

## **Objective**

**The following section explains :**

- **Structure of transaction**
- **The flow logic, Screen painter and Menu painter**
- **Input checks, changing of input values**
- **Error handling**
- **Step loops and table control**
- **Field help and value help**
- **Inserting / Updating of Database**

- **A Transaction is a program that conducts a dialog with the User**
- **In a typical dialog, the system displays the screen on which the user can enter or request information.**
- **As per the user input or request, transaction is used to**
  - **Branch to the next Screen**
  - **Display an output**
  - **Change/Update the database**

## ABAP/4 Development Workbench

**ABAP/4  
Dictionary**

**Screen  
Painter**

**ABAP/4  
Editor**

**Menu  
Painter**

## Runtime environment

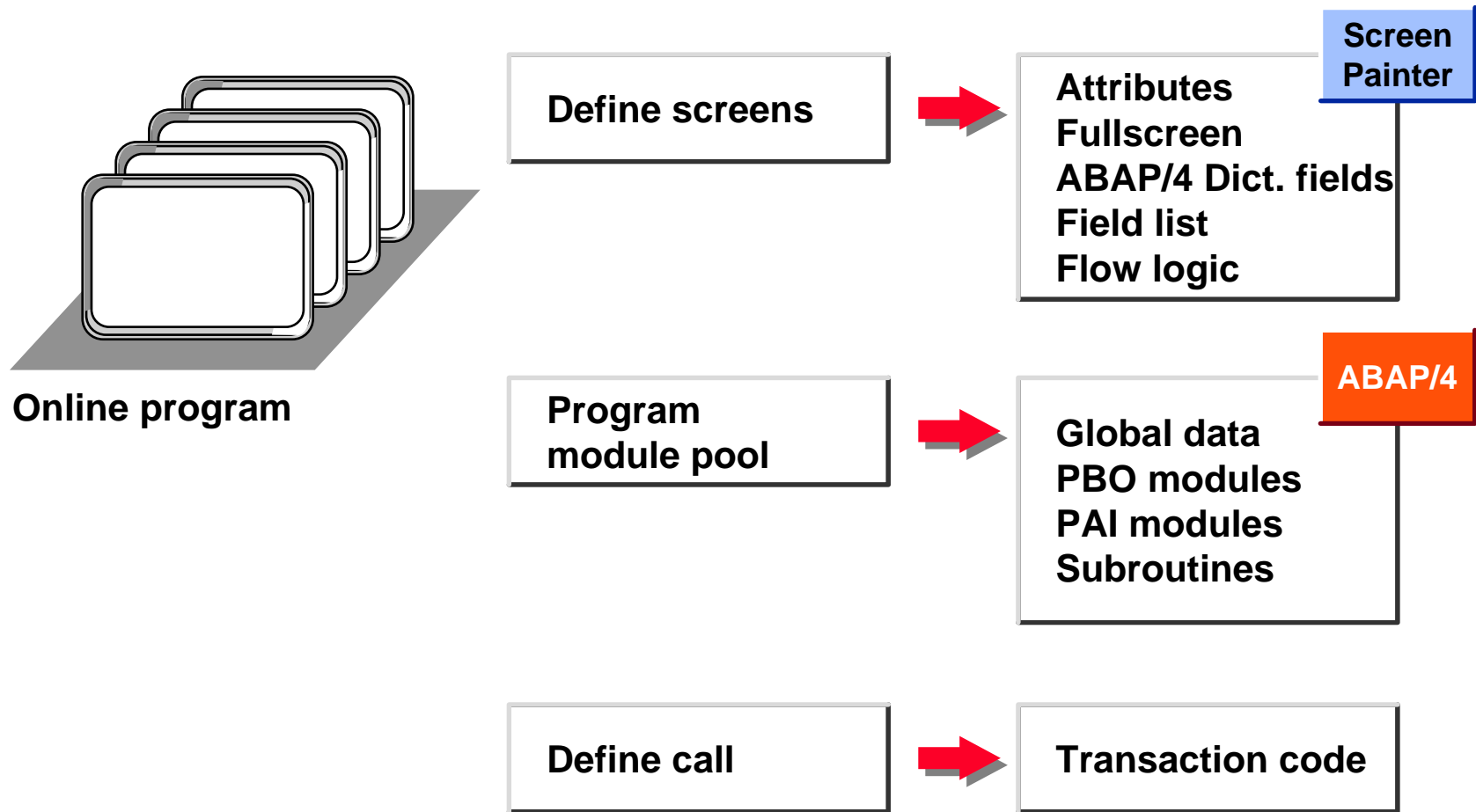
**Online processor**

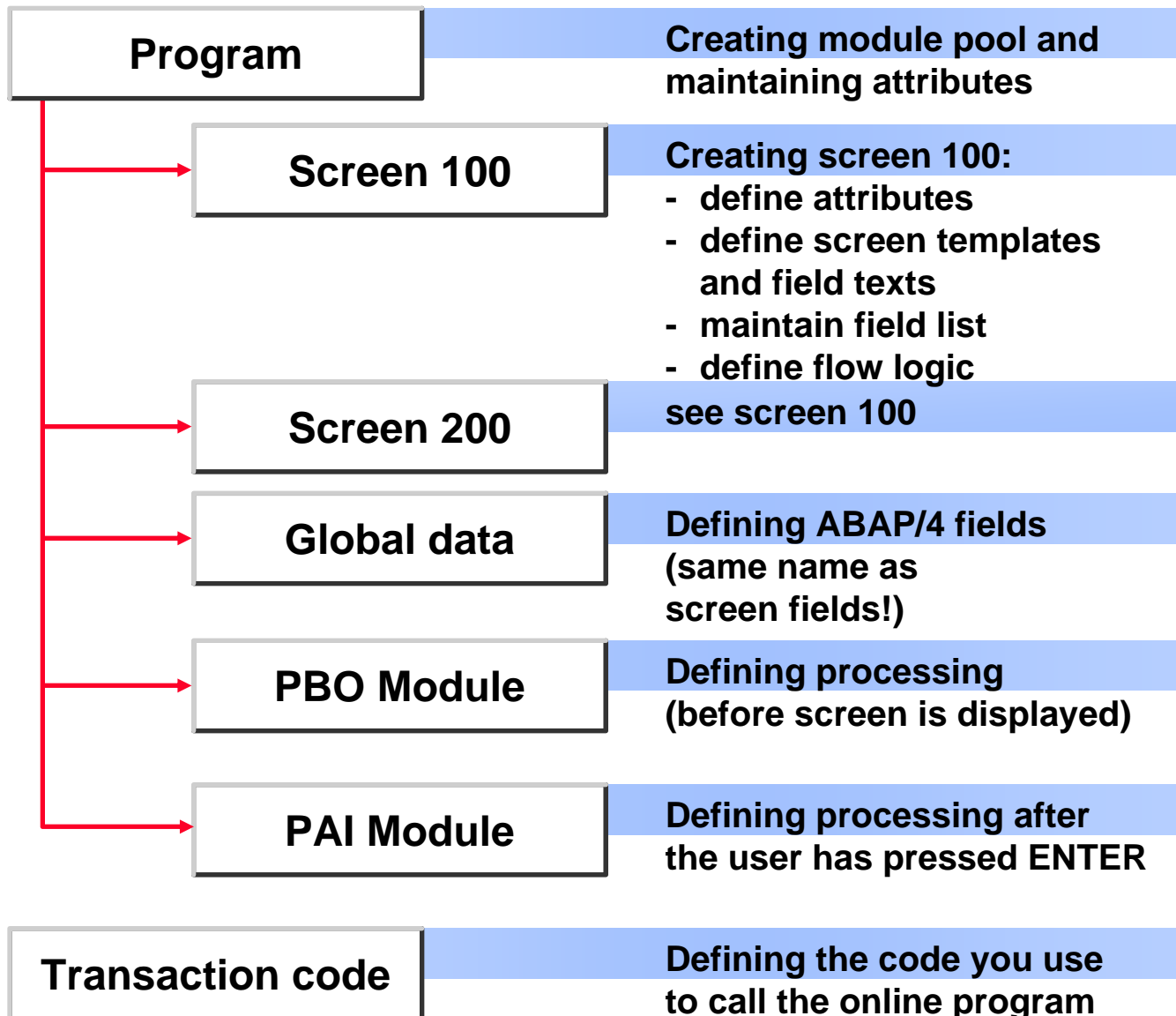
**ABAP/4 Processor**

- Structure of the Dialog Programming
- SCREEN Painter
- MENU Painter
- Input Checks
- Error Handling
- Flow Logic
- Screen Modification
- Table Control and Step Loop
- Branching to List Processing

## **Program Name**

- Dictionary Structure**
- Global data**
- PBO modules**
- PAI modules**
- Subroutines**
- Screens**
- GUI Status**
- Transaction code**





**Each Screen contains fields used to display or request Information. The fields can be text Sting, Input/Output fields, Radio Buttons, Check boxes or Pushbuttons .**

**Each screen consists of**

- **Screen Attributes**
- **Screen Elements**
- **Screen Fields**
- **Screen Flow Logic**



- **Program (type M)**

- **Screen Number** : A four-digit number, unique within the ABAP program, that identifies the screen within the program.

- **Screen Type** : A normal screen occupies a whole GUI window. Modal dialog boxes only cover a part of a GUI window. A subscreen is a screen that you can display in a subscreen area on a different screen in the same ABAP program.

- **Next Screen** : Specifies the next screen.

- **Hold Data** : If the user calls the screen more than once during a terminal session, he or she can retain changed data as default values.

**Text Fields**: Display elements, which cannot be changed either by the user or by the ABAP program.

**Input/Output Fields**: Used to display data from the ABAP program or for entering data on the screen. Linked to screen fields

**Radio Buttons**: Special input/output fields that are combined into groups. Within a radio button group, only a single button can be selected at any one time.

**Check boxes**: Special input/output fields which the user can select (value 'X') or deselect (value SPACE).

**Pushbuttons**: Elements on the screen that trigger the PAI event of the screen flow logic when chosen by the user. There is a function code attached to each pushbutton, which is passed to the ABAP program when it is chosen.

**Subscreen**: Area on the screen in which you can place another screen.

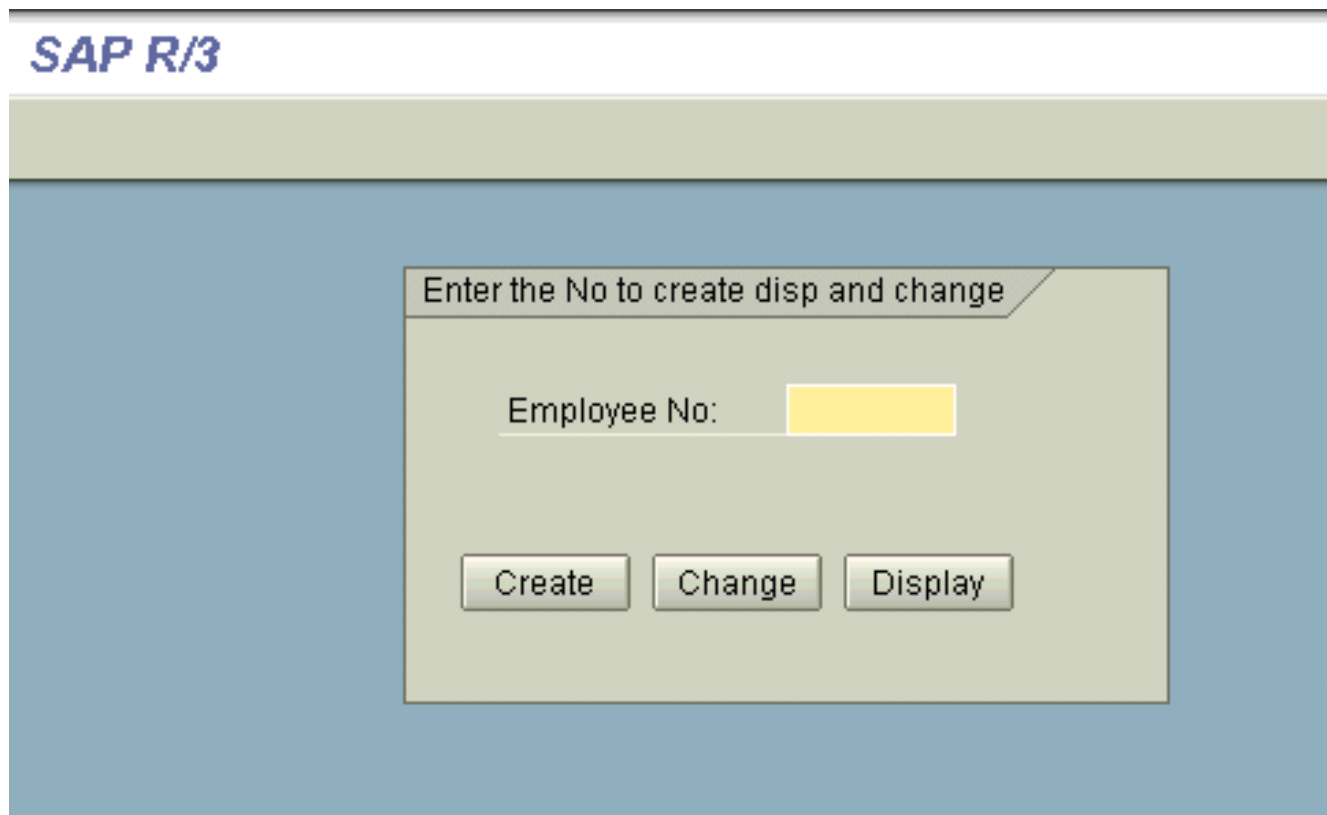
**Table Controls**: Tabular input/output fields.

**Tab Strip Controls**: Areas on the screen in which you can switch between various pages.

**Status Icons**: Display elements, indicating the status of the application program.

**Ok Code Field**: Every screen has a twenty-character OK\_CODE field (also known as the function code field), which is not displayed on the screen. User actions that trigger the PAI event also place the corresponding function code into this field, from where it is passed to the ABAP program. You can also use the command field in the standard toolbar to enter the function code. You must assign a name to the OK\_CODE field to be able to use it for a particular screen.

**Screen fields are fields in the working memory of a screen. Their contents are passed to identically-named fields in the ABAP program in the PAI event, and filled from the same identically-named fields in the program in the PBO event. The screen fields are linked with the input/output fields.**



The screenshot shows an SAP R/3 interface. At the top left, the text "SAP R/3" is displayed in blue. Below this is a light green horizontal bar. The main area has a blue background. In the center, there is a light green dialog box with a title bar that reads "Enter the No to create disp and change". Inside the dialog box, there is a label "Employee No:" followed by a yellow rectangular input field. Below the input field, there are three buttons: "Create", "Change", and "Display".

## Cont..

Screen number: 100 Active

Attributes Element list Flow logic

General attr. Texts/ I/O templates Special attr. Display attr. Mod. groups / functions

H..	MName	Type...	Li...	C...	D...	Vi...	H...	S...	Format	In...	O...	Out...	Di...	Dic...	Property list
	##AUTOTEXT005	Box	2	21	40	40	8			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	ZEMP-EMPNO	Text	4	26	12	12	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	F	➔ Properties
	EMPNO	I/O	4	41	8	8	1	<input type="checkbox"/>	NUMC	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	##AUTOTEXT002	Push	7	24	8	8	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		➔ Properties
	##AUTOTEXT003	Push	7	34	8	8	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		➔ Properties
	##AUTOTEXT004	Push	7	44	8	8	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		➔ Properties
	OK_CODE	OK	0	0	20	20	1	<input type="checkbox"/>	OK				<input type="checkbox"/>		

**Screen flow logic contains the procedural part of a screen. The language used to program screen flow logic has a similar syntax to ABAP, but is not part of ABAP itself. It is otherwise referred to as Screen Language. It contains no explicit data declarations. It serves as a container for processing blocks.**

**There are four event blocks, each of which is introduced with the screen keyword PROCESS:**

**PROCESS BEFORE OUTPUT.**

**...**

**PROCESS AFTER INPUT.**

**...**

**PROCESS ON HELP-REQUEST.**

**...**

**PROCESS ON VALUE-REQUEST.**

**The screen flow logic must contain at least the two statements PROCESS BEFORE OUTPUT and PROCESS AFTER INPUT in the correct order.**

**Cont..**

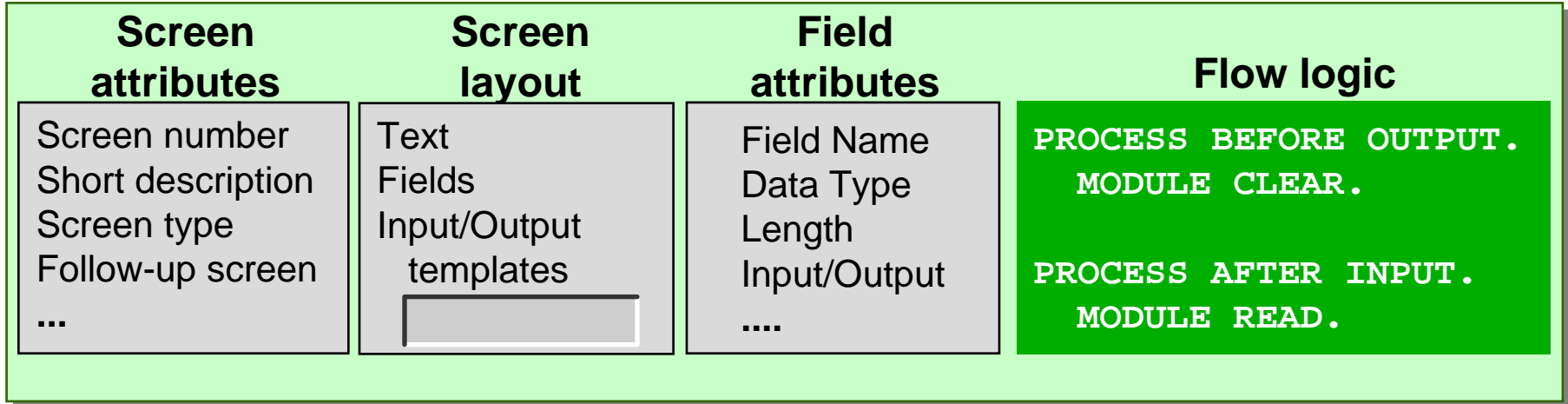
**PROCESS BEFORE OUTPUT (PBO)** is automatically triggered after the PAI processing of the previous screen and before the current screen is displayed. You can program the PBO processing of the screen in this block. At the end of the PBO processing, the screen is displayed.

**PROCESS AFTER INPUT (PAI)** is triggered when the user chooses a function on the screen. You can program the PAI processing of the screen in this block. At the end of the PAI processing, the system either calls the next screen or carries on processing at the point from which the screen was called.

**PROCESS ON HELP-REQUEST (POH)** and **PROCESS ON VALUE-REQUEST (POV)** are triggered when the user requests field help (F1) or possible values help (F4) respectively. You can program the appropriate coding in the corresponding event blocks. At the end of processing, the system carries on processing the current screen.

