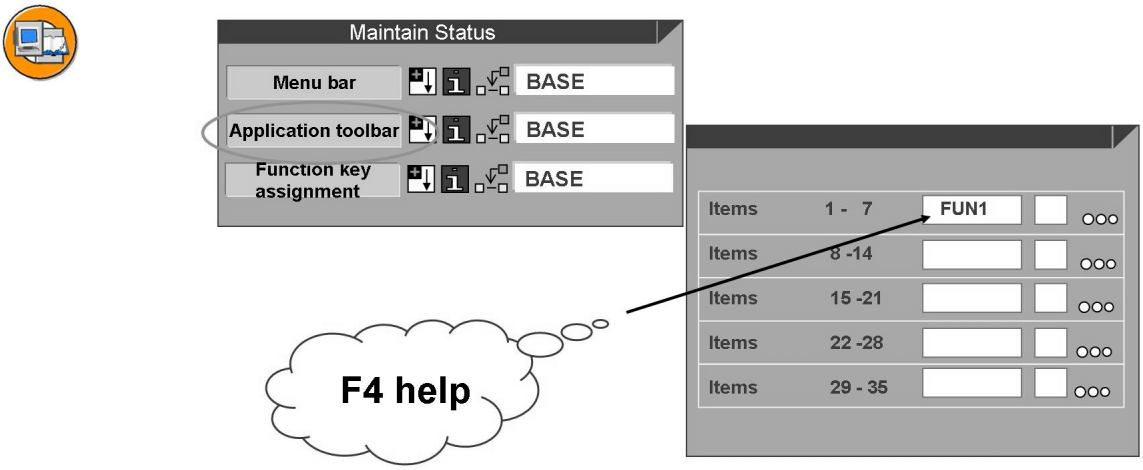


Figure 45: Creating a GUI Status: Application Toolbar

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 the quick info (tooltip). 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.

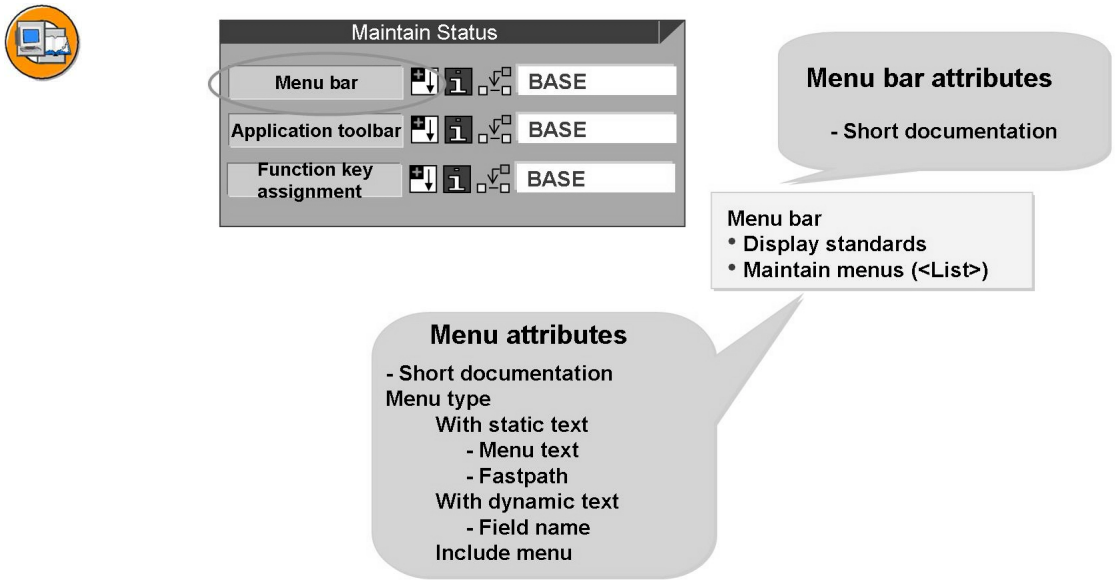


Figure 46: Creating a GUI Status: Menu Bar

A menu entry can be a function, a separator, another menu, or a 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.

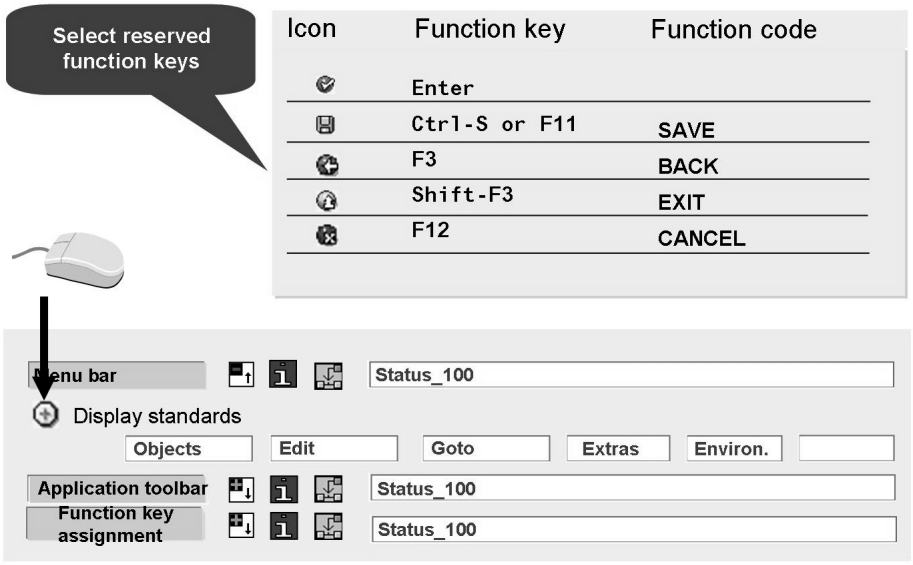


Figure 47: Displaying Standards

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 standards. This makes your program easier for users to understand and for you to maintain.

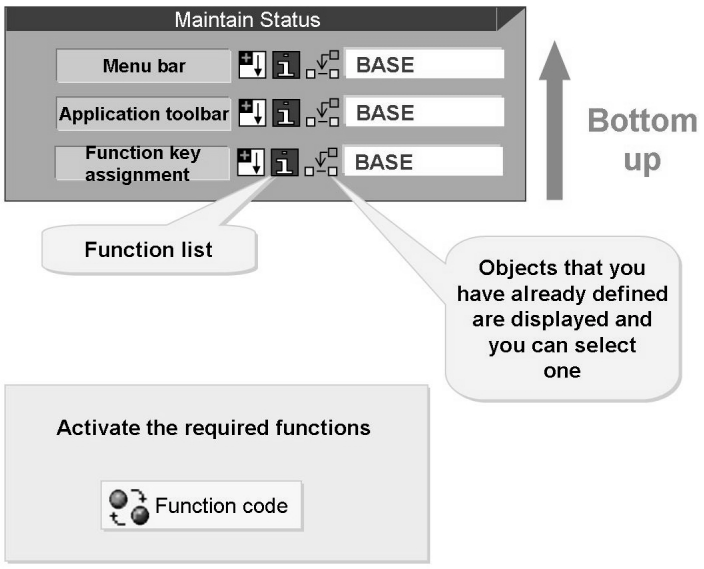


Figure 48: Including Existing Elements

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 use the Menu Painter, 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.

Using GUI Status

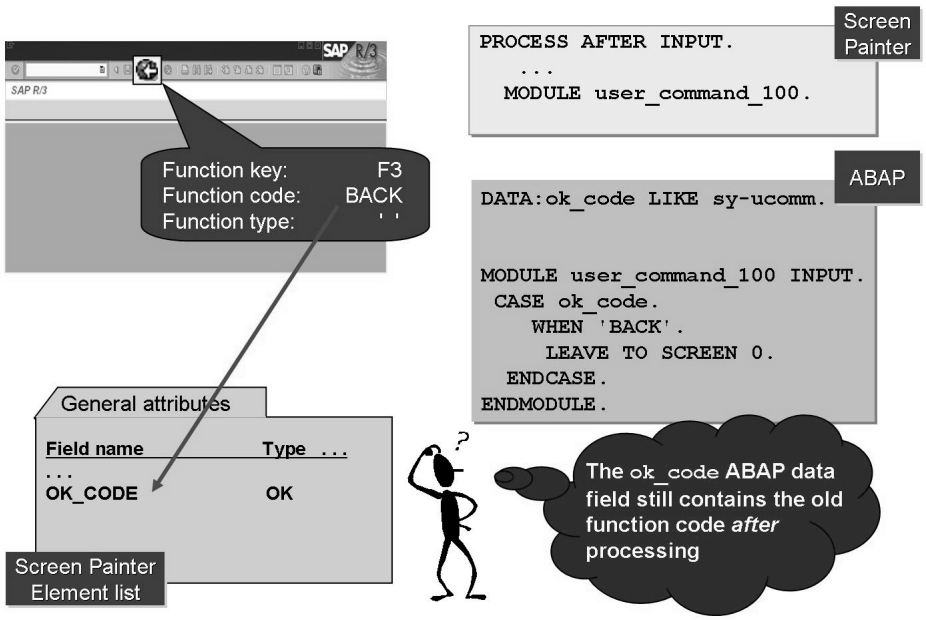


Figure 49: Processing the Function Code

When the user triggers a function with type ' ' (space) 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 (pressing the *Enter* key does not usually change the command field), you should initialize the identically named ABAP field.

The easiest way to achieve this is by writing a PBO module with the following line (provided you have named the user command field *ok_code* in the element list):

```
CLEAR ok_code.
```


Internal Use SAP Partner Only

Exercise 2: Creating GUI Status

Exercise Objectives

After completing this exercise, you will be able to:

- Create dialog statuses and use them in your applications

Business Example

After creating the programs for creating screens and modifying screens, you need to create GUI titles and assign functions to function keys or buttons. Use Menu Painter to create a GUI status and set GUI Title as Flight data. In addition, implement the field processing.

Task:

Use the Menu Painter to create a GUI status and GUI title and evaluate the user input.

1. 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.
2. In the PBO event for screen 100, call a module STATUS_100. Use the forward navigation to create the module in a new include. Accept the name proposed by the system.

   Set the GUI status *STATUS_100* there and create it using forward navigation. Choose Dialog Status as the status type and Maintenance Screen as the short text. Activate the standard function BACK (F3) with the function type ' '.

   You should also set the GUI title *TITLE_100*. It should be possible to change the contents of the title using a parameter : “Flight Data (&1)”. In this exercise, fill parameter &1 with “View”. Use a text symbol for the parameter, to ensure that it can be translated.
3. Assign the name *ok_code* to the command field on your screen, and create a corresponding variable in the top include of your program.
4. Implement the command field processing. Instead of controlling the next screen using the screen’s character field as it has been the case until now, the *ok_code* is now to be evaluated in the PAI module *USER_COMMAND_0100*: ensure that the user of screen 100 returns to the point from which the screen was called if he or she chooses BACK (F3).
5. To avoid undesired navigation, initialize the command field in a module at the PBO event of the screen.

Continued on next page

6. **Optional:** Completely replace the character field on screen 100 that has been used until now for the user dialog, so that the time can now also be displayed using the status and no longer by entering a T.

   To do this, extend your status *STATUS_100* with the function TIME and extend your PAI module *USER_COMMAND_0100*.

7. **Optional:** Set a separate title bar and a separate status for the dialog box showing the time. Create both of these objects in a PBO module *STATUS_150* using forward navigation.

Solution 2: Creating GUI Status

Task:

Use the Menu Painter to create a GUI status and GUI title and evaluate the user input.

1. 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.

   a) See sample solution.

2. In the PBO event for screen 100, call a module STATUS_100. Use the forward navigation to create the module in a new include. Accept the name proposed by the system.

   Set the GUI status *STATUS_100* there and create it using forward navigation. Choose Dialog Status as the status type and Maintenance Screen as the short text. Activate the standard function BACK (F3) with the function type ' '.

   You should also set the GUI title *TITLE_100*. It should be possible to change the contents of the title using a parameter : “Flight Data (&1)”. In this exercise, fill parameter &1 with “View”. Use a text symbol for the parameter, to ensure that it can be translated.

   a) See sample solution.

3. Assign the name *ok_code* to the command field on your screen, and create a corresponding variable in the top include of your program.

   a) See sample solution.

4. Implement the command field processing. Instead of controlling the next screen using the screen’s character field as it has been the case until now, the *ok_code* is now to be evaluated in the PAI module *USER_COMMAND_0100*: ensure that the user of screen 100 returns to the point from which the screen was called if he or she chooses BACK (F3).

   a) See sample solution.

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

   a) See sample solution.

6. **Optional:** Completely replace the character field on screen 100 that has been used until now for the user dialog, so that the time can now also be displayed using the status and no longer by entering a T.

Continued on next page

To do this, extend your status *STATUS_100* with the function TIME and extend your PAI module *USER_COMMAND_0100*.

a) See sample solution.

7. **Optional:** Set a separate title bar and a separate status for the dialog box showing the time. Create both of these objects in a PBO module *STATUS_150* using forward navigation.

a) See sample solution.

Result

Model Solution SAPMBC410ADIAS_GUI

Main program

```
INCLUDE MBC410ADIAS_GUITOP.

INCLUDE MBC410ADIAS_GUII01.

INCLUDE MBC410ADIAS_GUIO01.
```

Flow logic screen 100

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

PROCESS AFTER INPUT.
  MODULE user_command_0100.
  MODULE check_sflight.
```

Flow logic screen 150

```
PROCESS BEFORE OUTPUT.
  MODULE status_0150.
*
PROCESS AFTER INPUT.
* MODULE USER_COMMAND_0150.
```

Top include

```
PROGRAM  sapmbc410adias_dynpro.
* screen structure
TABLES: sdyn_conn.

DATA:
* workarea for database read
```

Continued on next page

Internal Use SAP Partner Only

```
  wa_sflight TYPE sflight,
* function code at PAI
  ok_code    LIKE sy-ucomm.
```

PBO module include

```
*&---------------------------------------------------------------------*
*&      Module  move_to_dynp  OUTPUT
*&---------------------------------------------------------------------*
*       copy data to screen structure
*---------------------------------------------------------------------*
MODULE move_to_dynp OUTPUT.
  MOVE-CORRESPONDING wa_sflight TO sdyn_conn.
ENDMODULE.                 " move_to_dynp  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  status_0100  OUTPUT
*&---------------------------------------------------------------------*
*       set status and title for screen 100
*---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'STATUS_100'.
  SET TITLEBAR  'TITLE_100' WITH text-vie. " View
ENDMODULE.                 " status_0100  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  status_0150  OUTPUT
*&---------------------------------------------------------------------*
*       set status and title for screen 150
*---------------------------------------------------------------------*
MODULE status_0150 OUTPUT.
  SET PF-STATUS 'STATUS_150'.
  SET TITLEBAR  'TITLE_150' WITH text-tim. " Time
ENDMODULE.                 " status_0150  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  clear_ok_code  OUTPUT
*&---------------------------------------------------------------------*
*       initialize ok_code
*---------------------------------------------------------------------*
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.                 " clear_ok_code  OUTPUT
```

PAI module include

```
*&---------------------------------------------------------------------*
*&      Module  check_sflight  INPUT
*&---------------------------------------------------------------------*
*       Read flight record from database
```

Continued on next page

```
*---------------------------------------------------------------------*
MODULE check_sflight INPUT.
  SELECT SINGLE *
    FROM sflight
* INTO CORRESPONDING FIELDS OF sdyn_conn  " direct read
    INTO wa_sflight         " Read into internal structure
    WHERE carrid = sdyn_conn-carrid AND
          connid = sdyn_conn-connid AND
          fldate = sdyn_conn-fldate.
  CHECK sy-subrc <> 0.
  CLEAR wa_sflight.
  MESSAGE i007(bc410).

ENDMODULE.                 " check_sflight  INPUT
*&---------------------------------------------------------------------*
*&      Module  user_command_0100  INPUT
*&---------------------------------------------------------------------*
*       process user command
*---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.

* display time on add'l screen
    WHEN 'TIME'.
      CALL SCREEN 150
        STARTING AT 10 10
        ENDING   AT 50 20.