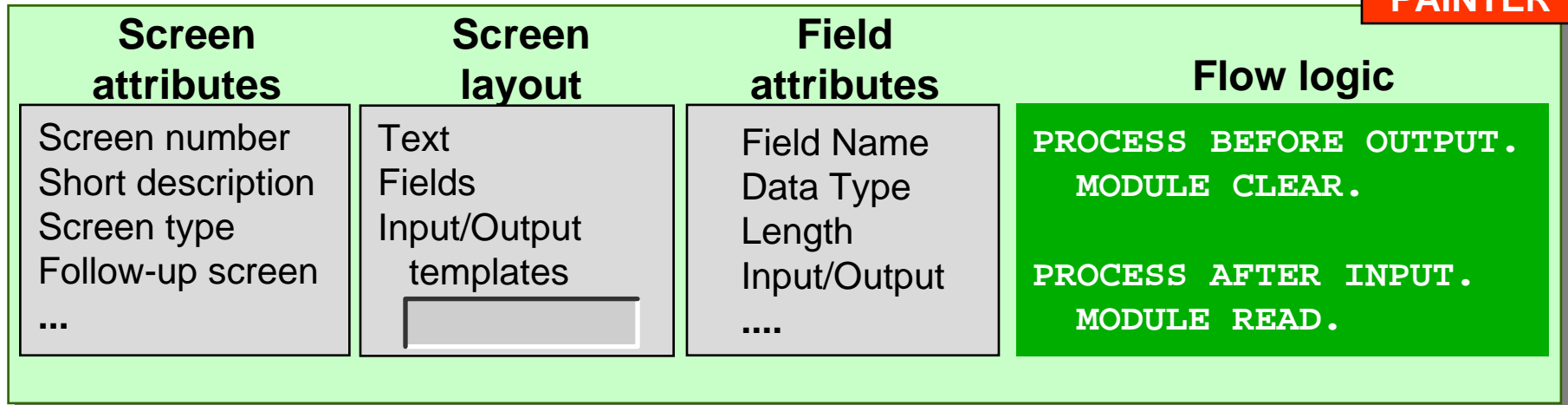
<b><u>Keyword</u></b>	<b><u>Function</u></b>
<b>MODULE</b>	<b>Calls a dialog module in an ABAP program</b>
<b>FIELD</b>	<b>Specifies the point at which the contents of a screen field should be transported</b>
<b>ON</b>	<b>Used in conjunction with FIELD</b>
<b>VALUES</b>	<b>Used in conjunction with FIELD</b>
<b>CHAIN</b>	<b>Starts a processing chain</b>
<b>ENDCHAIN</b>	<b>Ends a processing chain</b>
<b>CALL</b>	<b>Calls a subscreen</b>
<b>LOOP</b>	<b>Starts processing a screen table</b>
<b>ENDLOOP</b>	<b>Stops processing a screen table</b>



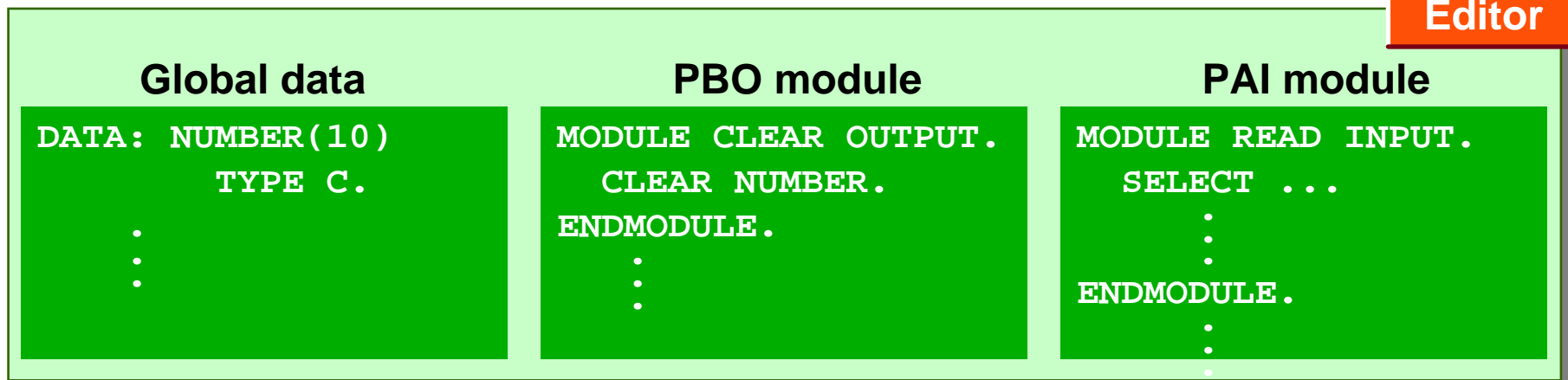


## Screen Painter To ABAP/4 Editor

**SCREEN  
PAINTER**



**ABAP/4  
Editor**



**To call a module, use the flow logic statement**

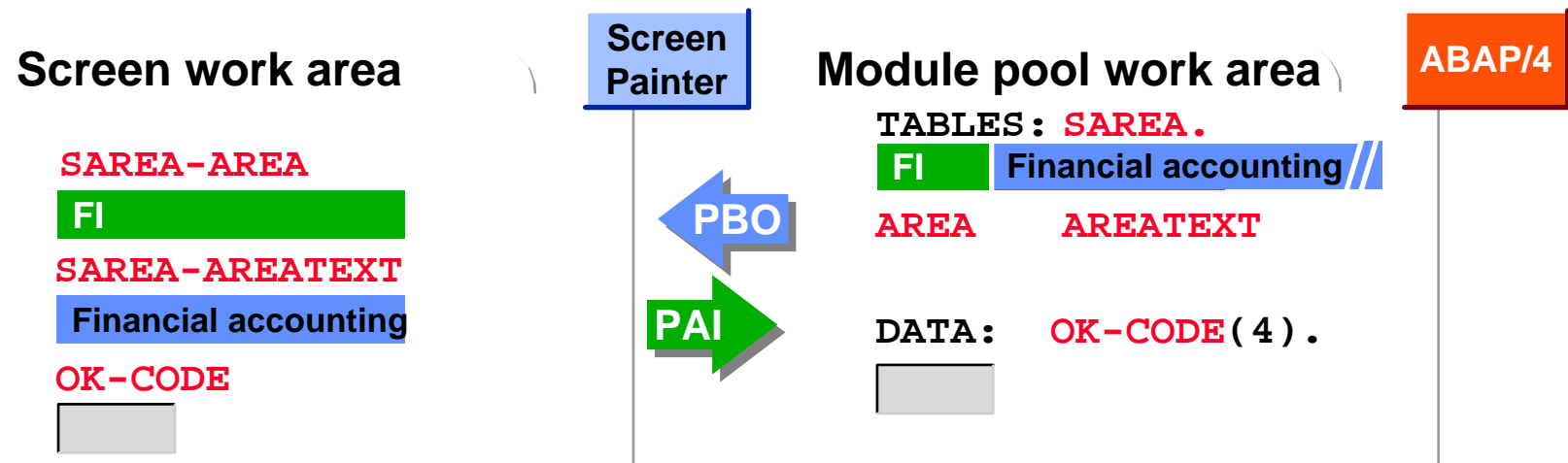
**MODULE <mod>.**

**The system starts the module <mod>, which must have been defined for the same event block in which the call occurs.**

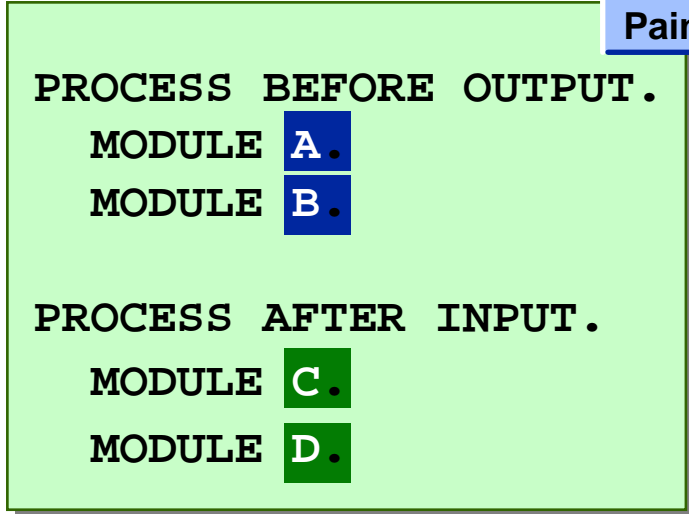
**If you only use simple modules in the screen flow logic, the data transport between the ABAP program and the screen is as follows:**

- **In the PAI event, all of the data from the screen is transported to the ABAP program (as long as there are program fields with the same names as the screen fields) after the automatic input checks and before the first PAI module is called. This includes the contents of the system fields (for example, SY-UCOMM, which contains the current function code).**
- **At the end of the last PBO module, and before the screen is displayed, all of the data is transported from the ABAP program to any identically-named fields in the screen.**

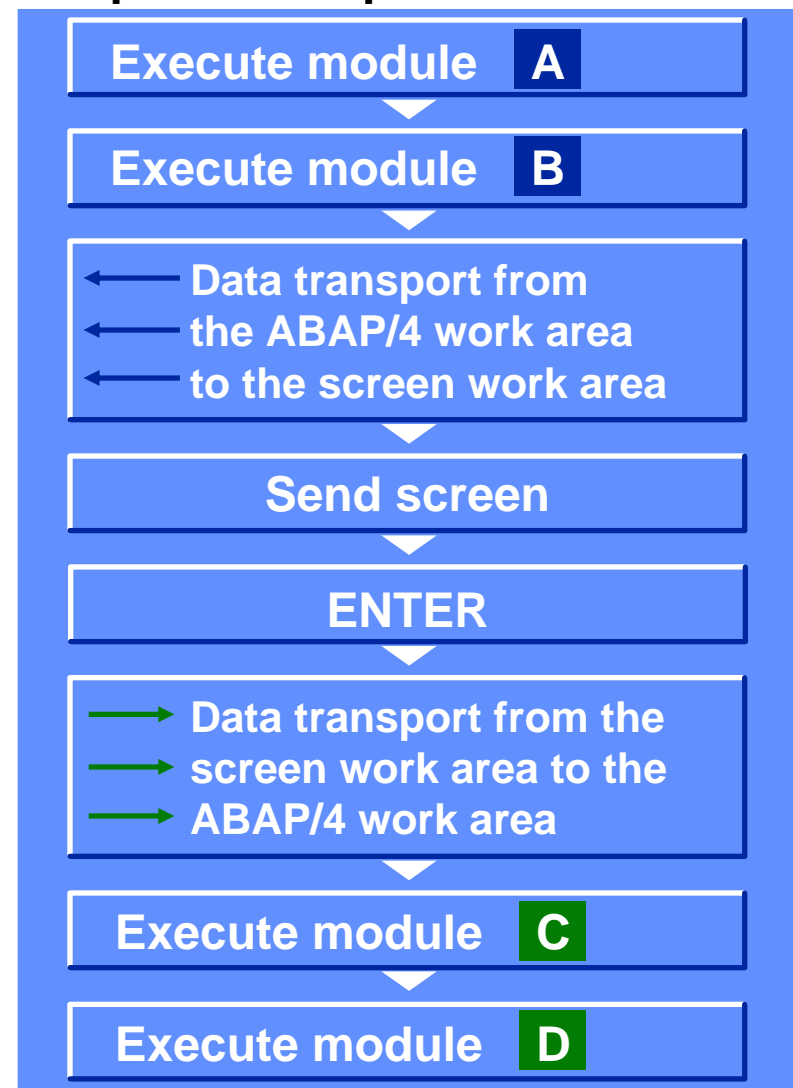
## Data Transfer within the Screen and Module Pool work Area .



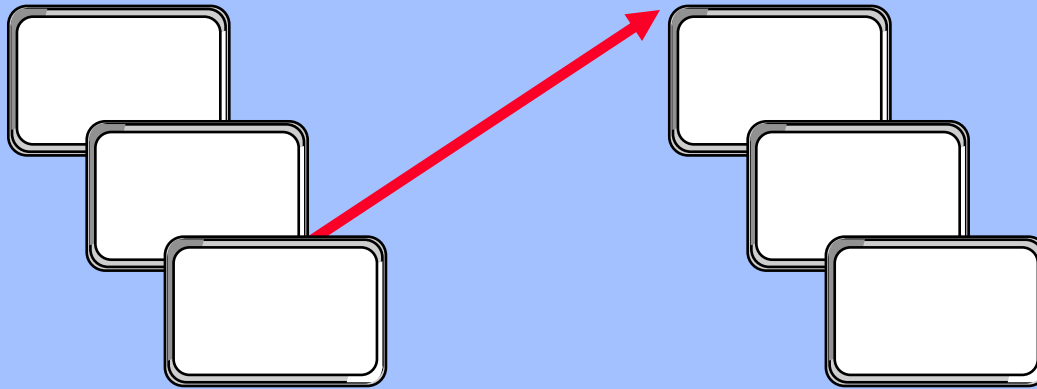
Screen Painter



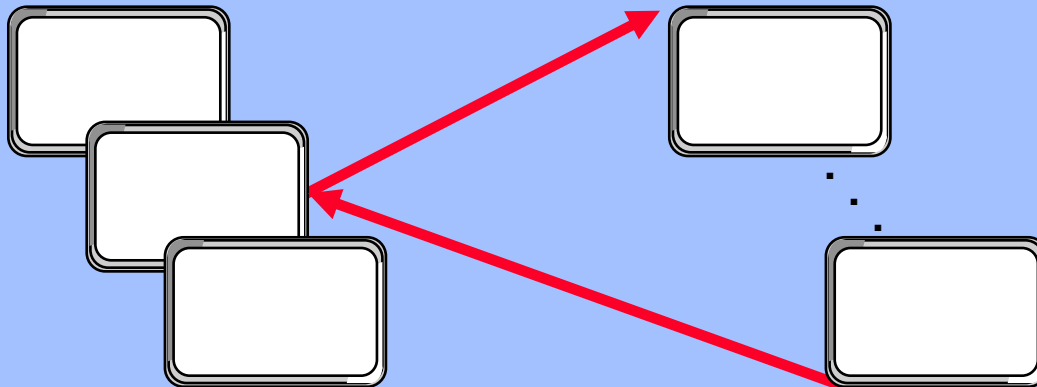
## Sequence of operations

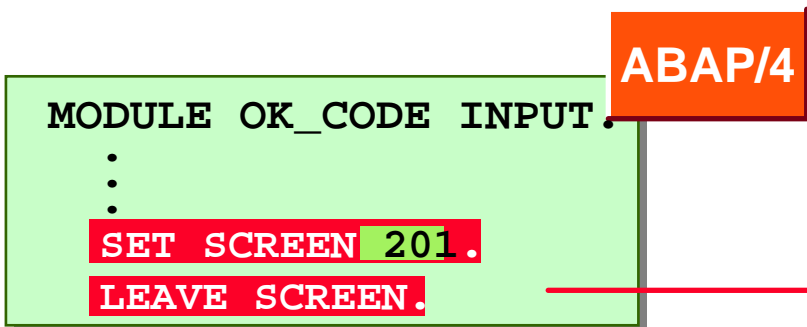
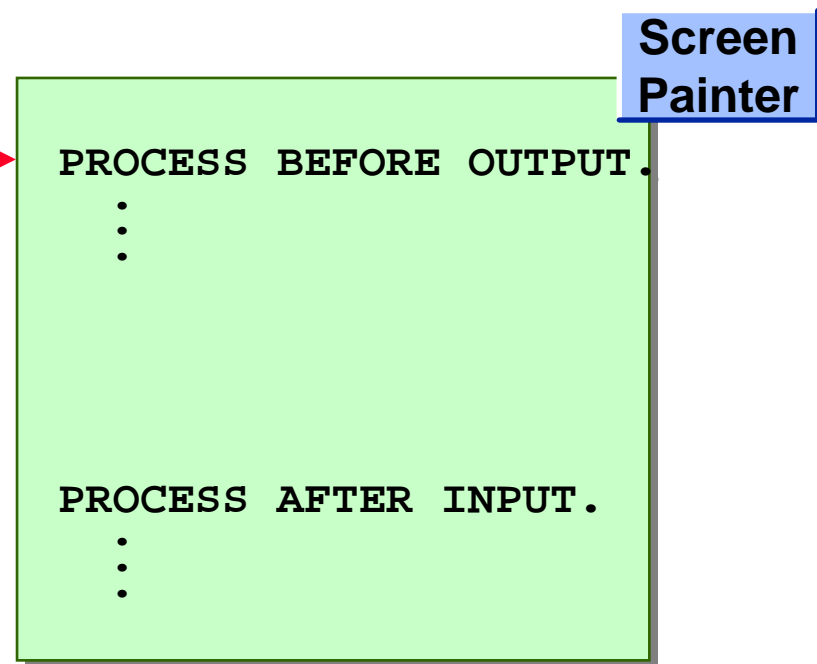
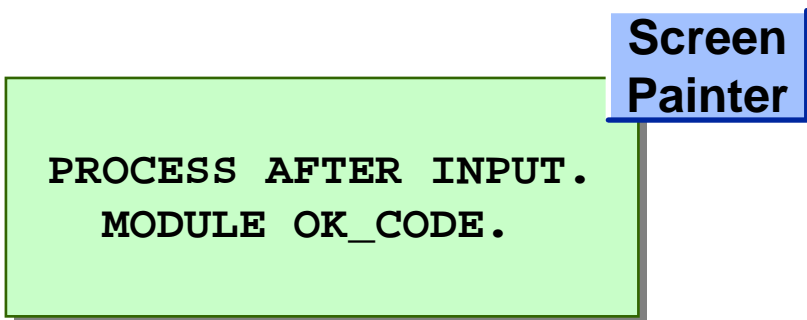
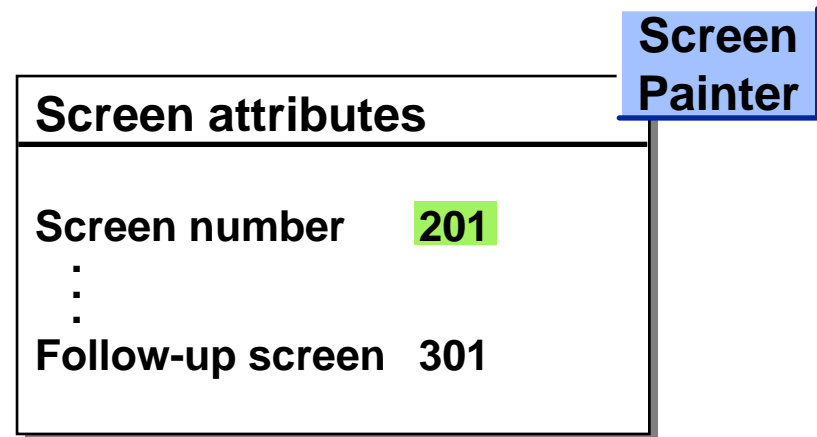
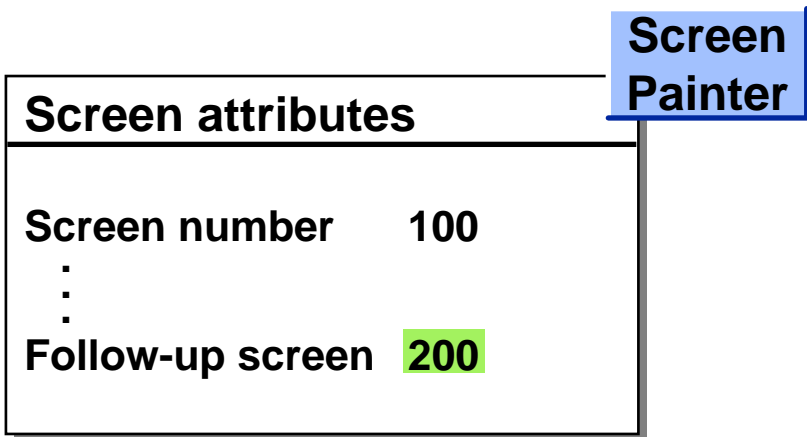


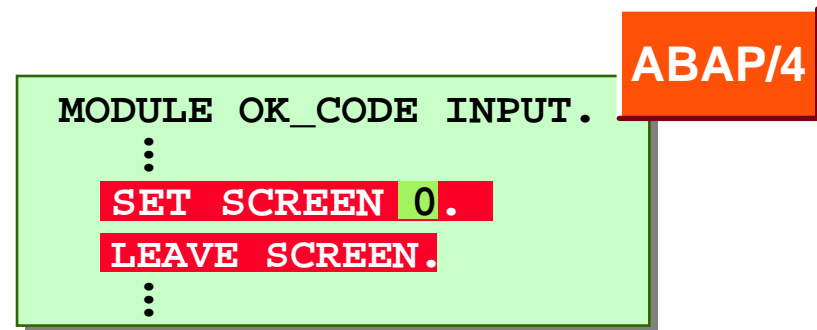
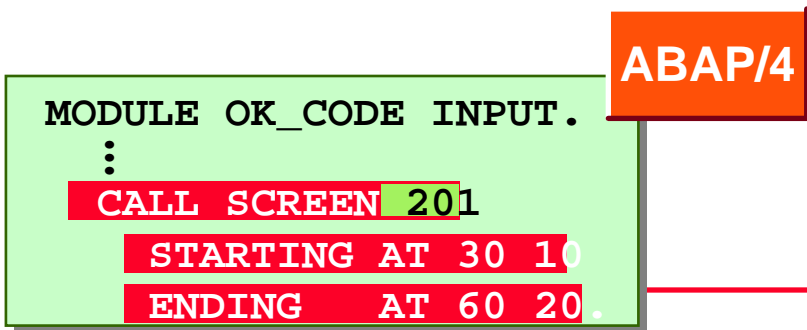
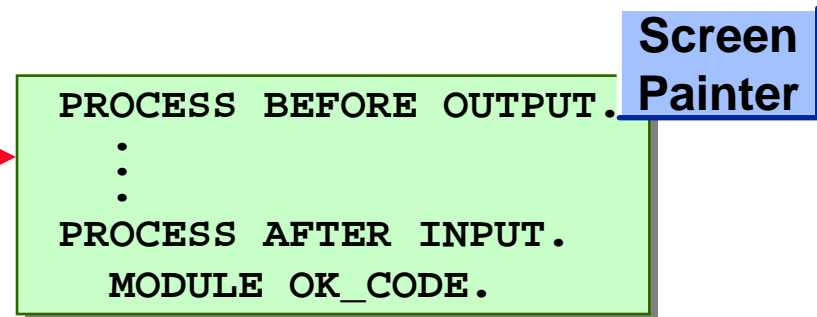
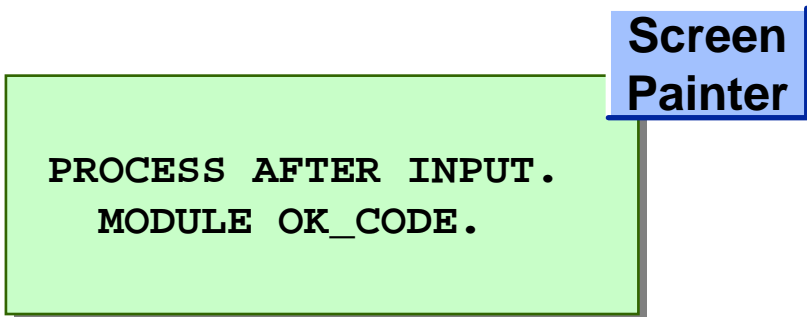
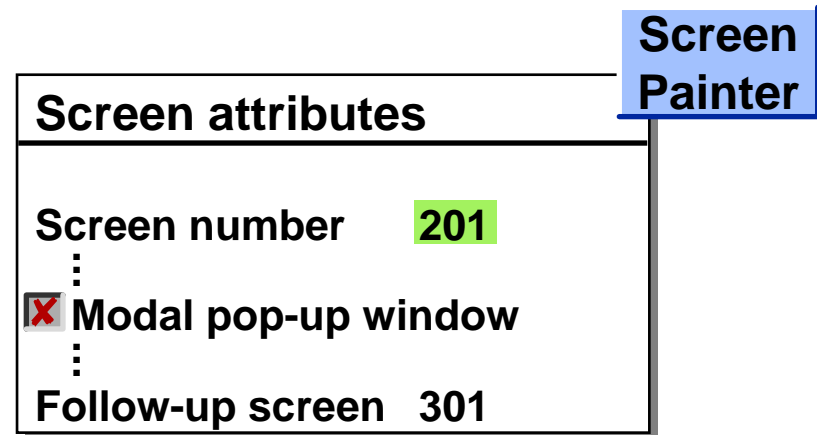
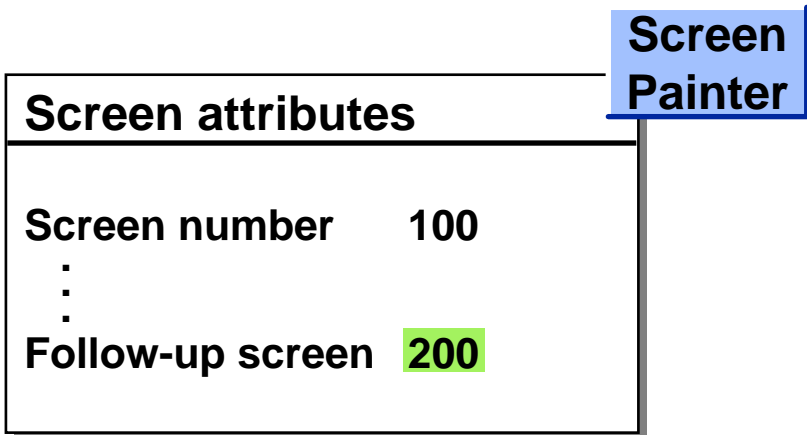
## SET SCREEN



## CALL SCREEN









## Syntax:

- LEAVE TO SCREEN <screen number>.  
(or)
- SET SCREEN <screen number>.  
LEAVE SCREEN.
  
- LEAVE TO SCREEN 0.
  - From called screen
  - From main screen

- SET TITLEBAR 'T01' WITH v1 v2 v3 v4

Example:

Module status\_100.

SET TITLEBAR 'T01' with EKKO-EBELN'.

Endmodule.

- SET PF-STATUS 'xxxxxxxx'.
- SET PF-STATUS 'xxxxxxxx' EXCLUDING  
<itab>.

Example:

Module Status\_100 on input

**Field format Check** :This format limits the kind of input that is valid. For ex. , a DATS field (Date field) is an 8 char string in YYYYMMDD format. All char must be numbers . For the given value entered, the system checks that the day value is valid.

**Required Check** : In the screen painter you can set a field's required Input Attribute . The system requires the user to enter the input before entering PAI Processing.

**Foreign Key Check**:The field can have a foreign key relationship with another table or its domain can specify a fixed value list for the field.The system checks the user input value can be found in the related check table or in the fixed-value lists.

Screen  
Painter

***Field list***

Field name	Format
DATE	DATE
⋮	
AMOUNT	DEC

Date

Amount

E: Invalid date

Date

Amount

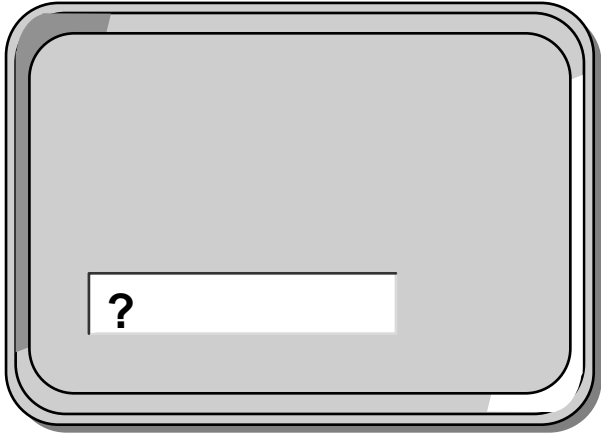
E: Please enter numeric value

Screen  
Painter

***Field list***

---

Field name	OBLIGATORY
TEST FIELD	X



**ABAP/4  
Dictionary**

Field name	Check table
⋮	
FIELD1	P1
⋮	

**Screen  
Painter**

<i>Field list</i>	
Field	Foreign key
FIELD1	X

Field1

Check

Check table	P1
KEY	
A	
B	
C	
⋮	

Use the FIELD..VALUES to check the field values in Screen Flow Logic.

FIELD <screen field> VALUES [<value list>]

```
PROCESS AFTER INPUT.
```

```
  FIELD <screen field>
```

```
    VALUES (<copy1>, <copy2>, ...).
```

Copy:

```
<value>
```

```
NOT <value>
```

```
BETWEEN      <value1> AND <value2>
```

```
NOT BETWEEN  <value1> AND <value2>
```

Screen  
Painter

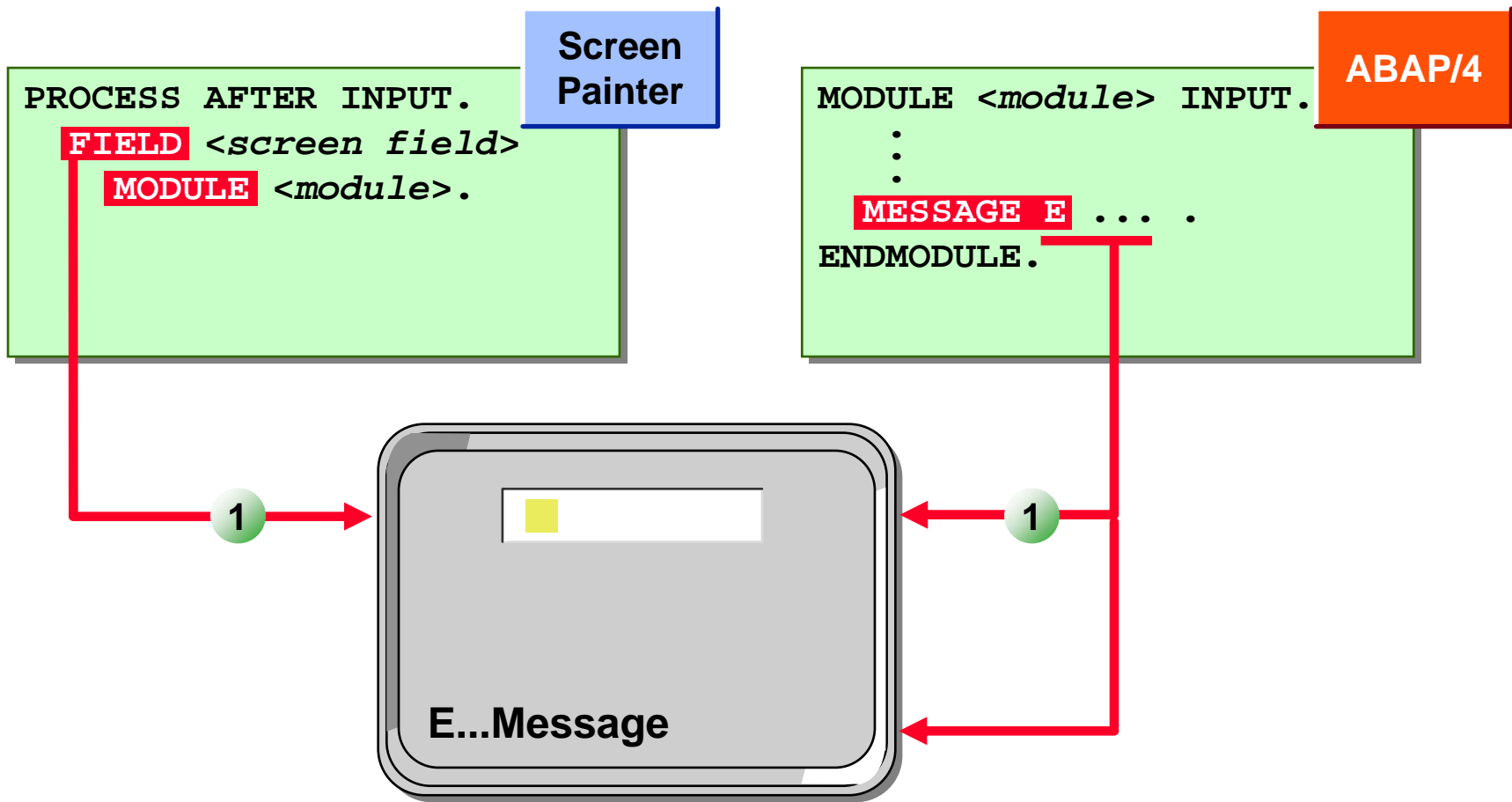
```
PROCESS AFTER INPUT.
```

```
  FIELD SCOUR-COURSE
```

```
    VALUES ('01', BETWEEN '20' AND '30', 'ABC').
```

Screen  
Painter

The `FIELD..MODULE` statement checks the validity for a particular screen field.





Screen  
Painter

```
PROCESS AFTER INPUT.  
  FIELD SCOUR-COURSE  
  MODULE CHECK_SCOUR.
```

ABAP/4

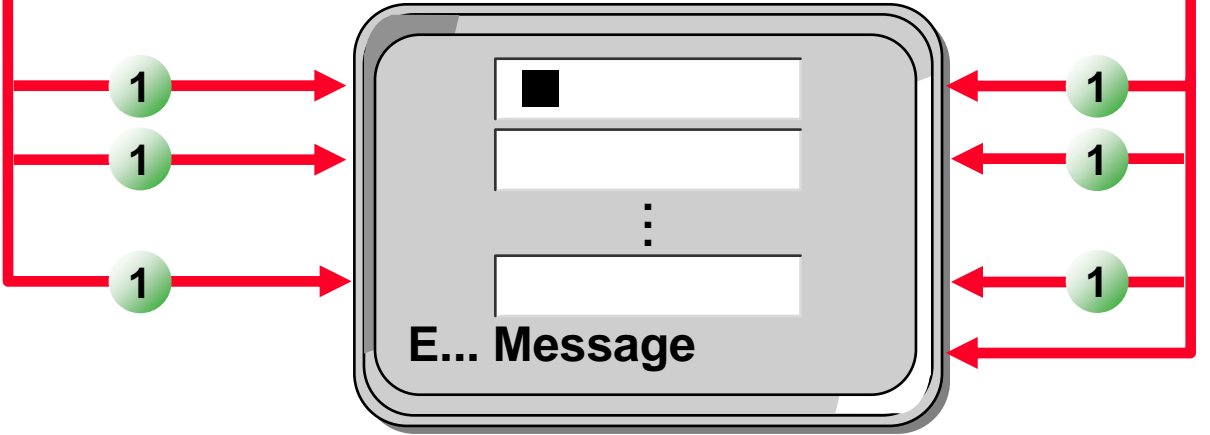
```
MODULE CHECK_SCOUR INPUT.  
  
  SELECT SINGLE * FROM SCOUR  
    WHERE AREA = SCOUR-AREA  
    AND COURSE = SCOUR-COURSE.  
  
  IF SY-SUBRC NE 0.  
    MESSAGE E123 WITH 'SCOUR'.  
  ENDIF.  
  
ENDMODULE.
```

```
PROCESS AFTER INPUT.
CHAIN.
  FIELD: <screen field1>,
         <screen field2>,
         ⋮
         <screen field n>.
  MODULE <module>.
ENDCHAIN.
```

Screen Painter

```
MODULE <module> INPUT.
  ⋮
  MESSAGE E ... .
ENDMODULE.
```

ABAP/4



1 ready for input

## Screen Painter

```
PROCESS AFTER INPUT.  
CHAIN.  
FIELD: SCOUR-AREA, SCOUR-COURSE.  
MODULE CHECK_FIELD.  
ENDCHAIN.
```

## ABAP/4

```
MODULE CHECK_SCOUR INPUT.  
  
SELECT SINGLE * FROM SCOUR  
WHERE AREA = SCOUR-AREA  
AND COURSE = SCOUR-COURSE.  
  
IF SY-SUBRC NE 0.  
MESSAGE E123 WITH 'SCOUR', 'COURSE'.  
ENDIF.  
  
ENDMODULE.
```

- **ON INPUT:**

If the field value is different from the initial value.

- **ON REQUEST:**

This module will be executed if a value has been entered in

the specific field since the screen was displayed.

- **AT EXIT-COMMAND:**

Screen  
Painter

```
PROCESS AFTER INPUT.  
  FIELD <screen field>  
    MODULE <module> ON INPUT.  
  :
```

Screen  
Painter

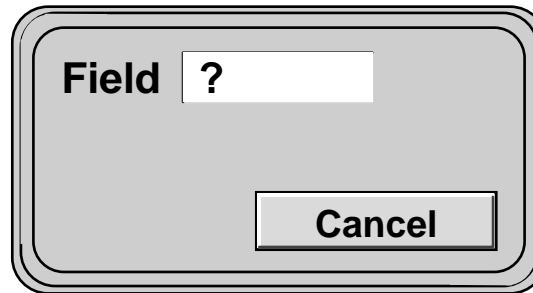
```
PROCESS AFTER INPUT.  
CHAIN.  
  FIELD: <screen field 1>,  
        <screen field 2>,  
        :  
        <screen field n>.  
  MODULE <module> ON CHAIN-INPUT.  
ENDCHAIN.  
  :
```

Screen  
Painter

```
PROCESS AFTER INPUT.  
  FIELD <screen field>  
    MODULE <module> ON REQUEST.  
  :
```

Screen  
Painter

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



Screen  
Painter

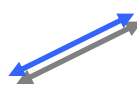
```
PROCESS AFTER INPUT.  
  MODULE X.  
  MODULE TERMINATE  
    AT EXIT-COMMAND.  
  :
```

ABAP/4

```
  :  
  MODULE TERMINATE INPUT.  
    SET SCREEN ... .  
    LEAVE SCREEN.  
  ENDMODULE.  
  :
```

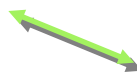
Field ?