  ENDCASE.
ENDMODULE.                 " user_command_0100  INPUT
```


Internal Use SAP Partner Only

Lesson Summary

You should now be able to:

- Create a GUI status
- Process the Function code

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Create a GUI title
- Identify function keys
- Create a GUI status
- Process the Function code

Test Your Knowledge

1. The function keys that contain proposals, which comply with the SAP system's ergonomic standards are:

   Choose the correct answer(s).

   - □ A Reserved
   - □ B Recommended
   - □ C Freely
   - □ D All of the above

2. There are _____ ways to create a title.

   Fill in the blanks to complete the sentence.

3. The content of the Icontext attribute appears in the status bar of the screen when the user chooses the function.

   Determine whether this statement is true or false.

   - □ True
   - □ False

4. You can initialize the command field only at PAI.

   Determine whether this statement is true or false.

   - □ True
   - □ False

Answers

1. The function keys that contain proposals, which comply with the SAP system's ergonomic standards are:

   Answer: B

   Recommended function keys contain proposals, which comply with the SAP System's ergonomic standards.

2. There are three ways to create a title.

   Answer: three

3. The content of the Icontext attribute appears in the status bar of the screen when the user chooses the function.

   Answer: False

   The content of the Infotext attribute appears in the status bar of the screen when the user chooses the function.

4. You can initialize the command field only at PAI.

   Answer: False

   You can initialize the command field at PAI or at PBO.

Unit 3

Screen Elements for Output

Unit Overview

This unit discusses how text fields are created and used on a screen. It also deals with status icons and explains how to create and fill the status icons. How to create group boxes will also be discussed in this unit.

Unit Objectives

After completing this unit, you will be able to:

- Create and change text fields
- Create and change status icons
- Create and change group boxes

Unit Contents

Lesson: Text Fields, Status Icons, and Group Boxes

Lesson Overview

In this lesson, you will understand how text fields are used on a screen. You will learn how to create text fields in two ways.

You will also learn that a status icon is a placeholder for an icon. You will also see how to create and fill the status icons.

In addition, you will learn about a group box, which is a display element that encloses a selection of elements that belong together.

Lesson Objectives

After completing this lesson, you will be able to:

- Create and change text fields
- Create and change status icons
- Create and change group boxes

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. On a screen, labels, such as flight number or booking number, need to be displayed in front of the corresponding fields.

Output Elements: Text Fields

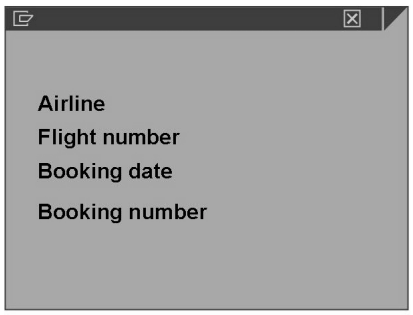


Figure 50: Text Fields

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 runtime. 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 checkbox, 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 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 *Goto* → *Translation* in the Screen Painter.

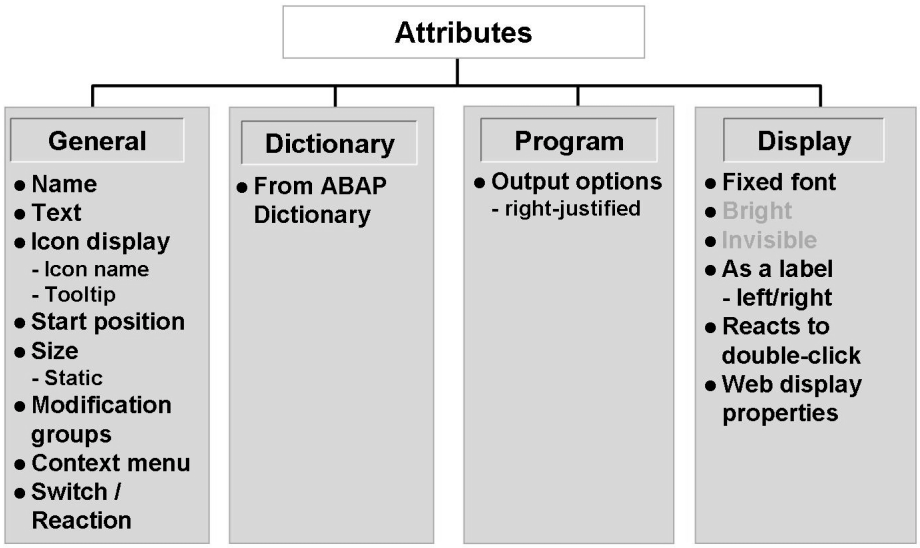


Figure 51: Text Field: Attributes

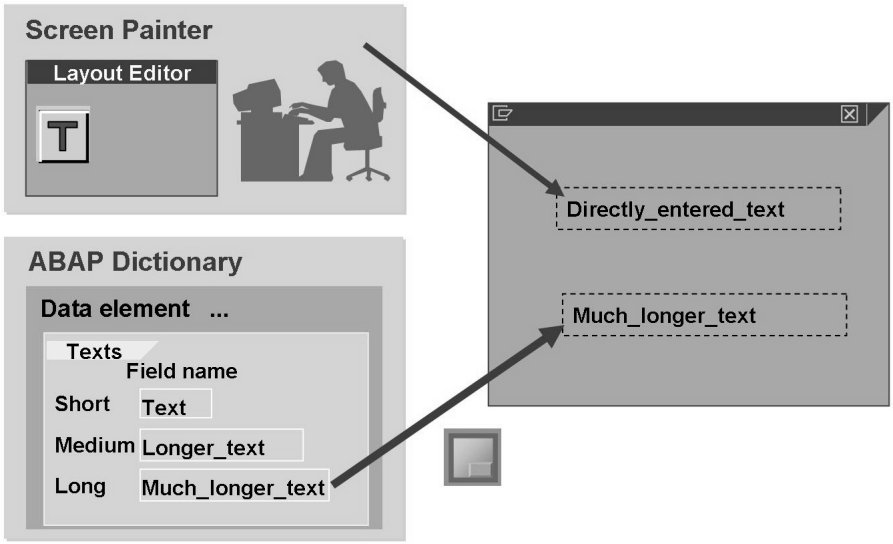


Figure 52: Creating Text Fields

You can create text fields in either of the following ways:

- By placing a text field object in the work area and entering the text in the *Text* attribute in the layout editor directly.
- By using the accompanying text of a data element from the ABAP Dictionary.
- When using 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.

Status Icons

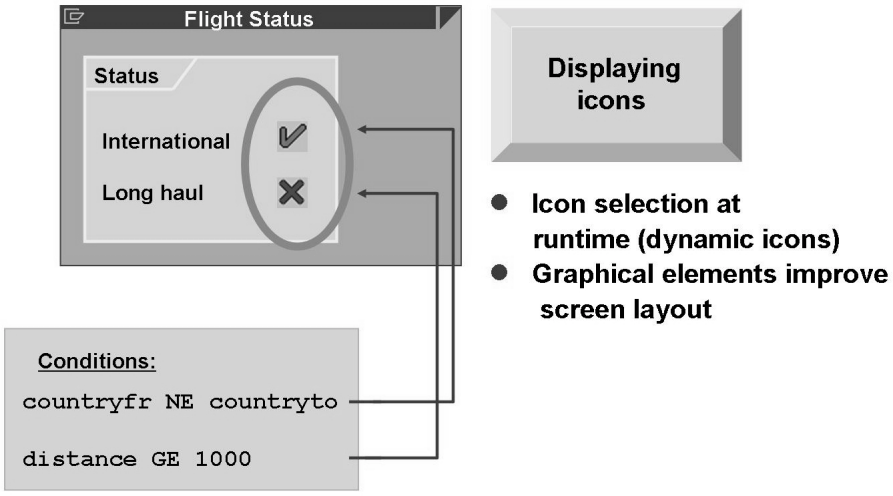


Figure 53: Status Icons

A status icon is a placeholder for an icon. You choose the relevant icon at runtime. 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, call report *SHOWICON*.

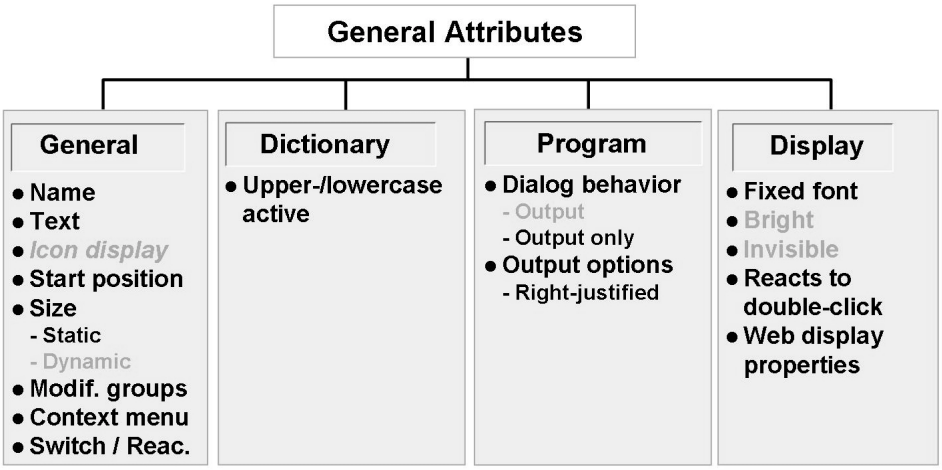


Figure 54: Status Icons: Attributes

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.

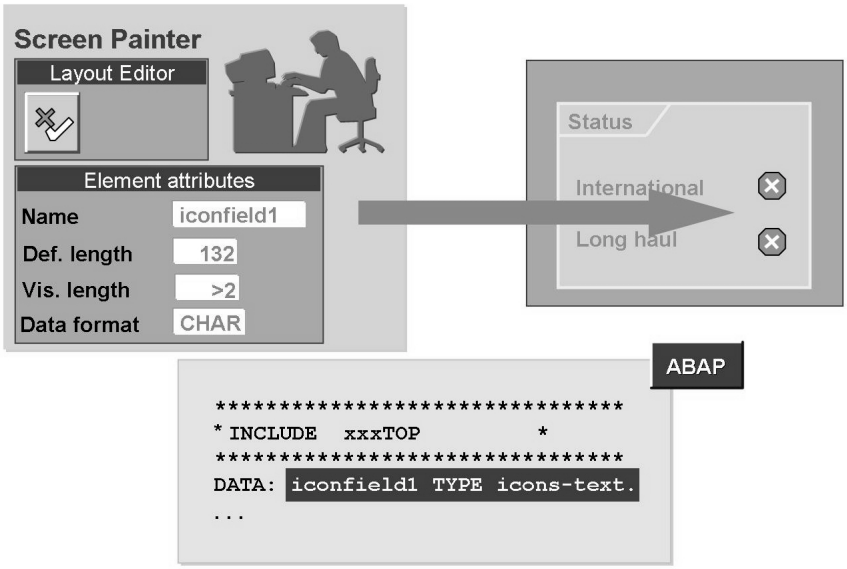


Figure 55: Creating Status Icons

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 runtime.

To ensure that you can display the *Quick info* that might be long, define the field with a defined length of 132 and visible length 2.

In the ABAP program, define a field with the same name as the screen field using the TEXT field from the ICONS structure. At runtime, this field contains the name of the icon you want to display.

At runtime, assign the required icon to this field using the *ICON_CREATE* function module.

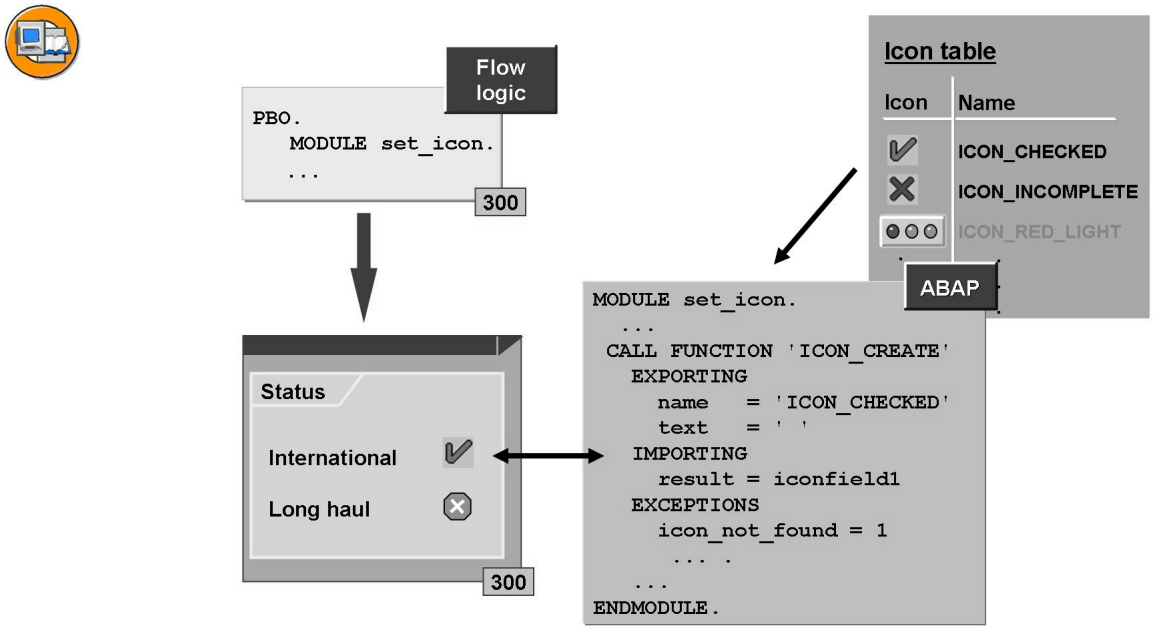


Figure 56: Filling a Status Icon

You select the icon you want to display from the ABAP program. Before the screen is displayed, you need to find 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 status icon field.

For further details about this function module, refer to its documentation.

Group Boxes

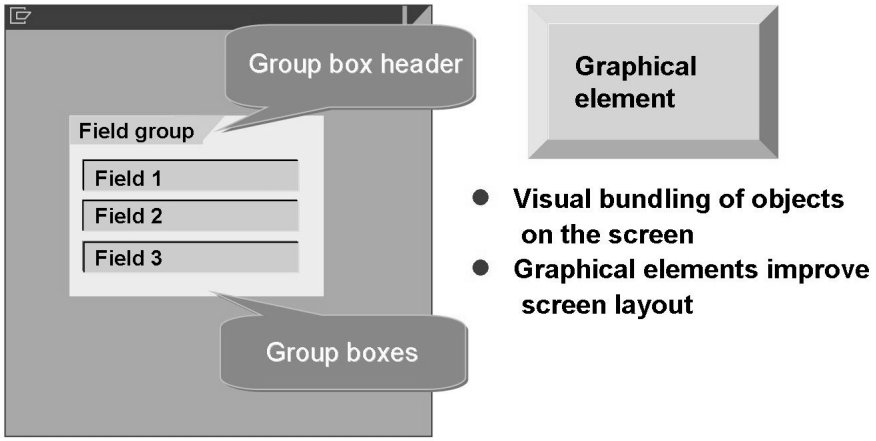


Figure 57: Group Boxes

Group boxes enclose a selection of elements that belong together, such as a group of fields or a radio button group. These 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.

Group boxes can have a title.

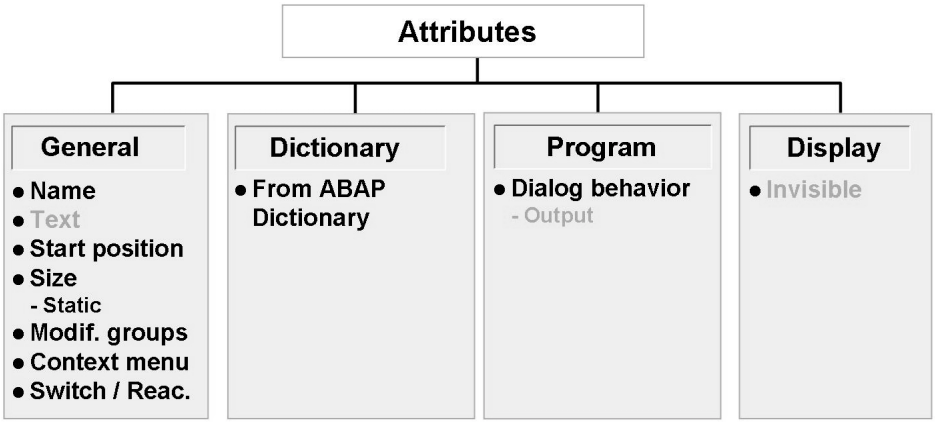


Figure 58: Group Boxes: Attributes

You can change *Visible length* and *Invisible* attributes using the SCREEN system table.

A group box may contain other screen elements.

At runtime, if the box contains only invisible elements and the screen attribute *Runtime compression* is set, the box is not displayed.

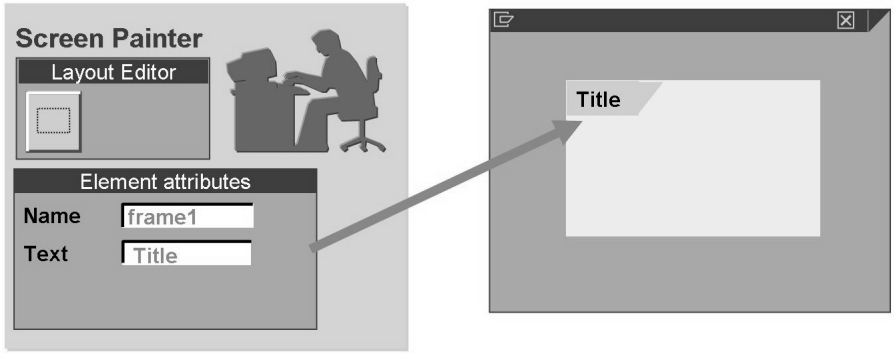


Figure 59: Creating Group Boxes

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.

Lesson Summary

You should now be able to:

- Create and change text fields
- Create and change status icons
- Create and change group boxes

Unit Summary

You should now be able to:

- Create and change text fields
- Create and change status icons
- Create and change group boxes

Test Your Knowledge

1. If your text consists of more than one word, use the following character as separators:

   Choose the correct answer(s).

   - □ A Underscore
   - □ B Comma
   - □ C Question mark
   - □ D Space

2. The group box text cannot be changed dynamically.

   Determine whether this statement is true or false.

   - □ True
   - □ False

3. _____, _____, and _____ attributes of a status icon can be changed dynamically.

   Fill in the blanks to complete the sentence.

Answers

1. If your text consists of more than one word, use the following character as separators:

   Answer: A

   If your text consists of more than one word, use the underscore characters as separators.

2. The group box text cannot be changed dynamically.

   Answer: False

   The group box text can be changed dynamically.

3. Visible length, Intensified, and Invisible attributes of a status icon can be changed dynamically.

   Answer: Visible length, Intensified, Invisible

Unit 4

Screen Elements for Input/Output

Unit Overview

This unit deals with input/output fields such as checkboxes, radio button groups, and pushbuttons. You will learn about input/output fields and the procedure to create them. You will also learn to perform error checking for the input/output fields. In addition, this unit covers how to create and use checkboxes, radio button groups, and pushbuttons and change attributes of these elements.

Unit Objectives

After completing this unit, you will be able to:

- Create input/output fields
- Check errors in input/output fields
- Create a dropdown box for an input field
- Create checkboxes in your programs
- Create radio button groups
- Create pushbuttons

Unit Contents

Internal Use SAP Partner Only

Lesson: Input/Output Fields Overview

Lesson Overview

This lesson will help you understand what input/output fields are and how to create them. You will learn how to perform error checking for input/output fields. In addition, you will learn how to navigate between screens using the *Back* and *Cancel* functions. Finally, you will learn that you can help the user with input by using dropdown list boxes containing the possible entries.

Lesson Objectives

After completing this lesson, you will be able to:

- Create input/output fields
- Check errors in input/output fields
- Create a dropdown box for an input field

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. The application should be able to accept inputs from or display the data to the user, and create input/output elements. It should also enable you to navigate between screens using the Back and Cancel functions.

Input/Output Fields

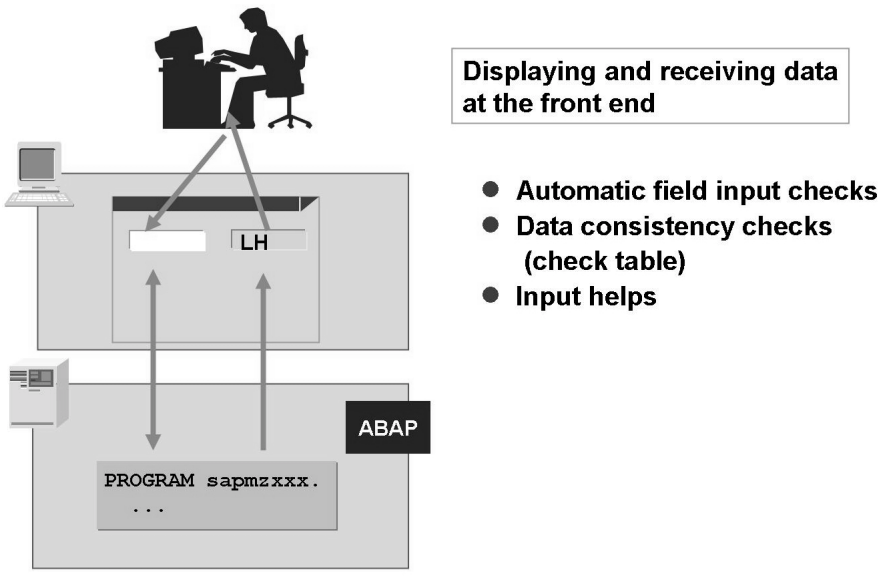


Figure 60: Input/Output Fields

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, *SAP Library → SAP NetWeaver 7.0 → SAP NetWeaver by Key Capability → Application Platform by Key Capability → ABAP Technology → ABAP Workbench → ABAP Workbench Tools → Screen Painter → Graphical Layout Editor → Overview of Screen layout → Screen elements*.

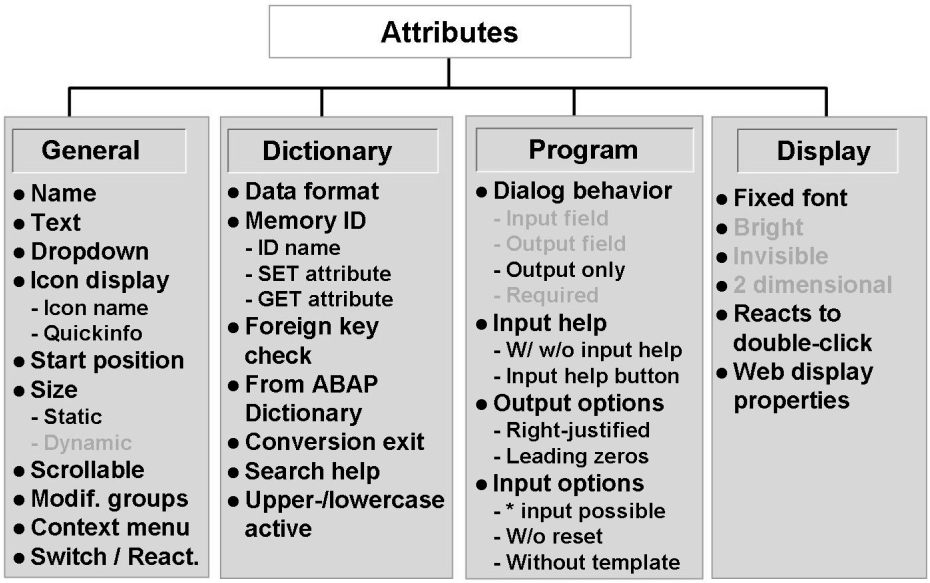


Figure 61: Input/Output Fields: Attributes

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 (*SAP Library → SAP NetWeaver 7.0 → SAP NetWeaver by Key Capability → Application Platform by Key Capability → ABAP Technology → ABAP Workbench → ABAP Workbench Tools → Screen Painter → Defining the element attributes → Choosing field formats*).

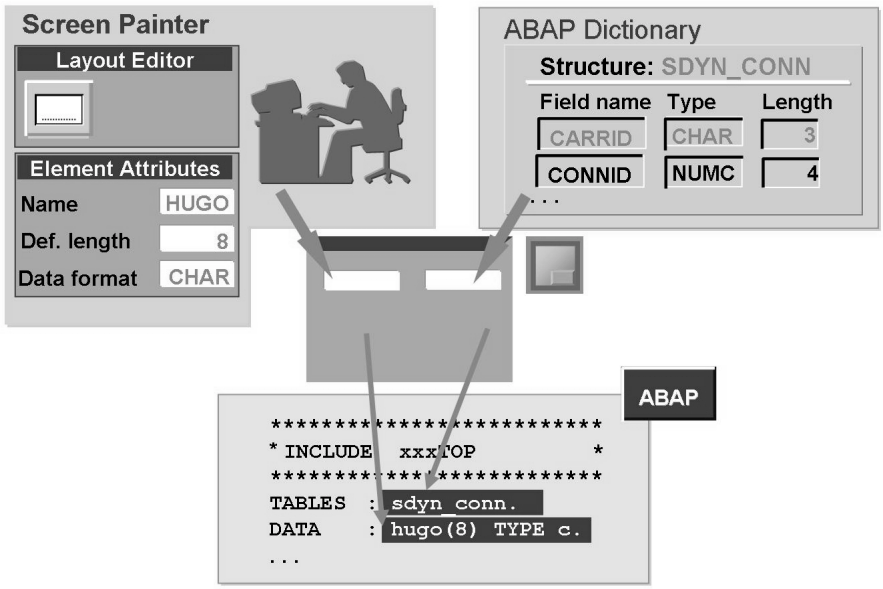


Figure 62: Creating Input/Output Fields

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.

Error Checking in Input/Output fields

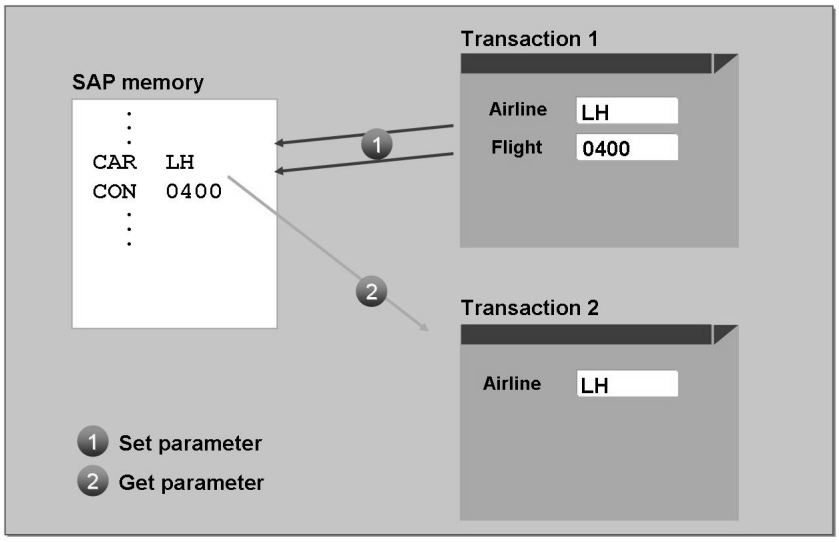


Figure 63: Default Values in SAP Memory

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.

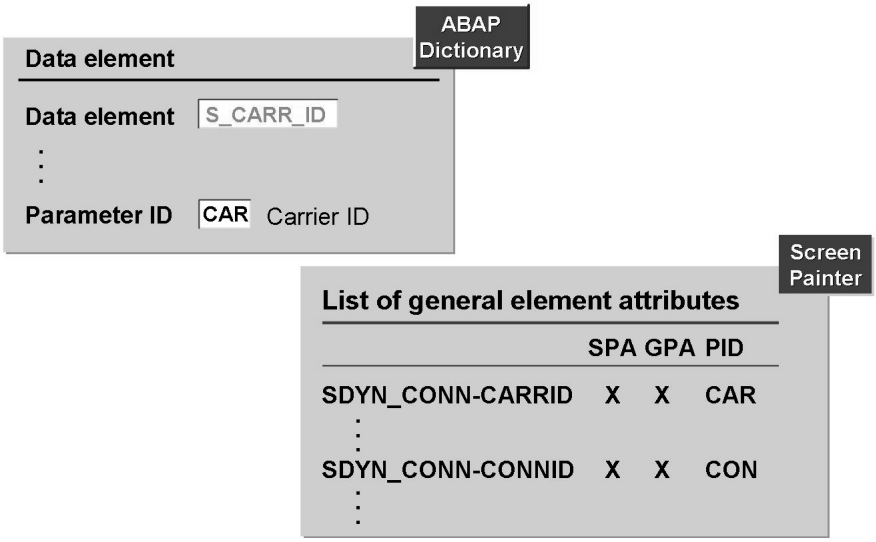


Figure 64: Defining SET and GET Parameter Attributes

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 the ABAP Workbench.

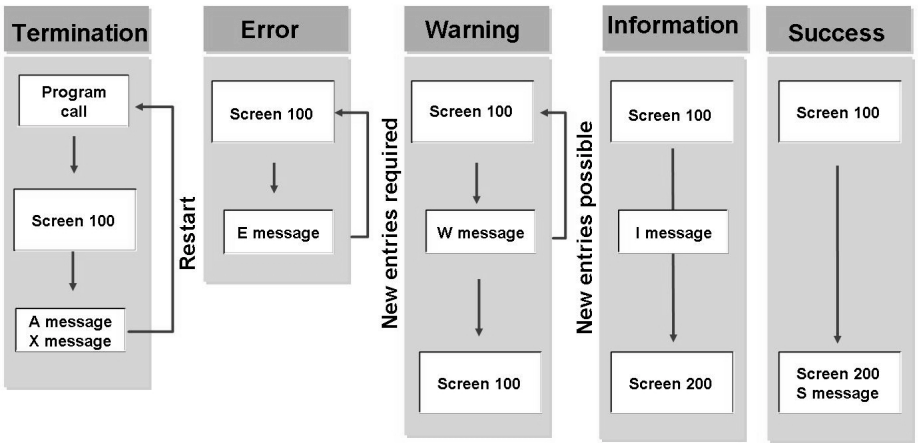


Figure 65: Dialog Message Categories

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

Message Types

Type	Text	Description
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 **must** correct the entry.
W	Warning	Processing is interrupted and the user can correct the entries.
I	Information	Processing is interrupted, but continues when the user has confirmed the message (by selecting Enter).
S	Success	Information is displayed on the next screen.

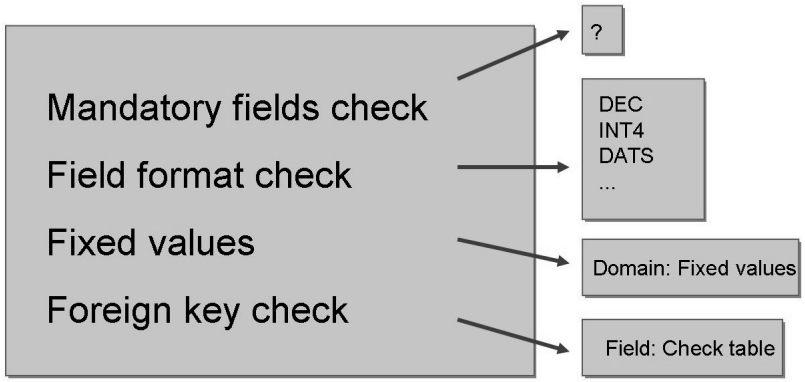


Figure 66: Automatic Field Input Checks

Before the PAI modules are processed, the system automatically checks the values the enters on the screen after the screen is displayed.

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.

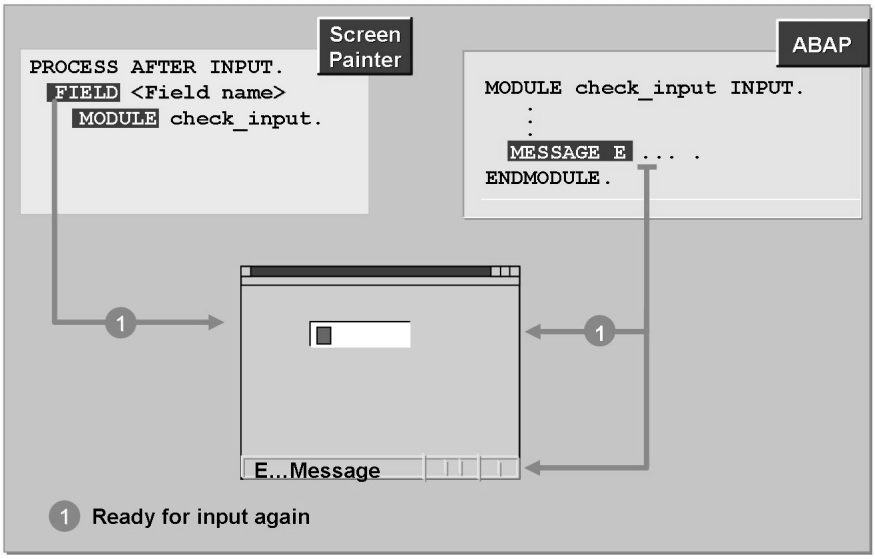


Figure 67: Field Input Checks with Error Dialog

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.

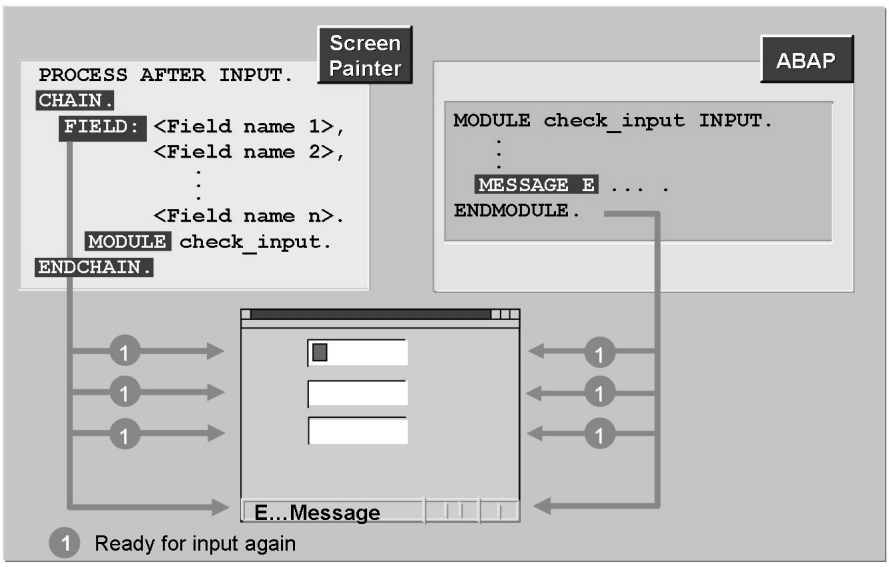


Figure 68: Checking Groups of Fields

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 fields 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 makes the field ready for input again. 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.

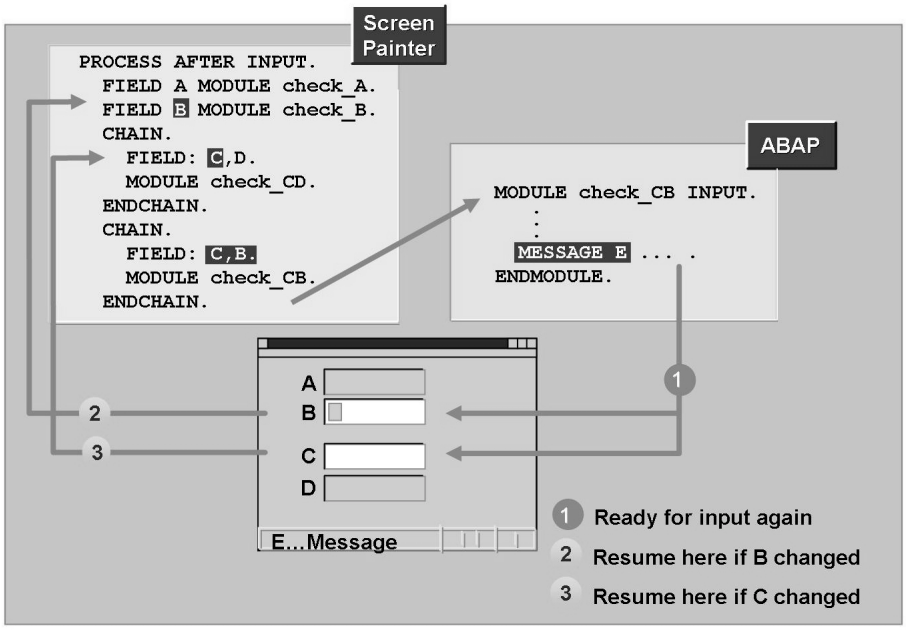


Figure 69: Controlling Error Dialogs

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.

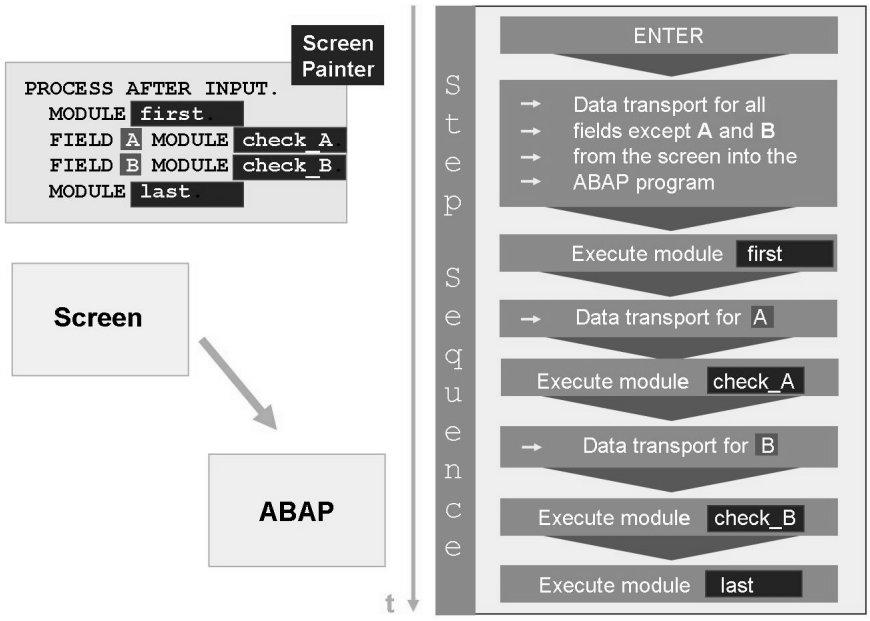
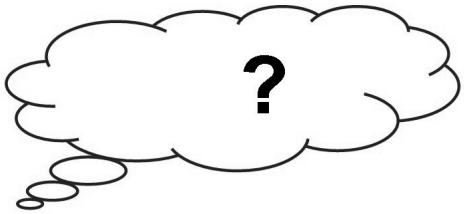


Figure 70: The FIELD Statement and Data Transport

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 block 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?**
- **How can I embed checks that are only executed when a switch is on?**

Figure 71: Conditional Module Calls

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 is 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.

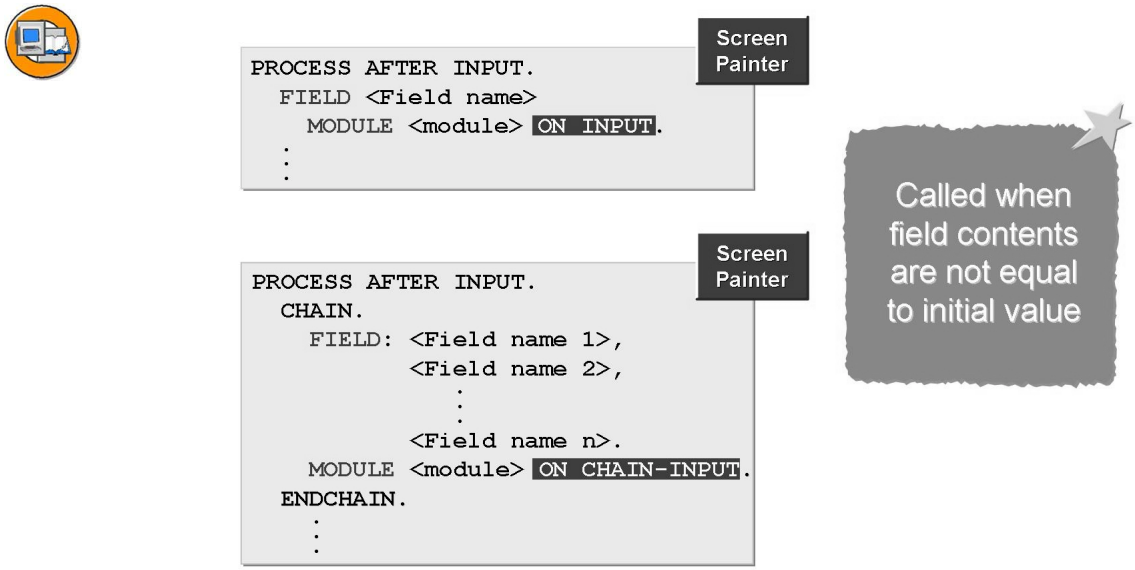


Figure 72: Execution when value is not initial

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 has changed from their initial value.

You may use the ON INPUT addition only if the MODULE statement is contained in a FIELD statement.

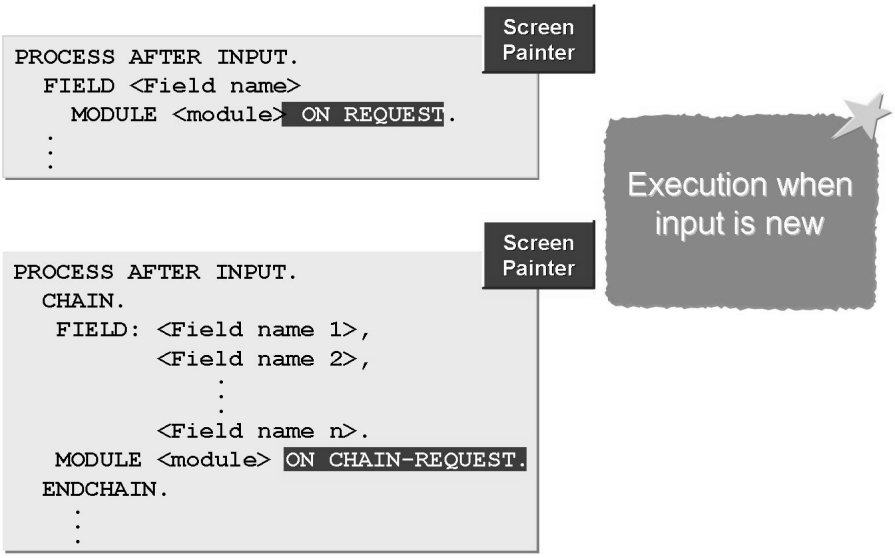


Figure 73: Execution on Change

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.

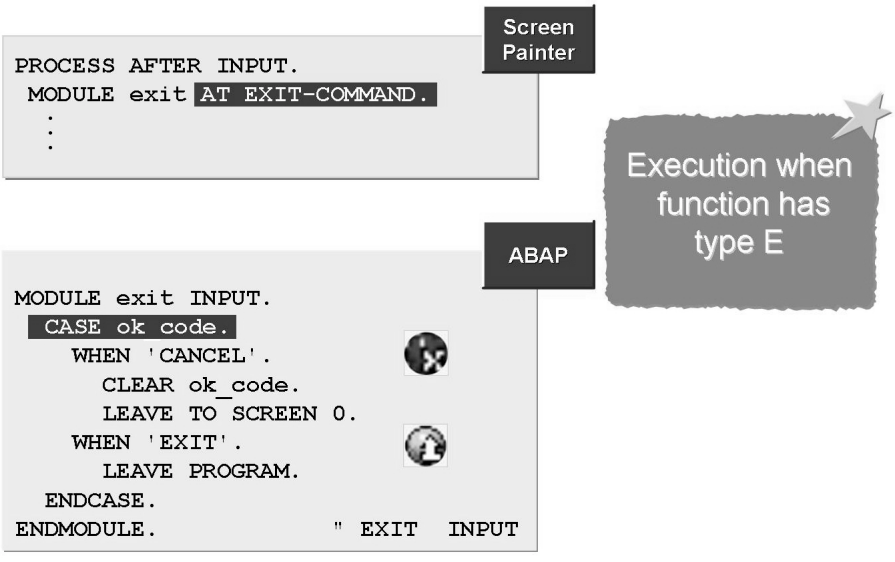


Figure 74: Avoiding Field Input Checks

The module with the AT EXIT-COMMAND addition is processed before the automatic field input checks, before the automatic data transport, and before all other PAI modules. 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.

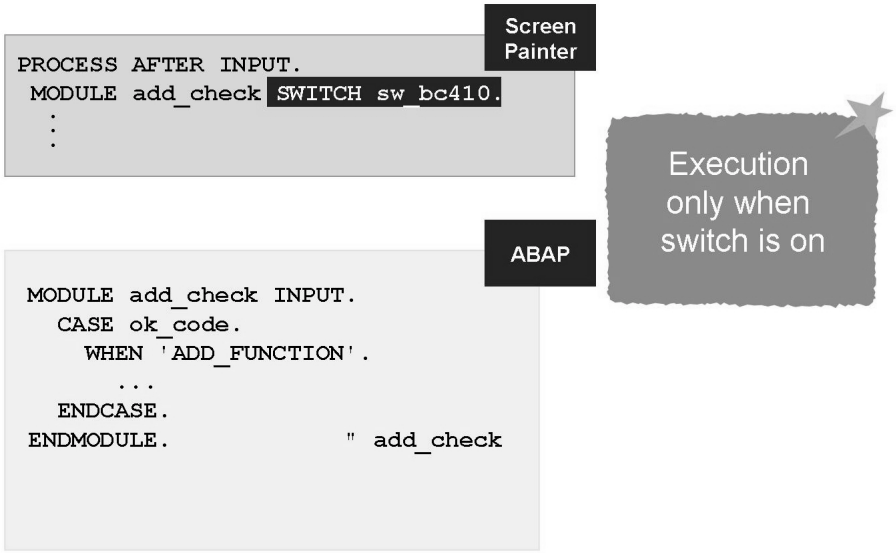


Figure 75: Switch-depending Module Call

If the addition *SWITCH* is specified, the dialog module is called only if the switch specified has the state ON. A switch defined in the Repository must be specified directly. If the specified switch does not exist, the dialog module will not be called.

You cannot specify the addition together with the statement *FIELD*. There, the switch assigned to the screen field in the Screen Painter applies.

Navigation

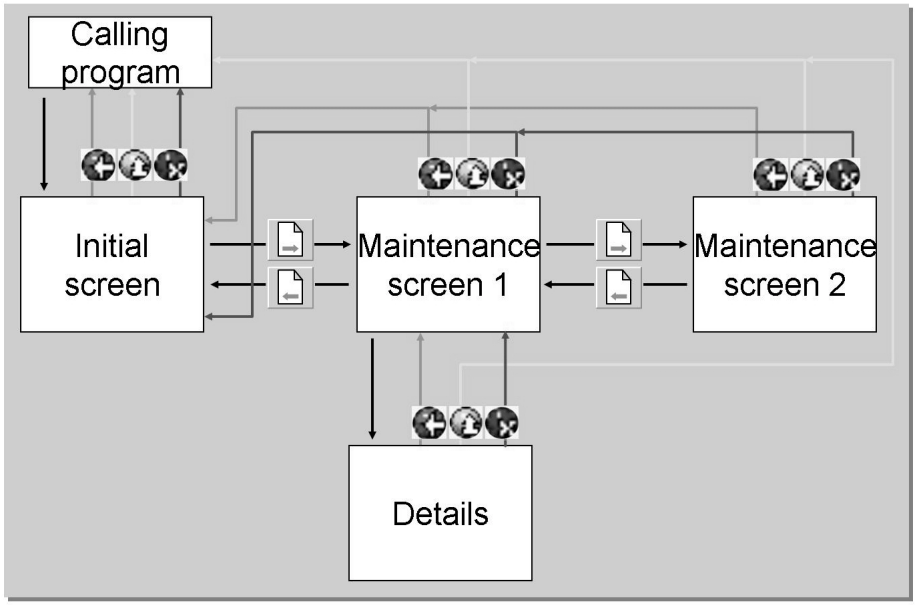


Figure 76: Navigation: Targets

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.

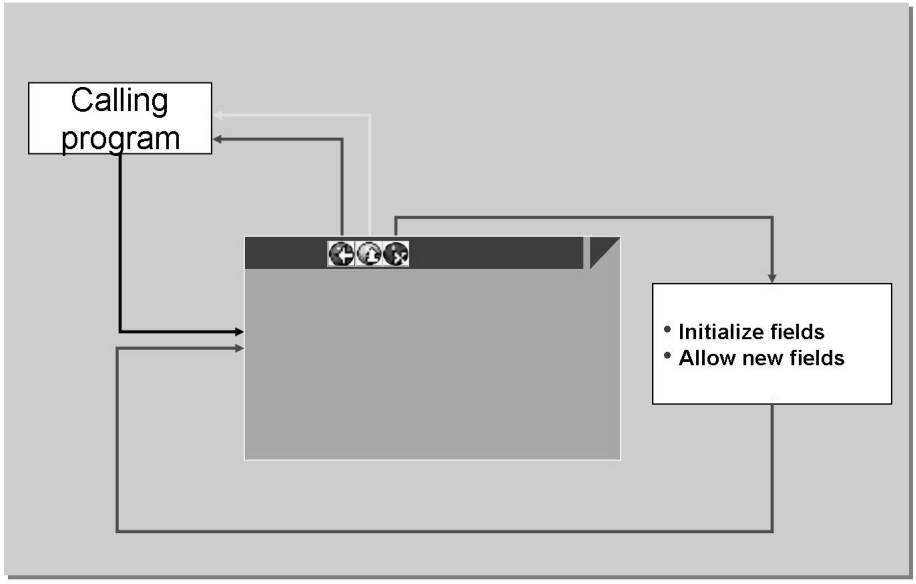


Figure 77: Navigation: Single-Screen Transaction

Back exits the current transaction and returns to the calling program.

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 Change Session	Exit	Cancel
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	`popup_to_confirm`	`popup_to_confirm`	`popup_to_confirm`

Figure 78: Navigation: Dialogs

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 SAP System contains a series of function modules that you can use for the user dialogs.

Input Help

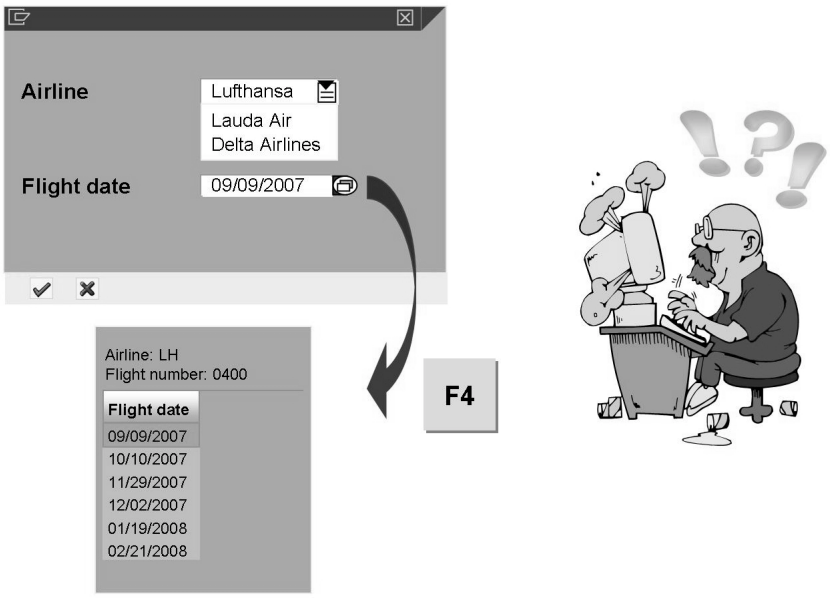


Figure 79: Input Help

Input help (F4 help) is a standard function in the SAP 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.

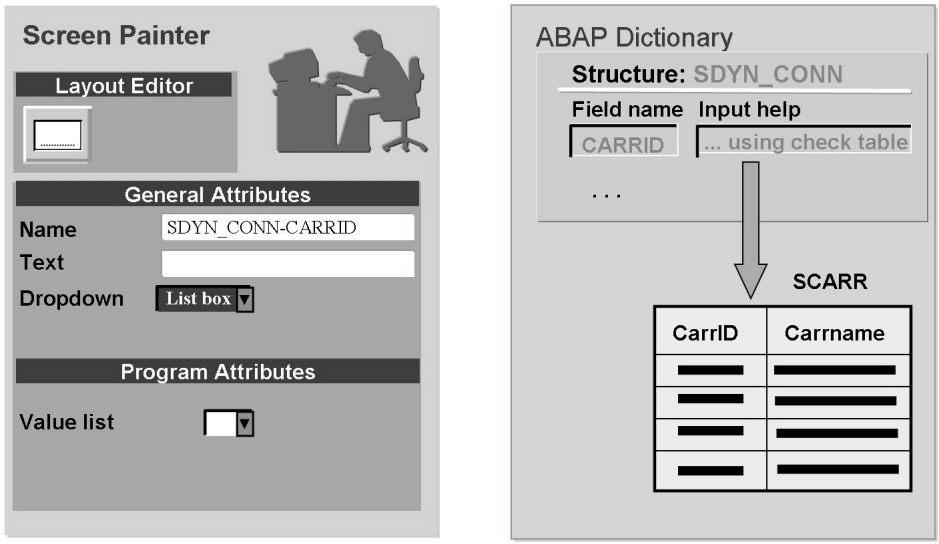


Figure 80: Dropdown Boxes

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* or to *Listbox with key*.

Change the *Visible Length* attribute to the displayed length of the descriptive text.

Set the Value list attribute to ' ' (space) to use value help from the ABAP Dictionary.

Listbox with key (available since *SAP Web Application Server 6.10*) enforces the display of the key as well as of the text, while *List box* leaves the choice to each user whether he likes to see the key or not.

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.

Exercise 3: Input checks, and input help

Exercise Objectives

After completing this exercise, you will be able to:

- Define details for input/output fields

Business Example

Support the user by checking the entries and providing input help.

Task:

Improve the input checks and extend the navigation options on screen 100.

1. Extend your program *SAPMZ##BC410_SOLUTION* from the previous exercise or copy the relevant model solution for the last exercise (*SAPMBC410ADIAS_GUI*).

   You can use the model solution *SAPMBC410AINPS_INPUT_FIELD* for orientation.

2. Ensure that the fields *Airline*, *Flight Number*, and *Flight Date* are automatically filled with values from the SAP memory.

   Change the information message from PAI module *CHECK_SFLIGHT* to an error message. Ensure that the fields are ready for input if the data record does not exist.

   Otherwise, a database select should only be performed if the user has entered new values on the screen.

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.

4. **Optional:** Ensure that appropriate values are read from the table *SFLIGHT* using the SAP Memory when the program is started, without the user having to trigger PAI first.

   Hint: Use the *LOAD-OF-PROGRAM* event to do this and create a new include (using forward navigation).

5. Offer the selection of the airline using a dropdown list: 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 ‘ ’.

Solution 3: Input checks, and input help

Task:

Improve the input checks and extend the navigation options on screen 100.

1. Extend your program *SAPMZ##BC410_SOLUTION* from the previous exercise or copy the relevant model solution for the last exercise (*SAPMBC410ADIAS_GUI*).

   You can use the model solution *SAPMBC410AINPS_INPUT_FIELD* for orientation.

   a) See sample solution.

2. Ensure that the fields *Airline*, *Flight Number*, and *Flight Date* are automatically filled with values from the SAP memory.

   Change the information message from PAI module *CHECK_SFLIGHT* to an error message. Ensure that the fields are ready for input if the data record does not exist.

   Otherwise, a database select should only be performed if the user has entered new values on the screen.

   a) See sample solution.

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.

   a) See sample solution.

4. **Optional:** Ensure that appropriate values are read from the table *SFLIGHT* using the SAP Memory when the program is started, without the user having to trigger PAI first.

   Hint: Use the *LOAD-OF-PROGRAM* event to do this and create a new include (using forward navigation).

   a) See sample solution.

5. Offer the selection of the airline using a dropdown list: 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 ‘ ’.

   a) See sample solution.

Continued on next page

Internal Use SAP Partner Only

Result

Model Solution SAPMBC410AINPS_INPUT_FIELD

Main program

```
* Data objects
INCLUDE MBC410AINPS_INPUT_FIELDTOP.