Cancel



Function list	
Function	Type
ABBR	E

**Menu  
Painter**



List of modification groups		
Field name	Fcode	Type
ABEND	ABBR	E

**Screen  
Painter**



- Error (E) - Displays Error Message on the current screen
- Warning (W) - Displays Warning Message on the current screen
- Information (I) - Displays Popup Message on the current screen
- Abend (A) - The current Transaction will be Terminated
- Success (S) - Message is displayed on the Following Screen

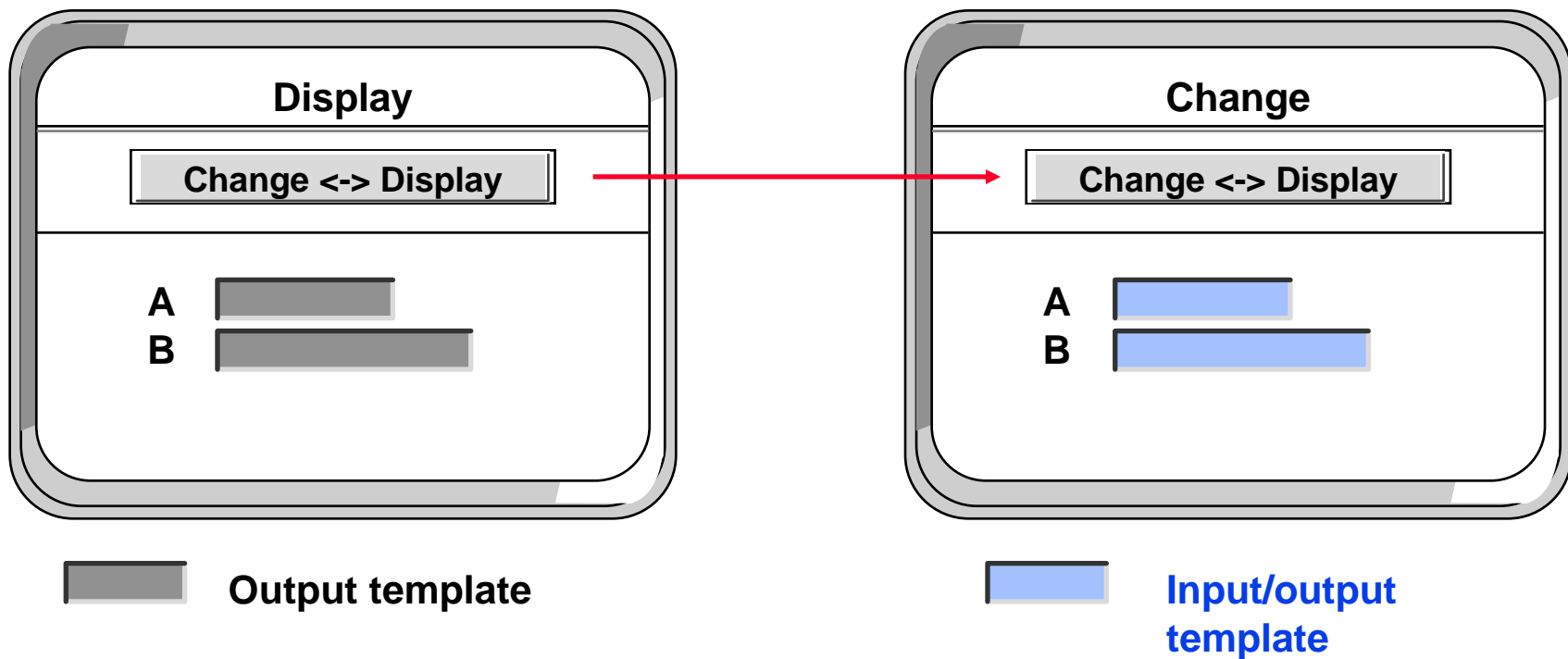
## Screen Painter

```
PROCESS AFTER INPUT.  
CHAIN.  
    FIELD: <screen field 1>,  
          <screen field 2>.  
    MODULE CHECK.  
ENDCHAIN.
```

## ABAP/4

```
PROGRAM B220MAIN MESSAGE-ID <id>.  
:  
:  
MODULE CHECK INPUT.  
:  
:  
IF SY-SUBRC ...  
    MESSAGE <qnnn>  
        WITH <value 1>..  
ENDIF.  
ENDMODULE.
```

At runtime , you may want to change the attributes depending on what user has requested in the previous screen.The attributes for each screen field are stored in the memory as SCREEN.You need not declare as table in your program.The system maintains it internally and updates during every screen change.



Field and its attrib. active	SCREEN-ACTIVE
Required entry field	SCREEN-REQUIRED
Input field	SCREEN-INPUT
Output field	SCREEN-OUTPUT
Highlighted	SCREEN-INTENSIFIED
Invisible	SCREEN-INVISIBLE
Shorter output length	SCREEN-LENGTH

Screen  
Painter

```
PROCESS BEFORE OUTPUT.  
  ⋮  
  MODULE MODIFY_SCREEN.  
  ⋮
```

ABAP/4

```
MODULE MODIFY_SCREEN OUTPUT.  
  ⋮  
  LOOP AT SCREEN.  
    IF SCREEN-GROUP1 = 'GR1'.  
      SCREEN-INPUT = 1.  
    ENDIF.  
    IF SCREEN-NAME = 'TAB-FELD'.  
      SCREEN-ACTIVE = 0.  
    ENDIF.  
    MODIFY SCREEN.  
  ENDLOOP.
```

**A screen table is a repeated series of table rows in a screen. Each entry contains one or more fields, and all rows have the same field structure.**

- **Table controls and step loops are types of screen tables you can add to a screen in the Screen Painter.**
- **These are the two mechanisms offered by ABAP/4 for displaying and using table data in a screen.**

**With table controls, the user can:**

- **Scroll through the table vertically and horizontally**
- **Re-size the width of a column**
- **Select table rows or columns**
- **Re-order the sequence of columns**

- **The feature of step loops is that their table rows can span more than one line on the screen. By contrast, the rows in a table control are always single lines, but can be very long.**



**You process a screen table by looping through it as you would through the rows of an internal table. To do this, you place a LOOP...ENDLOOP dynpro statement in the screen's flow logic.**

### **What the LOOP Statement Does?**

**The LOOP statement is responsible for getting screen table values passed back and forth between the screen and the ABAP/4 program. As a result, you must code a LOOP statement in both the PBO and PAI events for every table in your screen.**

### **Note :**

**Atleast, an empty LOOP...ENDLOOP must be there in PAI.**

**At PBO.**

**Loop at <itab> with control <name> cursor <name>-top\_line.**

**Module <mod\_name>.**

**Endloop.**

**At PAI.**

**Loop at <itab>.**

**Endloop.**

**Declaration for table control.**

**controls <name> type tableview using screen <no>.**

**At PBO.**

**Loop at <itab> cursor <var>.**  
        **Module <mod\_name>.**  
**Endloop.**

**At PAI.**

**Loop at <itab>.**  
  
**Endloop.**

**To branch to another transaction and end the current one, use the LEAVE TO TRANSACTION statement:**

## **Syntax**

**LEAVE TO TRANSACTION '<TRAN>'.**

**Once the new transaction starts, the user can not return to the previous transaction by pressing the Exit icon. Any data the user did not save in the previous transaction is lost.**

**If you want the user to be able to return to the initial transaction after processing an interim transaction, use the ABAP/4 statement:**

**Syntax**

**CALL TRANSACTION '<TRAN>'.**

**When you call a transaction, you can tell the system to suppress the transaction's initial screen and proceed directly to the next screen in the sequence:**

**syntax:**

**CALL TRANSACTION '<TRAN>' AND SKIP FIRST SCREEN.**

**The initial screen is processed but not displayed. while suppressing the first screen, for all required fields in the initial screen, your program must pass in data values when calling the transaction.**

**You can pass data to a called program using SPA/GPA parameters. SPA/GPA parameters are field values saved globally in memory. Each parameter is identified by a three-character code.**

**There are two ways to use SPA/GPA parameters:**

- **by setting field attributes in the Screen Painter**
- **by using the SET PARAMETER or GET PARAMETER statements**

### **Syntax**

**SET PARAMETER ID 'RID' FIELD <FIELD NAME1>.  
GET PARAMETER ID 'RID' FIELD <FIELD NAME1>.**

## **.SUBMIT**

Use the **SUBMIT** statement to start a separate report directly from the transaction.

**SUBMIT <prog>.**

**SUBMIT <prog> AND RETURN.**

**SUBMIT <prog> VIA SELECTION-SCREEN**

**SUBMIT <prog> WITH <prog sele var> = <fld>**

**SUBMIT <prog> WITH <para> IN <seltab>**

Produce the list from your module pool using

**LEAVE TO LIST-PROCESSING**

**LEAVE TO LIST-PROCESSING AND RETURN TO**

**SCREEN <screen number>.**

**LEAVE LIST-PROCESSING**



**You can code list-mode logic in PBO or PAI for the current screen.**

- **To display the list output in addition to the current screen:**

**Place the LEAVE TO LIST-PROCESSING logic at the end of PAI. On return from the list display, the system repeats processing for the current screen, starting with the beginning of PBO.**

- **To display the list output instead of the current screen:**

**Code the LEAVE TO LIST-PROCESSING logic in the PBO, and follow it with LEAVE SCREEN. This tells the system to display the list without displaying the current screen. PAI processing for the current screen is not executed.**

**Your program runs in list-mode until one of the following occurs**

**The system reaches a LEAVE LIST-PROCESSING statement in your code. The LEAVE LIST-PROCESSING statement returns control to the dialog screen. On return, the system re-starts processing at the beginning of PBO.**

**The user requests BACK or CANCEL from the basic-list level of the report.**

**If the user, exits the list using the BACK or CANCEL icons, you do not need to program an explicit LEAVE LIST-PROCESSING. When the user presses one of these, the system returns to the screen containing the LEAVE TO LIST-PROCESSING and re-starts PBO processing screen.**

**When returning to dialog-mode, your program can also re-route the user to a screen different from the one that started the list. To do this, use the keywords AND RETURN TO SCREEN when you first branch to list-mode:**

**syntax**

**LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 100.**

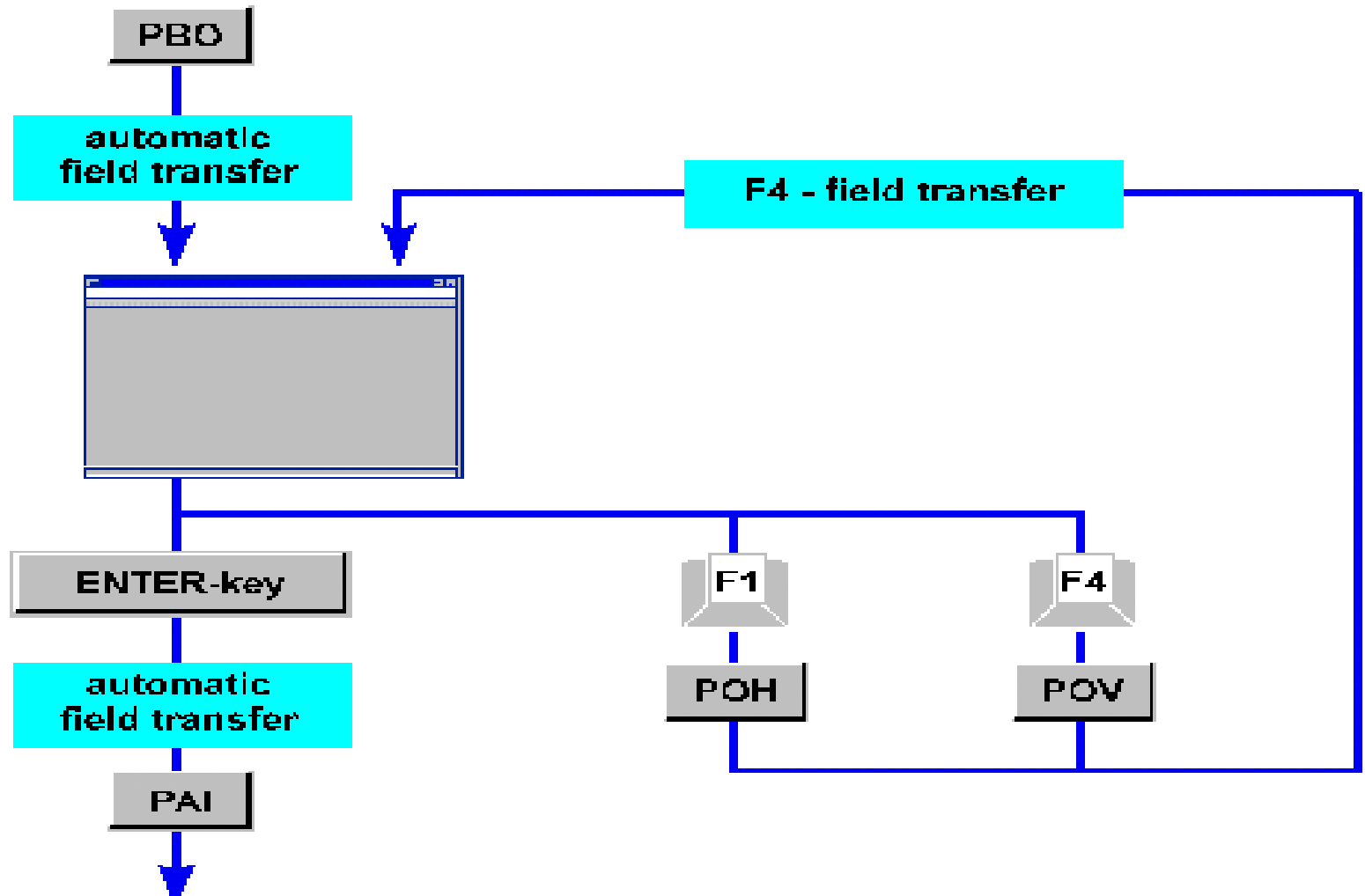
**You can program help texts and possible values lists using the PROCESS ON HELP-REQUEST (POH) and PROCESS ON VALUE-REQUEST (POV) events.**

## **Syntax**

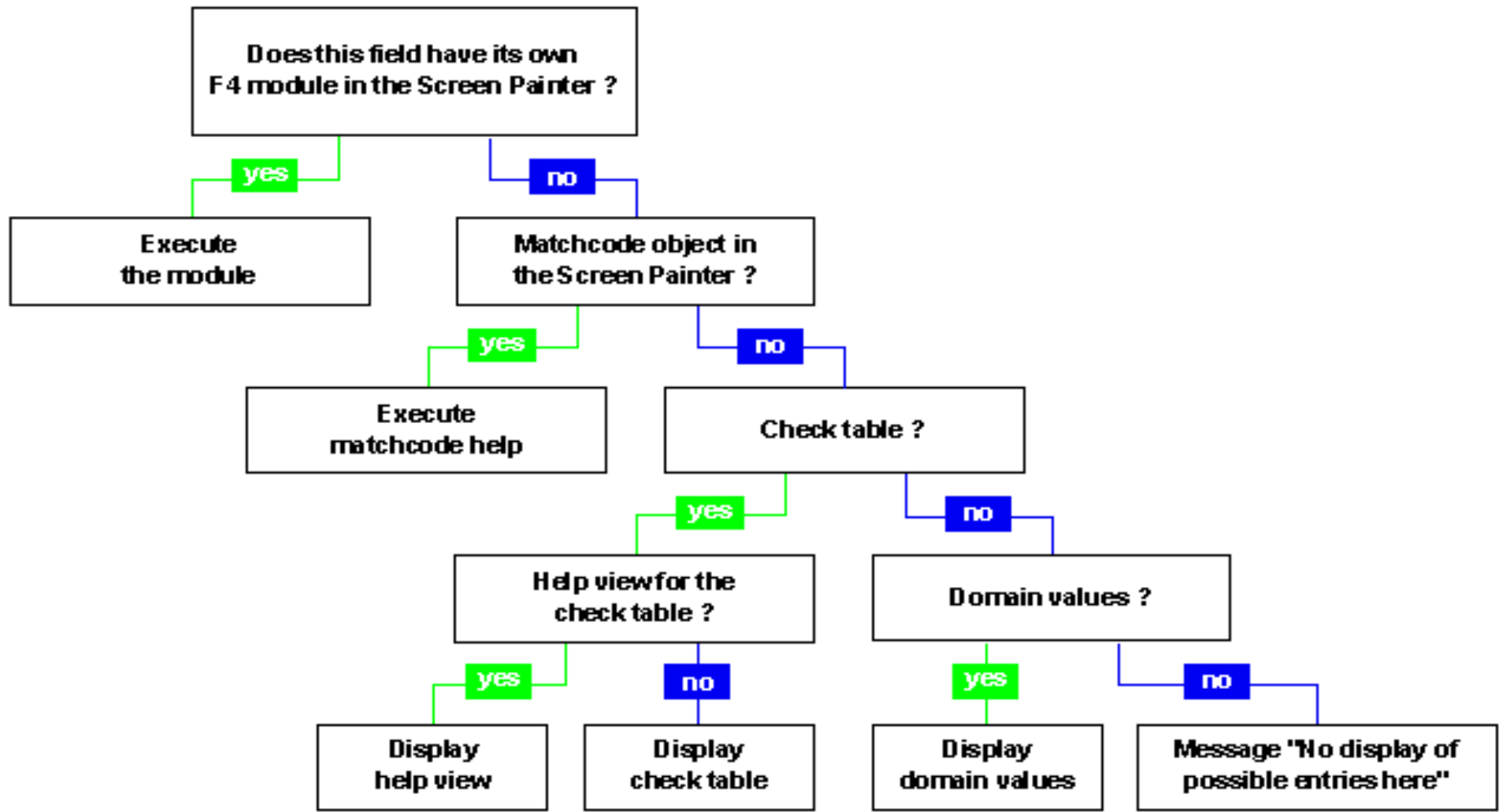
**PROCESS ON HELP-REQUEST.  
FIELD <field> MODULE <module>.**

**PROCESS ON VALUE-REQUEST.  
FIELD <field> MODULE <module> .**

# Programming Field- and Value-Help



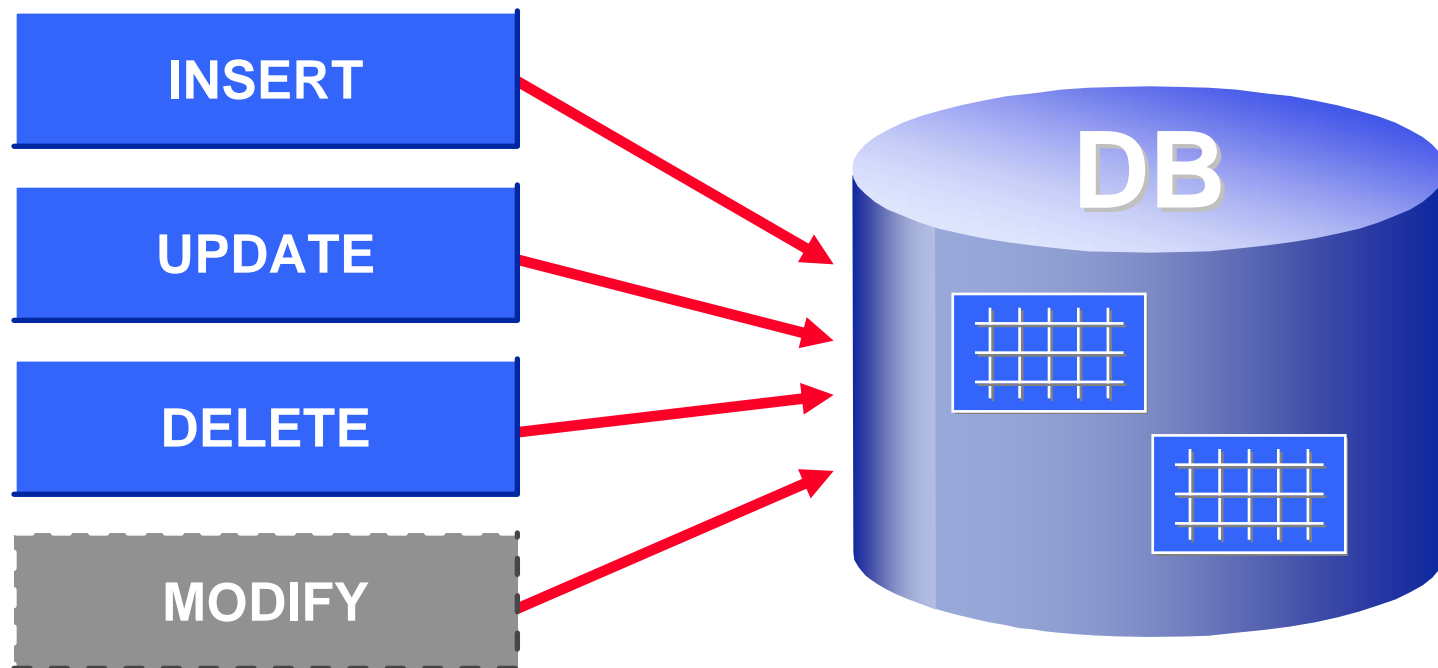
- **Matchcode help**
- **Check tables**
- **Help views**
- **Domain values**