* PAI modules
INCLUDE MBC410AINPS_INPUT_FIELDI01.

* PBO modules
INCLUDE MBC410AINPS_INPUT_FIELDO01.

* ABAP events
INCLUDE MBC410AINPS_INPUT_FIELDE02.
```

Flow logic screen 100

```
PROCESS BEFORE OUTPUT.
  MODULE status_0100.
  MODULE move_to_dynp.
  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.
  MODULE user_command_0100.
```

Top include

No changes with regard to previous exercise.

PBO module include

No changes with regard to previous exercise.

PAI module include

```
*&---------------------------------------------------------------------*
*&      Module  check_sflight  INPUT
*&---------------------------------------------------------------------*
*       Read flight record from database
*---------------------------------------------------------------------*
```

Continued on next page

```
MODULE check_sflight INPUT.
  SELECT SINGLE *
    FROM sflight
* INTO CORRESPONDING FIELDS OF sdyn_conn  " direct read
    INTO wa_sflight         " Read into internal structure
    WHERE carrid = sdyn_conn-carrid AND
          connid = sdyn_conn-connid AND
          fldate = sdyn_conn-fldate.
  CHECK sy-subrc <> 0.
  CLEAR wa_sflight.
  MESSAGE e007(bc410).

ENDMODULE.                 " check_sflight  INPUT
*&---------------------------------------------------------------------*
*&      Module  user_command_0100  INPUT
*&---------------------------------------------------------------------*
*       process user command
*---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.

* display time on add'l screen
    WHEN 'TIME'.
      CALL SCREEN 150
        STARTING AT 10 10
        ENDING   AT 50 20.

  ENDCASE.
ENDMODULE.                 " user_command_0100  INPUT
*&---------------------------------------------------------------------*
*&      Module  exit  INPUT
*&---------------------------------------------------------------------*
*       process EXIT functions (type 'E')
*---------------------------------------------------------------------*
MODULE exit INPUT.
  CASE ok_code.
    WHEN 'CANCEL'.
      CLEAR wa_sflight.
      SET PARAMETER ID:
        'CAR' FIELD wa_sflight-carrid,
        'CON' FIELD wa_sflight-connid,
        'DAY' FIELD wa_sflight-fldate.
      LEAVE TO SCREEN 100.
```

Continued on next page


```
    WHEN 'EXIT'.
      LEAVE PROGRAM.
  ENDCASE.

ENDMODULE.                 " exit  INPUT
```

Event Include

```
LOAD-OF-PROGRAM.
* get stored values from SAP memory
  GET PARAMETER ID:
    'CAR' FIELD sdyn_conn-carrid,
    'CON' FIELD sdyn_conn-connid,
    'DAY' FIELD sdyn_conn-fldate.

* retrieve detail data of flight
  SELECT SINGLE *
    FROM sflight
    INTO wa_sflight
    WHERE carrid = sdyn_conn-carrid AND
          connid = sdyn_conn-connid AND
          fldate = sdyn_conn-fldate.
```


Internal Use SAP Partner Only

Lesson Summary

You should now be able to:

- Create input/output fields
- Check errors in input/output fields
- Create a dropdown box for an input field

Lesson: Checkboxes, Radio Button Groups, and Pushbuttons

Lesson Overview

In this lesson, you will learn to create, use, and change the attributes of checkboxes, radio button groups, and pushbuttons. In addition, you will learn that you can assign a function code to a radio button after you have defined a radio button group. Finally, you will learn how to process pushbuttons.

Lesson Objectives

After completing this lesson, you will be able to:

- Create checkboxes in your programs
- Create radio button groups
- Create pushbuttons

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. The application should enable users to interact with the program, create checkboxes, radio button groups, and pushbuttons as screen elements.

Checkboxes and Radio Button Groups

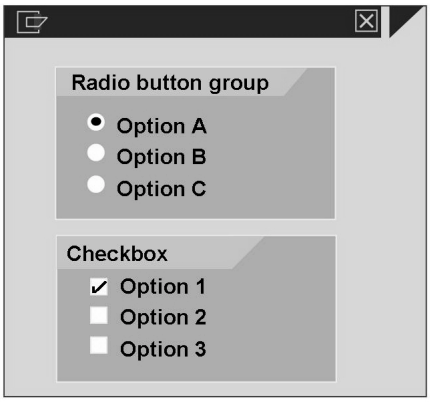


Figure 81: Checkboxes and Radio Button Groups

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.

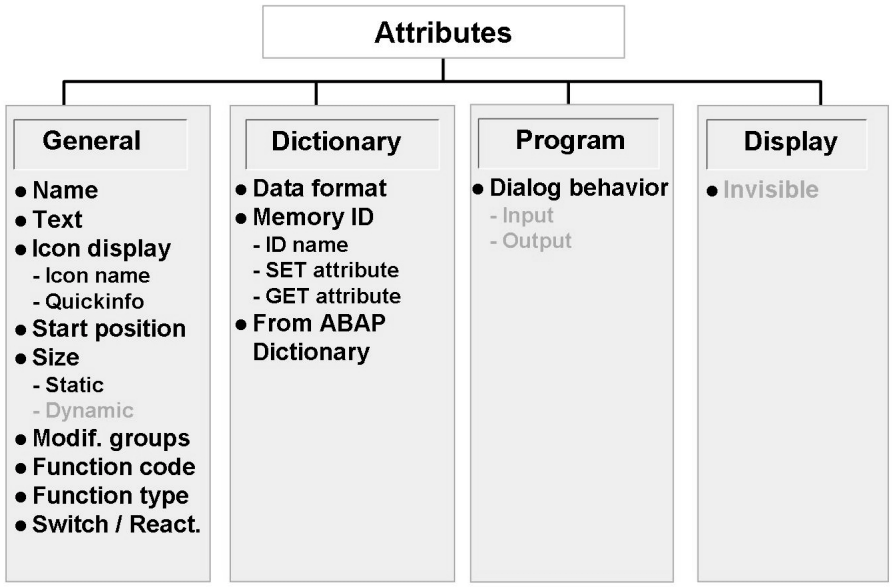


Figure 82: Radio Buttons and Checkboxes: Attributes

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.

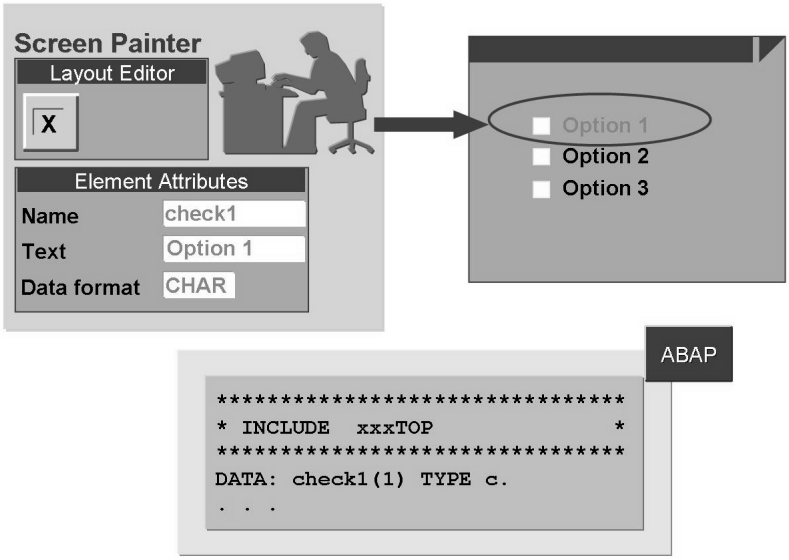


Figure 83: Creating a Checkbox

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.

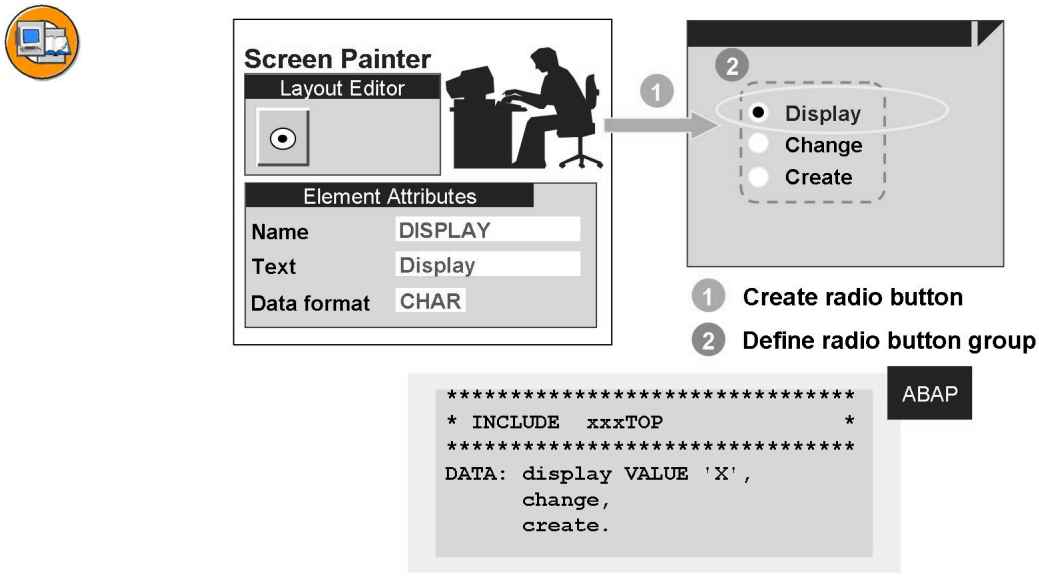


Figure 84: Creating a Radio Button Group

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.

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.

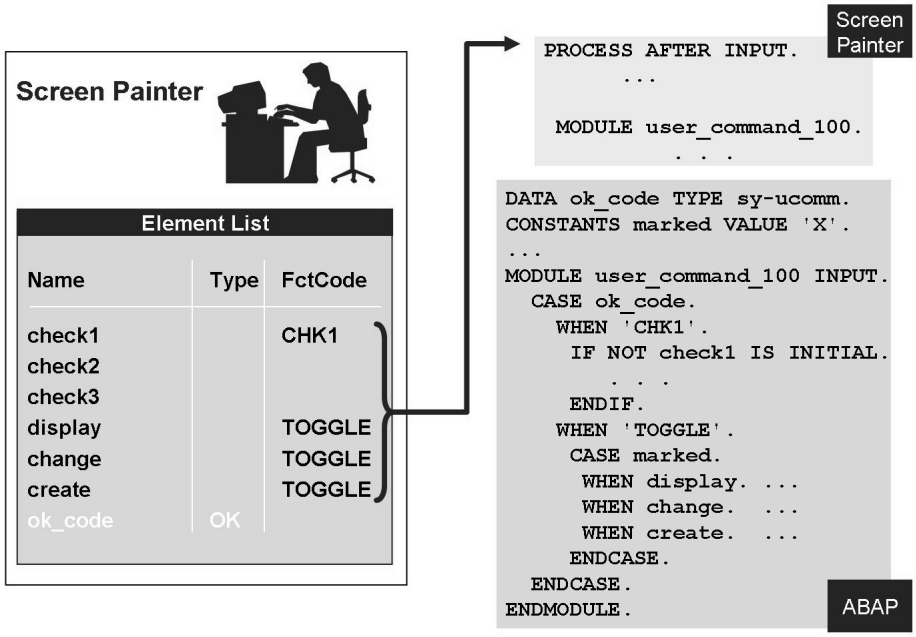
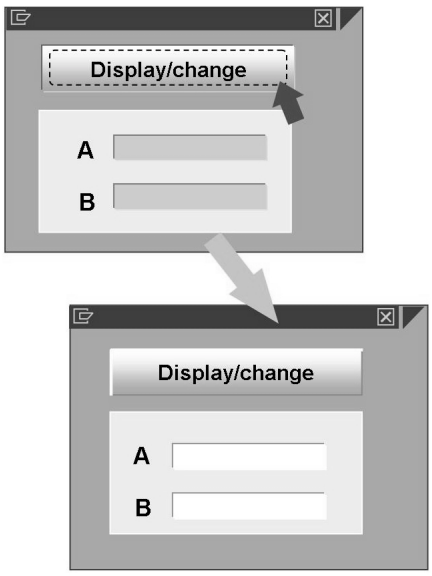


Figure 85: Program Flow for Radio Buttons and Checkboxes

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.

Pushbuttons



User input using the mouse

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

Figure 86: Pushbuttons

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.

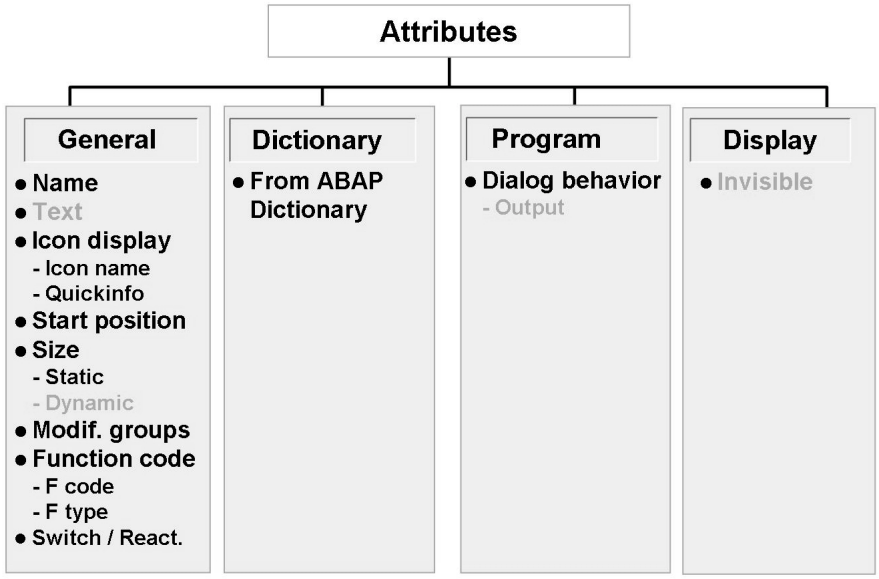


Figure 87: Pushbuttons: Attributes

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.

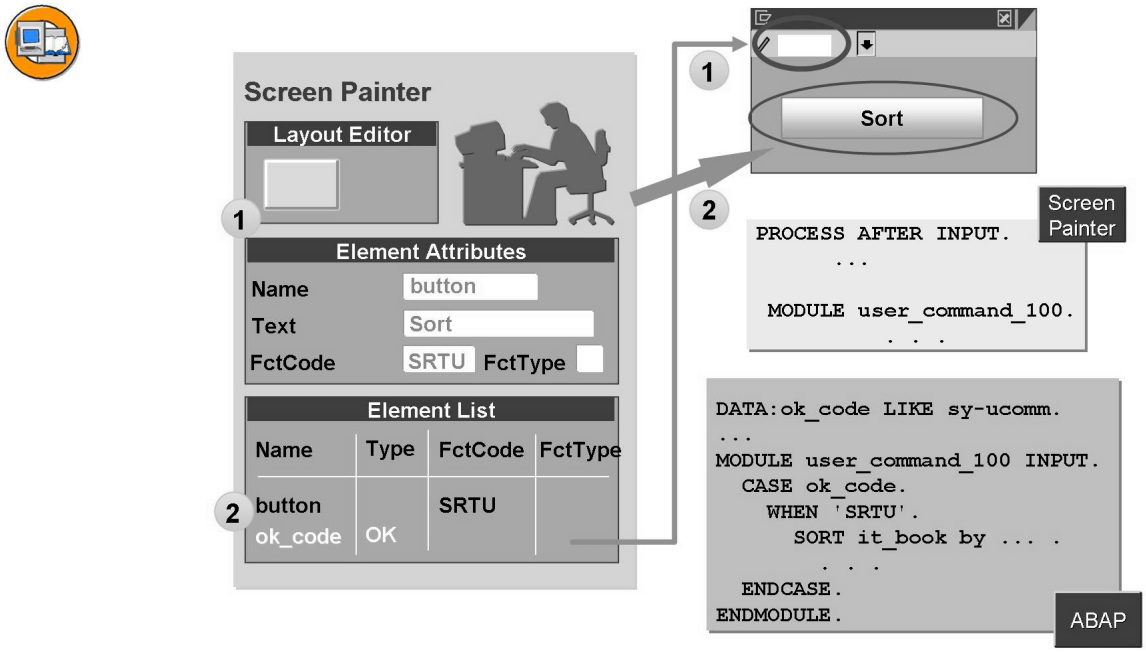


Figure 88: Creating and Processing Pushbuttons

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.

Exercise 4: Radio buttons and Input checks

Exercise Objectives

After completing this exercise, you will be able to:

- Create radio buttons and program the relevant logic
- Make dynamic changes to screens

Business Example

All the users are to be able to switch between different program modes:

- Display mode
- Flight data maintenance where the user can change the aircraft type
- Maintain bookings

Task:

Create a radio button group to allow the user to choose one of a range of program modes.

1. Extend your program *SAPMZ##BC410_SOLUTION* from the previous exercise or copy the previous model solution *SAPMBC410AINPS_INPUT_FIELD*. You can use the model solution *SAPMBC410AINPS_RADIOBUTTON* for orientation.

2. On screen 100, create a radio button group with the buttons *view*, *maintain_flights*, and *maintain_bookings*. Ensure that the function code MODE (with type ' ') is triggered when the user chooses a different mode.

   Declare the relevant data fields in your top include. Create a group box around the radio button group and assign it the text "Mode".

3. Set the GUI title in accordance with the mode chosen by the user. (Reminder: Your GUI title should have a parameter, meaning that you do not need to create a new title now.)

4. Program the Maintain flight data mode. In this mode, the input/output field *SDYN_CONN-PLANETYPE* should be ready for input. Code the relevant dynamic screen modification in a module called *modify_screen*.

   Optional:

Continued on next page

Internal Use SAP Partner Only

If the user enters a new aircraft type, 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*. If an error occurs, display message **109** from message class **BC410** as an error message. If an error occurs, transport the maximum number of seats back to the screen.

5. **Optional:** The new aircraft type is to be saved.

   Assign the function code SAVE (function type ' ') to the standard key CTRL-S (Save). The function should only be available if the radio button is set to Maintain flight data. (Tip: Use the addition `EXCLUDING` at the command `SET PF-STATUS.`)

   If the user chooses this function, save the new flight data record in the database. To do this, use a direct database update in the form:

```
UPDATE sflight FROM wa_sflight.
IF sy-subrc NE 0.
  MESSAGE a008(bc410).
ENDIF.
MESSAGE s009(bc410).
```

Solution 4: Radio buttons and Input checks

Task:

Create a radio button group to allow the user to choose one of a range of program modes.

1. Extend your program *SAPMZ##BC410_SOLUTION* from the previous exercise or copy the previous model solution *SAPMBC410AINPS_INPUT_FIELD*. You can use the model solution *SAPMBC410AINPS_RADIOBUTTON* for orientation.

   a) See sample solution.

2. On screen 100, create a radio button group with the buttons *view*, *maintain_flights*, and *maintain_bookings*. Ensure that the function code MODE (with type ' ') is triggered when the user chooses a different mode.

   Declare the relevant data fields in your top include. Create a group box around the radio button group and assign it the text "Mode".

   a) See sample solution.

3. Set the GUI title in accordance with the mode chosen by the user. (Reminder: Your GUI title should have a parameter, meaning that you do not need to create a new title now.)

   a) See sample solution.

4. Program the Maintain flight data mode. In this mode, the input/output field *SDYN_CONN-PLANETYPE* should be ready for input. Code the relevant dynamic screen modification in a module called *modify_screen*.

   Optional:

   If the user enters a new aircraft type, 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*. If an error occurs, display message **109** from message class **BC410** as an error message. If an error occurs, transport the maximum number of seats back to the screen.

   a) See sample solution.

5. **Optional:** The new aircraft type is to be saved.

   Assign the function code SAVE (function type ' ') to the standard key CTRL-S (Save). The function should only be available if the radio button is set to Maintain flight data. (Tip: Use the addition `EXCLUDING` at the command `SET PF-STATUS.`)

Continued on next page

If the user chooses this function, save the new flight data record in the database. To do this, use a direct database update in the form:

```
UPDATE sflight FROM wa_sflight.
IF sy-subrc NE 0.
  MESSAGE a008(bc410).
ENDIF.
MESSAGE s009(bc410).
```

a) See sample solution.

Result

Model Solution SAPMBC410AINPS_RADIOBUTTON

Main program

No changes with regard to previous exercise.

Flow logic screen 100

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

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-seatsmax,
      sdyn_conn-planetype.
    MODULE check_planetype ON CHAIN-REQUEST.
  ENDCHAIN.

  MODULE user_command_0100.
```

Top include

Continued on next page

Internal Use SAP Partner Only

```
PROGRAM  sapmbc410adias_dynpro.
* screen structure
TABLES: sdyn_conn.

DATA:
* workarea for database access
  wa_sflight TYPE sflight,
* function code at PAI
  ok_code    LIKE sy-ucomm,
* radio buttons
  view VALUE 'X',
  maintain_flights,
  maintain_bookings.
```

PBO module include

```
*&---------------------------------------------------------------------*
*&      Module  move_to_dynp  OUTPUT
*&---------------------------------------------------------------------*
*       copy data to screen structure
*---------------------------------------------------------------------*
MODULE move_to_dynp OUTPUT.
  MOVE-CORRESPONDING wa_sflight TO sdyn_conn.
ENDMODULE.                 " move_to_dynp  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  status_0100  OUTPUT
*&---------------------------------------------------------------------*
*       set status and title for screen 100
*---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  CASE 'X'.
    WHEN view.
      SET TITLEBAR 'TITLE_100' WITH 'Display'(vie).
    WHEN maintain_flights.
      SET TITLEBAR 'TITLE_100' WITH 'Maintain Flights'(fli).
    WHEN maintain_bookings.
      SET TITLEBAR 'TITLE_100' WITH 'Maintain Bookings'(boo).
  ENDCASE.

ENDMODULE.                 " status_0100  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  status_0150  OUTPUT
*&---------------------------------------------------------------------*
*       set status and title for screen 150
*---------------------------------------------------------------------*
MODULE status_0150 OUTPUT.
```

Continued on next page

```
  SET PF-STATUS 'STATUS_150'.
  SET TITLEBAR  'TITLE_150' WITH text-vie.
ENDMODULE.                 " status_0150  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  clear_ok_code  OUTPUT
*&---------------------------------------------------------------------*
*       initialize ok_code
*---------------------------------------------------------------------*
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.                 " clear_ok_code  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  modify_screen  OUTPUT
*&---------------------------------------------------------------------*
*       change elements dynamically
*---------------------------------------------------------------------*
MODULE modify_screen OUTPUT.
  IF maintain_flights = 'X'.
    SET PF-STATUS 'STATUS_100'.
    LOOP AT SCREEN.
      IF screen-name = 'SDYN_CONN-PLANETYPE'.
        screen-input = 1.
        MODIFY SCREEN.
      ENDIF.
    ENDLOOP.
  ELSE.
    SET PF-STATUS 'STATUS_100' EXCLUDING 'SAVE'.
  ENDIF.

ENDMODULE.                 " modify_screen  OUTPUT
```

PAI module include

```
*&---------------------------------------------------------------------*
*&      Module  check_sflight  INPUT
*&---------------------------------------------------------------------*
*       Read flight record from database
*---------------------------------------------------------------------*
MODULE check_sflight INPUT.
  SELECT SINGLE *
    FROM sflight
* INTO CORRESPONDING FIELDS OF sdyn_conn  " direct read
    INTO wa_sflight         " Read into internal structure
    WHERE carrid = sdyn_conn-carrid AND
          connid = sdyn_conn-connid AND
          fldate = sdyn_conn-fldate.
```

Continued on next page

```
  CHECK sy-subrc <> 0.
  CLEAR wa_sflight.
  MESSAGE e007(bc410).

ENDMODULE.                 " check_sflight  INPUT
*&---------------------------------------------------------------------*
*&      Module  user_command_0100  INPUT
*&---------------------------------------------------------------------*
*       process user command
*---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'BACK'.
      LEAVE TO SCREEN 0.

* display time on add'l screen
    WHEN 'TIME'.
      CALL SCREEN 150
        STARTING AT 10 10
        ENDING   AT 50 20.

* save changes to database
    WHEN 'SAVE'.
      UPDATE sflight
      FROM wa_sflight.
      IF sy-subrc <> 0.
        MESSAGE a008(bc410).
      ENDIF.
      MESSAGE s009(bc410).

  ENDCASE.
ENDMODULE.                 " user_command_0100  INPUT
*&---------------------------------------------------------------------*
*&      Module  exit  INPUT
*&---------------------------------------------------------------------*
*       process EXIT functions (type 'E')
*---------------------------------------------------------------------*
MODULE exit INPUT.
  CASE ok_code.
    WHEN 'CANCEL'.
      CLEAR wa_sflight.
      SET PARAMETER ID:
        'CAR' FIELD wa_sflight-carrid,
        'CON' FIELD wa_sflight-connid,
        'DAY' FIELD wa_sflight-fldate.
```

Continued on next page

```
      LEAVE TO SCREEN 100.
    WHEN 'EXIT'.
      LEAVE PROGRAM.
  ENDCASE.

ENDMODULE.                 " exit  INPUT
*&---------------------------------------------------------------------*
*&      Module  check_planetype  INPUT
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
MODULE check_planetype INPUT.
  IF sdyn_conn-planetype IS INITIAL.
    MESSAGE e555(bc410) WITH 'Plane type needed'(plt).
  ENDIF.

  SELECT SINGLE seatsmax
    FROM saplane
    INTO sdyn_conn-seatsmax
    WHERE planetype = sdyn_conn-planetype.

  IF sdyn_conn-seatsocc > sdyn_conn-seatsmax.
    MESSAGE e109(bc410).
*   Number of seats booked exceeds aircraft capacity
  ELSE.
    MOVE-CORRESPONDING sdyn_conn TO wa_sflight.
  ENDIF.

ENDMODULE.                 " check_planetype  INPUT
```

Lesson Summary

You should now be able to:

- Create checkboxes in your programs
- Create radio button groups
- Create pushbuttons

Unit Summary

You should now be able to:

- Create input/output fields
- Check errors in input/output fields
- Create a dropdown box for an input field
- Create checkboxes in your programs
- Create radio button groups
- Create pushbuttons

Test Your Knowledge

1. The following message appears when the processing is interrupted and the user can correct the entries:
   Choose the correct answer(s).
   - □ A E Error
   - □ B W Warning
   - □ C I Information
   - □ D X Exit

2. The _____ of the field determines the width of the field, including the button, and the selection list.
   Fill in the blanks to complete the sentence.

3. The Back function differs from Cancel in its dialog behavior.
   Determine whether this statement is true or false.
   - □ True
   - □ False

4. List the steps required to create a pushbutton.

   _____
   _____
   _____
   _____

Internal Use SAP Partner Only

Answers

1. The following message appears when the processing is interrupted and the user can correct the entries:

   Answer: B

   Message 'W Warning' appears when the processing is interrupted and the user can correct the entries.

2. The visible length of the field determines the width of the field, including the button, and the selection list.

   Answer: visible length

3. The Back function differs from Cancel in its dialog behavior.

   Answer: False

   The Cancel function differs from Back in its dialog behavior.

4. List the steps required to create a pushbutton.

   Answer: 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.

Unit 5

Screen Elements: Subscreens and Tabstrip Controls

Unit Overview

This unit covers basics of using subscreens and tabstrip controls. Two techniques to use subscreens for screen modularization are discussed. The mechanism to allow the user to display related information on a single screen without having to navigate through multiple detail screens using tabstrip control is explained in this unit.

Unit Objectives

After completing this unit, you will be able to:

- Create subscreens on screens
- Use function groups in subscreens
- Create tabstrip controls
- Scroll in tabstrip controls

Unit Contents

Lesson: Subscreens

Lesson Overview

This lesson explains that a subscreen area is a reserved rectangular area on a screen in which you place another screen at runtime. You will learn about the general and subscreen attributes of the subscreen area. You will also learn how to create and call a subscreen area and how to encapsulate subscreens in a function group.

Lesson Objectives

After completing this lesson, you will be able to:

- Create subscreens on screens
- Use function groups in subscreens

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. For these screens, you need to create subscreens to display additional information.

Using Subscreens

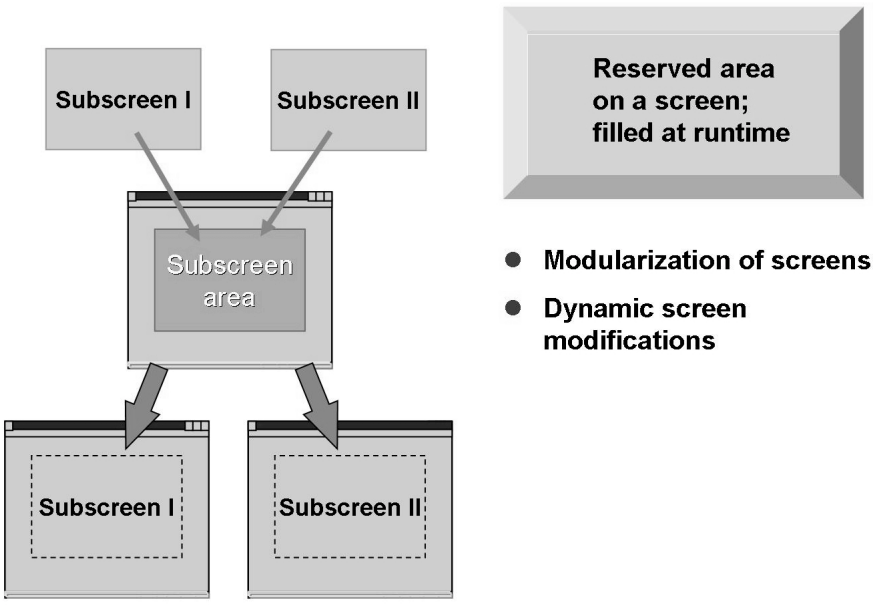


Figure 89: Subscreens (1)

A subscreen area is a reserved rectangular area on a screen, in which you place another screen at runtime. 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.

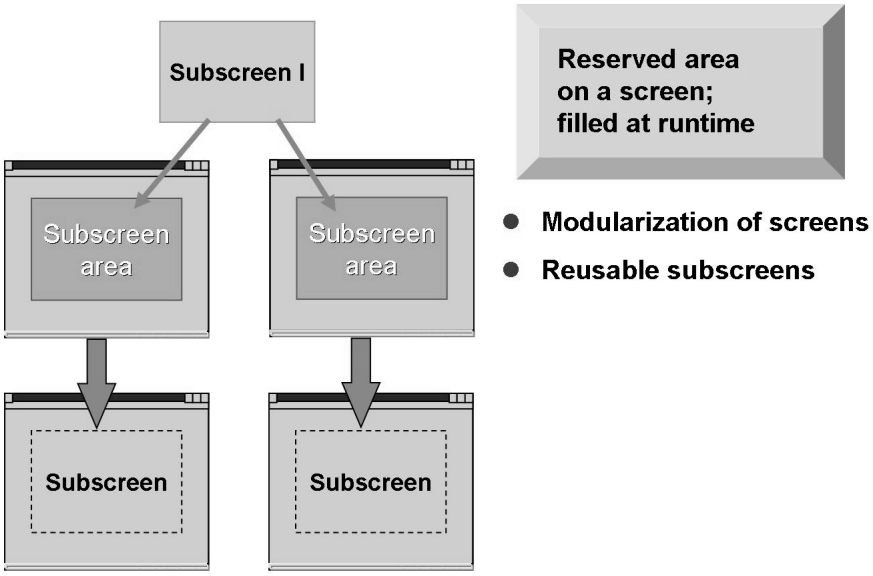


Figure 90: Subscreens (2)

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 runtime.

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

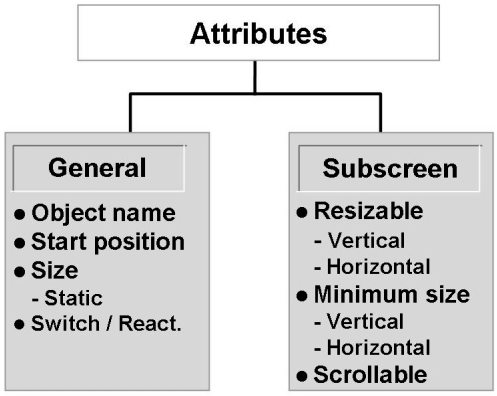


Figure 91: Subscreen Area: Attributes

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.

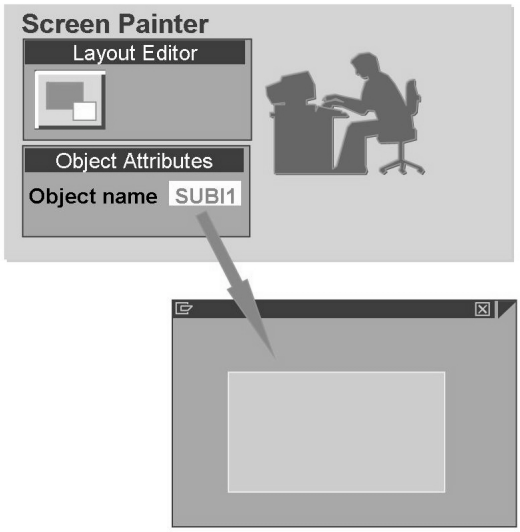


Figure 92: Creating a Subscreen Area

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.

Processing Subscreens

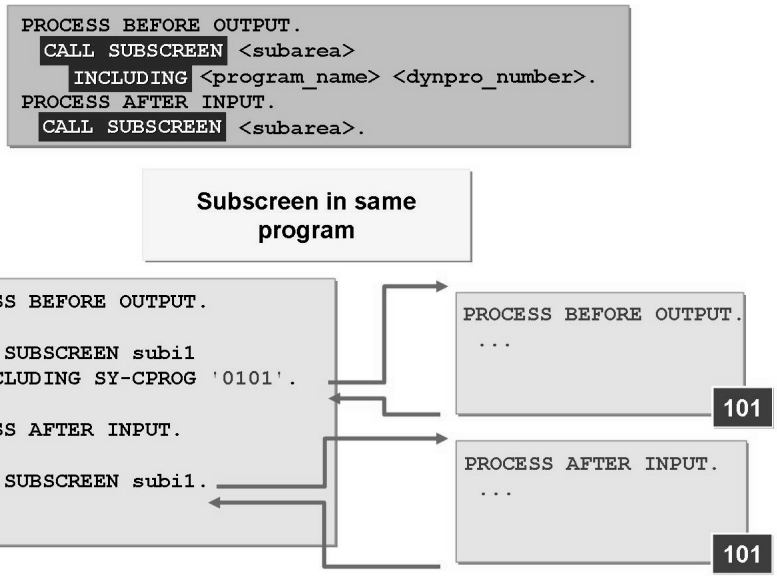


Figure 93: Calling a Subscreen

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 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).

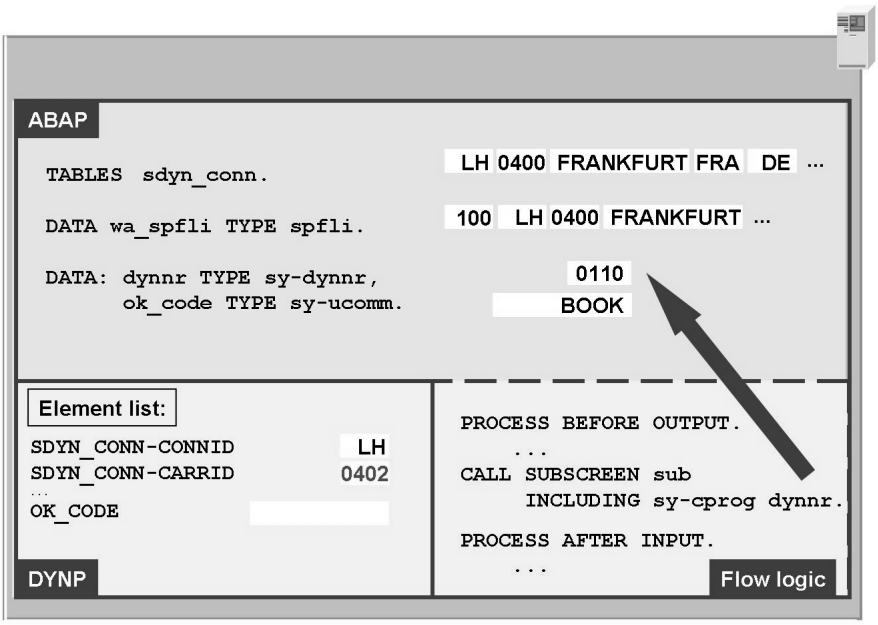


Figure 94: Special Case: Visibility of Data

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

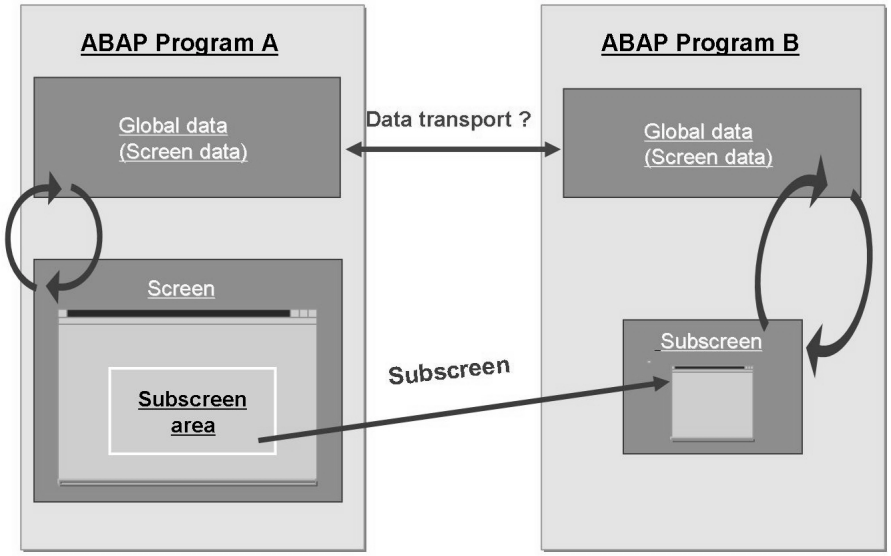


Figure 95: Subscreens from External Programs

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 in Function Groups

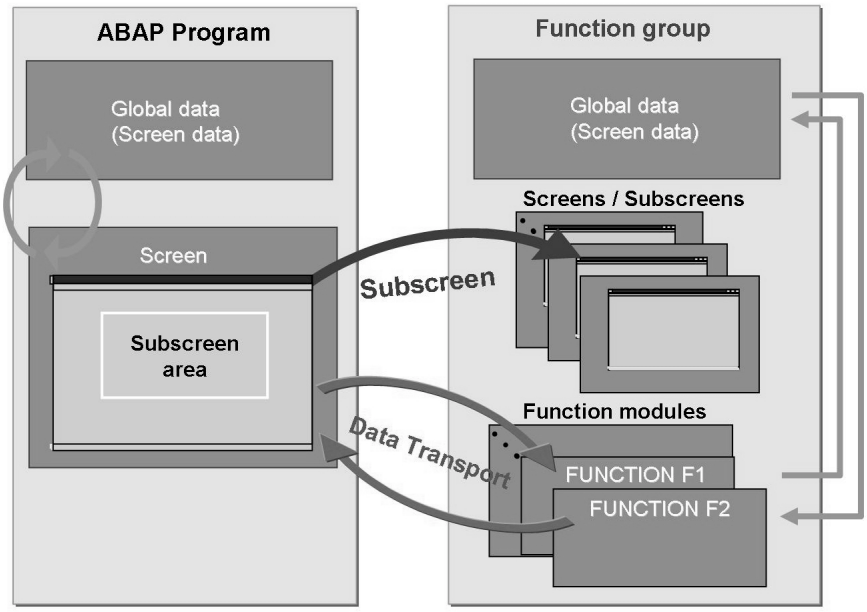


Figure 96: Subscreens: Encapsulation in Function Groups

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).

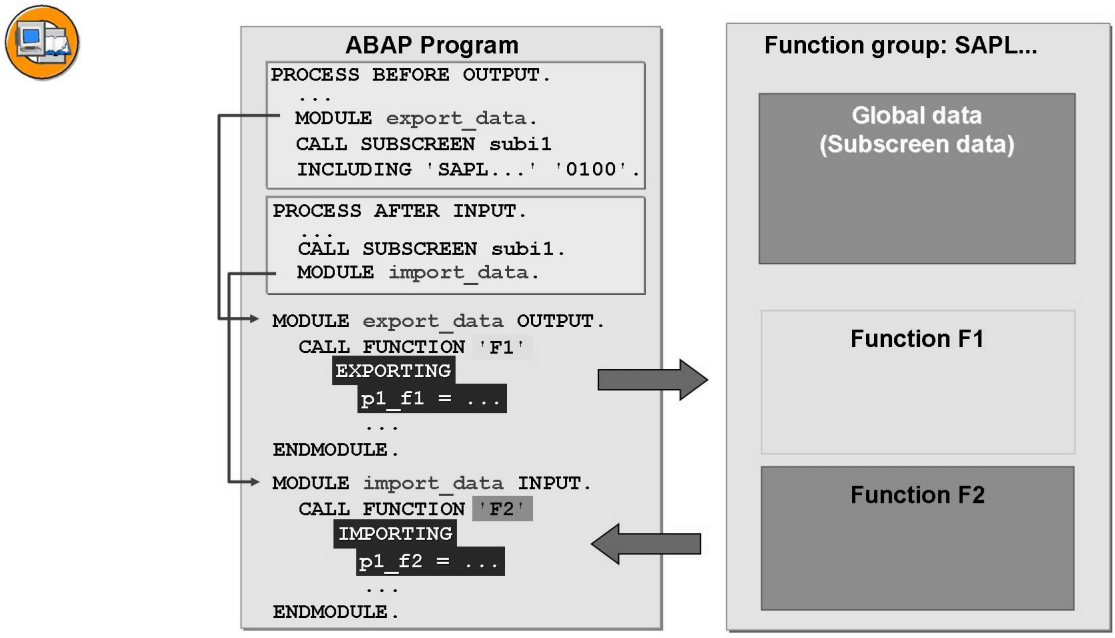


Figure 97: Subscreens in Function Groups: Call Sequence

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.

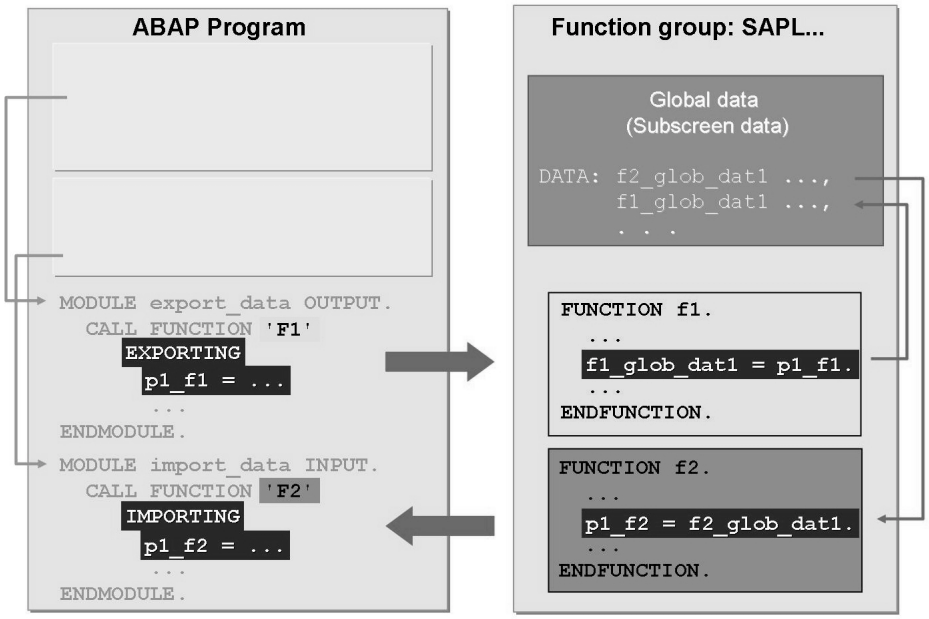


Figure 98: Subscreens in Function Groups: Data Transport

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 the 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.

Exercise 5: Subscreens

Exercise Objectives

After completing this exercise, you will be able to:

- Use subscreens on screens in your programs

Business Example

To display additional information on your screen, depending on the mode in which the user is working, add subscreens in the maintenance screen. Create three subscreens to display the view flight data, technical details for aircraft and bookings.

Task:

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

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

Subscreen	SUB	**Attributes:** Vertical and horizontal Resizing: ON

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

Continued on next page

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

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.

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.

Continued on next page

6. In the flow logic of screen 110, read the flight information from table *SPFLI* using the key field values.

   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.

Internal Use SAP Partner Only

Solution 5: Subscreens

Task:

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

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.

   a) See sample solution.

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

Subscreen	SUB	**Attributes:** Vertical and horizontal Resizing: ON

   a) See sample solution.

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

Continued on next page

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

a) See sample solution.

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.

   a) See sample solution.

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

Continued on next page

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.

a) See sample solution.

6. In the flow logic of screen 110, read the flight information from table *SPFLI* using the key field values.

   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.

   a) See sample solution.

Result

Model Solution SAPMBC410ASUBS_SUBSCREEN

Main program

No changes with regard to previous exercise.

Flow logic screen 100

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

  MODULE fill_dynnr.

  CALL SUBSCREEN sub INCLUDING sy-cprog dynnr.

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-seatsmax,
      sdyn_conn-planetype.
```

Continued on next page


```
    MODULE check_planetype ON CHAIN-REQUEST.
  ENDCHAIN.

  MODULE user_command_0100.
```

Flow logic screen 110

```
PROCESS BEFORE OUTPUT.
  MODULE get_spfli.
*
PROCESS AFTER INPUT.
* MODULE USER_COMMAND_0110.
```

Flow logic screen 120

```
PROCESS BEFORE OUTPUT.
  module get_saplane.
*
PROCESS AFTER INPUT.
* MODULE USER_COMMAND_0120.
```

Flow logic screen 130

Remains empty yet.

Top include

```
PROGRAM  sapmbc410adias_dynpro.
* screen structure
TABLES: sdyn_conn,
        saplane.

DATA:
* workarea for database read
  wa_sflight TYPE sflight,
* function code at PAI
  ok_code    LIKE sy-ucomm,
* radio buttons
  view       VALUE 'X',
  maintain_flights,
  maintain_bookings,
* subscreen number
  dynnr      TYPE sy-dynnr.
```

PBO module include

```
*&---------------------------------------------------------------------*
*&      Module  move_to_dynp  OUTPUT
*&---------------------------------------------------------------------*
*       copy data to screen structure
```

Continued on next page

```
*---------------------------------------------------------------------*
MODULE move_to_dynp OUTPUT.
  MOVE-CORRESPONDING wa_sflight TO sdyn_conn.
ENDMODULE.                 " move_to_dynp  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  status_0100  OUTPUT
*&---------------------------------------------------------------------*
*       set status and title for screen 100
*---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'STATUS_100'.
  SET TITLEBAR  'TITLE_100' WITH text-vie.

  CASE 'X'.
    WHEN view.
      SET TITLEBAR 'TITLE_100' WITH 'Display'(vie).
    WHEN maintain_flights.
      SET TITLEBAR 'TITLE_100' WITH 'Maintain Flights'(fli).
    WHEN maintain_bookings.
      SET TITLEBAR 'TITLE_100' WITH 'Maintain Bookings'(boo).
  ENDCASE.

ENDMODULE.                 " status_0100  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  status_0150  OUTPUT
*&---------------------------------------------------------------------*
*       set status and title for screen 150
*---------------------------------------------------------------------*
MODULE status_0150 OUTPUT.
  SET PF-STATUS 'STATUS_150'.
  SET TITLEBAR  'TITLE_150' WITH text-vie.
ENDMODULE.                 " status_0150  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  clear_ok_code  OUTPUT
*&---------------------------------------------------------------------*
*       initialize ok_code
*---------------------------------------------------------------------*
MODULE clear_ok_code OUTPUT.
  CLEAR ok_code.
ENDMODULE.                 " clear_ok_code  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  modify_screen  OUTPUT
*&---------------------------------------------------------------------*
*       change elements dynamically
*---------------------------------------------------------------------*
```

Continued on next page


```
MODULE modify_screen OUTPUT.
  IF maintain_flights = 'X'.
    SET PF-STATUS 'STATUS_100'.
    LOOP AT SCREEN.
      IF screen-name = 'SDYN_CONN-PLANETYPE'.
        screen-input = 1.
        MODIFY SCREEN.
      ENDIF.
    ENDLOOP.
  ELSE.
    SET PF-STATUS 'STATUS_100' EXCLUDING 'SAVE'.
  ENDIF.

ENDMODULE.                 " modify_screen  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  fill_dynnr  OUTPUT
*&---------------------------------------------------------------------*
*       determine subscreen number
*---------------------------------------------------------------------*
MODULE fill_dynnr OUTPUT.
  CASE 'X'.
    WHEN view.
      dynnr = 110.
    WHEN maintain_flights.
      dynnr = 120.
    WHEN maintain_bookings.
      dynnr = 130.
  ENDCASE.
ENDMODULE.                 " fill_dynnr  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  get_spfli  OUTPUT
*&---------------------------------------------------------------------*
*       read SPFLI data from database
*---------------------------------------------------------------------*
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
```

Continued on next page

```
*&---------------------------------------------------------------------*
*       read SAPLANE data from database
*---------------------------------------------------------------------*
MODULE get_saplane OUTPUT.
  ON CHANGE OF wa_sflight-planetype.
    SELECT SINGLE * FROM saplane
    WHERE planetype = wa_sflight-planetype.
  ENDON.
ENDMODULE.                 " get_saplane  OUTPUT
```

PAI module include

No changes with regard to previous exercise.

Lesson Summary

You should now be able to:

- Create subscreens on screens
- Use function groups in subscreens

Lesson: Tabstrip Controls

Lesson Overview

This lesson explains the creation and uses of tabstrip controls, which 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. In addition, you will learn how to scroll between the pages locally at the front end. Finally, you learn how to process scrolling in a tabstrip control using a program.

Lesson Objectives

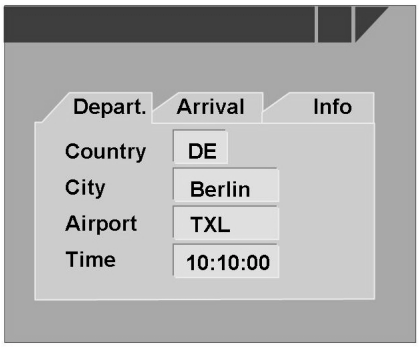
After completing this lesson, you will be able to:

- Create tabstrip controls
- Scroll in tabstrip controls

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. In the screens, users should be able to create tabstrip controls to display various components of an application on a single screen. They should also be able to navigate between components.

Screen Elements: Tabstrip Controls



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**

Figure 99: Screen Elements: Tabstrip Controls

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.

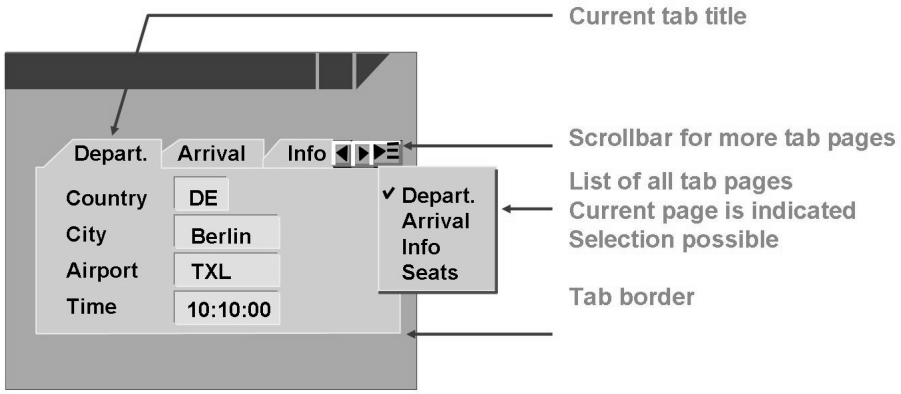


Figure 100: Tabstrip Elements

A tabstrip control consists of individual pages with a tab page and tab title.

A 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.

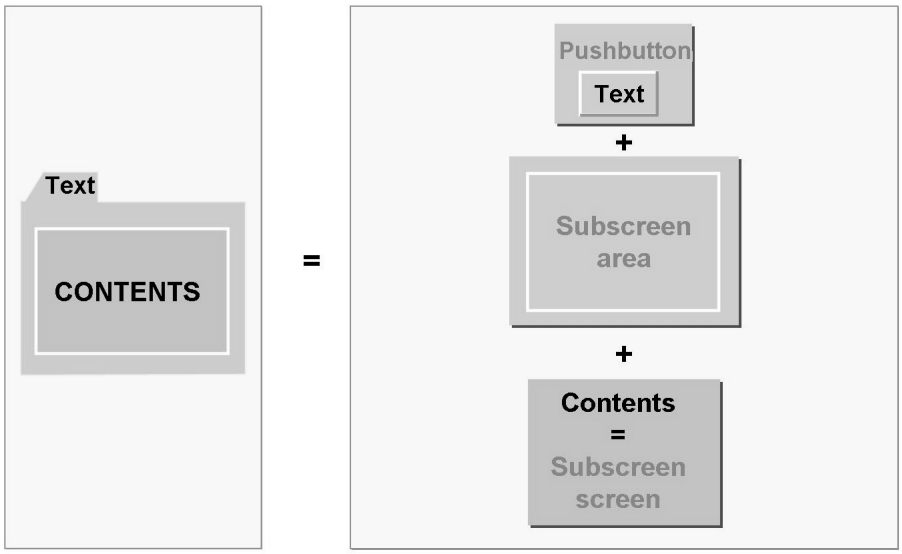


Figure 101: Page Elements: Technical View

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.

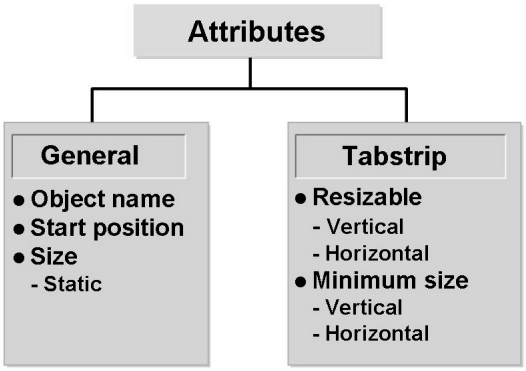


Figure 102: Tabstrip Controls: Attributes

In addition to the general attributes, *Object name*, *Starting position* and *static size*, tabstrip controls also have special attributes.

Creating a Tabstrip Control

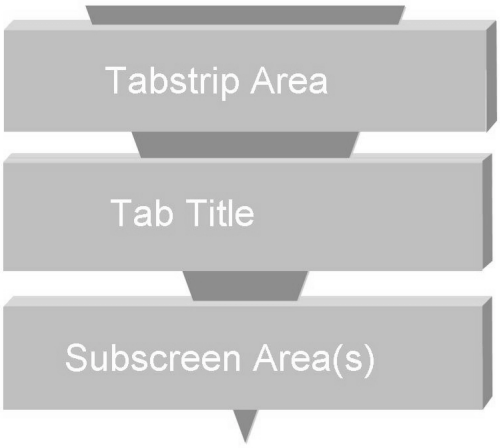


Figure 103: Creating a Tabstrip Control

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.

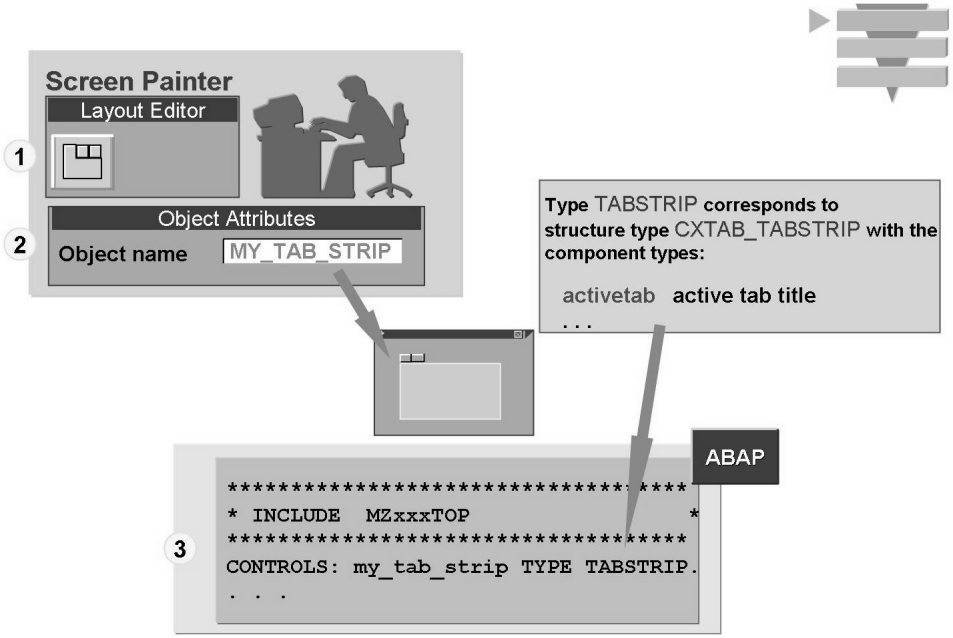


Figure 104: Creating a Tabstrip Control: Tabstrip Area

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.

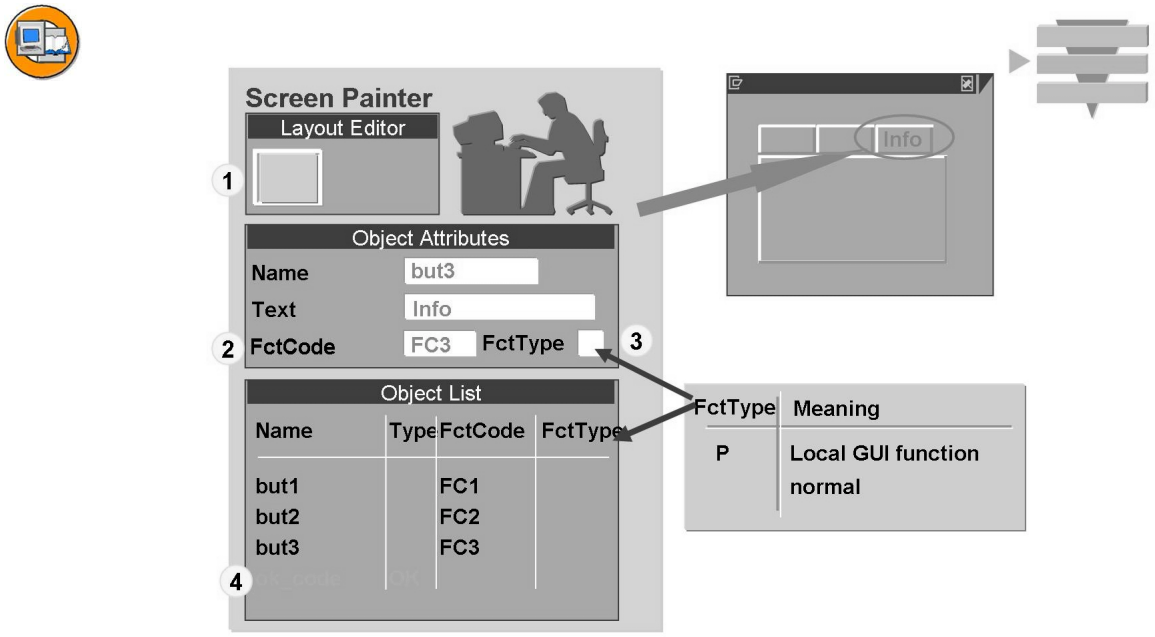


Figure 105: Creating a Tabstrip Control: Tab Title

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.

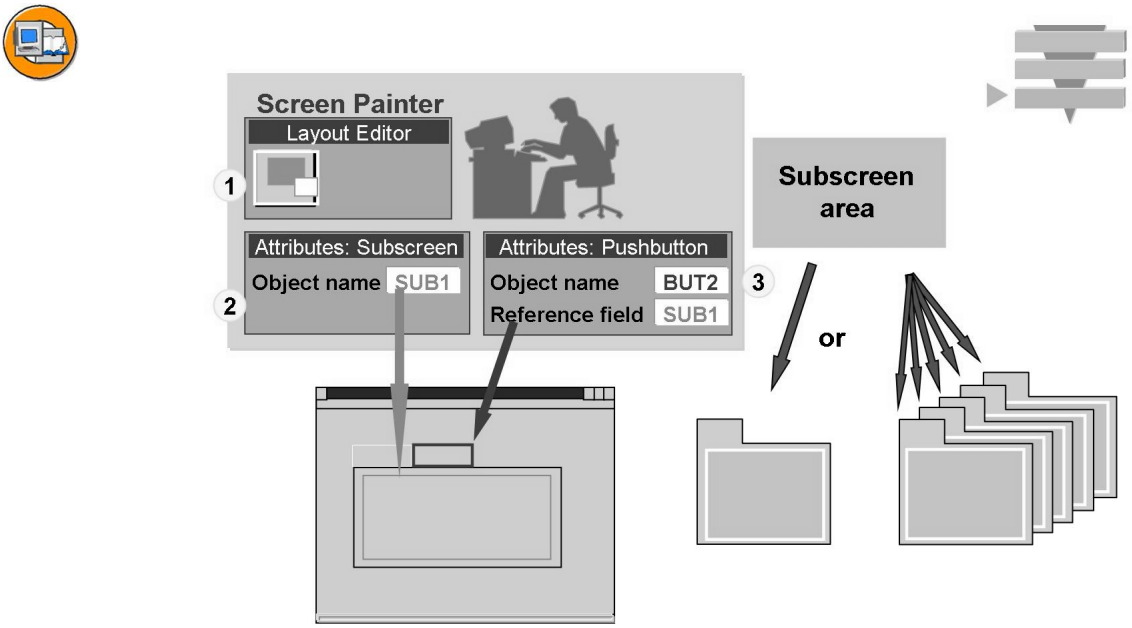


Figure 106: Creating Tabstrip Control: Tabstrip Subscreens

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 Locally in Tabstrip Controls

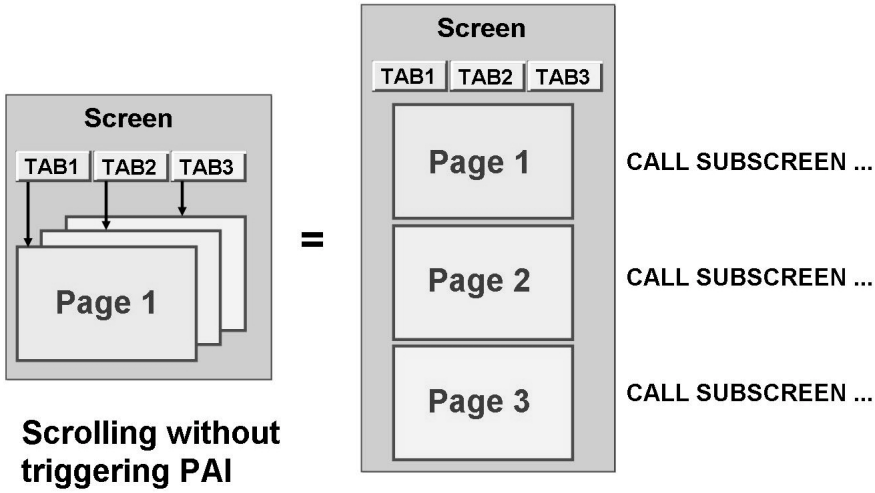


Figure 107: Scrolling Locally in Tabstrip Controls

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.

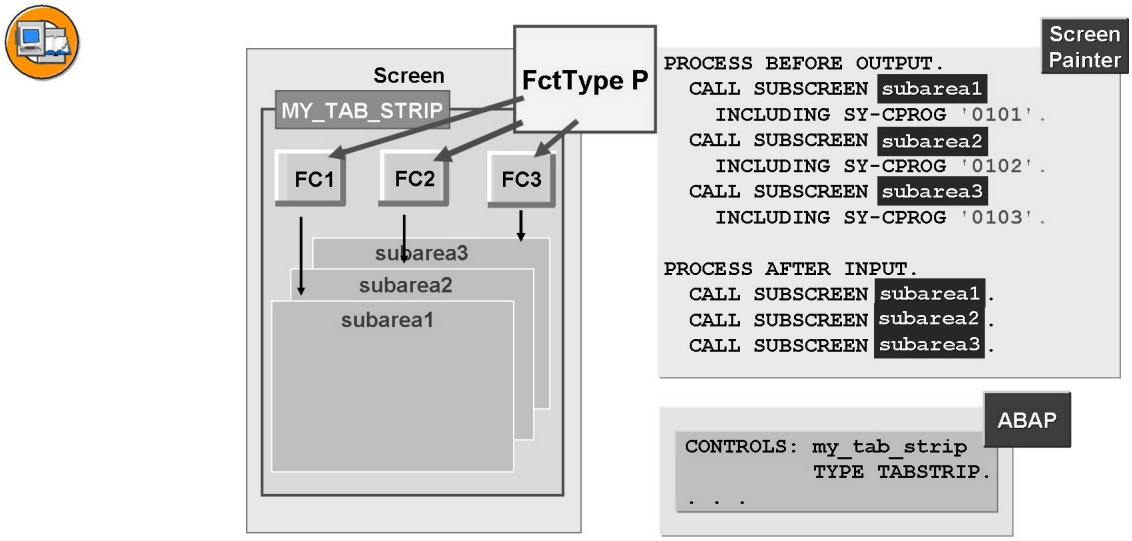


Figure 108: Scrolling Locally in Tabstrip Controls: Programming

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.

PAI Scrolling in Tabstrip Controls

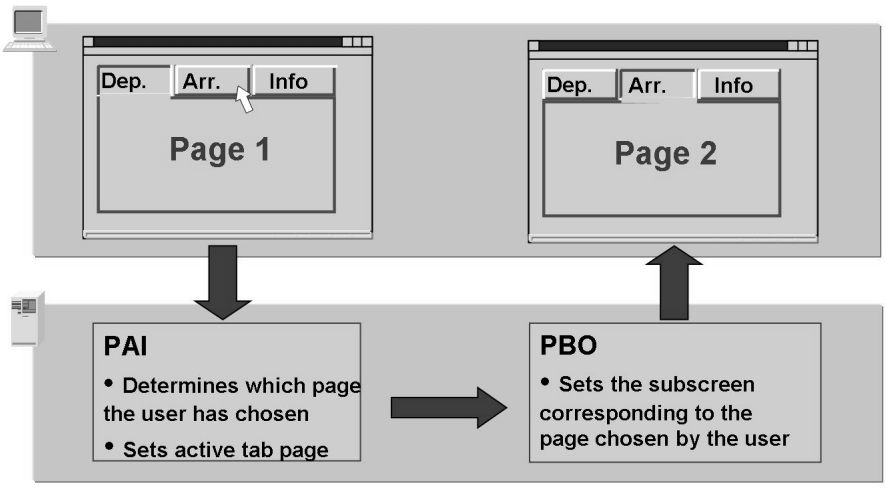


Figure 109: PAI Scrolling in Tabstrip Controls

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.

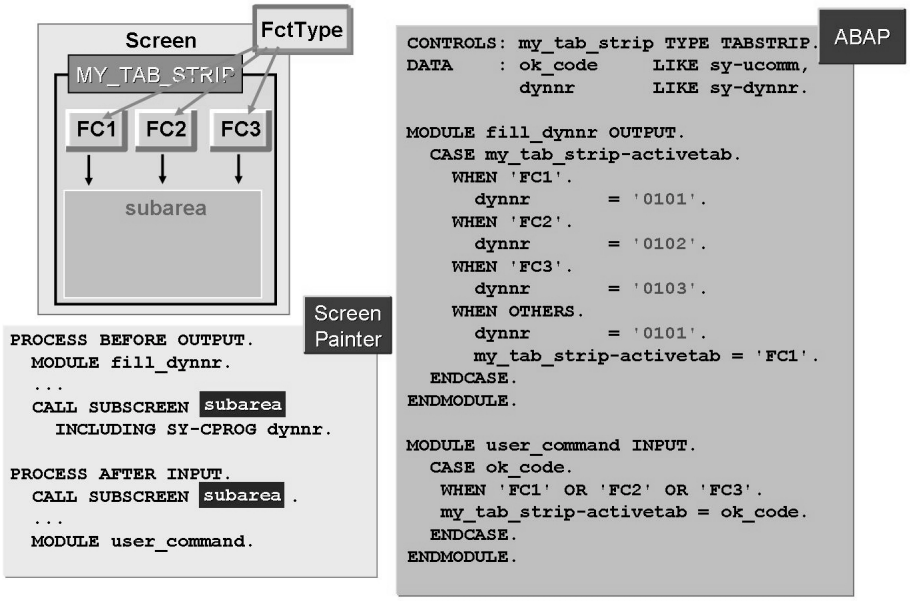


Figure 110: PAI Scrolling in Tabstrip Controls: Programming

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 runtime 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.

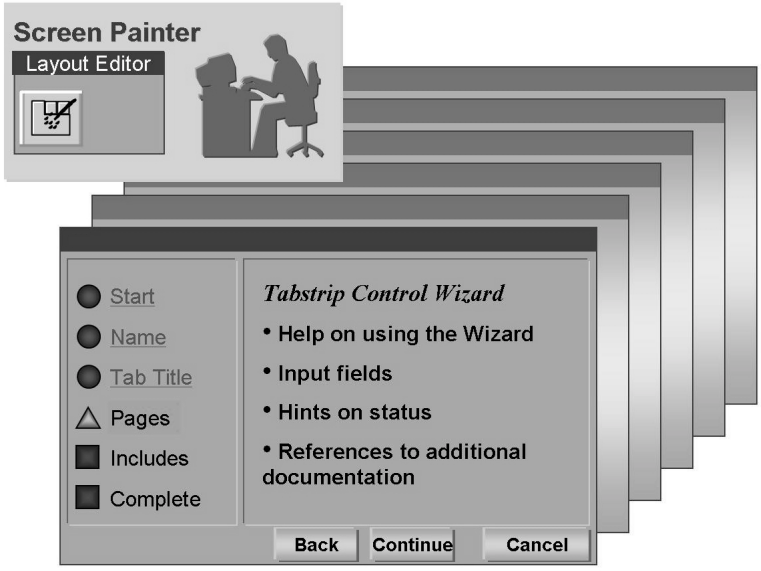


Figure 111: Creating Tabstrip Controls Using the Wizard

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:

Empty subscreens for the individual tabstrip control pages.

Includes for data definitions, PBO modules, PAI modules, and INCLUDE statements for these includes.

All objects are placed in the inactive object list.

Exercise 6: Creating Tabstrip controls

Exercise Objectives

After completing this exercise, you will be able to:

- Use tabstrip controls on screens in your programs

Business Example

In the last exercise, you created subscreens to display additional information on your screen. Now, create a tabstrip control on screen 100 for displaying extra flight information and details of the aircraft type.

Task:

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

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

Continued on next page

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

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

   ```
   CONTROLS MY_TABSTRIP ...
   ```

4. 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.

5. 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*.

   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*.

Solution 6: Creating Tabstrip controls

Task:

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

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.

   a) See sample solution.

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

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

   a) See sample solution.

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

Continued on next page

```
CONTROLS MY_TABSTRIP ...
```

a) See sample solution.

4. 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.

   a) See sample solution.

5. 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*.

   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*.

   a) See sample solution.

Result

Model Solution SAPMBC410ASUBS_TABSTRIP

Main program

No changes with regard to previous exercise.

Flow logic screen 100

No changes with regard to previous exercise.

Flow logic screen 110

No changes with regard to previous exercise.

Flow logic screen 120

No changes with regard to previous exercise.

Flow logic screen 130

No changes with regard to previous exercise.

Top include

Add the following coding:

```
CONTROLS:
  my_tabstrip TYPE TABSTRIP.
```

PBO module include

Change module *FILL_DYNNR*:

Continued on next page


```
*&---------------------------------------------------------------------*
*&      Module  fill_dynnr  OUTPUT
*&---------------------------------------------------------------------*
*       determine subscreen number
*---------------------------------------------------------------------*
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.                 " fill_dynnr  OUTPUT
```

PAI module include

Change module *USER_COMMAND_0100*:

```
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'FC1' OR 'FC2' OR 'FC3'.
      my_tabstrip-activetab = ok_code.