**The ABAP/4 development environment provides a number of ways of designing a context-sensitive F1 help:**

- **Data element documentation**
- **Using the PROCESS ON HELP-REQUEST event.**





**Insert a new record into a database**

**Syntax:**

**INSERT <table>[<workarea>]**

**Eg:**

**MOVE 'BC' TO SPLAN-AREA.  
MOVE 'BC200' TO SPLAN-COURSE  
MOVE ....  
INSERT SPLAN.**

**MOVE 'BC' TO REC-AREA.  
MOVE 'BC200' TO REC-COURSE  
MOVE ....  
INSERT INTO SPLAN VALUES REC.**

**UPDATE Changes a record in the database**

**syntax: UPDATE <table>.**

**UPDATE <TABLE> SET <F1> = <V1>...<FN> = <VN>  
WHERE <f1> = <x1> .**

**Eg:**

**SELECT SINGLE \* FROM SPLAN WHERE AREA = 'BC'  
AND COURSE = 'BC200'  
AND WEEK = '23'.**

**SPLAN-TID1 = '007'.  
UPDATE SPLAN.**

**UPDATE SPLAN  
SET TID1 = '007'  
TID2 = '003'  
WHERE AREA = 'BC' AND COURSE = 'BC200'  
AND WEEK = '23'.**

**Delete record from the database.**

**Syntax: DELETE <table> .**

**DELETE FROM <table> WHERE <F1> = <V1>..**

**Eg:**

**MOVE 'BC' TO SPLAN-AREA.**

**MOVE 'BC200' TO SPLAN-COURSE**

**MOVE ....**

**DELETE SPLAN.**

**DELETE FROM SPLAN WHERE AREA = 'BC' AND COURSE = 'BC200'  
AND WEEK = '23'.**

**ARRAY operations improve the performance of the database updates**

**Syntax:**

**INSERT <table< FROM <itab>.  
UPDATE <table< FROM <itab>.  
DELETE <table< FROM <itab>.**

**Eg:**

**Data: begin of itab occurs 10.  
Include structure splan.**

**Data: end of itab.**

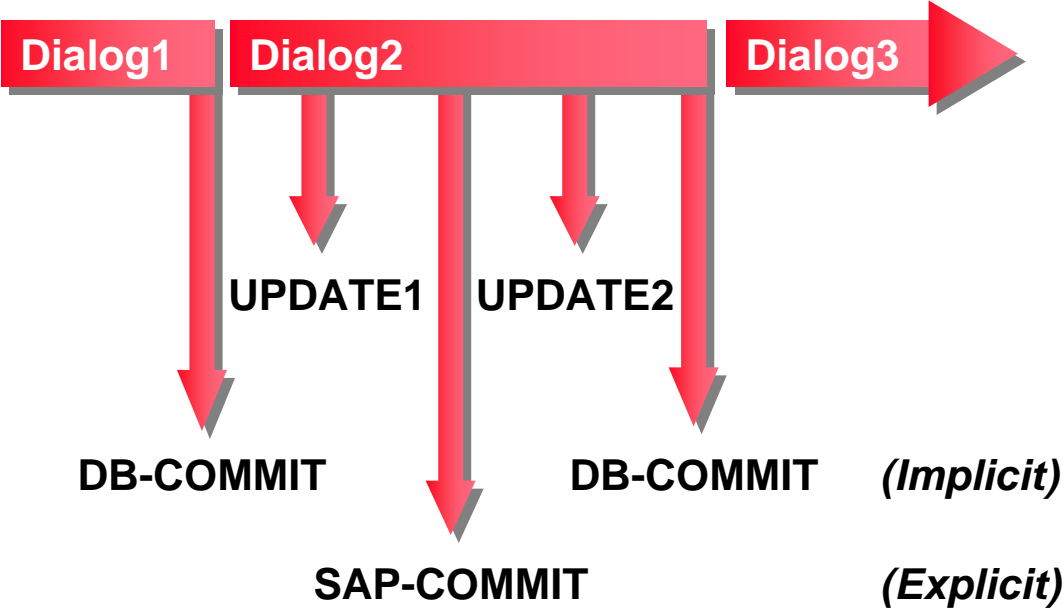
**Move 'BC' to itab-area.  
Append itab.**

**.....**

**INSERT SPLAN FROM ITAB.  
UPDATE SPLAN FROM ITAB.  
DELETE SPLAN FROM ITAB.**

**ABAP/4**

```
⋮  
MODULE UPDATE INPUT.  
⋮  
UPDATE <table1>.  
IF SY-SUBRC EQ 0.  
  COMMIT WORK.  
ELSE.  
  ROLLBACK WORK.  
  MESSAGE E ... .  
ENDIF.  
UPDATE <table2>.  
⋮  
ENDMODULE.  
⋮
```



**To program database updates effectively, programmers are mainly concerned with:**

- **maintaining database correctness.**
- **optimizing response times for users.**

- **If the transaction runs successfully, all changes should be carried out.**
- **If the transaction encounters an error, no changes should be carried out, not even partially.**

**In the database world, an "all-or-nothing" transaction is called an LUW (Logical Unit of Work). There are two types of LUW's.**

- **Database LUW**
- **SAP LUW**



**With Update Bundling you can execute updates at the end of the update transaction, rather than at every screen change.**

**You can avoid your updates being committed at each screen change.**

**You can lock the objects to be updated across multiple screens.**

**With update bundling, you package your updates in special routines that run only when your program issues a ABAP/4 commit/rollback. To do this, you use:**

- PERFORM ON COMMIT**
- CALL FUNCTION IN UPDATE TASK**
- CALL FUNCTION IN BACKGROUND TASK**

**These statements specify that a given FORM routine or function module be executed not immediately, but rather at the next ABAP/4 commit/rollback.**

**The PERFORM ON COMMIT statement calls a form routine in the dialog task, but delays its execution until the system encounters the next COMMIT WORK statement.**

**Updating in the update task:**

**The CALL FUNCTION IN UPDATE TASK statement logs a function module for execution in the update task. The subsequent COMMIT WORK statement triggers actual execution.**

**The CALL FUNCTION IN BACKGROUND TASK statement logs a function module to run in a background task. Normally, this statement is used to execute functions on remote hosts (by specifying an additional DESTINATION parameter).**

**Background-task functions are processed as low-priority requests, but all requests for the same destination run in a common update transaction.**

**In this Exercise we see how to write a simple Transaction**  
**First create a program with naming convention SAPMZ<initials>.**  
**Eg. SAPMZ\_EMPDET**

The screenshot shows the 'ABAP: Program Attributes SAPMZ\_EMPDET Change' dialog box. The title bar indicates the program name is SAPMZ\_EMPDET. The main area contains the following fields and values:

Title	Program SAPMZ_EMPDET	
Original language	EN	English
Created	20.12.2001	DEVELOP
Last changed by	21.12.2001	DEVELOP
Status	Active	

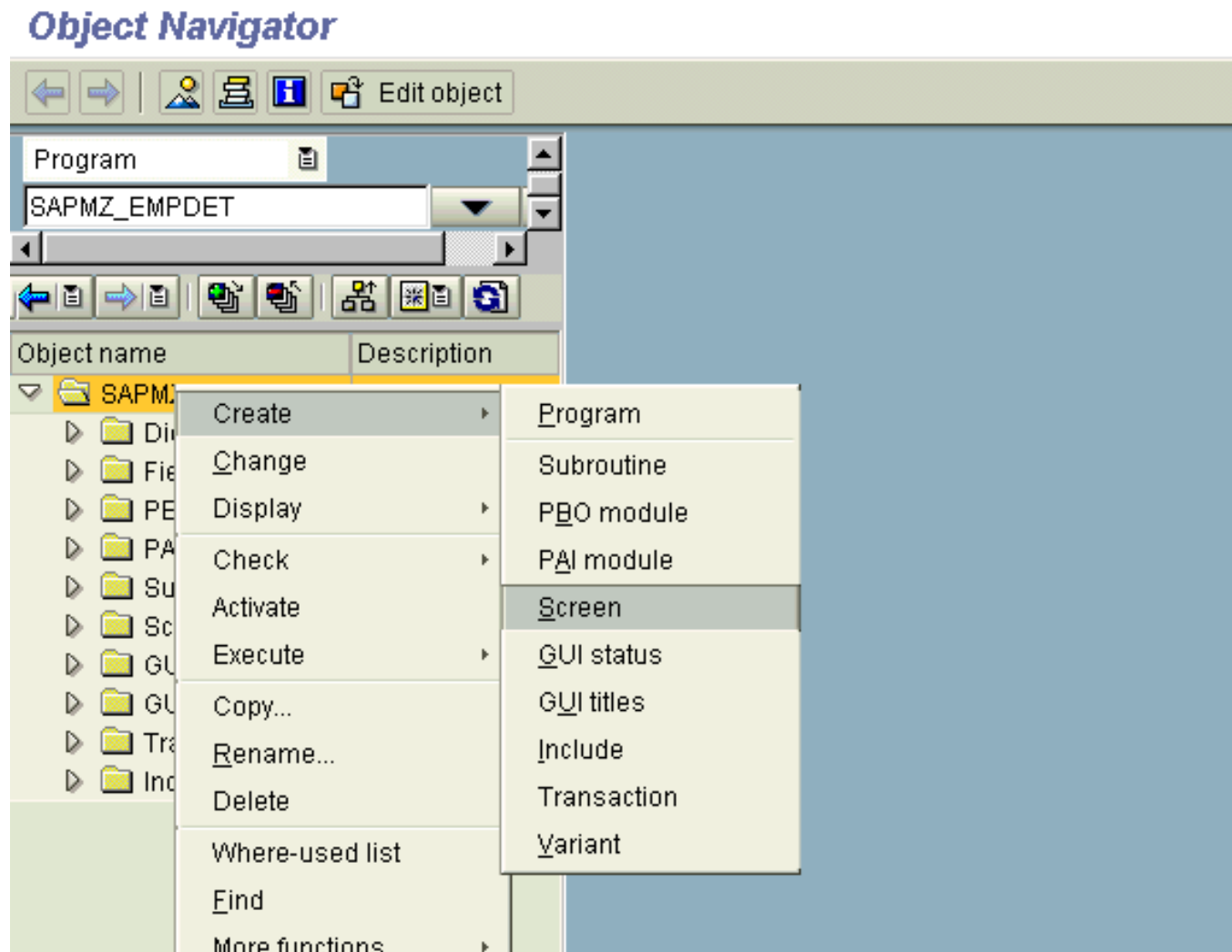
The 'Attributes' section is expanded, showing the following details:

Type	Module pool
Status	Test program
Application	
Authorization groups	
Development class	\$TMP Temporary Objects (never transported!)

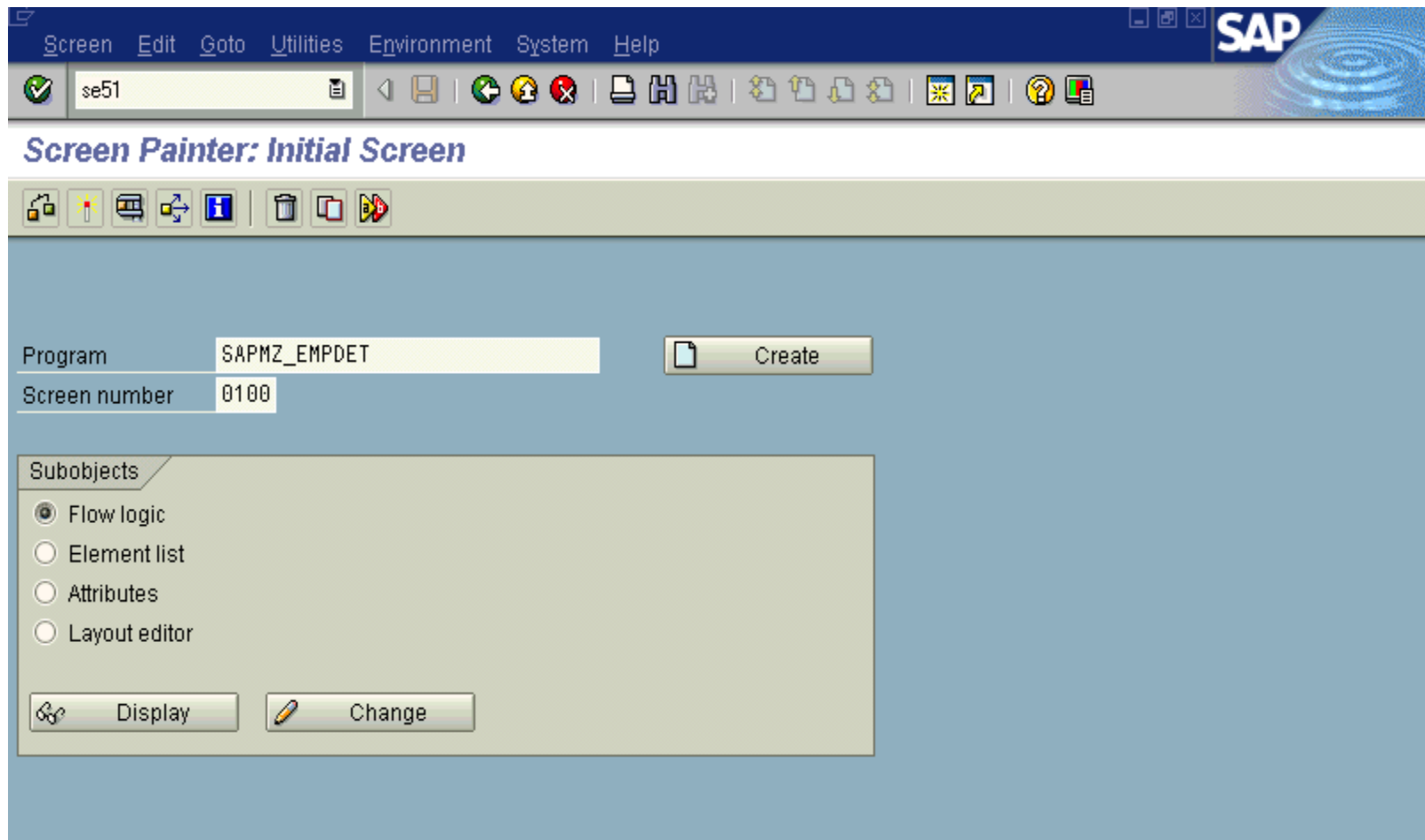
At the bottom of the attributes section, there are two checkboxes: 'Editor lock' (unchecked) and 'Fixed point arithmetic' (checked).

The bottom toolbar contains icons for Save, Edit, Find, Print, and Close.