    WHEN 'BACK'.
      LEAVE TO SCREEN 0.

....

  ENDCASE.
```

Lesson Summary

You should now be able to:

- Create tabstrip controls
- Scroll in tabstrip controls

Unit Summary

You should now be able to:

- Create subscreens on screens
- Use function groups in subscreens
- Create tabstrip controls
- Scroll in tabstrip controls

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Test Your Knowledge

1. The _____ 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.

   Fill in the blanks to complete the sentence.

2. When using an encapsulated subscreen, the subscreen call must occur before the function module call.

   Determine whether this statement is true or false.

   □ True

   □ False

3. To create a tabstrip control area, choose Tabstrip control from the Screen Painter and place it on the screen.

   Fill in the blanks to complete the sentence.

4. To program a tabstrip control to scroll locally at the frontend, you must call all subscreens from the flow logic.

   Determine whether this statement is true or false.

   □ True

   □ False

Answers

1. 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.

   Answer: CALL SUBSCREEN <subarea>

2. When using an encapsulated subscreen, the subscreen call must occur before the function module call.

   Answer: False

   The function module call must occur before the subscreen call.

3. To create a tabstrip control area, choose Tabstrip control from the Screen Painter and place it on the screen.

   Answer:

4. To program a tabstrip control to scroll locally at the frontend, you must call all subscreens from the flow logic.

   Answer: True

   To program a tabstrip control to scroll locally at the frontend, you must call all subscreens from the flow logic.

Unit 6

Screen Elements: Table Controls

Unit Overview

This unit explains how to create table control area and table control fields to create a table control. In addition, this unit helps you understand the principle of processing a table control. Procedure to transfer changed values from the table control back to the internal table is explained. This unit also describes how to change the attributes of a table control by overwriting the field contents of the structure created in the CONTROLS statement. Finally, how to sort table controls is explained in this unit.

Unit Objectives

After completing this unit, you will be able to:

- Create table controls
- Identify table controls attributes
- Fill a table control
- Change the contents of a table control
- Change a table control
- Change the attributes of a table control

Unit Contents

Lesson: Table Controls Overview

Lesson Overview

This lesson explains that a table control is used to display data in tabular form. You will also learn that the table control contains a series of actions that are controlled at the presentation server. In addition, you will learn how to create a table control area and table control fields to create a table control.

Lesson Objectives

After completing this lesson, you will be able to:

- Create table controls
- Identify table controls attributes

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. The application should create a table control in an area on the screen to display data in tabular form.

Table Controls

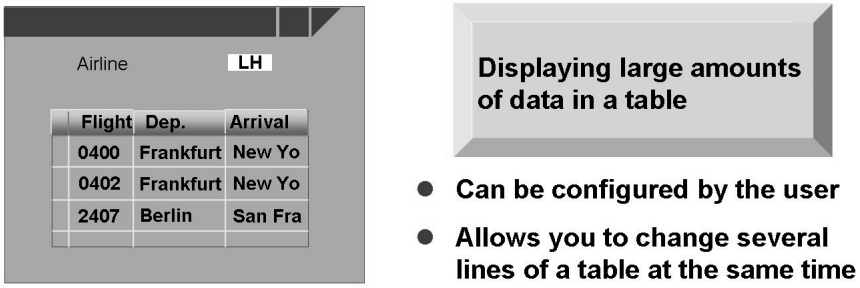


Figure 112: Table Controls

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 and each column may have a title.

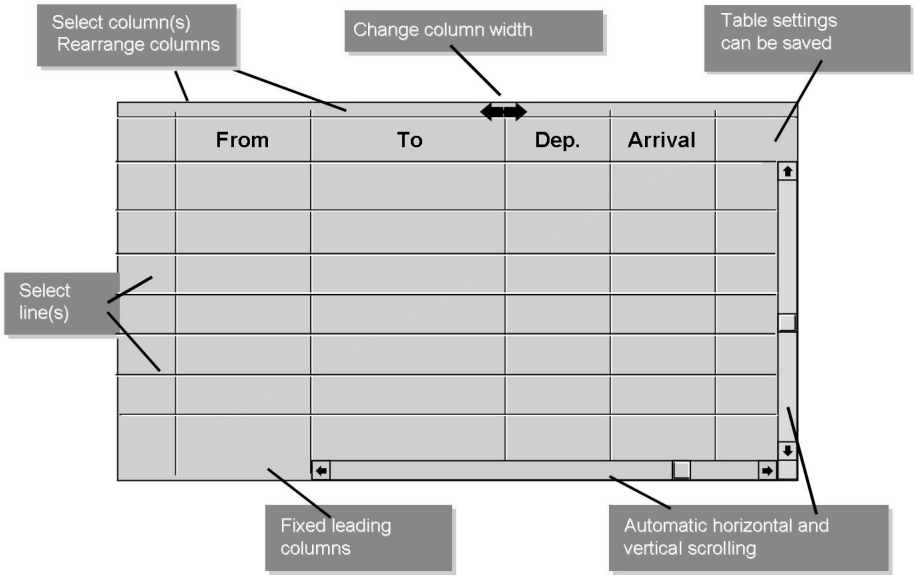


Figure 113: Table Controls: Features

You can display or enter single structured lines of data using a table control.

Benefits of using a table control are:

- 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 runtime.

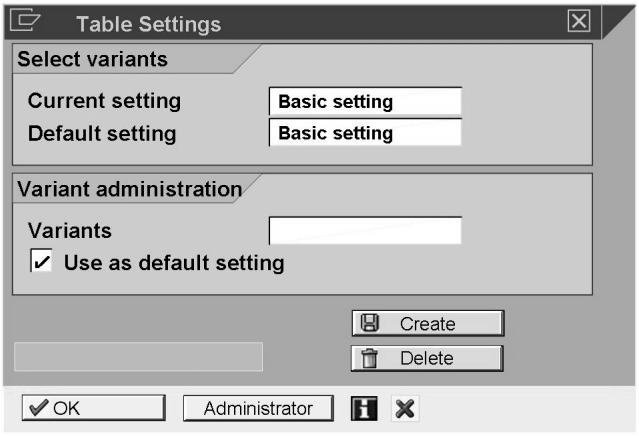


Figure 114: Table Controls: Table Settings

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

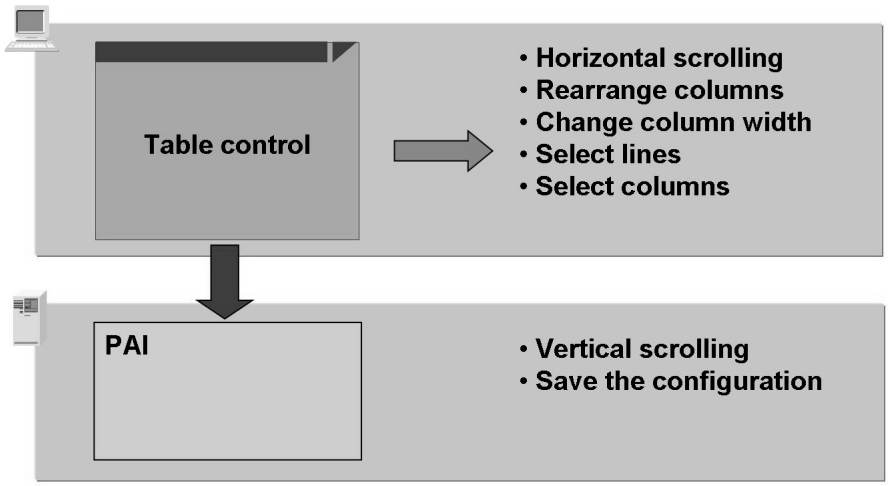


Figure 115: Actions in Table Controls

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.

Creating Table Controls

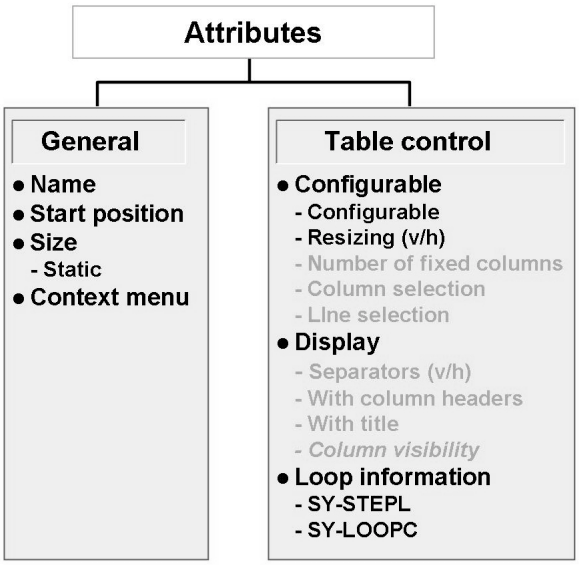


Figure 116: Table Controls: Attributes

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 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 the structure *SYST* contain information about the loop processing used with table controls.

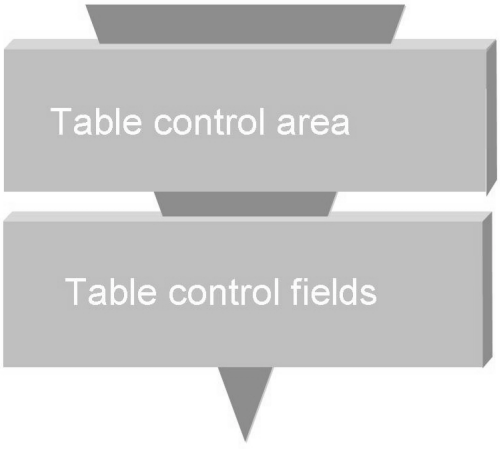


Figure 117: Creating Table Controls

When you create a table control, you must create:

- A table control area
- Table control fields

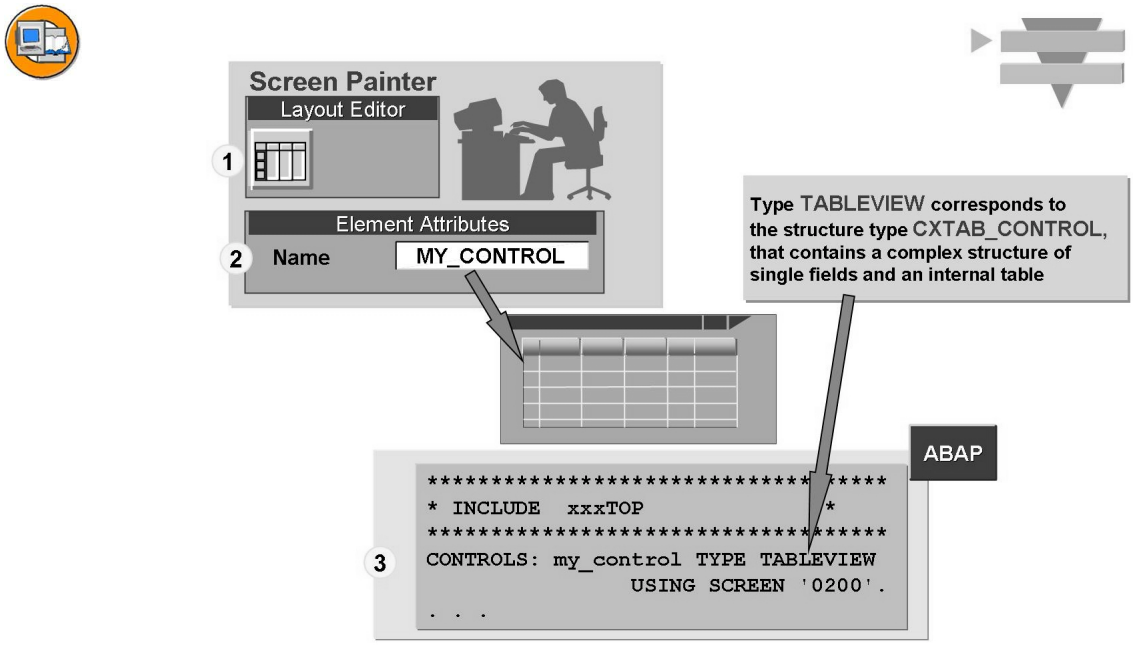


Figure 118: Creating a Table Control: Table Control Area

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 runtime, 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.

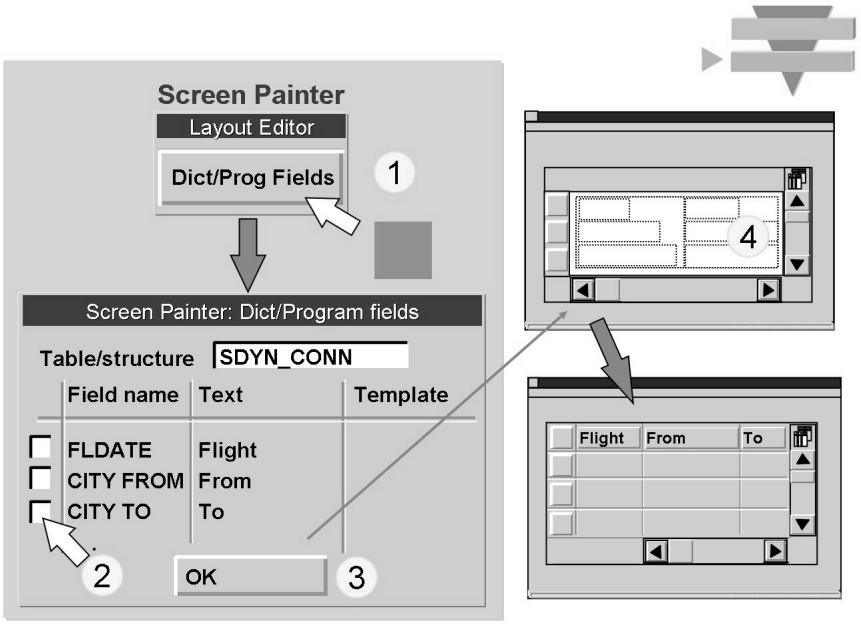


Figure 119: Creating Table Controls: Fields

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 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.

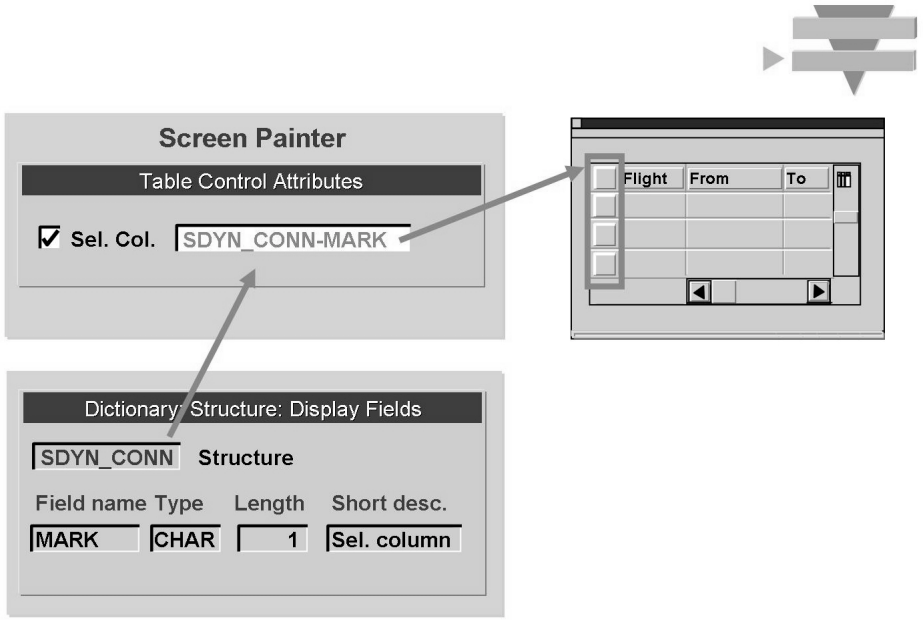


Figure 120: Creating Table Controls: Selection Column

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.

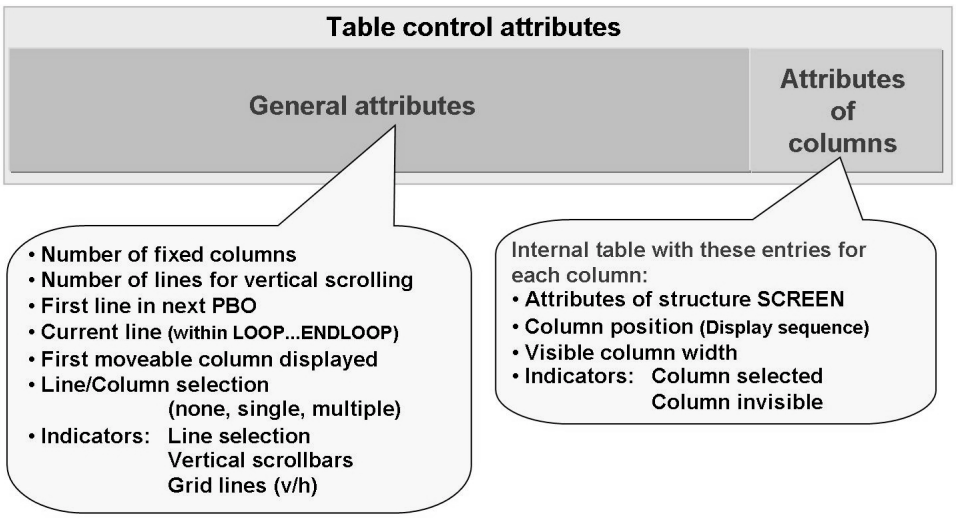


Figure 121: Table Control Attributes at Runtime

The table control attributes, saved at runtime 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*).

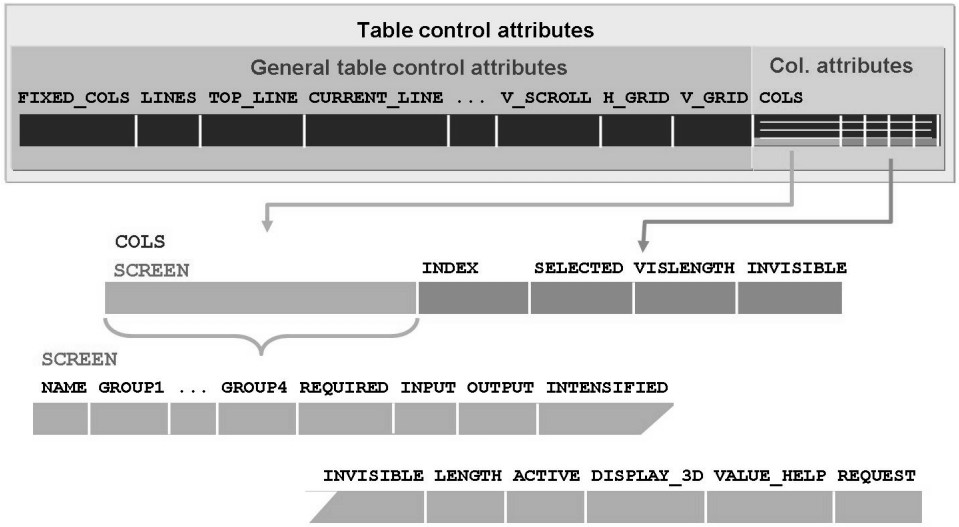


Figure 122: Table Control Attributes (Structure)

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.

Lesson Summary

You should now be able to:

- Create table controls
- Identify table controls attributes

Internal Use SAP Partner Only

Lesson: Processing Table Controls

Lesson Overview

This lesson will help you understand the principle of processing a table control. You will learn how to display buffered data from the internal table in the table control. In addition, you will learn how to transfer changed values from the table control back to the internal table.

Lesson Objectives

After completing this lesson, you will be able to:

- Fill a table control
- Change the contents of a table control

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. After creating table controls, you should be able to display buffered data from the internal table in the table control.

Principle of Processing Table Controls

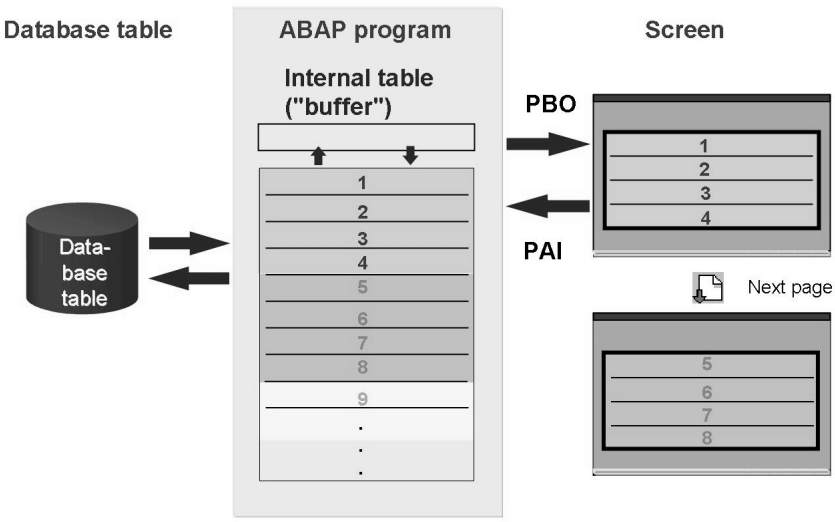


Figure 123: Processing a Table Control (Principle)

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.

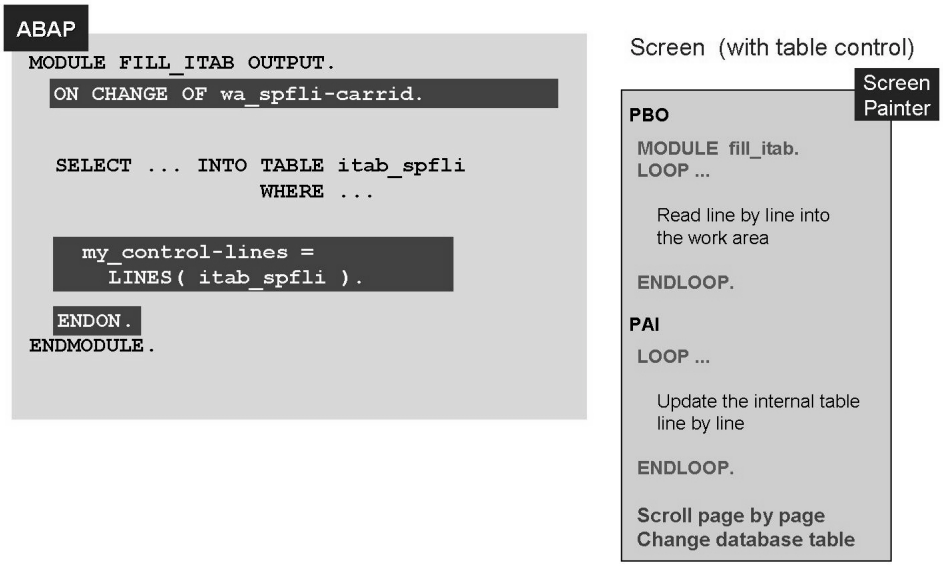


Figure 124: Table Controls: Applications (Principle)

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.

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 *LINES* function 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 moduls, 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.

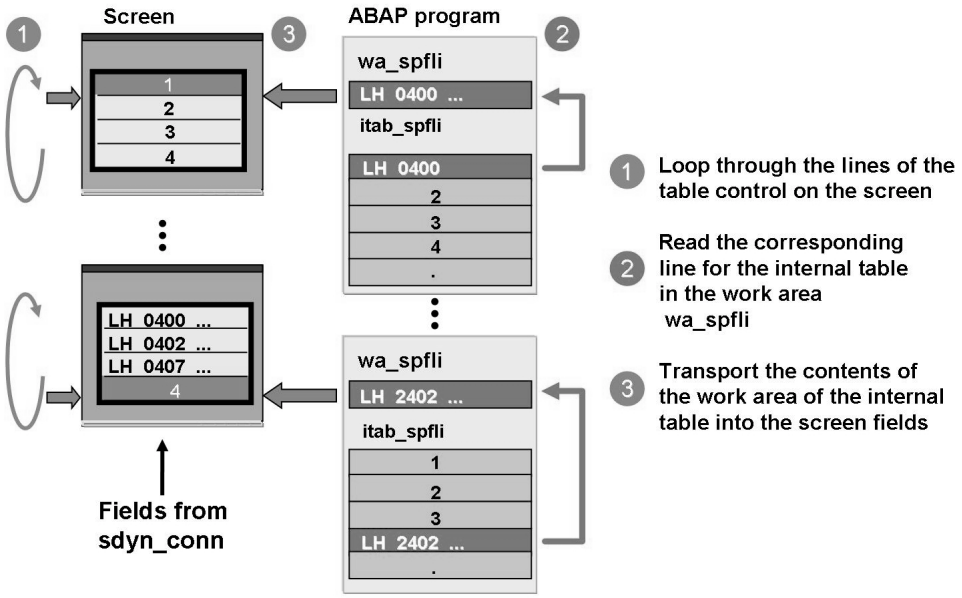


Figure 125: Filling a Table Control

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.

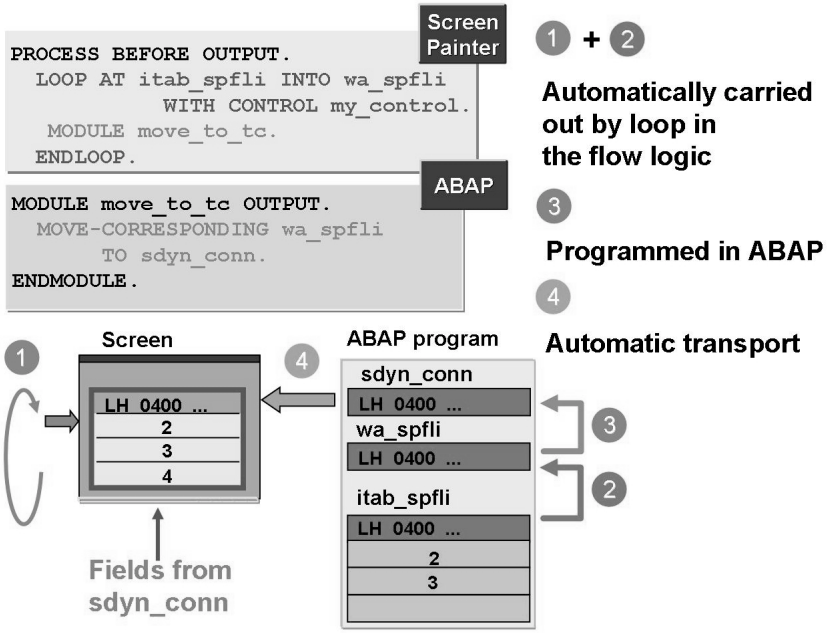


Figure 126: Code: Filling a Table Control

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.

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 <wa_screen>.*, where <wa_screen> should be defined with a *TABLES* statement.)

The system calculates the value of <ctrl>-TOP_LINE when you scroll.

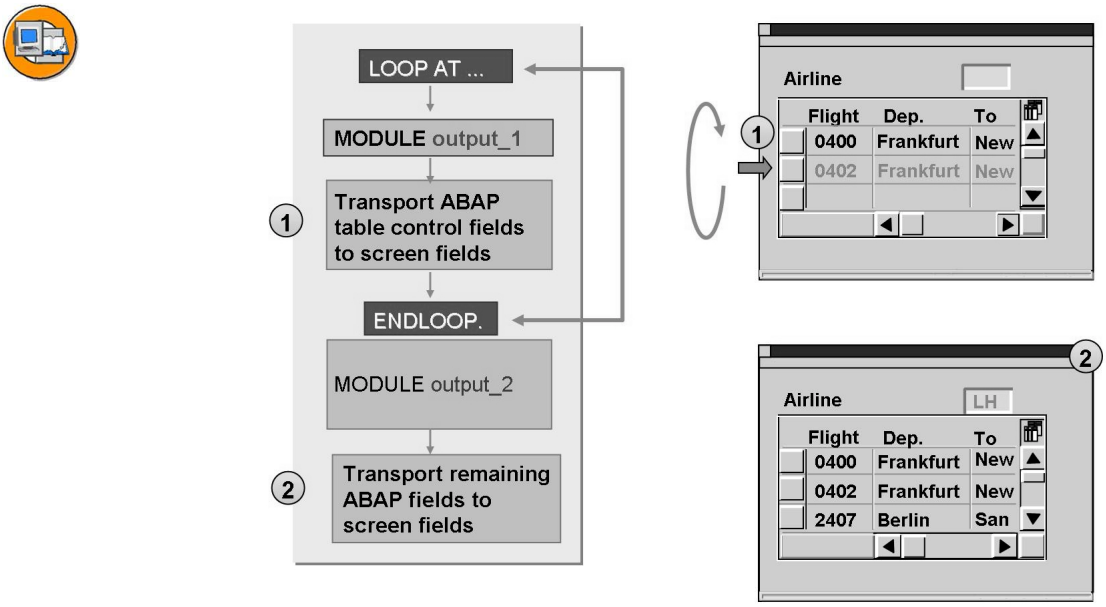


Figure 127: Table Controls: Field Transport in the PBO

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 Table Contents

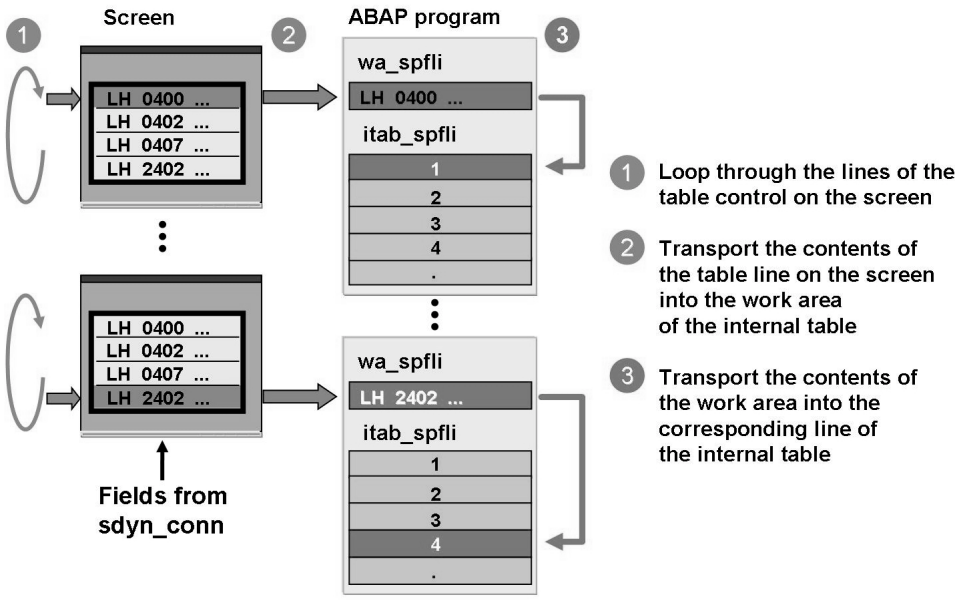


Figure 128: Changing the Contents of a Table Control

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 and transports the data to the TABLES structure..
- If you are not using the TABLES structure as the work area of the internal table you must program the transport of the contents of the TABLES structure to the work area of the internal table.
- You must program the transport of the contents of the work area into the corresponding line of the internal table.

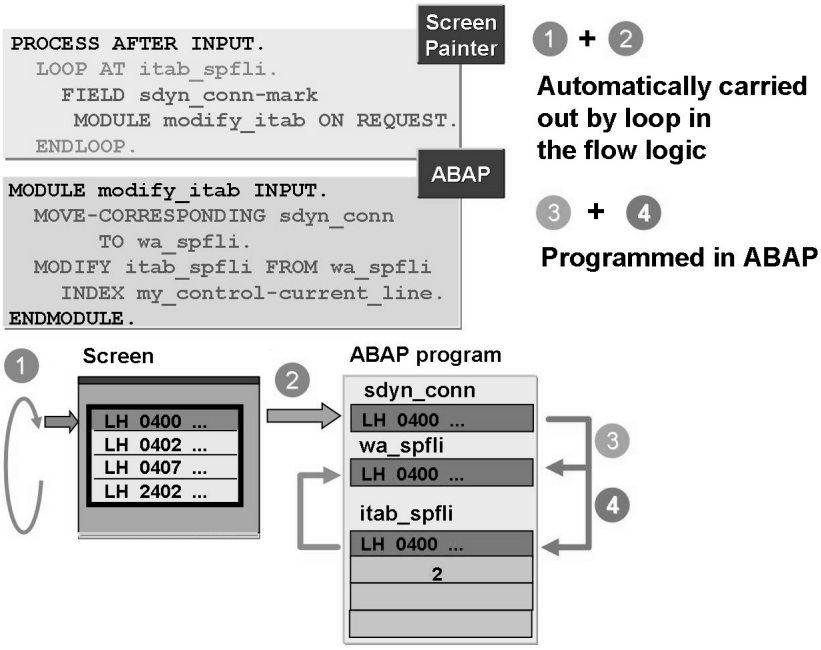


Figure 129: Code: Changing the Contents of Table Control

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.

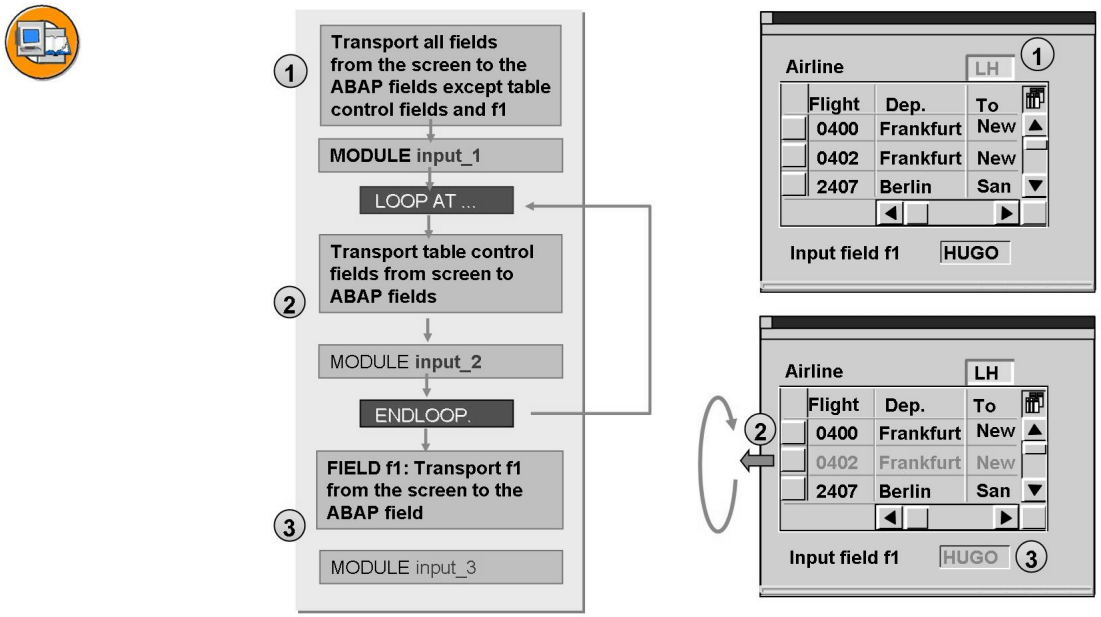


Figure 130: Table Controls: Field Transport in the PAI

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.

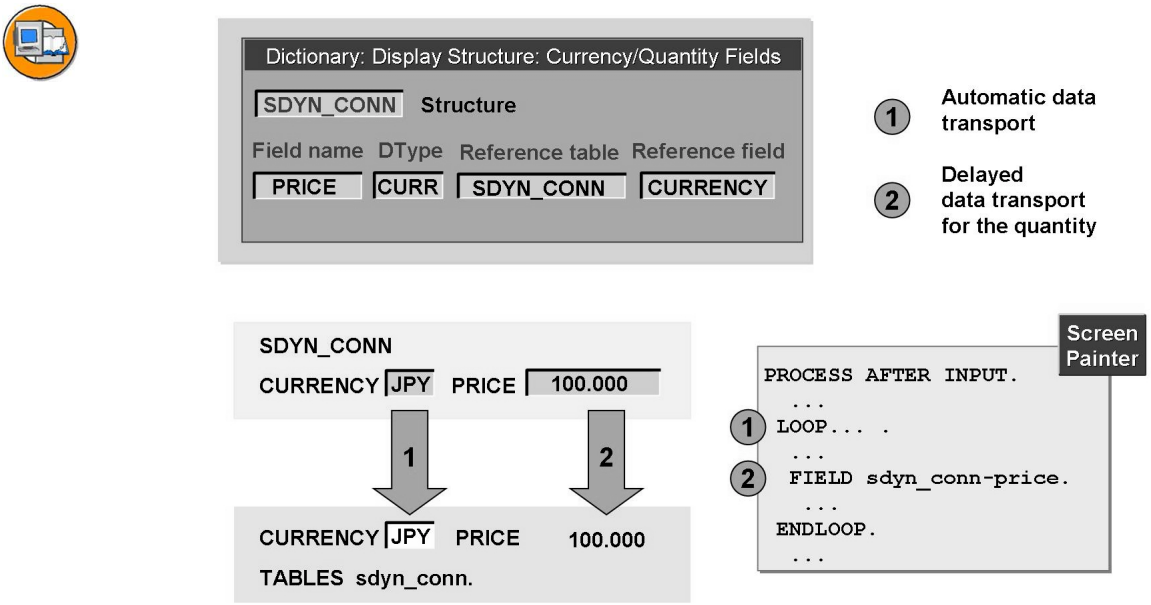


Figure 131: Field Transports for Formatted Fields

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 fields are followed by the amounts. To do this, delay the data transport of the quantity fields using a corresponding FIELD statement.

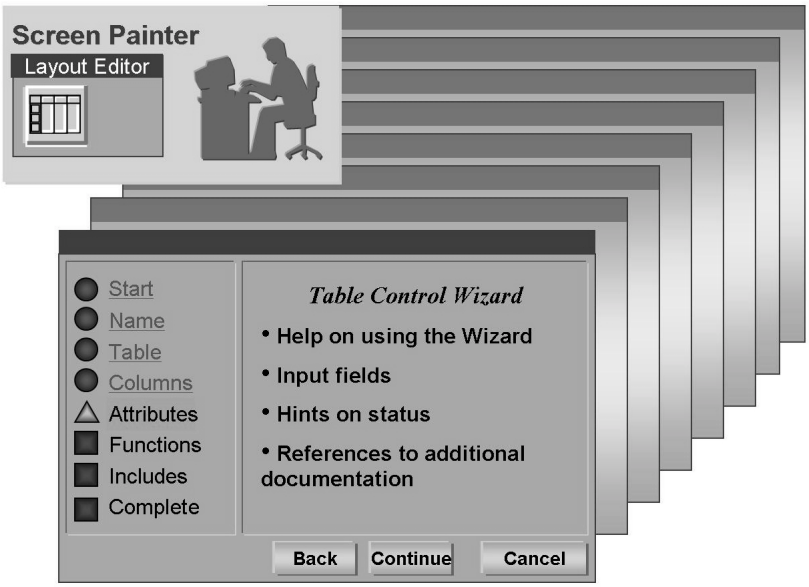


Figure 132: Creating Table Controls Using the Wizard

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, 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.

Exercise 7: Creating and Filling a Table Control

Exercise Objectives

After completing this exercise, you will be able to:

- Create a table control
- Create an internal table
- Fill a table control

Business Example

After creating tabstrip controls, create a table control on the third page of the tabstrip control to maintain bookings for your flight. Also, create a PBO module in which you read all of the bookings for the flight selected that have not been canceled. In addition, to copy data from the work area of the internal table into the screen fields.

Task:

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

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.

2. **Table Control Area:**

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

Continued on next page

Table control	**Name:** *MY_TABLE_CONTROL*	**Attributes:** **Vertical and horizontal Resizing:** ON **Vertical and horizontal Separators:** ON **Column selection:** SINGLE **Line selection:** MULTIPLE **Column title:** ON **Configurable:** ON **Selection column:** SDYN_BOOKMARK **No. of fixed columns:** 2

Hint: 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_CONTROL ...
```

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

Continued on next page

Internal Use SAP Partner Only

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

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

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.

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 *LINES(itab)* function.

6. **Implement a table control:** Program a loop for the table control in both the PBO and PAI events of screen 130 to 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

Continued on next page

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.`).

Hint: You can test whether your changes or 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.

Solution 7: Creating and Filling a Table Control

Task:

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

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.

   a) See sample solution.

2. **Table Control Area:**

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

Table control	**Name:** *MY_TABLE_CON-TROL*	**Attributes:** **Vertical and horizontal Resizing:** ON **Vertical and horizontal Separators:** ON **Column selection:** SINGLE **Line selection:** MULTIPLE **Column title:** ON **Configurable:** ON **Selection column:** SDYN_BOOK-MARK **No. of fixed columns:** 2

Hint: 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_CONTROL ...
```

   a) See sample solution.

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

Continued on next page

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

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

a) See sample solution.

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.

   a) See sample solution.

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 *LINES(itab)* function.

   a) See sample solution.

Continued on next page

Internal Use SAP Partner Only

6. **Implement a table control:** Program a loop for the table control in both the PBO and PAI events of screen 130 to 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.`).

Hint: You can test whether your changes or 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.

a) See sample solution.

Result

Model Solution SAPMBC410ATABS_TABLE_CONTROL1

Main program

No changes with regard to previous exercise.

Flow logic screen 100

No changes with regard to previous exercise.

Flow logic screen 110

No changes with regard to previous exercise.

Flow logic screen 120

No changes with regard to previous exercise.

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 sdyn_book-mark
    MODULE trans_from_tc ON REQUEST.
  ENDLOOP.
```

Continued on next page

Top include

Add the following coding:

```
TABLES: sdyn_book.

ATA:
* internal table for bookings table control
  it_sdyn_book TYPE TABLE OF sdyn_book,
  wa_sdyn_book LIKE LINE OF it_sdyn_book.
DATA  not_cancelled VALUE space.
CONTROLS:
  my_table_control TYPE TABLEVIEW USING SCREEN '0130'.
```

PBO module include

New module *GET_SBOOK*:

```
MODULE get_sbook OUTPUT.
  ON CHANGE OF wa_sflight-carrid OR
               wa_sflight-connid OR
               wa_sflight-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.

    my_table_control-lines = lines( it_sdyn_book ).
  ENDON.
ENDMODULE.                 " get_sbook  OUTPUT
```

New module *TRANS_TO_TC*:

```
*&---------------------------------------------------------------------*
*&      Module  trans_to_tc  OUTPUT
*&---------------------------------------------------------------------*
*       copy booking data to TABLES structure
*---------------------------------------------------------------------*
MODULE trans_to_tc OUTPUT.
  MOVE wa_sdyn_book TO sdyn_book.
ENDMODULE.                 " trans_to_tc  OUTPUT
```

PAI module include

Change module *TRANS_FROM_TC*:

```
*&---------------------------------------------------------------------*
*&      Module  trans_from_tc  INPUT
```

Continued on next page

Internal Use SAP Partner Only

```
*&---------------------------------------------------------------------*
*       copy booking data from TABLES structure to work area
*       and modify internal table
*---------------------------------------------------------------------*
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
```


Internal Use SAP Partner Only

Lesson Summary

You should now be able to:

- Fill a table control
- Change the contents of a table control

Lesson: Table Controls Further Techniques

Lesson Overview

This lesson will help you learn how to change the attributes of a table control by overwriting the field contents of the structure created in the CONTROLS statement. You will also learn to change the attributes of table control fields temporarily and to sort table controls.

Lesson Objectives

After completing this lesson, you will be able to:

- Change a table control
- Change the attributes of a table control

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. In table controls created for a screen, the application should allow you to change the attributes by overwriting the field contents of the structure created in the CONTROLS statement.

Setting Table Control

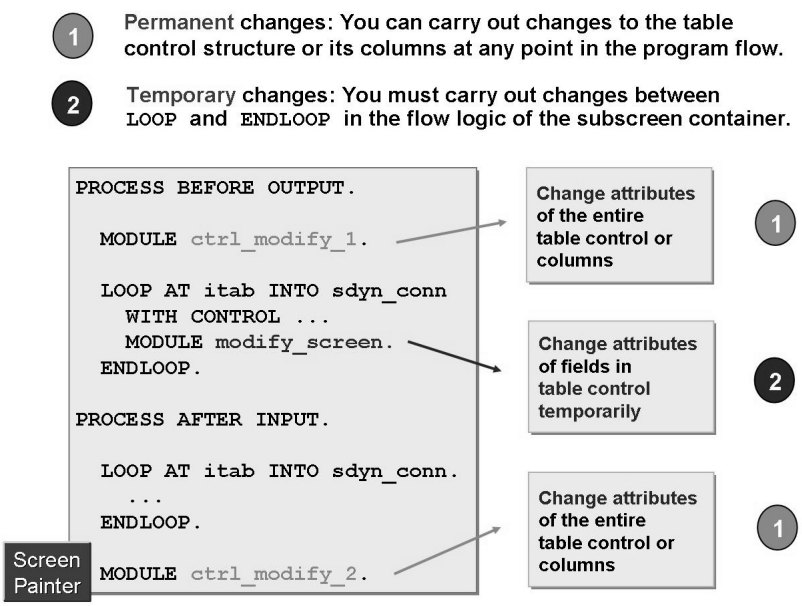


Figure 133: Changing Table Control

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.

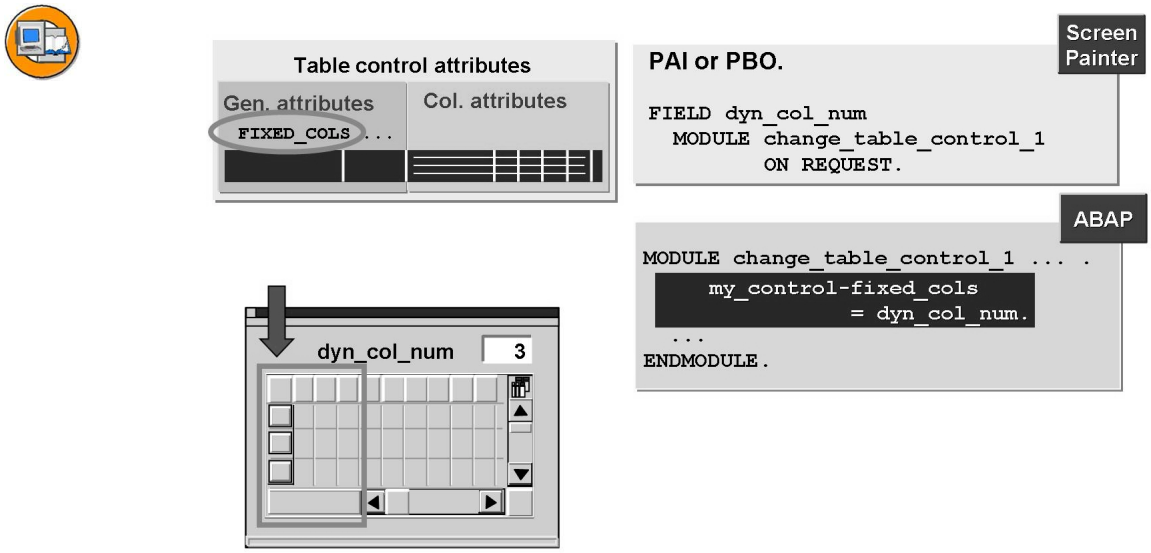


Figure 134: Changing the Attributes of Table Control (1)

You can change a table control dynamically by modifying the contents of the fields of its structure.

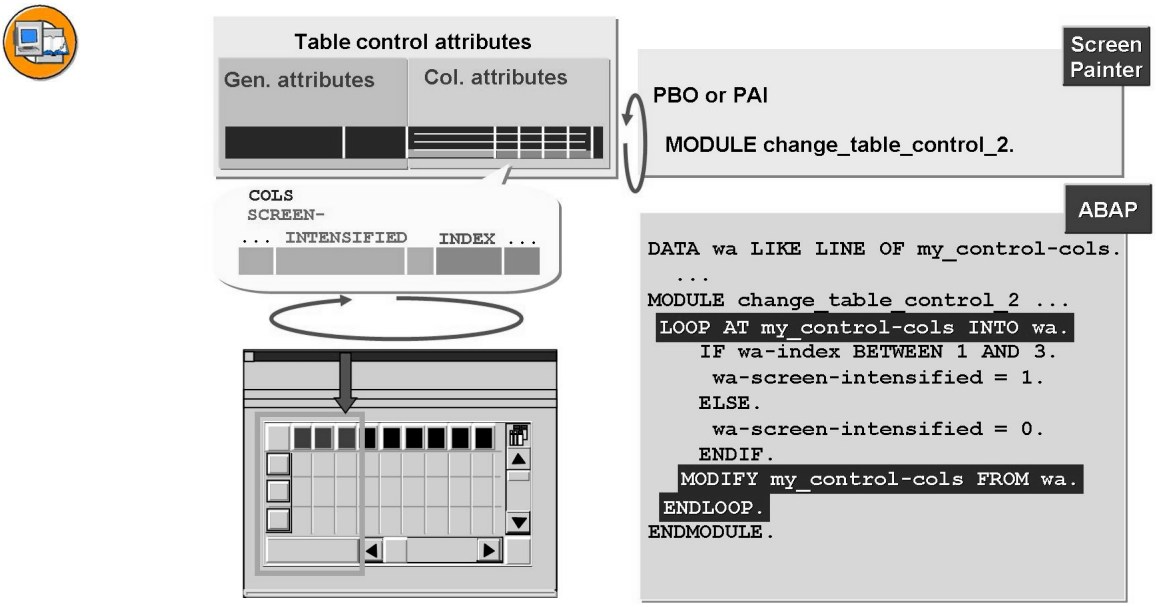


Figure 135: Changing the Attributes of Table Control (2)

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.

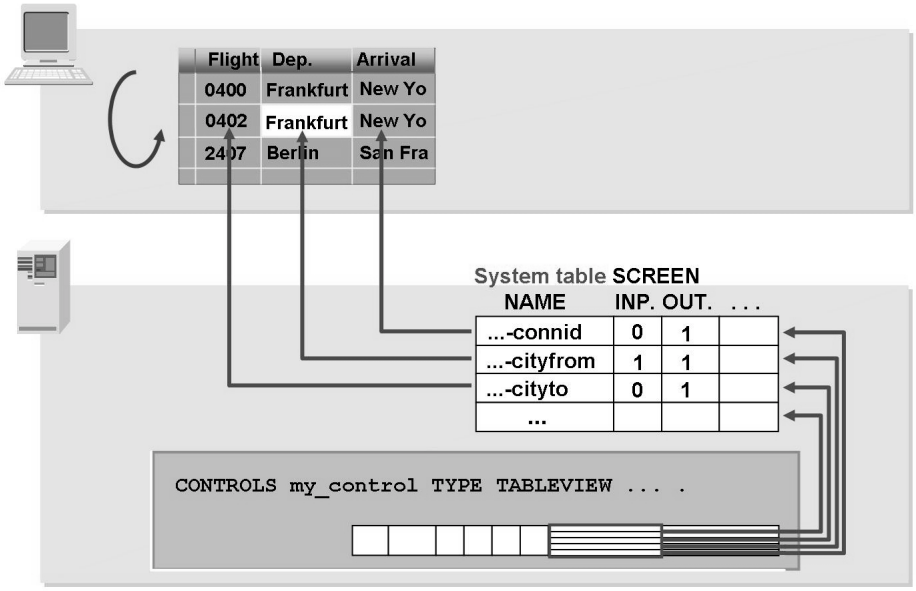


Figure 136: Table Controls: Field Attributes

During the loop processing of the table control at PBO, a system program copies some of the column attributes from the table control structure to the **SCREEN** system table. The program copies the parts of the **COLS** line structure that are also called **SCREEN-<fname>** Attributes are copied for all fields of a table control line.

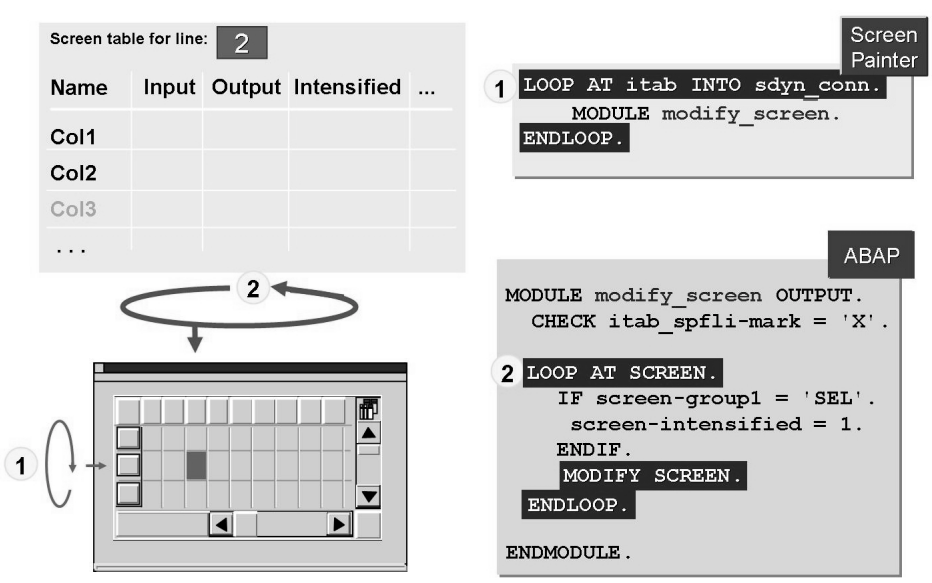


Figure 137: Table Controls: Changing Field Attributes Temporarily

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 PBO 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.

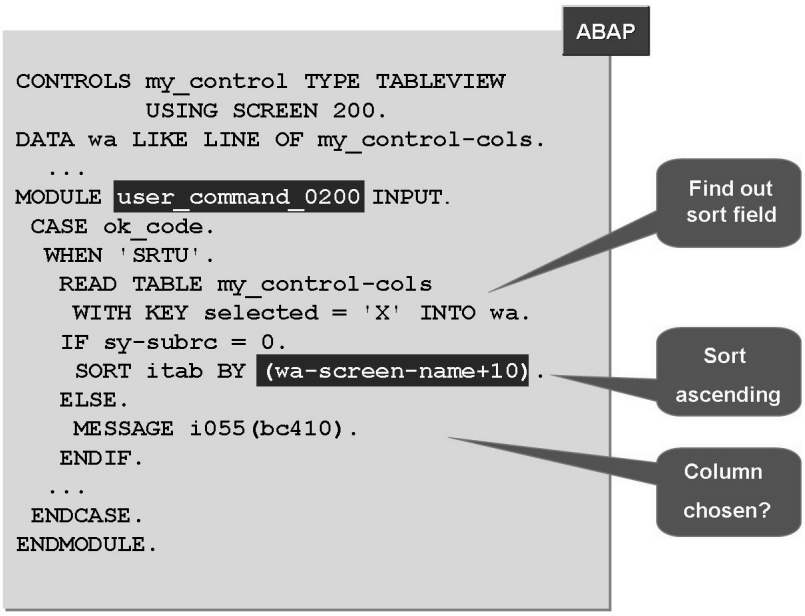


Figure 138: Sorting Table Controls: Example

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.

Line Counters in Table Control

Line Counter	Description
SY-LOOPC	At PBO: How many lines can be displayed in the table control? At PAI: How many lines are actually displayed?
SY-STEPL	Loop counter - counts during *LOOP ... ENDLOOP* of the flow logic. Which line of the **table control** is being processed? The statement *GET CURSOR LINE* refers to *SY-STEPL*.
<TC>-*CURRENT_LINE*	Loop counter - counts during *LOOP ... ENDLOOP* of the flow logic. Which line of the **internal table** is being processed?
<TC>-*TOP_LINE*	Which line of the internal table is momentarily at the top of the table control?
<TC>-*LINES*	How many lines contains 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 LIKE sy-stepl,
      tabix LIKE sy-tabix.
  ...

GET CURSOR LINE selline.
tabix = my_control-TOP_LINE
        + selline - 1.
READ TABLE spfli_itab
     INDEX tabix.
```

Figure 139: Table Controls: Cursor Position (Example)

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 = <ctr-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.

Internal Use SAP Partner Only

Exercise 8: Table Control: Further Techniques

Exercise Objectives

After completing this exercise, you will be able to:

- Create pushbuttons for a table control
- Change the attributes of a table control at runtime

Business Example

Implement functions for selecting all unmarked table control lines, deselecting all marked table control lines, and sorting in ascending and descending order. To do all these tasks, create pushbuttons.

The application should allow the user to change the attributes for horizontal and vertical gridlines by overwriting the field contents of the structure created in the *CONTROLS* statement.

Finally there should be a function to highlight bookings of smokers.

Task:

Implement functions for selecting all unmarked table control lines, and deselecting all marked table control lines, and sorting in ascending and descending order.

Allow the user to change the attributes for horizontal and vertical gridlines.

Implement a function to highlight bookings of smokers.

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.
2. **Create pushbuttons:** Create the following pushbuttons on screen 130:

Pushbutton	**Name:** SELECT_ALL	**Function code:** SELE **Function type:** <blank> **Icon:** ICON_SELECT_ALL
Pushbutton	**Name:** DESELECT_ALL	**Function code:** DSELE **Function type:** <blank> **Icon:** ICON_DESELECT_ALL

Continued on next page

3. **Implement select functions:** Extend the command field processing of screen 130 to include the *Select all* and *Deselect all* functions. This task deals with a data change: If a row is selected, the field *MARK* in the internal table contains the value 'X'. This means that you only need to change the contents of the internal table.Loop through the internal table and change all values of field *MARK* either to 'X' or to space.

4. **Activate or deactivate the horizontal and/or vertical separators:** Allow the user to activate or deactivate the horizontal and/or vertical separators of the table control using checkboxes on the screen.

Hint: This task deals with changes to Table Control properties, that is, “permanent” changes – you can make these at PBO or PAI.

It is sufficient to add a checkbox called *MY_TABLE_CONTROL-H_GRID* and another called *MY_TABLE_CONTROL-V_GRID* to screen 130 (it is best to assign any function code so that the user does not have to press *Return* for the changes to take effect.)

Checkboxes can have the values 'X' or space – exactly like the fields *V_GRID* and *H_GRID* of a table control. This means that no additional coding is required, since the values of the checkboxes are automatically copied to the two fields of the table control structure at PAI.

5. **Optional: Highlighting the Smokers:** Offer a checkbox on the screen 130 to highlight all smokers in the table control (SMOKER = 'X'), by switching the affected rows to intensive.

Hint: This task deals with changes to field properties, that is, “temporary” changes – these must be made at PBO.

Again assign a function code to the checkbox. If the box is checked, highlight the rows, otherswise display them normally.

6. **Optional: 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:

Continued on next page

Internal Use SAP Partner Only

Pushbutton	**Name:** P_SRTU	**Function code:** SRTU **Function type:** <blank> **Icon:** ICON_SORT_UP
Pushbutton	**Name:** P_SRTD	**Function code:** SRTD **Function type:** <blank> **Icon:** ICON_SORT_DOWN

Implement sort 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 read 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.

Solution 8: Table Control: Further Techniques

Task:

Implement functions for selecting all unmarked table control lines, and deselecting all marked table control lines, and sorting in ascending and descending order.

Allow the user to change the attributes for horizontal and vertical gridlines.

Implement a function to highlight bookings of smokers.

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.

   a) See sample solution.

2. **Create pushbuttons:** Create the following pushbuttons on screen 130:

Pushbutton	**Name:** SELECT_ALL	**Function code:** SELE **Function type:** <blank> **Icon:** ICON_SELECT_ALL
Pushbutton	**Name:** DESELECT_ALL	**Function code:** DSELE **Function type:** <blank> **Icon:** ICON_DESELECT_ALL

   a) See sample solution.

3. **Implement select functions:** Extend the command field processing of screen 130 to include the *Select all* and *Deselect all* functions. This task deals with a data change: If a row is selected, the field *MARK* in the internal table contains the value 'X'. This means that you only need to change the contents of the internal table.Loop through the internal table and change all values of field *MARK* either to 'X' or to space.

   a) See sample solution.

4. **Activate or deactivate the horizontal and/or vertical separators:** Allow the user to activate or deactivate the horizontal and/or vertical separators of the table control using checkboxes on the screen.

Hint: This task deals with changes to Table Control properties, that is, “permanent” changes – you can make these at PBO or PAI.

Continued on next page

It is sufficient to add a checkbox called *MY_TABLE_CONTROL-H_GRID* and another called *MY_TABLE_CONTROL-V_GRID* to screen 130 (it is best to assign any function code so that the user does not have to press *Return* for the changes to take effect.)

Checkboxes can have the values 'X' or space – exactly like the fields *V_GRID* and *H_GRID* of a table control. This means that no additional coding is required, since the values of the checkboxes are automatically copied to the two fields of the table control structure at PAI.

a) See sample solution.

5. **Optional: Highlighting the Smokers:** Offer a checkbox on the screen 130 to highlight all smokers in the table control (SMOKER = 'X'), by switching the affected rows to intensive.

Hint: This task deals with changes to field properties, that is, "temporary" changes – these must be made at PBO.

Again assign a function code to the checkbox. If the box is checked, highlight the rows, otherswise display them normally.

a) See sample solution.

6. **Optional: 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	**Name:** P_SRTU	**Function code:** SRTU **Function type:** <blank> **Icon:** ICON_SORT_UP
Pushbutton	**Name:** P_SRTD	**Function code:** SRTD **Function type:** <blank> **Icon:** ICON_SORT_DOWN

Implement sort 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 read for the

Continued on next page

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.

a) See sample solution.

Result

Model Solution SAPMBC410ATABS_TABLE_CONTROL2

Main program

No changes with regard to previous exercise.

Flow logic screen 100

No changes with regard to previous exercise.

Flow logic screen 110

No changes with regard to previous exercise.

Flow logic screen 120

No changes with regard to previous exercise.

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.
    module modify_tc_line.
  ENDLOOP.

PROCESS AFTER INPUT.
  LOOP AT it_sdyn_book.
    FIELD sdyn_book-mark
    MODULE trans_from_tc ON REQUEST.
  ENDLOOP.

  MODULE user_command_0130.
```

Top include

Continued on next page

Internal Use SAP Partner Only

Add the following coding:

```
* checkbox highlight smokers
DATA: smoker TYPE c LENGTH 1.

* work area for columns' internal table of Table Control structure
DATA: wa_cols LIKE LINE OF my_table_control-cols.
```

PBO module include

New module *MODIFY_TC_LINE*:

```
*&---------------------------------------------------------------------*
*&      Module  modify_tc_line  OUTPUT
*&---------------------------------------------------------------------*
*       highlite smokers if wanted
*---------------------------------------------------------------------*
MODULE modify_tc_line OUTPUT.
  CHECK wa_sdyn_book-smoker = 'X' AND smoker = 'X'.

  LOOP AT SCREEN.
    screen-intensified = 1.
    MODIFY SCREEN.
  ENDLOOP.

ENDMODULE.                 " modify_tc_line  OUTPUT
```

PAI module include

Change module *USRE_COMMAND_0130*:

```
*&---------------------------------------------------------------------*
*&      Module  user_command_0130  INPUT
*&---------------------------------------------------------------------*
*       process commands from subscreen 0130
*---------------------------------------------------------------------*
MODULE user_command_0130 INPUT.
  CASE ok_code.
    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.
```

Continued on next page

```
      ENDLOOP.
    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.  ENDCASE.
ENDMODULE.                 " user_command_0130  INPUT
```

Lesson Summary

You should now be able to:

- Change a table control
- Change the attributes of a table control

Unit Summary

You should now be able to:

- Create table controls
- Identify table controls attributes
- Fill a table control
- Change the contents of a table control
- Change a table control
- Change the attributes of a table control

Test Your Knowledge

1. The following attributes contain information about the properties of the entire table control, such as the number of fixed columns:

   Choose the correct answer(s).

   - □ A Column
   - □ B General
   - □ C Special
   - □ D Row

2. You create fields in a table control using the _____ function.

   Fill in the blanks to complete the sentence.

3. The contents of the _____ are transported line by line to the corresponding work area in the ABAP program in the appropriate loop.

   Fill in the blanks to complete the sentence.

4. Briefly describe the steps involved in displaying buffered data from the internal table in the table control.

   _____
   _____
   _____
   _____

5. You can easily sort the table control display by a particular column using the table control attributes, _____ and _____.

   Fill in the blanks to complete the sentence.

6. Briefly explain the process of changing attributes temporarily.

   _____
   _____
   _____
   _____

Answers

1. The following attributes contain information about the properties of the entire table control, such as the number of fixed columns:

   Answer: B

   The general attributes contain information about the properties of the entire table control, such as the number of fixed columns.

2. You create fields in a table control using the Dict./Program fields function.

   Answer: Dict./Program fields

3. The contents of the table control are transported line by line to the corresponding work area in the ABAP program in the appropriate loop.

   Answer: table control

4. Briefly describe the steps involved in displaying buffered data from the internal table in the table control.

   Answer: The various steps are:

   - 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 fields of the current line of the table control in the work area with identical field names in the TOP Include.
   - For each line, the system copies the data from the work area with identical field names in the TOP Include to the corresponding fields of the relevant line of the table control.

5. 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.

   Answer: <wa_cols>-selected, <wa_cols>-screen-name

6. Briefly explain the process of changing attributes temporarily.

   Answer: 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.

Unit 7

Context Menus

Unit Overview

This unit explains how context menus are created, used, and modified. Mechanisms to include list output within screens are also described.

Unit Objectives

After completing this unit, you will be able to:

- Create context menus in your programs

Unit Contents

Lesson: Context Menus

Lesson Overview

In this lesson, you will learn to create a context menu from within the object list of the Object Navigator and to create a context menu directly in the Menu Painter. You will also learn to modify the context menus dynamically.

Lesson Objectives

After completing this lesson, you will be able to:

- Create context menus in your programs

Business Example

A travel agency needs to maintain flight booking data for the customers using screens in the application. You need to create context menus for the functions that are frequently used.

Creating Context Menus

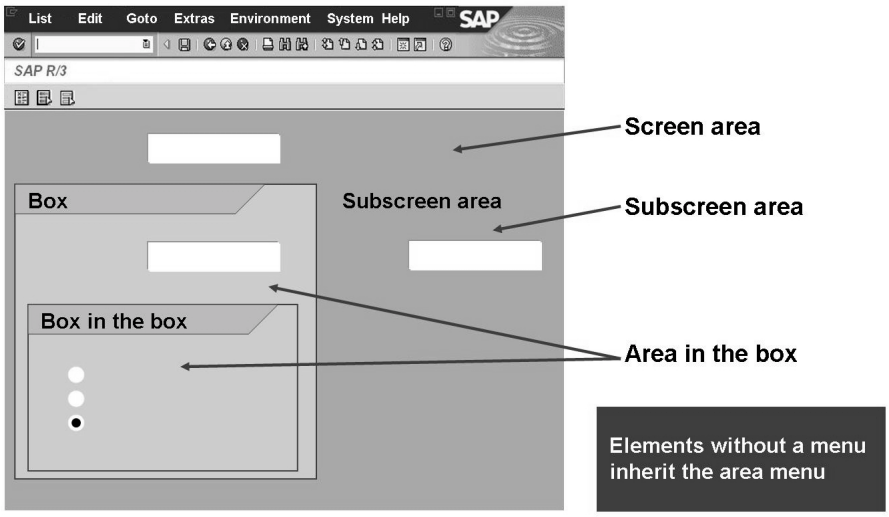


Figure 140: Context Menus

Context menus, available when users click the right mouse button or press 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, the cursor for SHIFT-F10, the mouse location for the 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 determine 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 context menu.