## Now create a screen using Transaction SE80

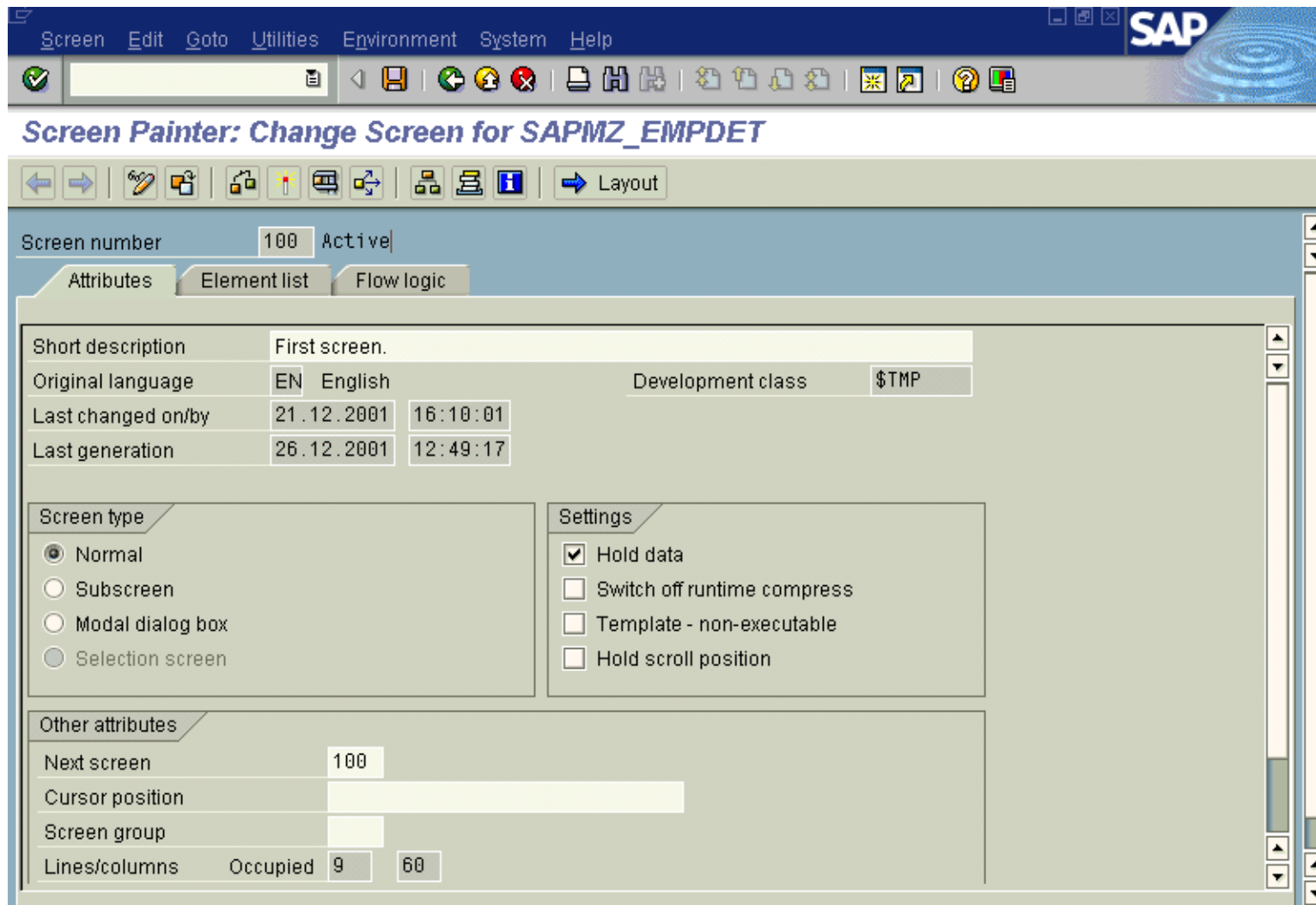


**We can also create a screen using Transaction SE51**



**Click the create button**

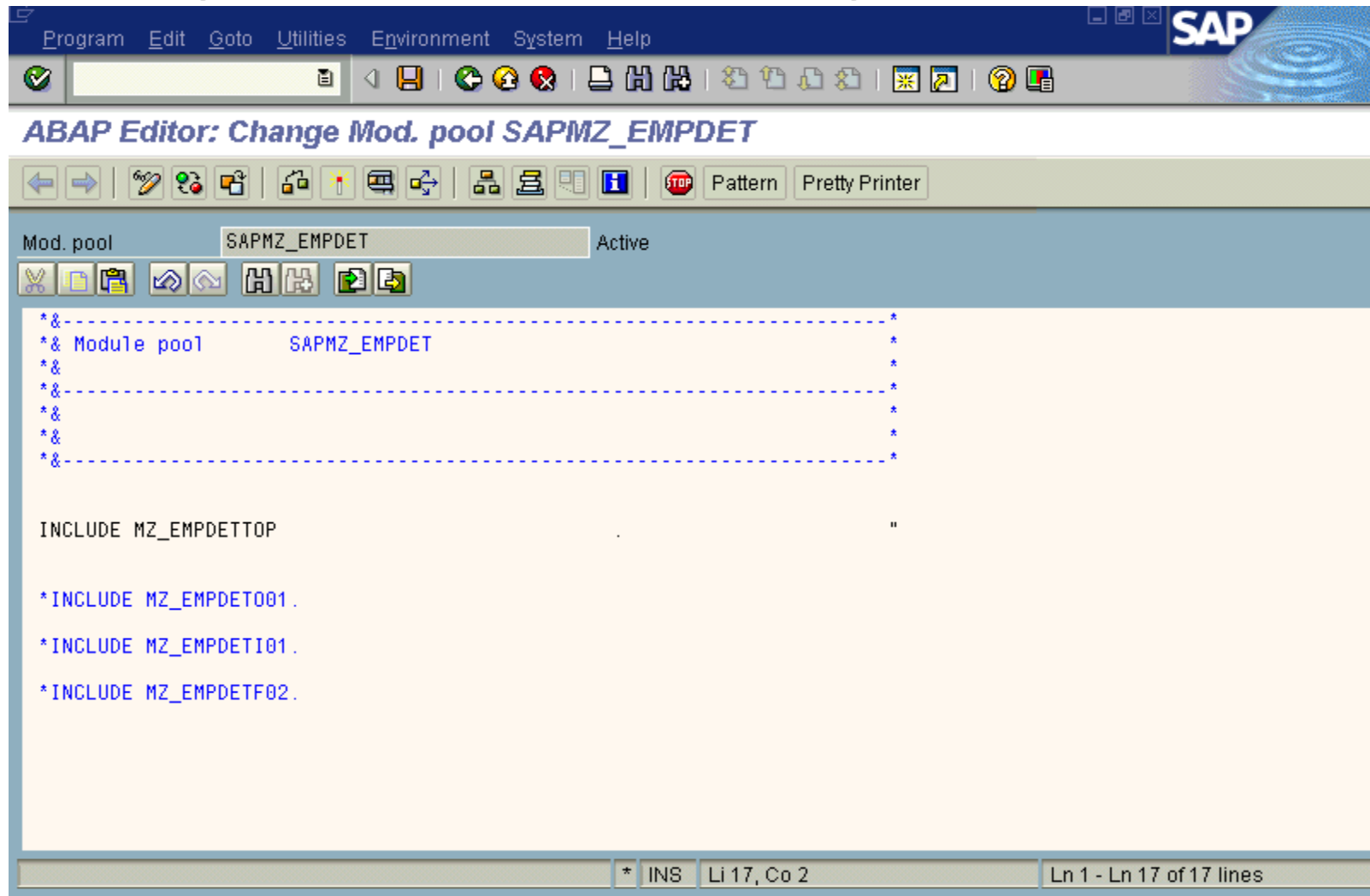
**Enter a meaningful description for the screen and select the screen type**



**Click on the save button to save the entries**

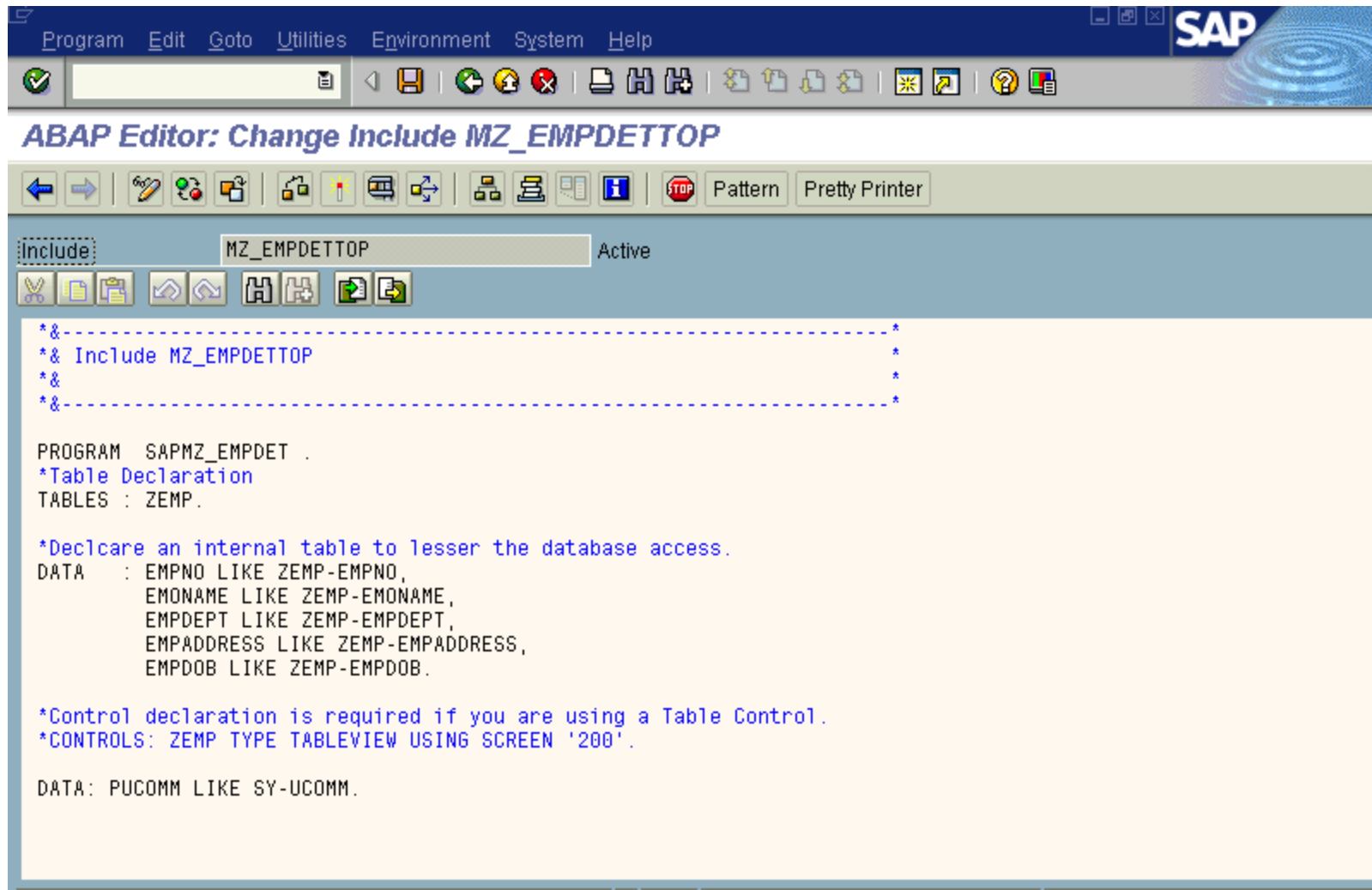


Select the program name and click the change icon.



Double click on the Include MZ<initial>TOP

## Declare the Global variables in this include section



The screenshot shows the SAP ABAP Editor interface. The title bar reads "ABAP Editor: Change Include MZ\_EMPDETTOP". The menu bar includes "Program", "Edit", "Goto", "Utilities", "Environment", "System", and "Help". The toolbar contains various icons for editing and navigation. The main editor area shows the following code:

```
*&-----*  
*& Include MZ_EMPDETTOP  
*&  
*&-----*  
  
PROGRAM SAPMZ_EMPDET .  
*Table Declaration  
TABLES : ZEMP.  
  
*Declare an internal table to lesser the database access.  
DATA : EMPNO LIKE ZEMP-EMPNO,  
        EMONAME LIKE ZEMP-EMONAME,  
        EMPDEPT LIKE ZEMP-EMPDEPT,  
        EMPADDRESS LIKE ZEMP-EMPADDRESS,  
        EMPDOB LIKE ZEMP-EMPDOB.  
  
*Control declaration is required if you are using a Table Control.  
*CONTROLS: ZEMP TYPE TABLEVIEW USING SCREEN '200'.  
  
DATA: PUCOMM LIKE SY-UCOMM.
```

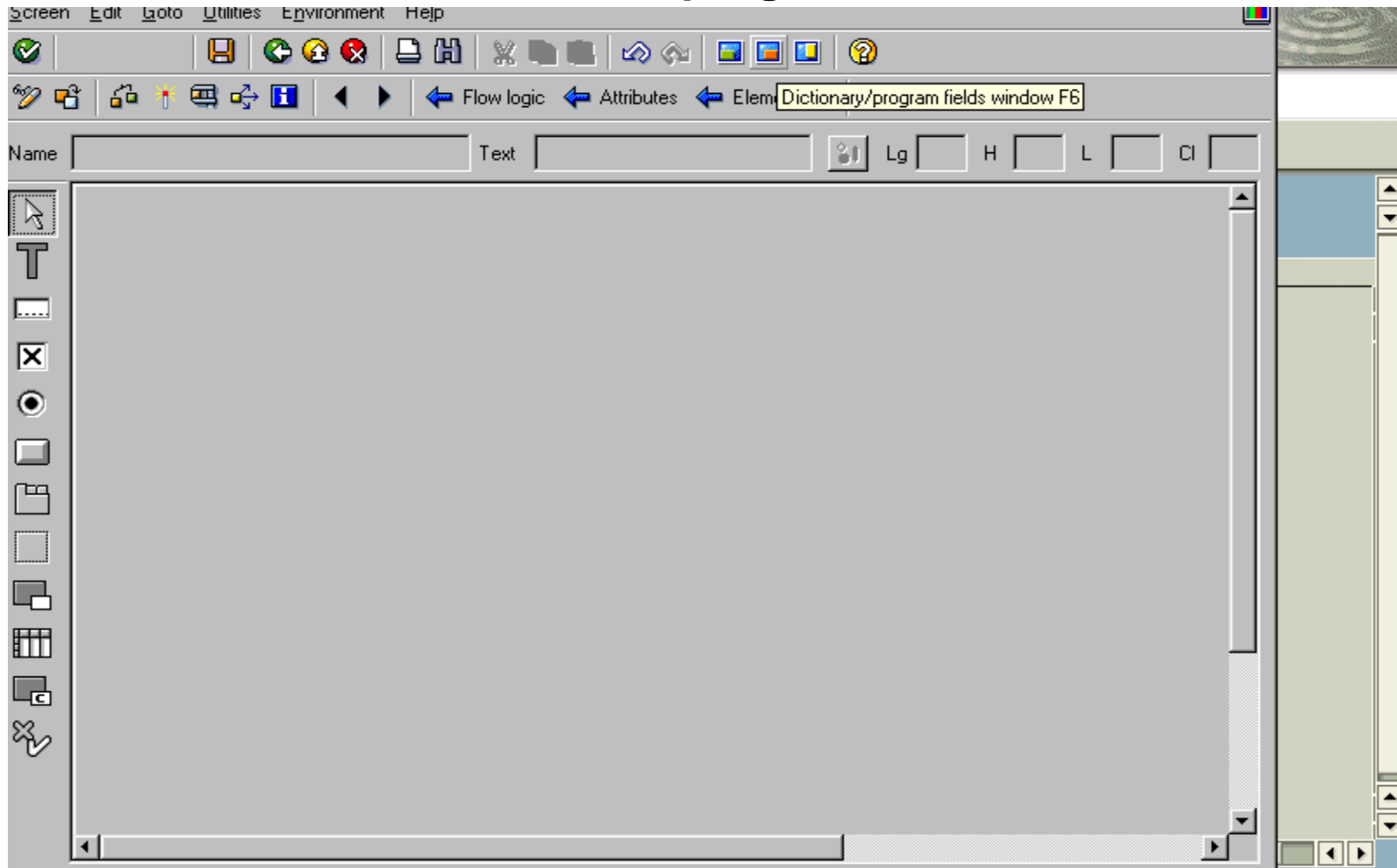
**Cont. Of the previous screen**

**Click on the SAVE icon to save the code.**

**Click the BACK icon to come out.**

**Click on the LAYOUT BUTTON on the application toolbar to design the screen.**

**Click on the DICT/PROGRAM fields button on the application toolbar to include the fields on the screen from the dictionary tables or internal tables or other fields declared in the program**



Enter the Internal or Table or field name on the Table/Field name and click on the Get from program

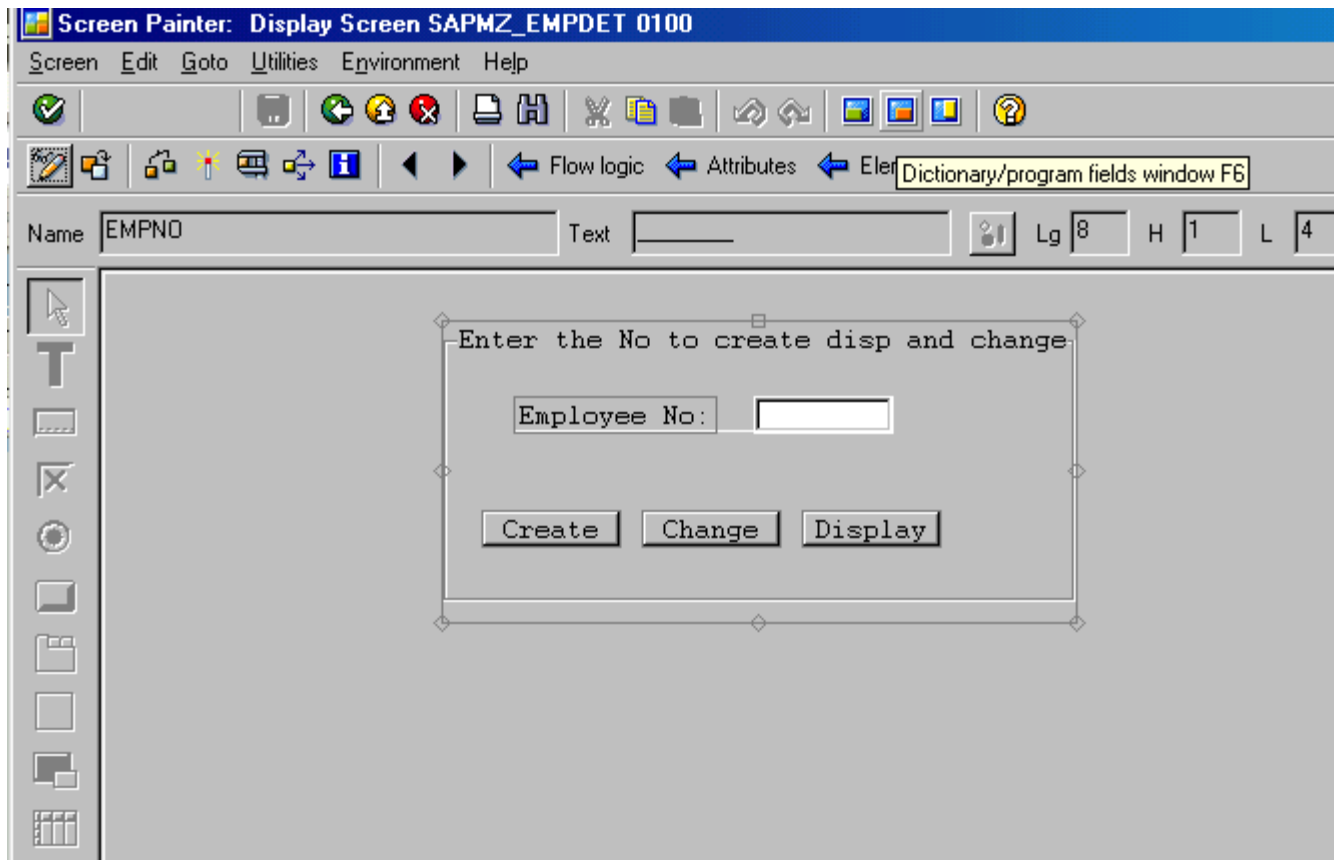
The screenshot shows a software interface with a screen painter window at the top and a data dictionary table below it. The screen painter window contains a text input field with the text "Employee No:" and a label "Enter the No to create disp and change". The data dictionary table below it lists various fields and their attributes.

Screen Painter: Dict./program fields

Table/field name: [EMPNO]      Get from Dict.      Get from pr

Table/field name		Description	I/O field	Text					Copy as		
Table name	Field name			None	Short	Medium	Long	Header	Text	ChkB	Re
	EMONAME		<input checked="" type="checkbox"/> CHAR 30	●	●	●	●	●		●	●
	EMPADDRESS		<input checked="" type="checkbox"/> CHAR 30	●	●	●	●	●		●	●
	EMPDEPT		<input checked="" type="checkbox"/> CHAR 25	●	●	●	●	●		●	●
	EMPDOB		<input checked="" type="checkbox"/> DATS 10	●	●	●	●	●		●	●
	EMPNO		<input checked="" type="checkbox"/> NUMC 8	●	●	●	●	●		●	●
	PUCOMM		<input checked="" type="checkbox"/> CHAR 70	●	●	●	●	●		●	●
ZEMP	MANDT		<input checked="" type="checkbox"/> CHAR 3	●	●	●	●	●		●	●
ZEMP	EMPNO		<input checked="" type="checkbox"/> NUMC 8	●	●	●	●	●		●	●
ZEMP	EMONAME		<input checked="" type="checkbox"/> CHAR 30	●	●	●	●	●		●	●
ZEMP	EMPSEX		<input checked="" type="checkbox"/> CHAR 1	●	●	●	●	●		●	●
ZEMP	EMPDEPT		<input checked="" type="checkbox"/> CHAR 25	●	●	●	●	●		●	●
ZEMP	EMPADDRESS		<input checked="" type="checkbox"/> CHAR 30	●	●	●	●	●		●	●

**Using the corresponding icon (Text, Entry, Check and so on) on the object bar and drag and place the object on he screen.**