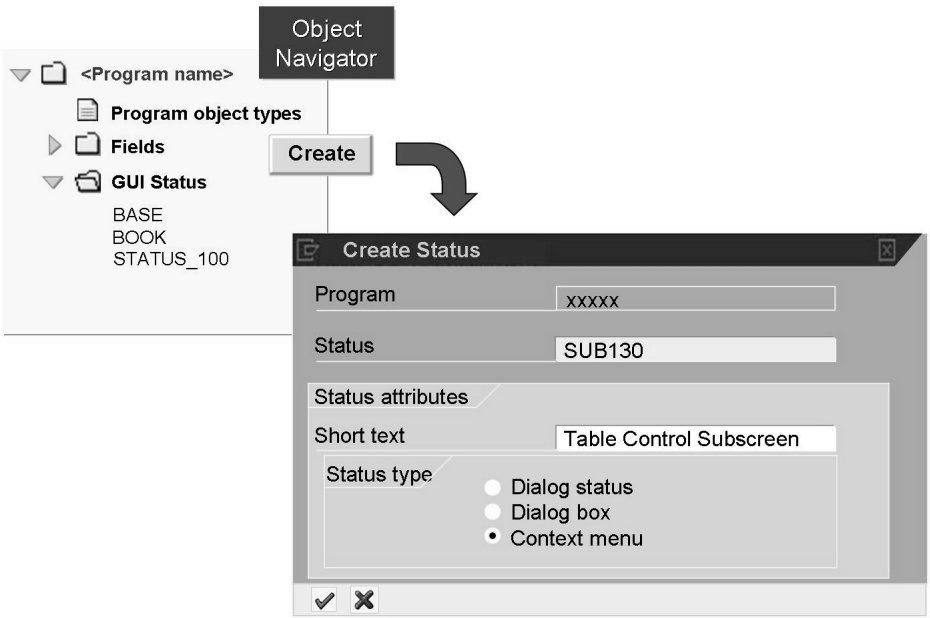


Figure 141: Creating a Context Menu

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 the status type *Context menu*.

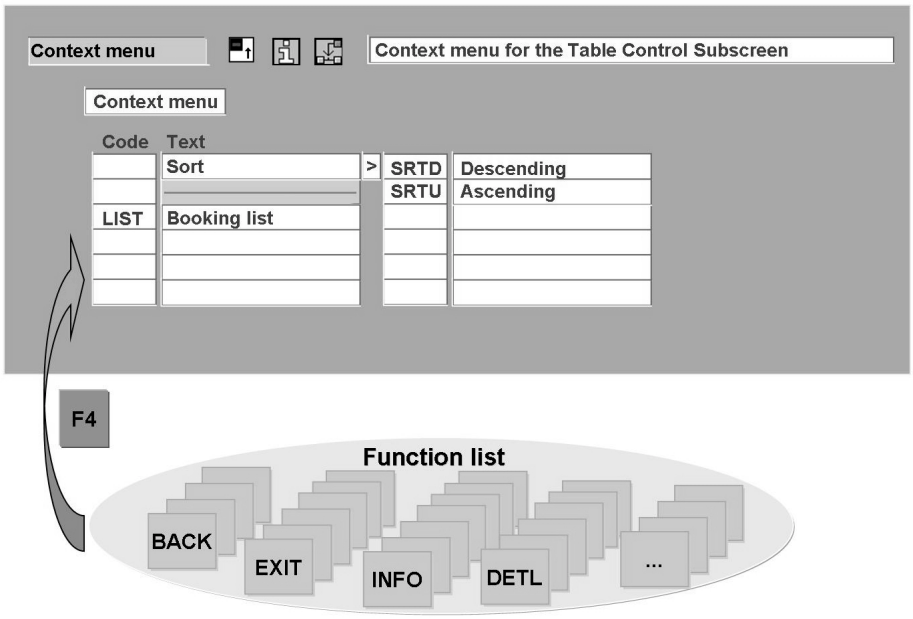


Figure 142: Creating a Context Menu: Assigning Functions

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.

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.

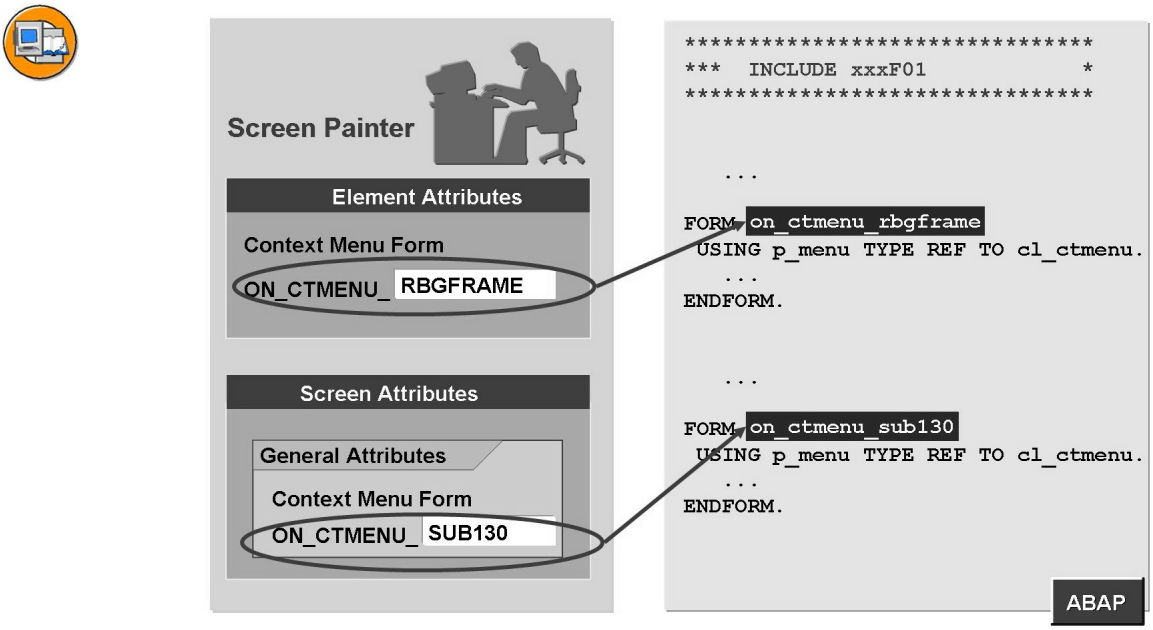


Figure 143: Linking Screen Objects

Right-clicking triggers a callback routine in your program. You must create the respective callback routines in your application program. They are named *ON_CTMENU_<arbitrarily>*. 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_USING p_menu TYPE REF TO cl_ctmenu.

ENDFORM.

You create the structure of the context menu by loading a statically defined menu or by adding entries dynamically with methods of class *CL_CTMENU*.

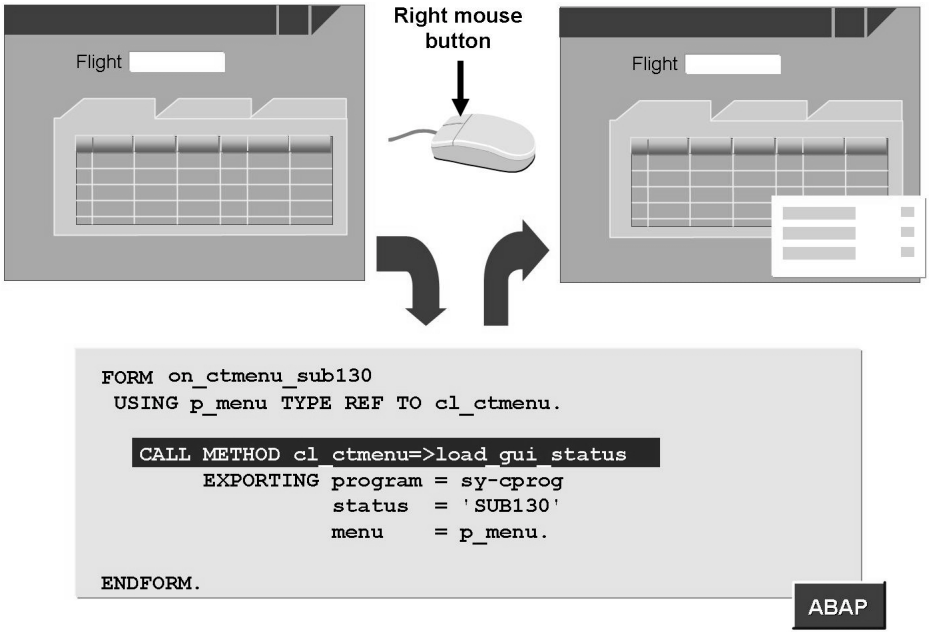


Figure 144: Using the Context Menu

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 the 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.

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

Method	Meaning
ADD_FUNCTION	Add a function
ADD_SEPARATOR	Add a separator
HIDE_FUNCTIONS	Hide functions
SHOW_FUNCTIONS	Show functions
DISABLE_FUNCTIONS	Disable functions
ENABLE_FUNCTIONS	Enable functions

Figure 145: Modifying Context Menus Dynamically

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 runtime, 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.

Exercise 9: Optional: Creating and Using a Context Menu

Exercise Objectives

After completing this exercise, you will be able to:

- Use context menus in your programs

Business Example

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

Task:

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_CONTROL2**. You can use the model solution **SAPMBC410ACONS_CONTEXTMENU** for orientation.

2. Create the GUI status *sub0130* with type *context menu* and the short description *Table control subscreen*.

   Assign the following functions to the menu:

Context menu	**Function code:**	**Function text:**
	SELE	Select all
	DSELE	Deselect all
	Separator	...
	SRTU	Sort in Ascending…
	SRTD	Sort in Descending…

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

3. In the screen attributes of 130, declare that you want to use subroutine *on_ctmenu_sub0130* to create the context menu.

4. Write the subroutine to create the context menu.

5. Create another GUI status *dyn0100* with type *context menu* for screen 100.

   Assign the following functions to the menu:

Continued on next page

Context menu	**Function code:**	**Function text:**
	EXIT	Exit
	BACK	Back
	CANCEL	Cancel
	Separator	...
	SAVE	Save
	Separator	...
	TIME	Time

Assign function type ' ' (space) to BACK, SAVE and TIME, but type 'E' to EXIT and CANCEL.

6. In the screen attributes of 100, declare that you want to use subroutine *on_ctmenu_dyn0100* to create the context menu.
7. Write the subroutine to create the context menu.
8. Activate the SAVE function for screen 100 if the user is in flight data maintenance mode.

Hint: You must pass the function code to the method in a table with type *ui_functions*.

Solution 9: Optional: Creating and Using a Context Menu

Task:

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_CONTROL2**. You can use the model solution **SAPMBC410ACONS_CONTEXTMENU** for orientation.

   a) See sample solution.

2. Create the GUI status *sub0130* with type *context menu* and the short description *Table control subscreen*.

   Assign the following functions to the menu:

Context menu	**Function code:**	**Function text:**
	SELE	Select all
	DSELE	Deselect all
	Separator	...
	SRTU	Sort in Ascending…
	SRTD	Sort in Descending…

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

   a) See sample solution.

3. In the screen attributes of 130, declare that you want to use subroutine *on_ctmenu_sub0130* to create the context menu.

   a) See sample solution.

4. Write the subroutine to create the context menu.

   a) See sample solution.

5. Create another GUI status *dyn0100* with type *context menu* for screen 100.

   Assign the following functions to the menu:

Continued on next page

Context menu	**Function code:**	**Function text:**
	EXIT	Exit
	BACK	Back
	CANCEL	Cancel
	Separator	...
	SAVE	Save
	Separator	...
	TIME	Time

Assign function type ' ' (space) to BACK, SAVE and TIME, but type 'E' to EXIT and CANCEL.

a) See sample solution.

6. In the screen attributes of 100, declare that you want to use subroutine *on_ctmenu_dyn0100* to create the context menu.

   a) See sample solution.

7. Write the subroutine to create the context menu.

   a) See sample solution.

8. Activate the SAVE function for screen 100 if the user is in flight data maintenance mode.

Hint: You must pass the function code to the method in a table with type *ui_functions*.

a) See sample solution.

Result

Model Solution SAPMBC410ACONS_CONTEXTMENU

Main program

```
* Data objects
INCLUDE MBC410ACONS_CONTEXTMENUTOP.

* PAI modules
INCLUDE MBC410ACONS_CONTEXTMENUI01.

* PBO modules
INCLUDE MBC410ACONS_CONTEXTMENUO01.
```

Continued on next page

Internal Use SAP Partner Only

```
* ABAP events
INCLUDE MBC410ACONS_CONTEXTMENUE02.

* Forms
INCLUDE MBC410ACONS_CONTEXTMENUF01.
```

Flow logic screen 100

No changes with regard to previous exercise.

Flow logic screen 110

No changes with regard to previous exercise.

Flow logic screen 120

No changes with regard to previous exercise.

Flow logic screen 130

No changes with regard to previous exercise.

Top include

No changes with regard to previous exercise.

PBO module include

No changes with regard to previous exercise.

PAI module include

No changes with regard to previous exercise.

Forms include

```
*---------------------------------------------------------------------*
* FORM    :  on_ctmenu_sub0130
*---------------------------------------------------------------------*
FORM on_ctmenu_sub0130
  USING p_menu TYPE REF TO cl_ctmenu.
  CALL METHOD cl_ctmenu=>load_gui_status
    EXPORTING
      program = sy-cprog
      status  = 'SUB0130'
      menu    = p_menu.
ENDFORM. "on_ctmenu_sub0130
*---------------------------------------------------------------------*
* FORM    :  on_ctmenu_dyn0100
*---------------------------------------------------------------------*
FORM on_ctmenu_dyn0100 USING p_menu TYPE REF TO cl_ctmenu.
```

Continued on next page


```
  DATA: l_fcodes TYPE ui_functions.
  CALL METHOD cl_ctmenu=>load_gui_status
    EXPORTING
      program = sy-cprog
      status  = 'DYN0100'
      menu    = p_menu.

  IF maintain_flights = 'X'.
    APPEND 'SAVE' TO l_fcodes.
    CALL METHOD p_menu->enable_functions
      EXPORTING
        fcodes = l_fcodes.

  ENDIF.

ENDFORM. "on_ctmenu_dyn0100
```


Internal Use SAP Partner Only

Lesson Summary

You should now be able to:

- Create context menus in your programs

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Create context menus in your programs

Test Your Knowledge

1. List the rules followed when designing context menus.

Internal Use SAP Partner Only

Answers

1. List the rules followed when designing context menus.

   Answer: The rules followed are:

   - Do not use any functions that cannot be found elsewhere in the system.
   - 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.

Course Summary

You should now be able to:

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

Appendix 1

Appendix

This section contains additional material to be used for reference purposes.

This material is not part of the standard course.

Therefore, it may not be covered during the course presentation.

Lists in Screen Programming

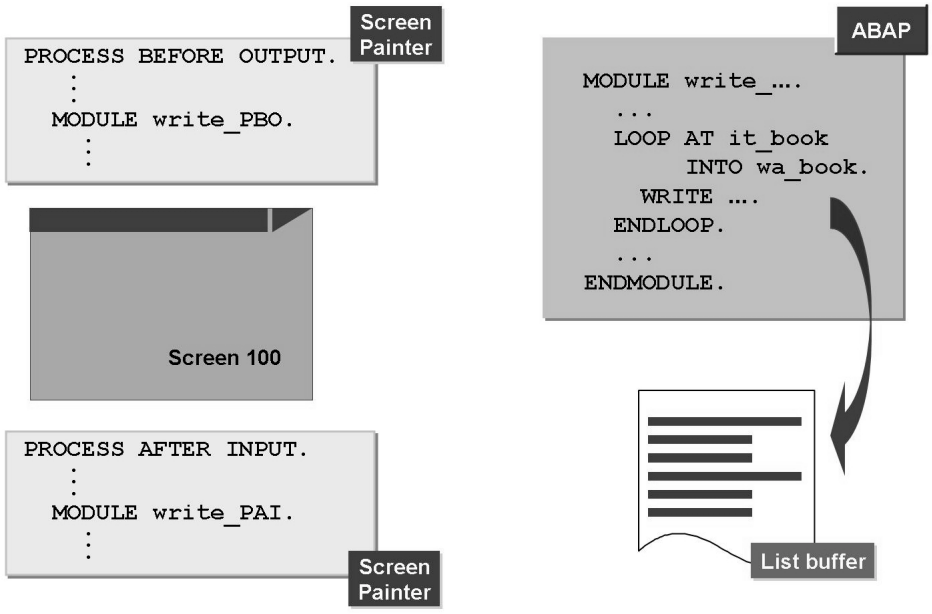


Figure 146: Creating a List Buffer

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.

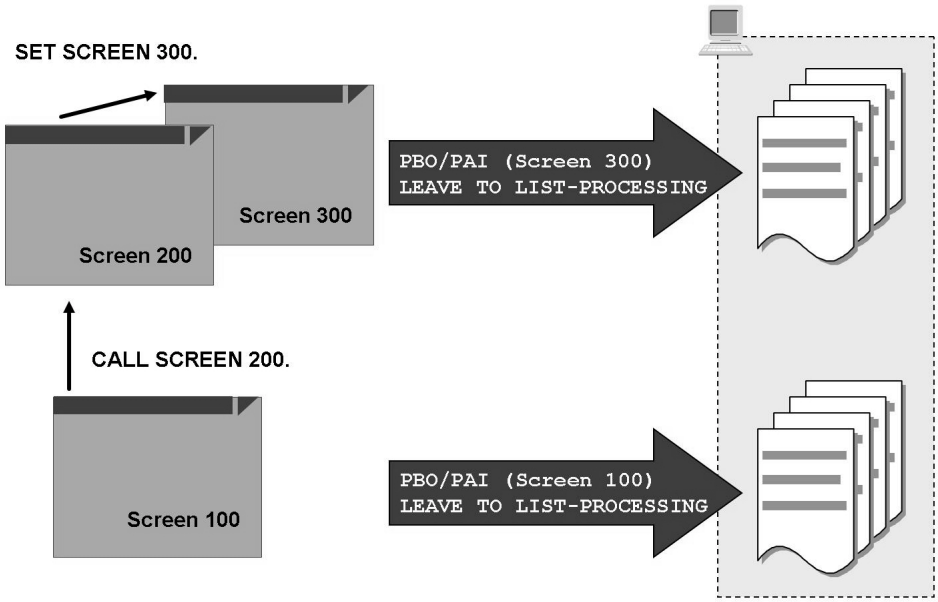


Figure 147: List Display at the Front End

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.

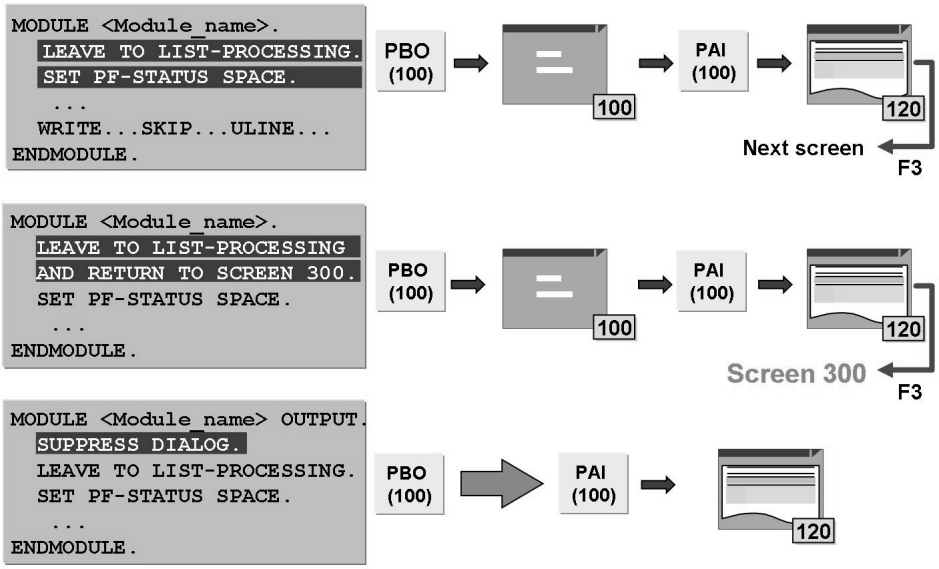


Figure 148: Displaying List on a Screen

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, it 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 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.

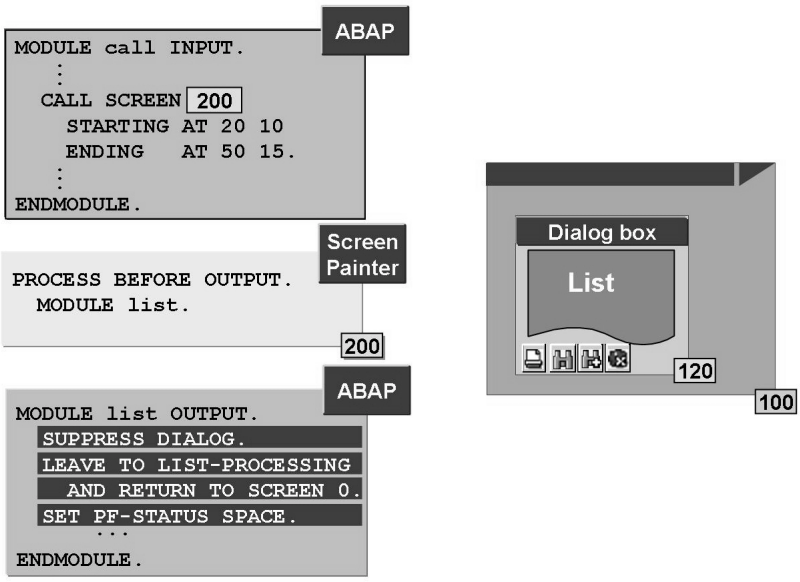


Figure 149: Lists in Modal Dialog Boxes

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.

To return to the calling screen when the user leaves the list, use the statement

LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.

Transaction Variants

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, screens)**
- **Different scope:**
  - **System: Global fixed values**
  - **Transaction: Transaction variants**
- **Standard variants or individual variants**
- **WYSIWYG maintenance with special recording function**

Figure 150: Transaction Variants: Objectives

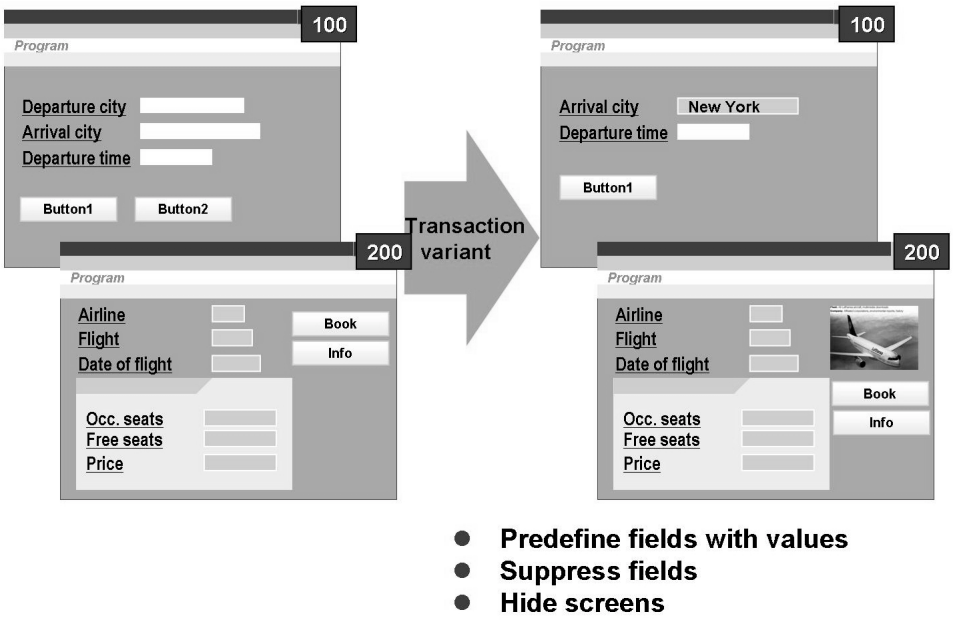


Figure 151: Transaction Variants: Example

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.

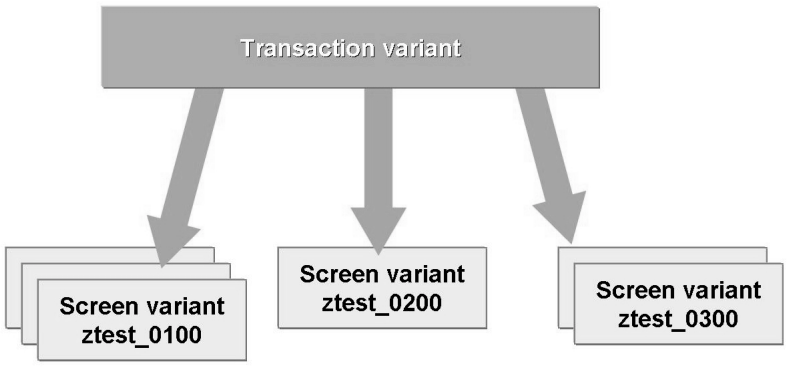


Figure 152: Transaction Variants: Principle

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.

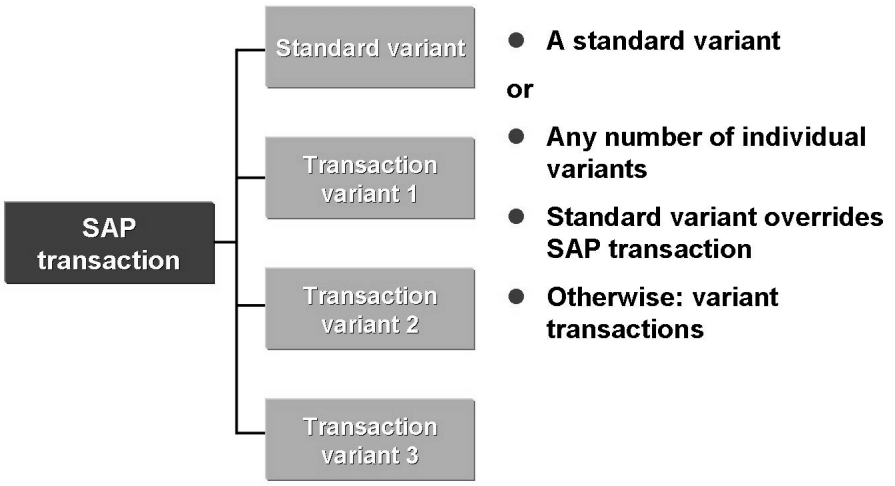


Figure 153: Transaction Variants: Options

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**

Figure 154: Creating Transaction Variants

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.

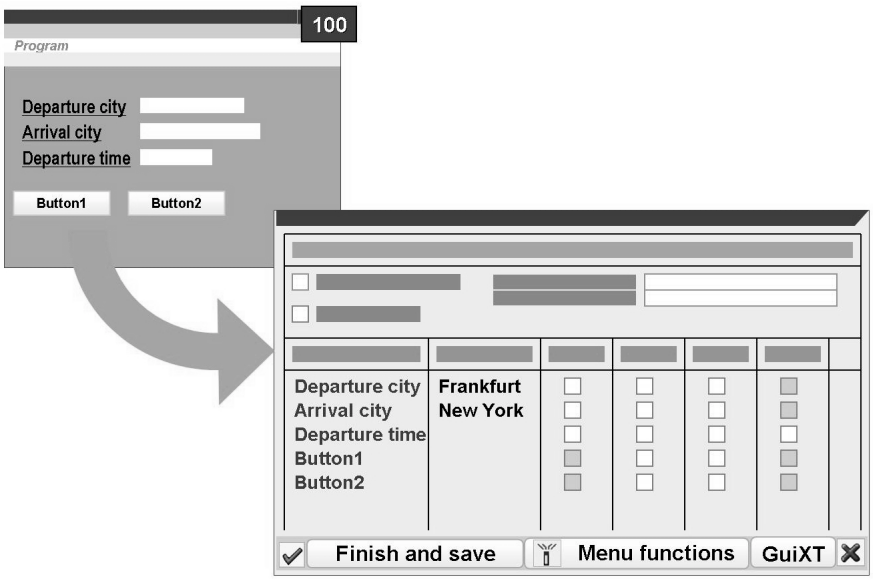


Figure 155: Transaction Variants: Evaluating Fields

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.

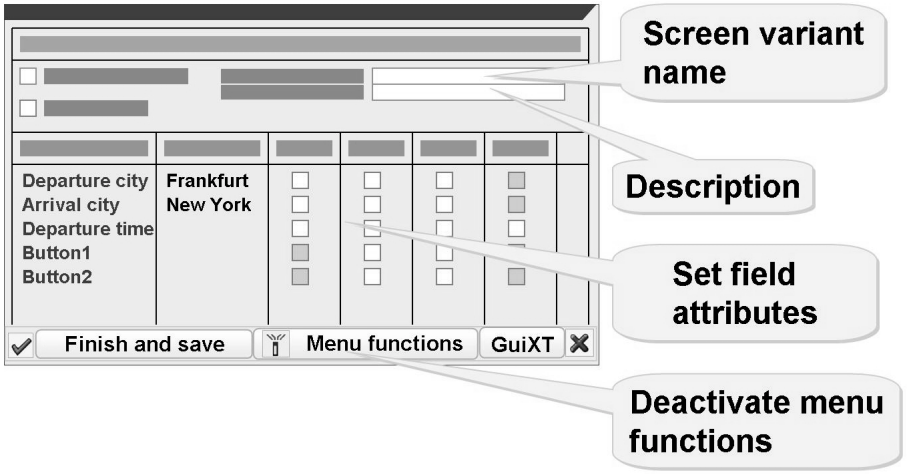


Figure 156: Screen Variants

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

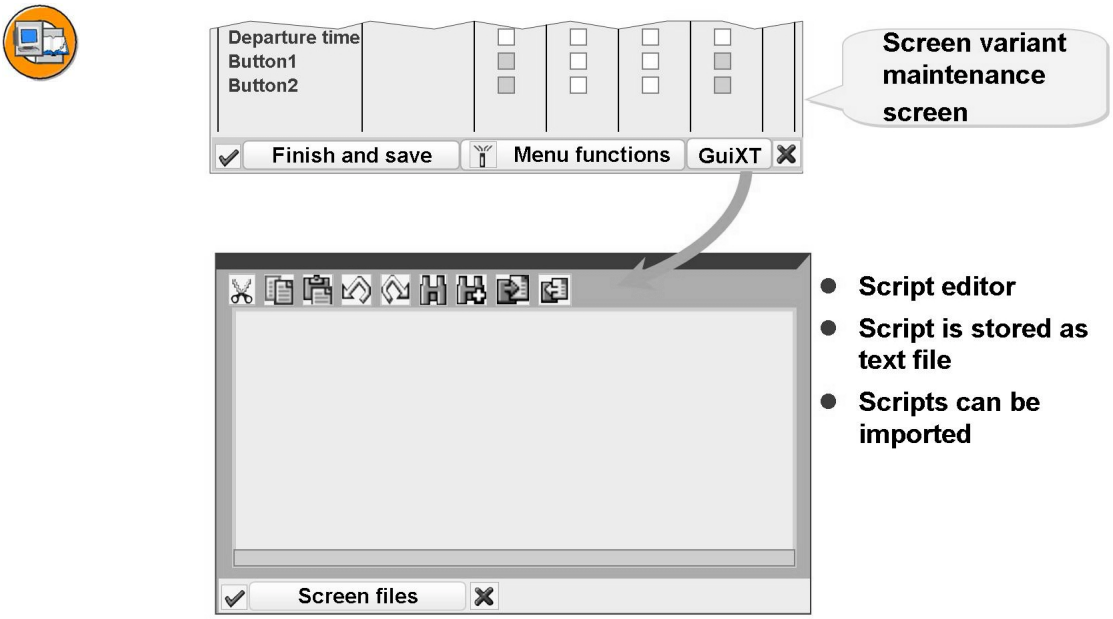


Figure 157: GuiXT

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.

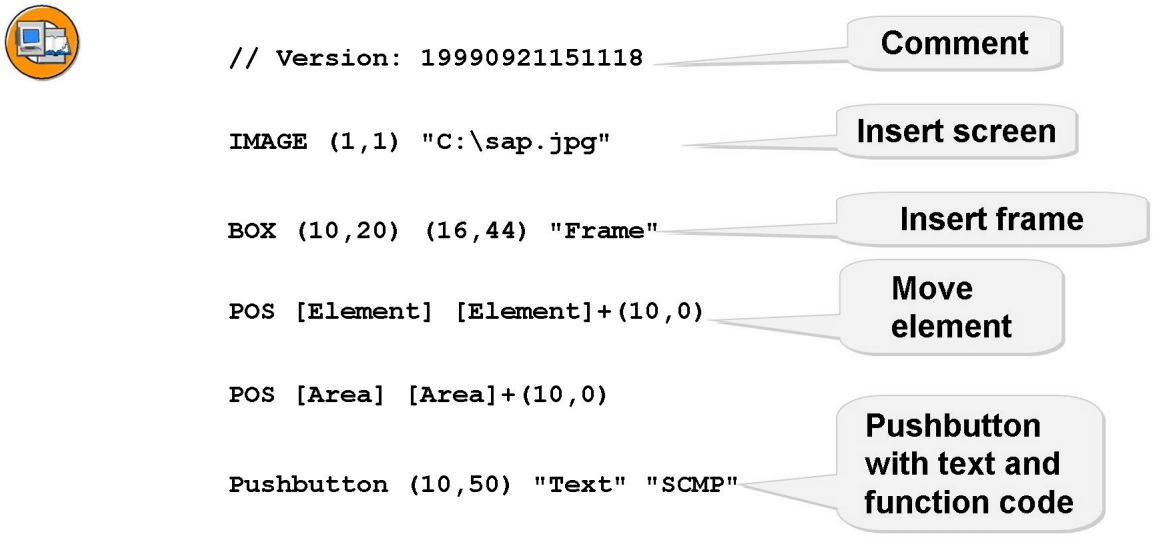


Figure 158: GuiXT: Script Language

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.

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*.

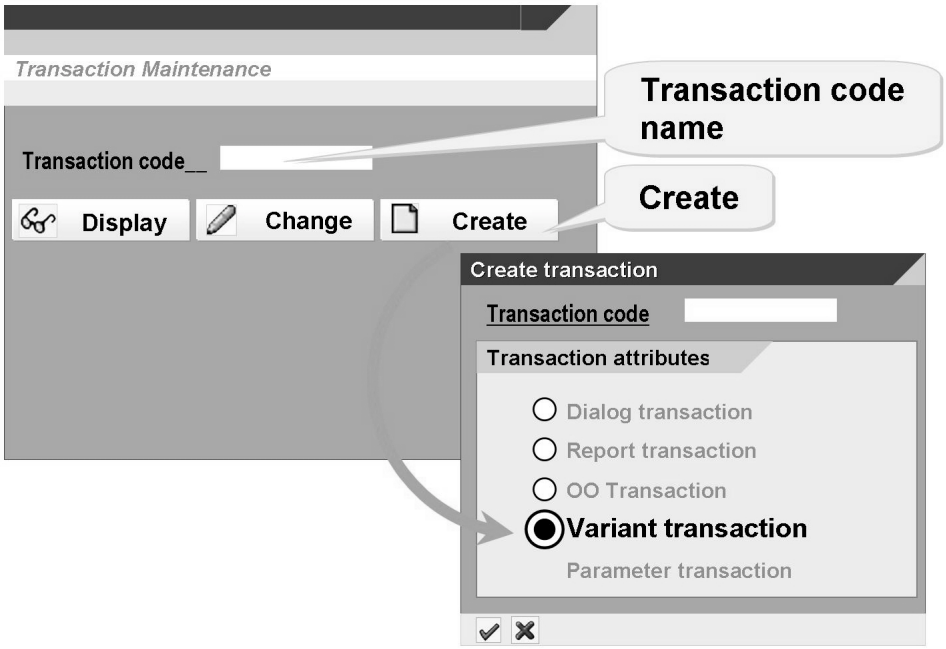


Figure 159: Creating Variant Transactions

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.

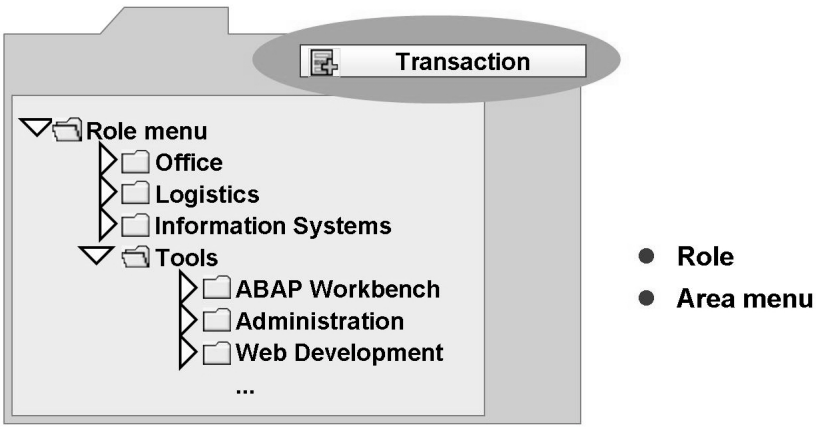


Figure 160: Inserting Variant Transactions into Menus

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.

Control Framework

Basic Terminology

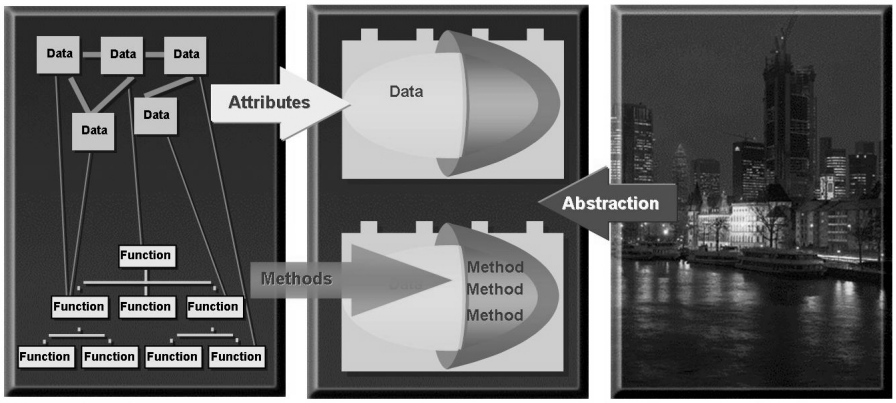


Figure 161: Object Orientation

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 or data. The behavior of the objects is described with methods or 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.

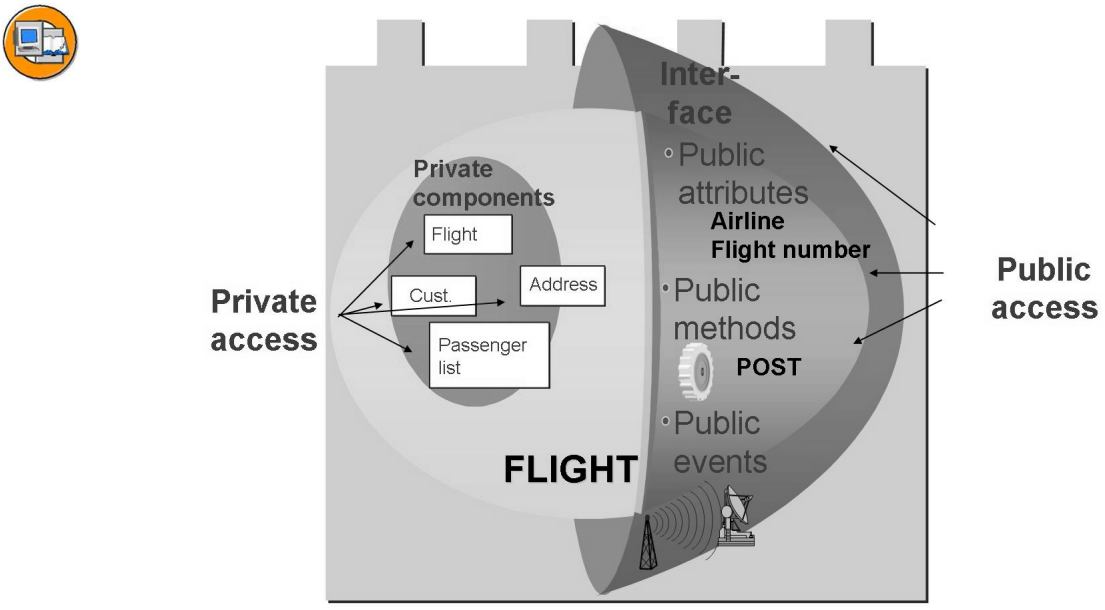


Figure 162: Objects

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 or 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.

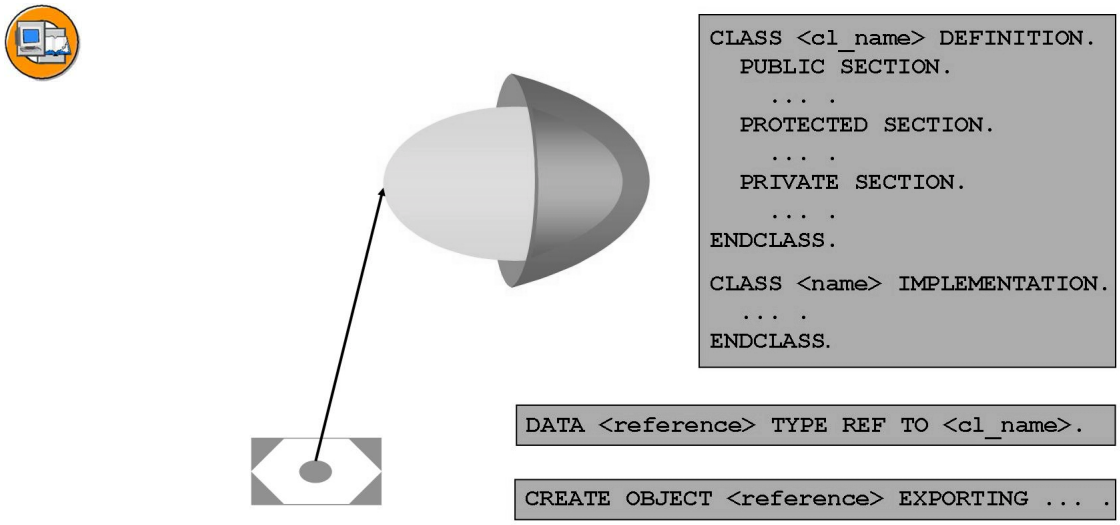


Figure 163: Classes, References, and Objects

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. You can then create or instantiate an object of this class using the ABAP object keyword CREATE OBJECT.

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.

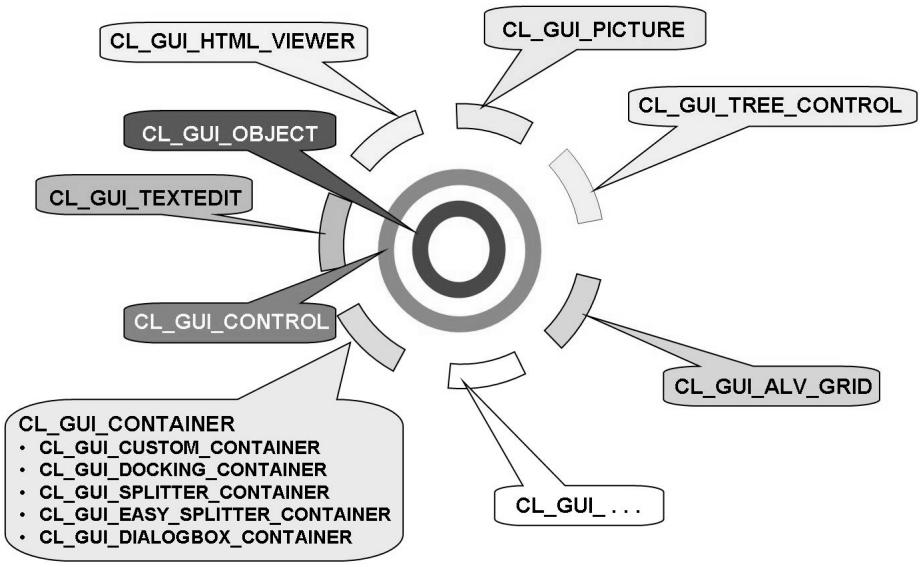


Figure 164: CFW as Seen by the Programmer: Classes

To use the Control Framework as described here, you need at least SAP R/3 4.6A with a locally installed SAP GUI with at least 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

Implementation of controls

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**

Figure 165: Principles of Control Processing

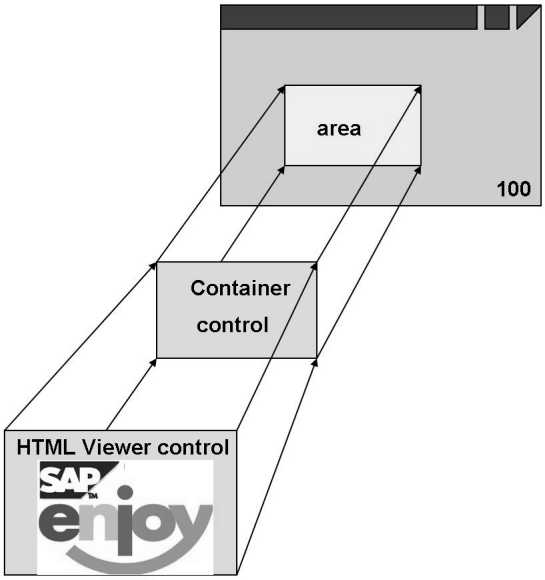


Figure 166: Screen Element Custom Container: Use

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 runtime and then reserves space for your application control.

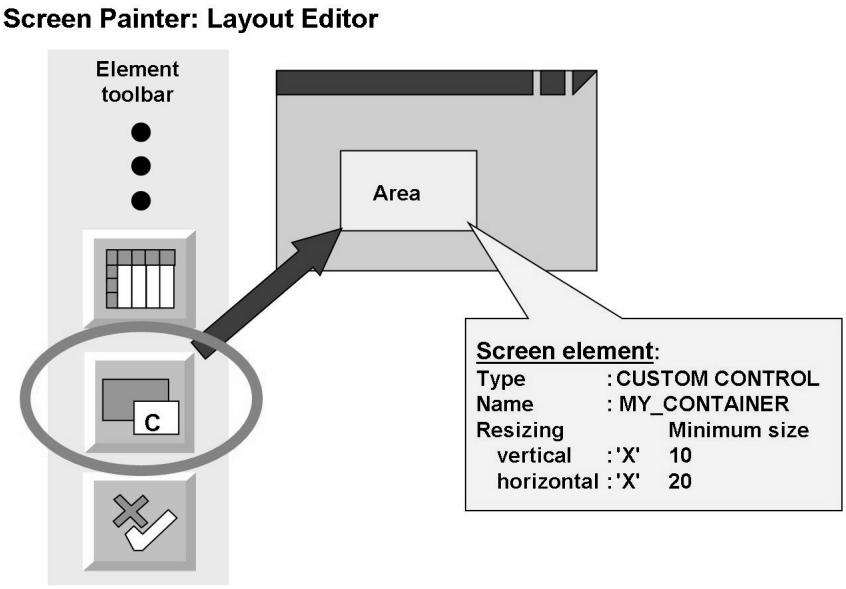


Figure 167: Creating a Screen Element Custom Container

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.
```

Figure 168: Creating a Container Control Instance

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.

```
  ...
DATA: container TYPE REF TO cl_gui_custom_container,
      html_viewer TYPE REF TO cl_gui_html_viewer,
      url(255) value 'HTTP://WWW.SAP-AG.DE' .

START-OF-SELECTION.

  CALL SCREEN '0100'.
```

```
PROCESS BEFORE OUTPUT.

MODULE create_objects.

MODULE set_url.

PROCESS AFTER INPUT.

MODULE exit AT EXIT-COMMAND.
```

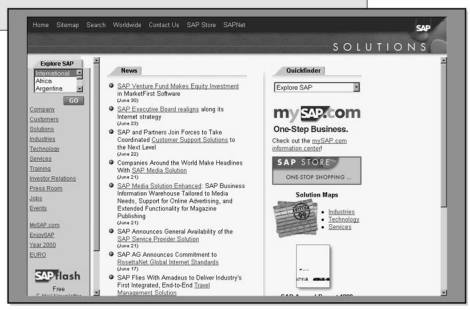


Figure 169: Example HTML Viewer I

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.
```

Figure 170: Example HTML Viewer II

You need to generate only a single instance of the HTML Viewer at runtime. 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.

Communication with controls

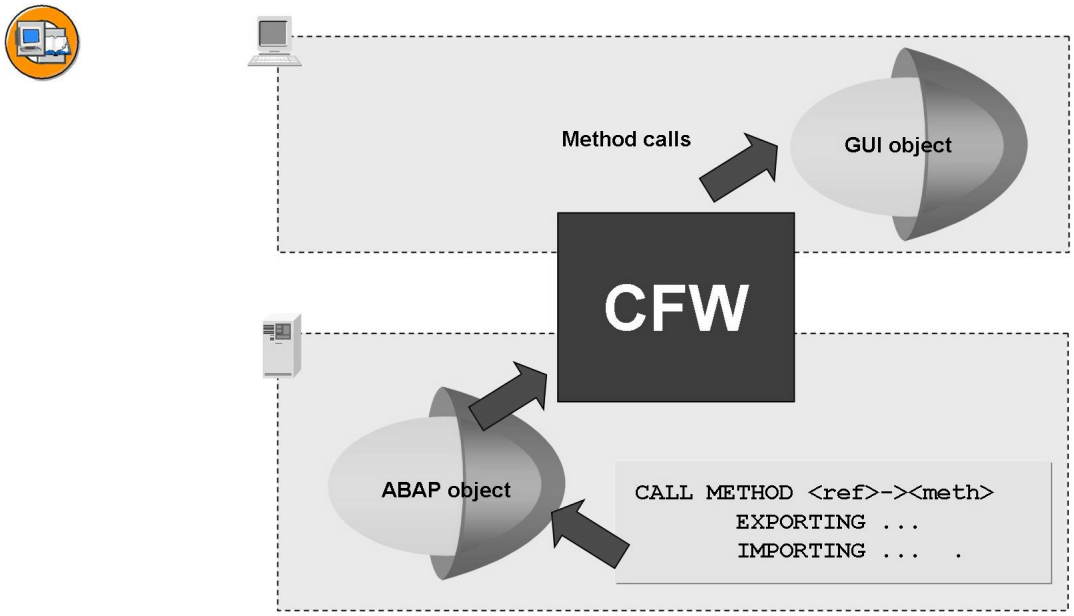


Figure 171: Method Calls

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.

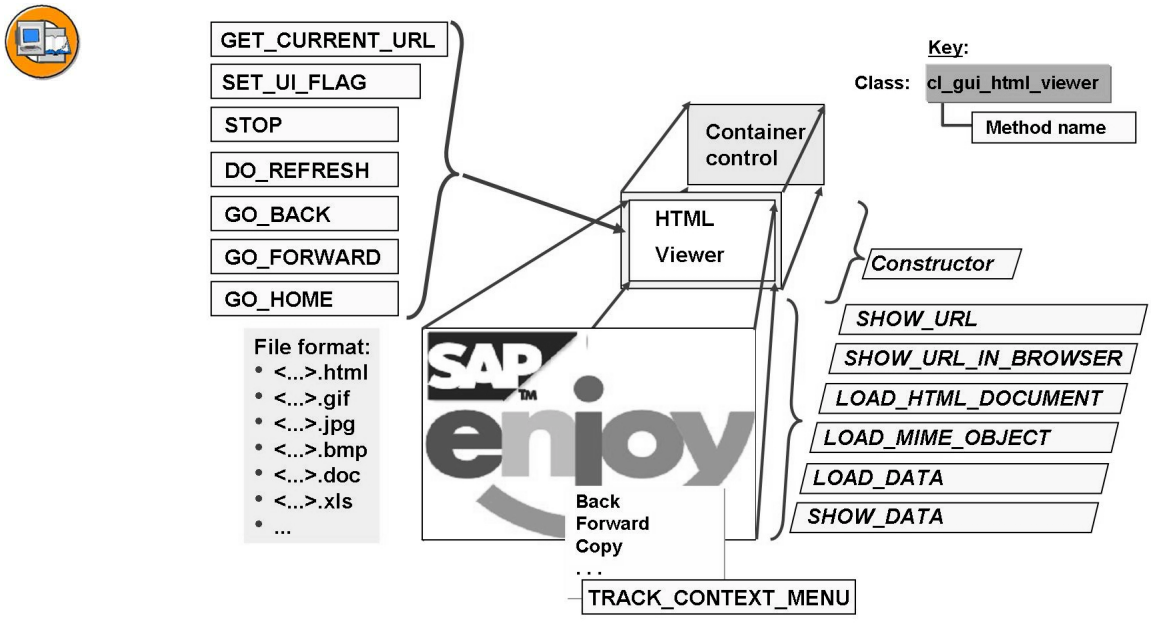


Figure 172: SAP HTML Viewer: Features

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.

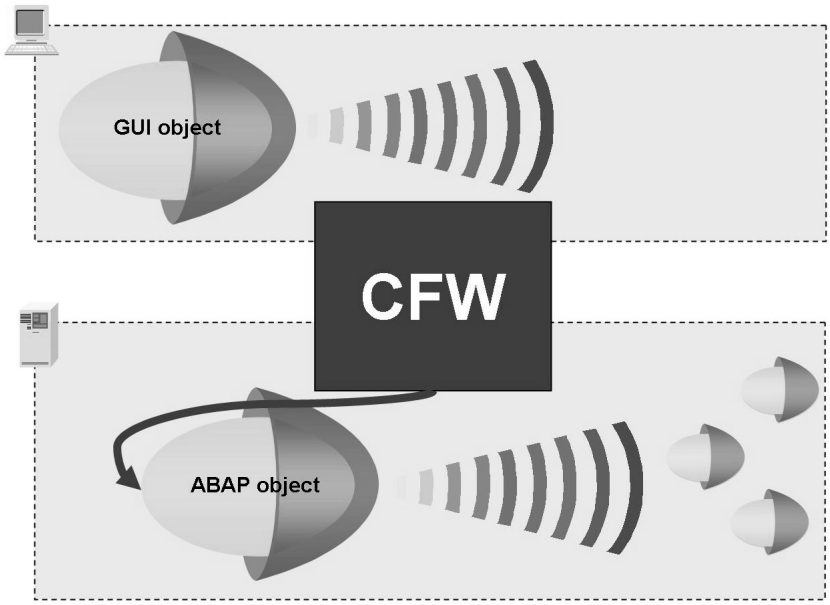


Figure 173: Events

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 runtime.

A class can thus define static events and an object can trigger events at runtime 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.

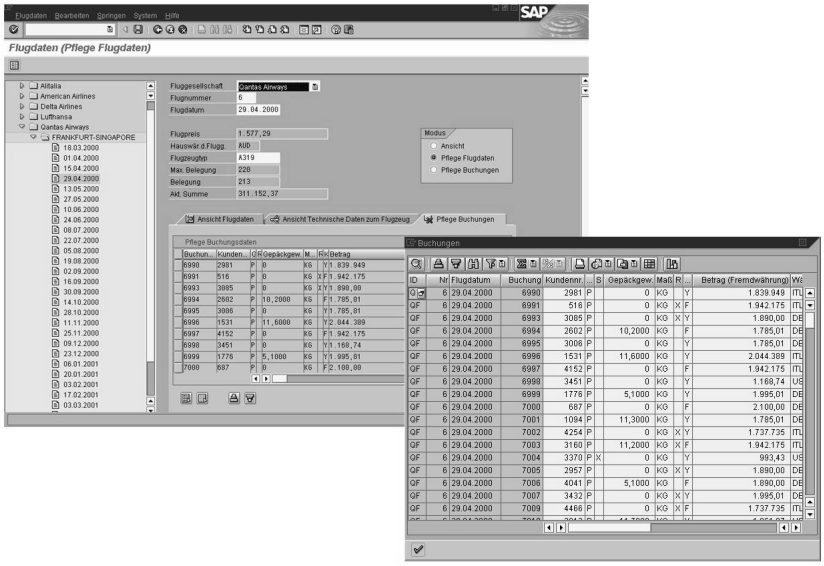


Figure 174: Creating a Model Solution Using the EnjoySAP Controls

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.

Sources of information

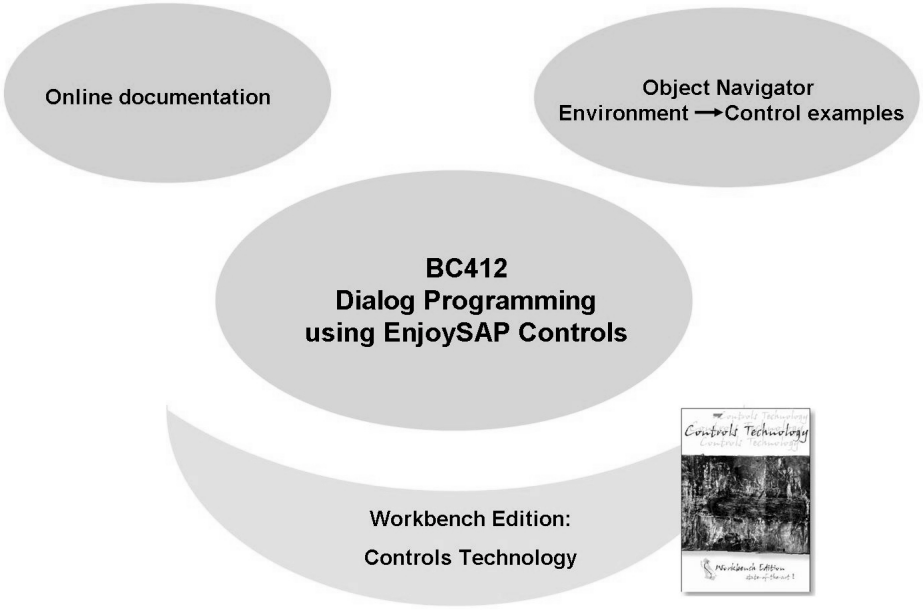


Figure 175: Sources of Information

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/.

Search Helps

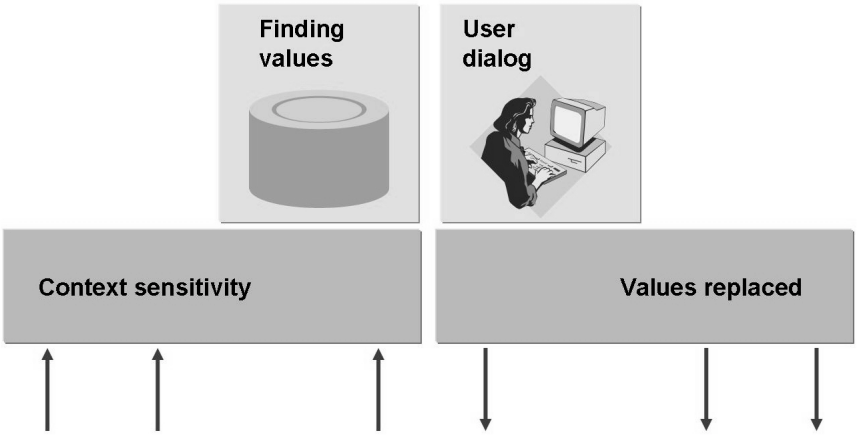


Figure 176: Requirements of F4 Help

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.

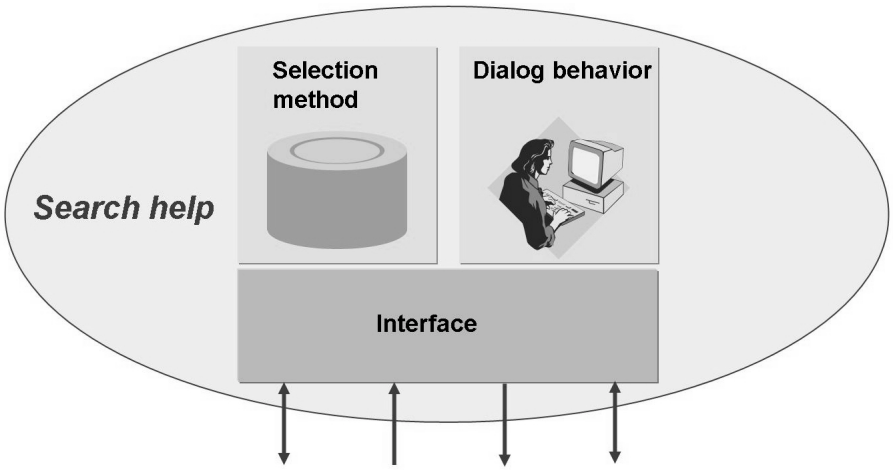


Figure 177: ABAP Dictionary Object: Search Help

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.

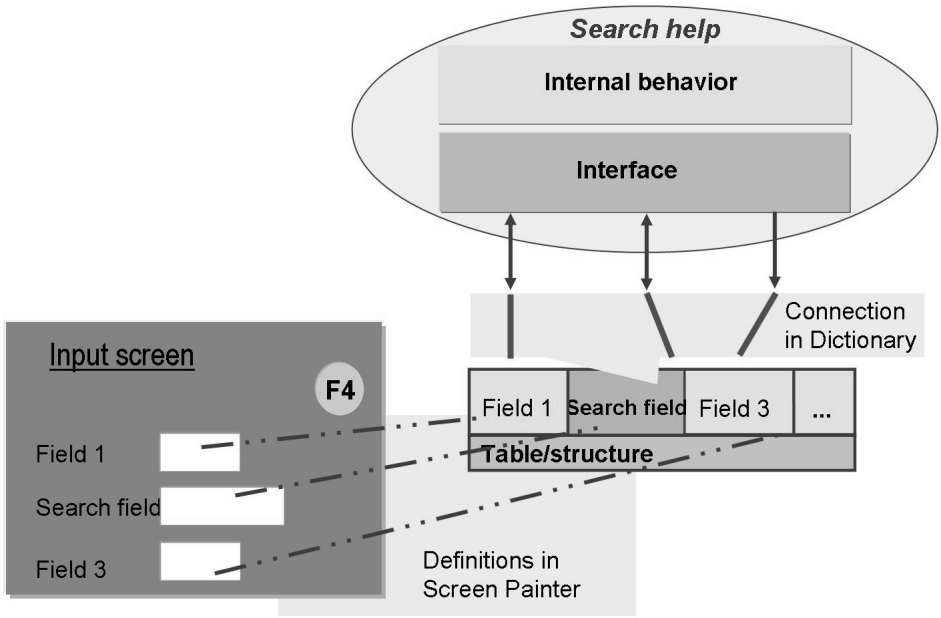


Figure 178: Using Search Helps

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.

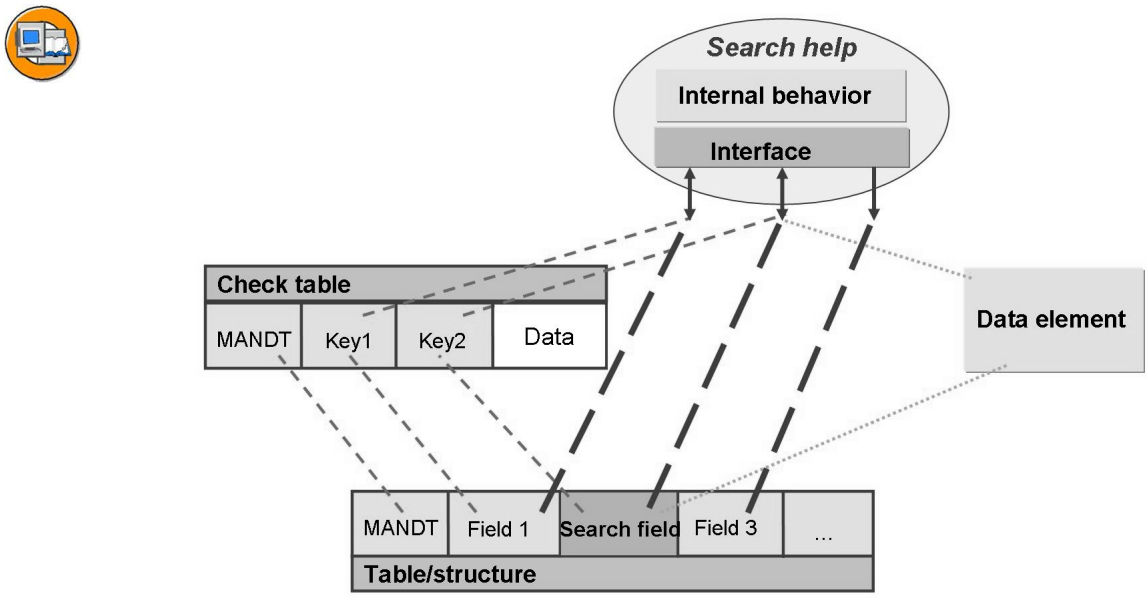


Figure 179: Search Help Assignment in ABAP Dictionary

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.

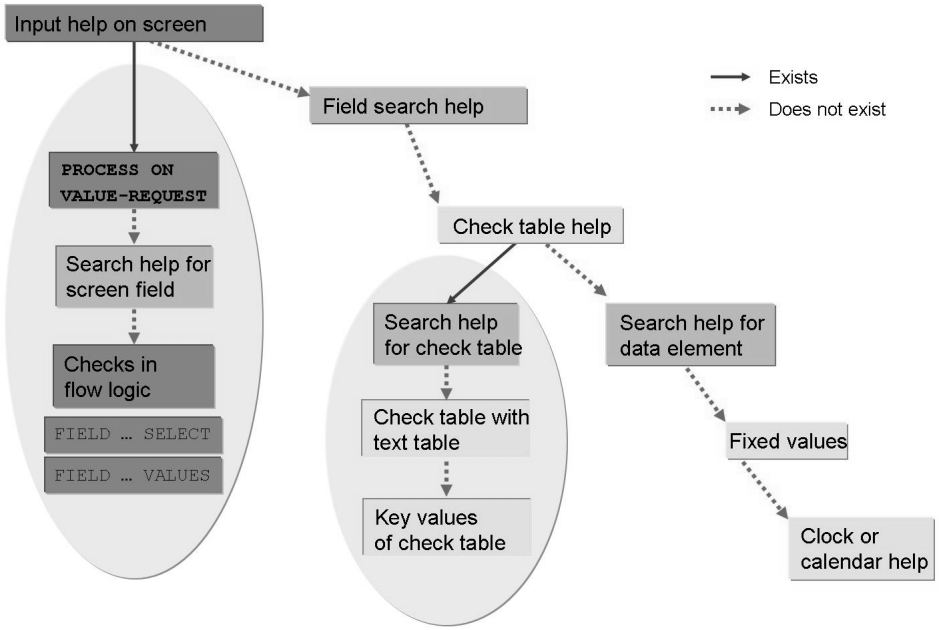


Figure 180: Input Help Mechanisms

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.

Tabstrip Control on selection screen

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
```

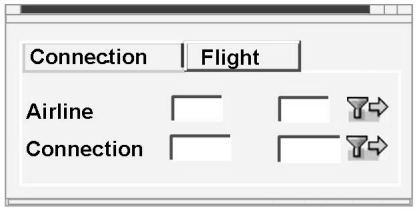


Figure 181: Defining Tabstrip Controls on the Selection Screen

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 /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 runtime. You can also change an existing assignment at runtime. 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.

Table Controls: Scrolling Page by Page

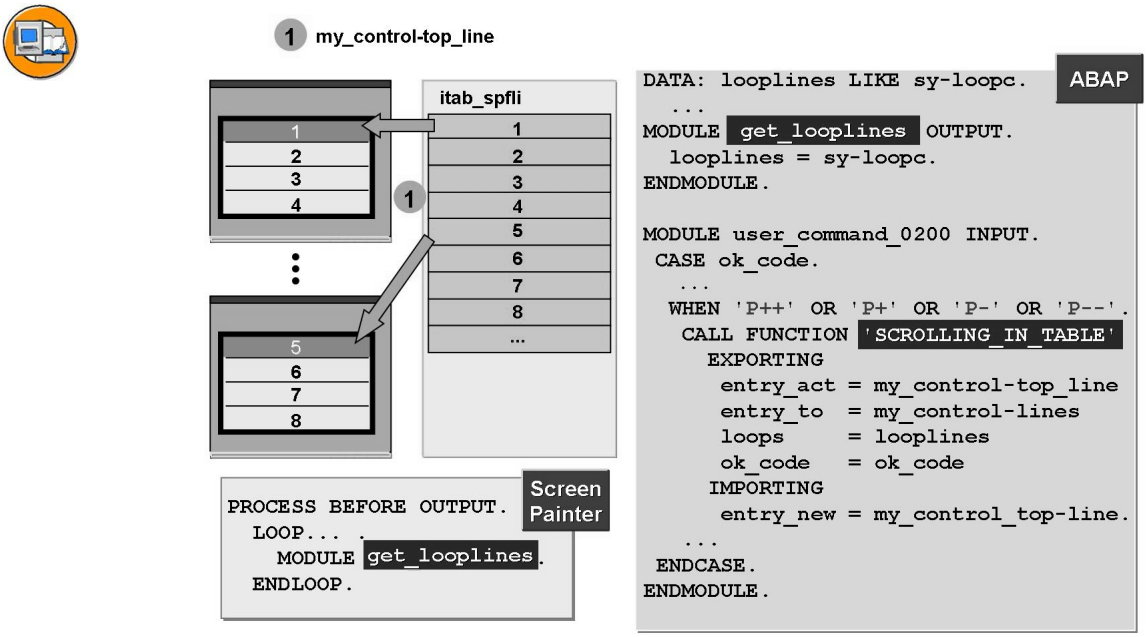


Figure 182: Table Controls: Scrolling Page by Page - Example

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.

Index

Internal Use SAP Partner Only

R

S

T

Internal Use SAP Partner Only

Feedback

SAP AG has made every effort in the preparation of this course to ensure the accuracy and completeness of the materials. If you have any corrections or suggestions for improvement, please record them in the appropriate place in the course evaluation.

Internal Use SAP Partner Only