**Click on the SAVE icon to save the code .**

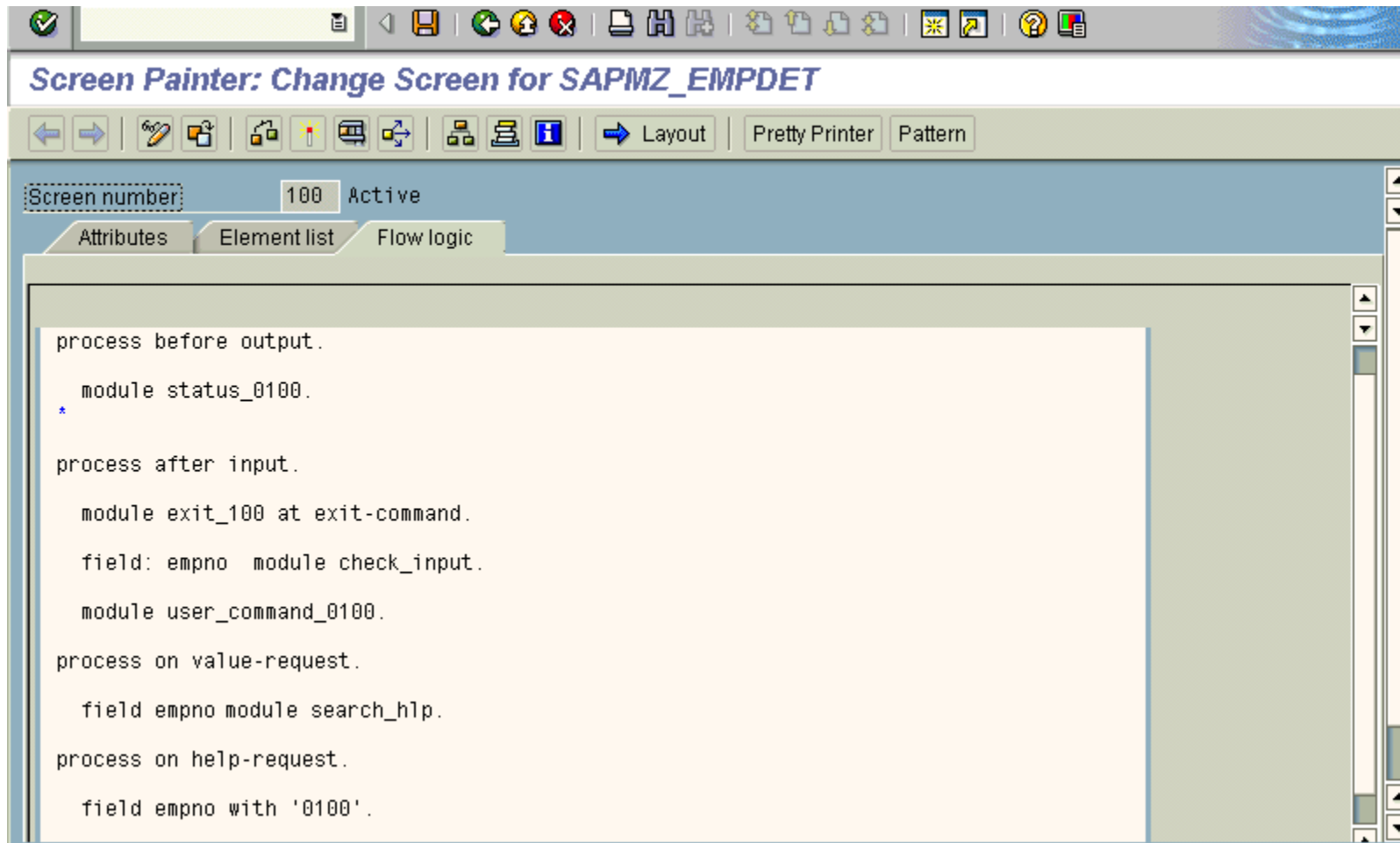
## Double Click on the table Control object to view the attributes or properties

The screenshot displays a software development environment with a central workspace and a right-hand properties panel. The workspace shows a form with a text label "Enter the No to create disp and cl", an input field "Employee No:", and three buttons: "Create", "Change", and "Display". The properties panel on the right is titled "Attributes" and contains the following fields:

- El. type: Input/output field
- Name: EMPNO
- Text: [Empty field]
- Dropdown: [Dropdown menu]
- With icon:
- Scrollable:
- Line: 4
- Def. Lengt: 8
- Column: 41
- Vis. Length: 8
- Height: 1
- Groups: [Four empty input fields]
- FctCode: [Empty field]
- FctType: [Dropdown menu]
- Context menu form: ON\_CTMENU\_ [Empty field]
- Attributes section:
  - Dict: [Empty field]
  - Program: [Empty field]
  - Display: [Empty field]
  - Format: NUMC
  - From dict.:
  - Modify: [Dropdown menu]
  - Conv. Exit: [Empty field]
  - Search help: [Empty field]
  - Ref. field: [Empty field]
  - Parameter ID: PER

The bottom right corner of the interface shows the text "P04 OVR".

**Now write the code for the PBO, PAI, POV and POH for this screen flow logic**

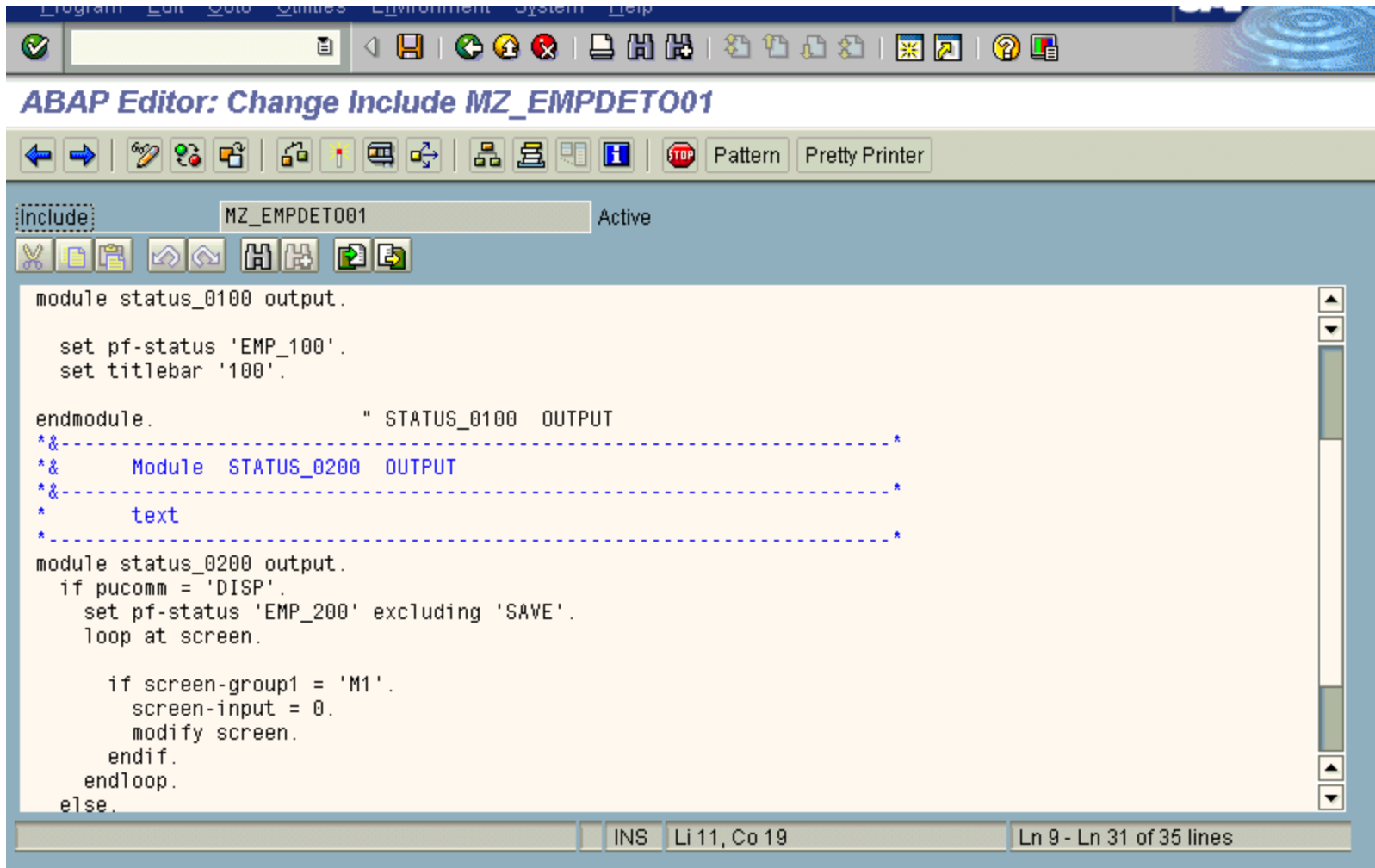


The screenshot shows the SAP Screen Painter interface for screen 100. The title bar reads "Screen Painter: Change Screen for SAPMZ\_EMPDET". The interface includes a toolbar with navigation and editing tools, and a tabbed view with "Attributes", "Element list", and "Flow logic" selected. The "Flow logic" tab displays the following code:

```
process before output.  
  module status_0100.  
*  
process after input.  
  module exit_100 at exit-command.  
  field: empno module check_input.  
  module user_command_0100.  
process on value-request.  
  field empno module search_hlp.  
process on help-request.  
  field empno with '0100'.
```



**In the PBO write the relevant code for PF-STATUS and TITLE BAR**



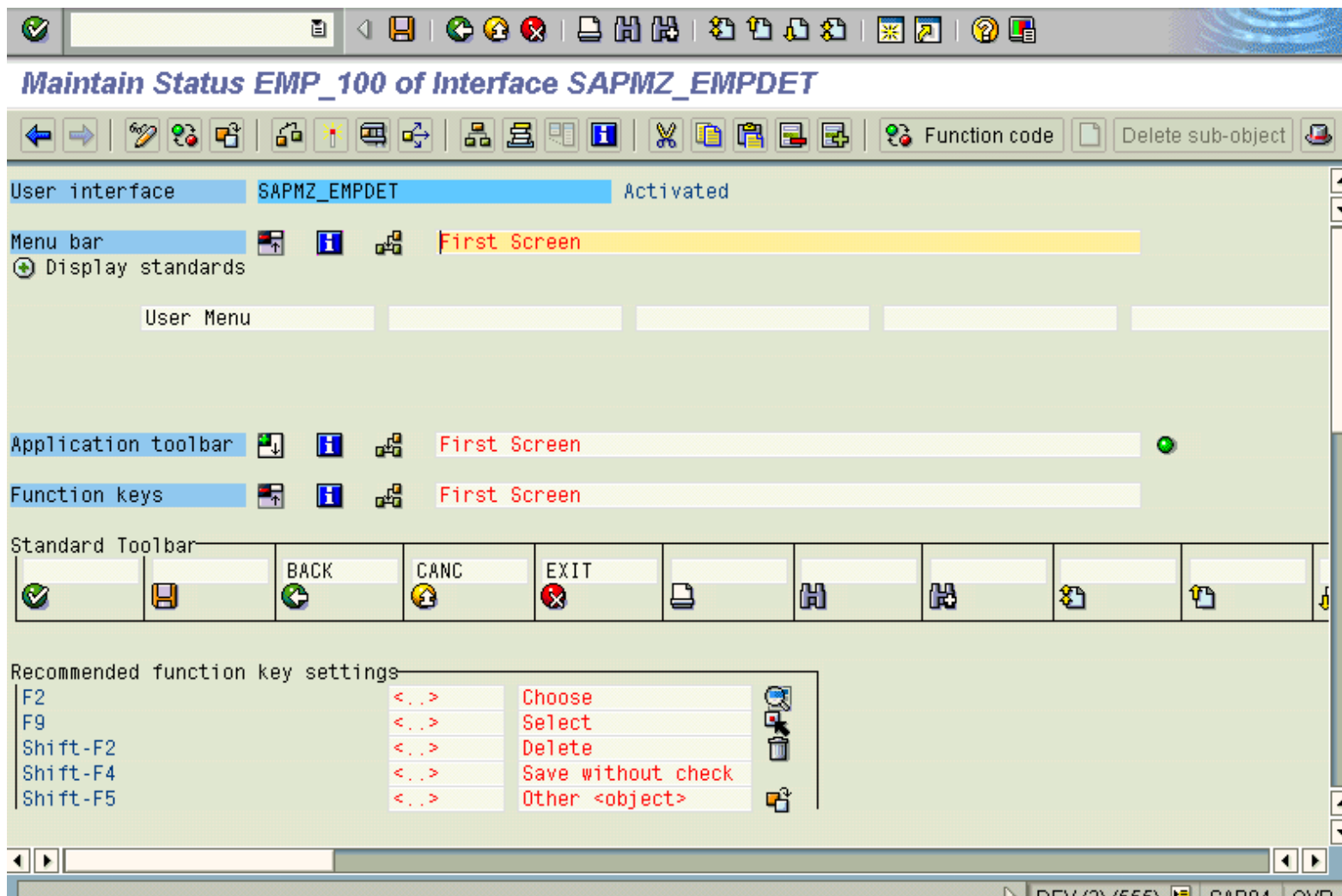
The screenshot shows the ABAP Editor interface. The title bar reads "ABAP Editor: Change Include MZ\_EMPDET001". The editor window displays the following code:

```
module status_0100 output.  
    set pf-status 'EMP_100'.  
    set titlebar '100'.  
  
endmodule.                " STATUS_0100  OUTPUT  
*&-----*  
*&   Module STATUS_0200  OUTPUT  
*&-----*  
*   text  
*&-----*  
module status_0200 output.  
    if pucomm = 'DISP'.  
        set pf-status 'EMP_200' excluding 'SAVE'.  
        loop at screen.  
  
            if screen-group1 = 'M1'.  
                screen-input = 0.  
                modify screen.  
            endif.  
        endloop.  
    else.
```

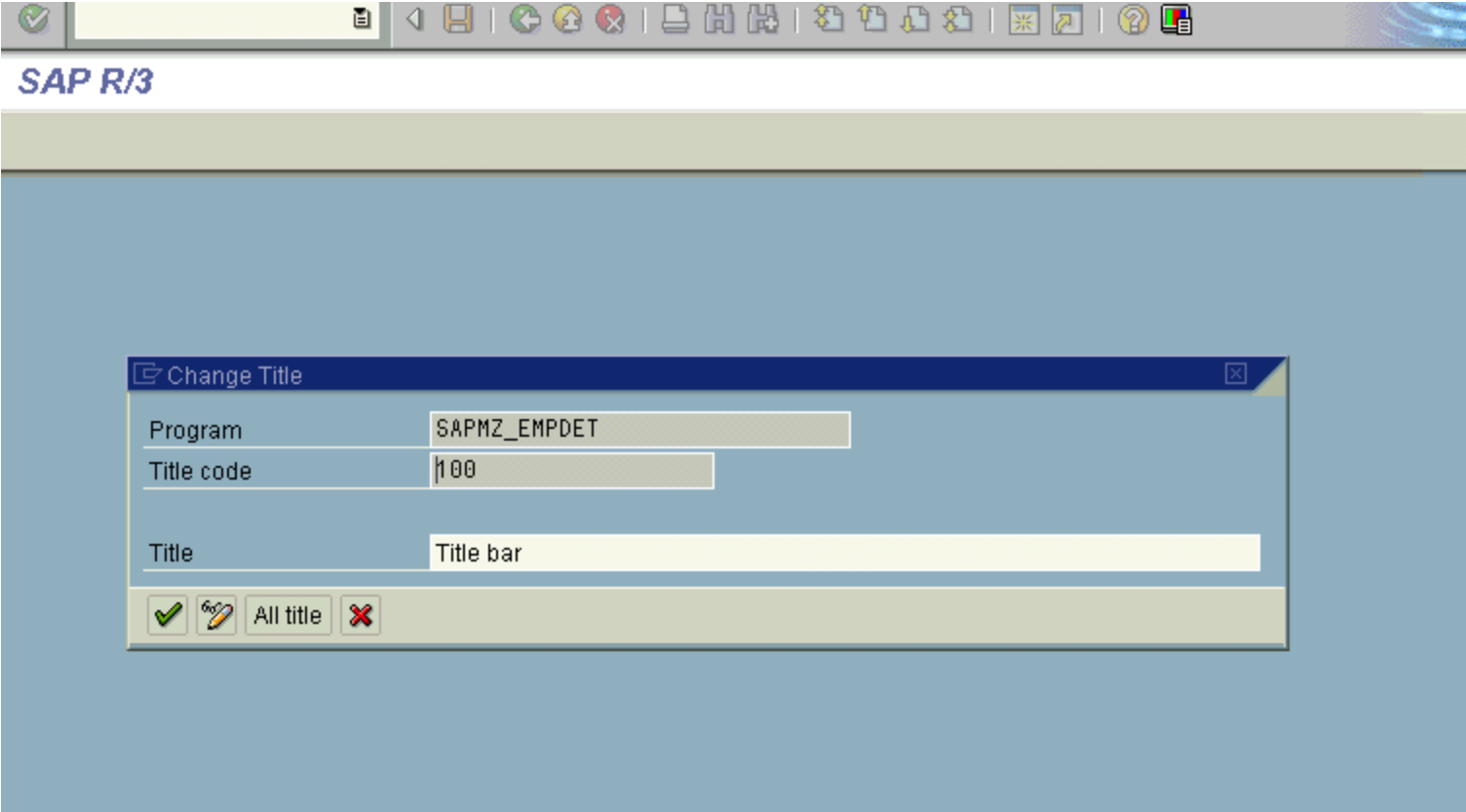
The status bar at the bottom indicates "INS Li 11, Co 19" and "Ln 9 - Ln 31 of 35 lines".

**Click on the SAVE icon to save the code .**

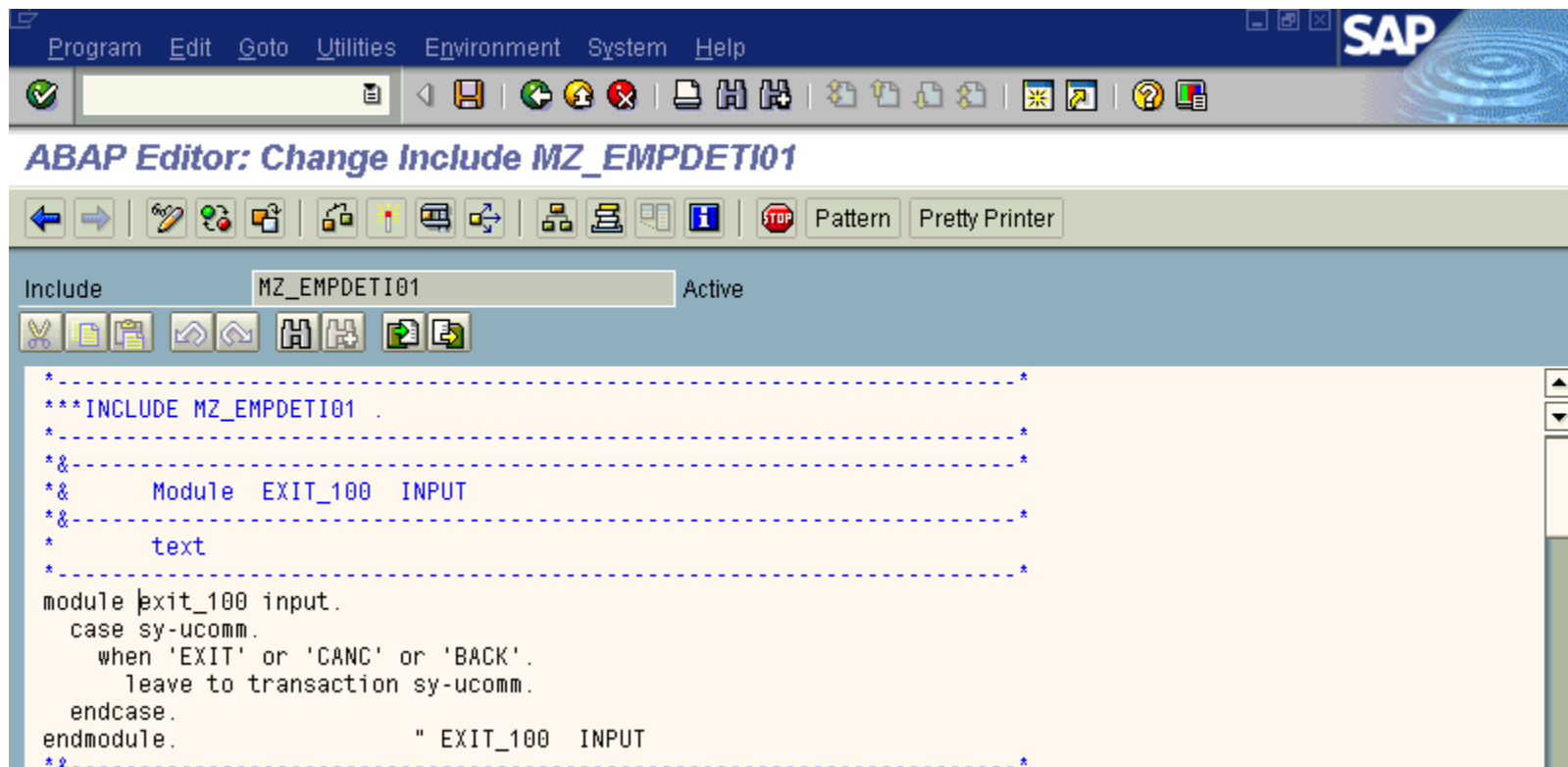
This is a GUI status containing buttons and menus for the screen. All ABAP programs will have a default GUI status.



The title Bar of the window that you will use to display.



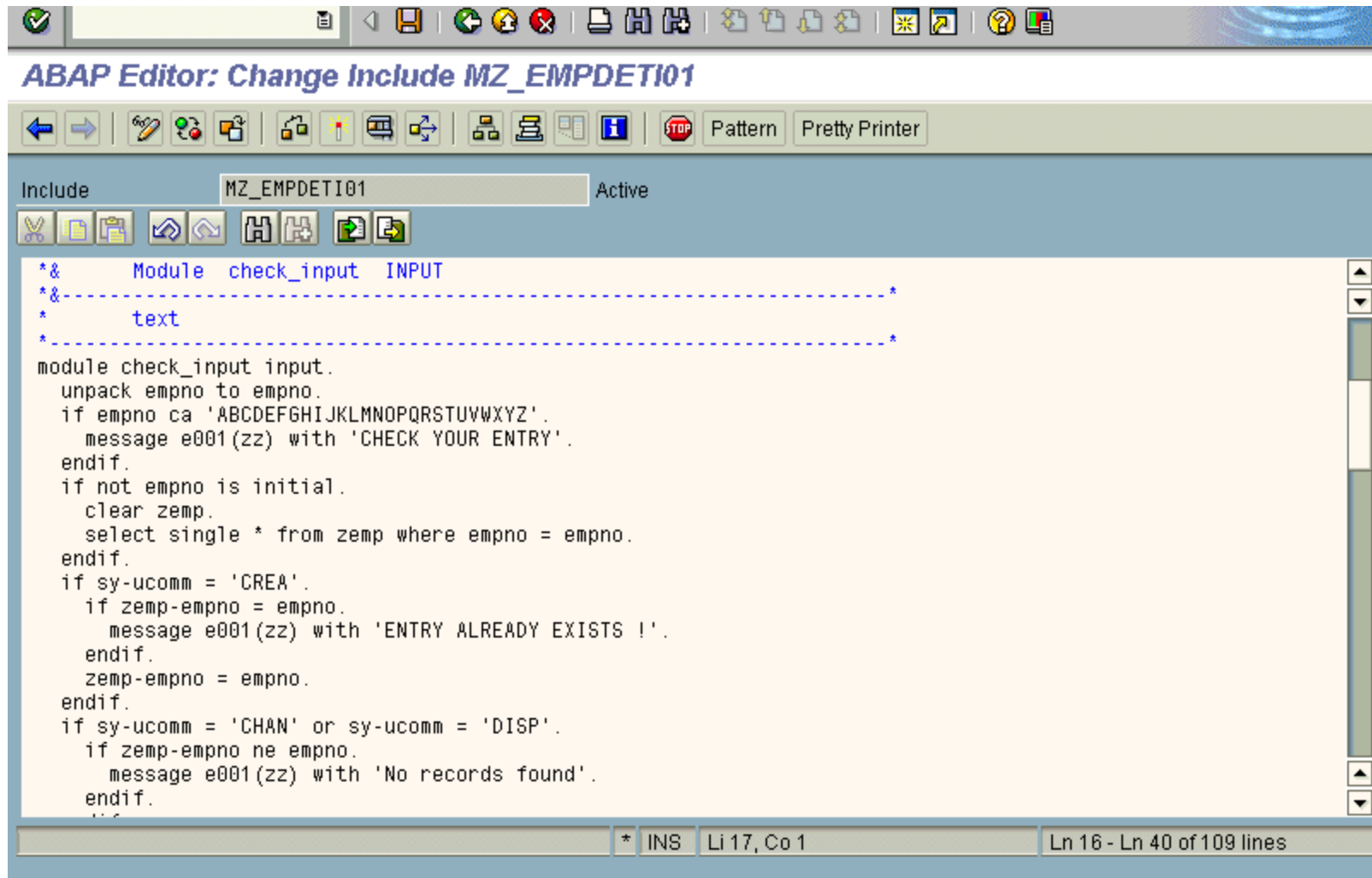
**In the PAI of the flow logic we can use the conditional statement AT EXIT-COMMAND as below.**



```
*-----*
**INCLUDE MZ_EMPDETI01 .
*-----*
*&
*&   Module  EXIT_100  INPUT
*&
*&   text
*-----*
module |exit_100 input.
  case sy-ucomm.
    when 'EXIT' or 'CANC' or 'BACK'.
      leave to transaction sy-ucomm.
    endcase.
endmodule.
" EXIT_100  INPUT
*&
*-----*
```

**Click on the SAVE icon to save the code .**

## Code for the Field check in the PAI module.

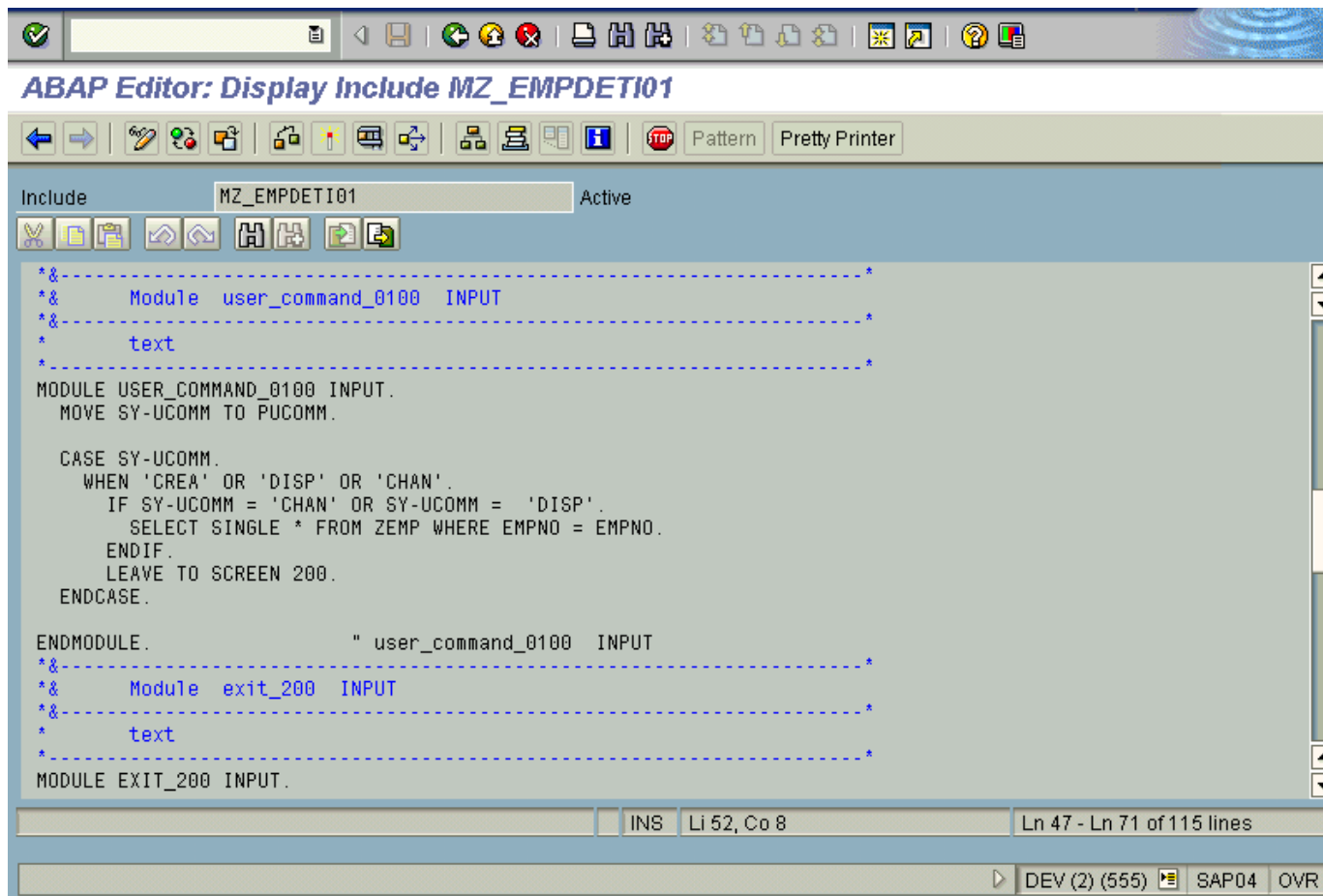


```
*&      Module  check_input  INPUT
*&-----*
*      text
*&-----*
module check_input input.
  unpack empno to empno.
  if empno ca 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
    message e001(zz) with 'CHECK YOUR ENTRY'.
  endif.
  if not empno is initial.
    clear zemp.
    select single * from zemp where empno = empno.
  endif.
  if sy-ucomm = 'CREA'.
    if zemp-empno = empno.
      message e001(zz) with 'ENTRY ALREADY EXISTS !'.
    endif.
    zemp-empno = empno.
  endif.
  if sy-ucomm = 'CHAN' or sy-ucomm = 'DISP'.
    if zemp-empno ne empno.
      message e001(zz) with 'No records found'.
    endif.
  endif.
endmodule.
```

\* INS Li 17, Co 1 Ln 16 - Ln 40 of 109 lines

Click on the **SAVE** icon to save the code .

**In this module we can even write the code to Retrieve the Data, Branching to Different Screen and also use function codes .**



```
ABAP Editor: Display Include MZ_EMPDETI01

Include MZ_EMPDETI01 Active

*&-----*
*&   Module user_command_0100 INPUT
*&-----*
*   text
*-----*

MODULE USER_COMMAND_0100 INPUT.
  MOVE SY-UCOMM TO PUCOMM.

  CASE SY-UCOMM.
    WHEN 'CREA' OR 'DISP' OR 'CHAN'.
      IF SY-UCOMM = 'CHAN' OR SY-UCOMM = 'DISP'.
        SELECT SINGLE * FROM ZEMP WHERE EMPNO = EMPNO.
      ENDIF.
      LEAVE TO SCREEN 200.
    ENDCASE.

ENDMODULE.          " user_command_0100 INPUT

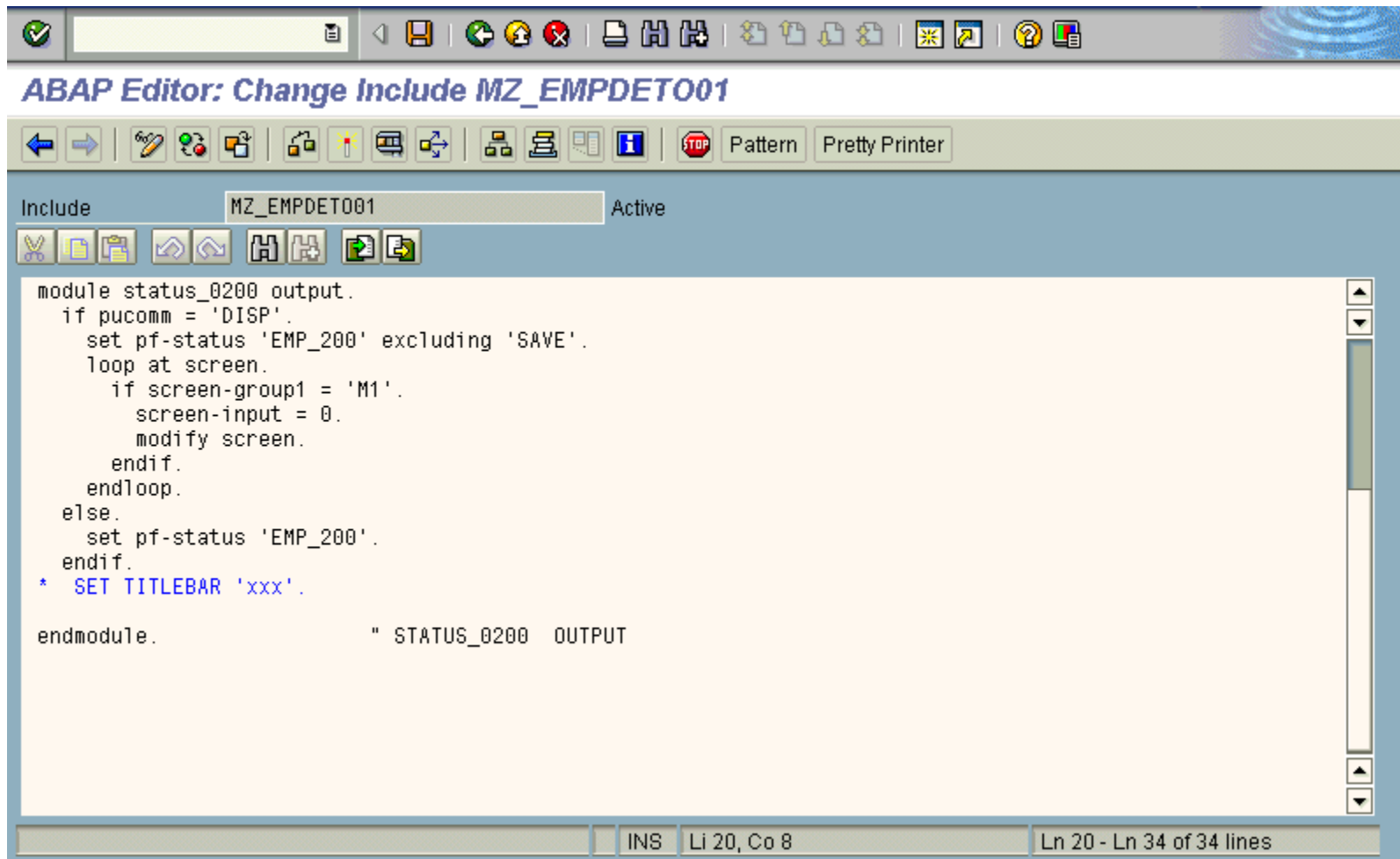
*&-----*
*&   Module exit_200 INPUT
*&-----*
*   text
*-----*

MODULE EXIT_200 INPUT.
```

INS Li 52, Co 8 Ln 47 - Ln 71 of 115 lines

DEV (2) (555) SAP04 OVR

Depending on the user requirements in the previous screen 100, you can dynamically change the Attributes of the object using SCREEN Attributes in the screen 200.

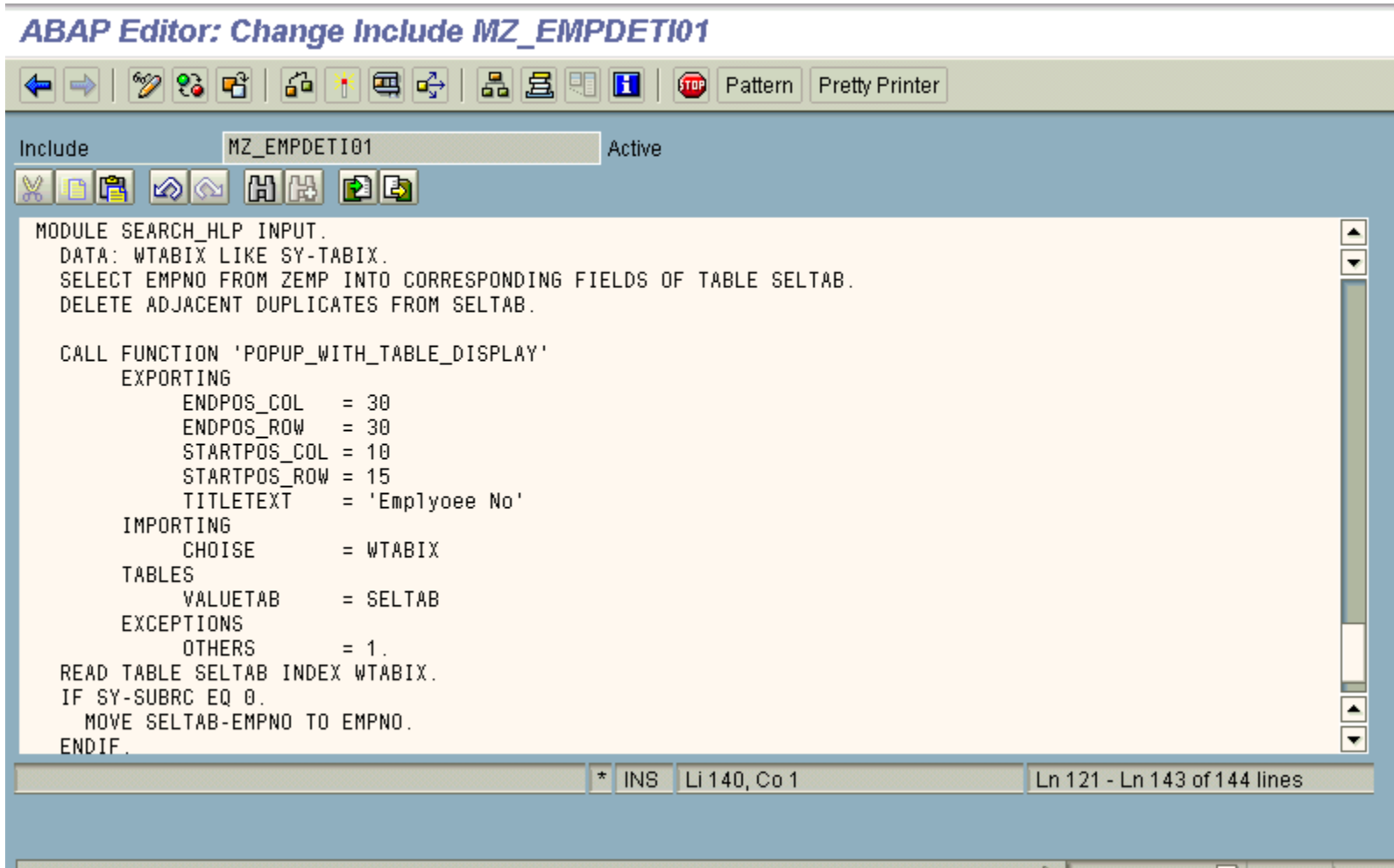


The screenshot shows the ABAP Editor interface. The title bar reads "ABAP Editor: Change Include MZ\_EMPDET001". The editor window displays the following code:

```
module status_0200 output.  
  if pucomm = 'DISP'.  
    set pf-status 'EMP_200' excluding 'SAVE'.  
    loop at screen.  
      if screen-group1 = 'M1'.  
        screen-input = 0.  
        modify screen.  
      endif.  
    endloop.  
  else.  
    set pf-status 'EMP_200'.  
  endif.  
  * SET TITLEBAR 'xxx'.  
endmodule.                " STATUS_0200  OUTPUT
```

The status bar at the bottom indicates "INS Li 20, Co 8" and "Ln 20 - Ln 34 of 34 lines".

**In the Process on value request (POV) of the flow logic we can use the search help for a particular field**



```
ABAP Editor: Change Include MZ_EMPDETI01

Include MZ_EMPDETI01 Active

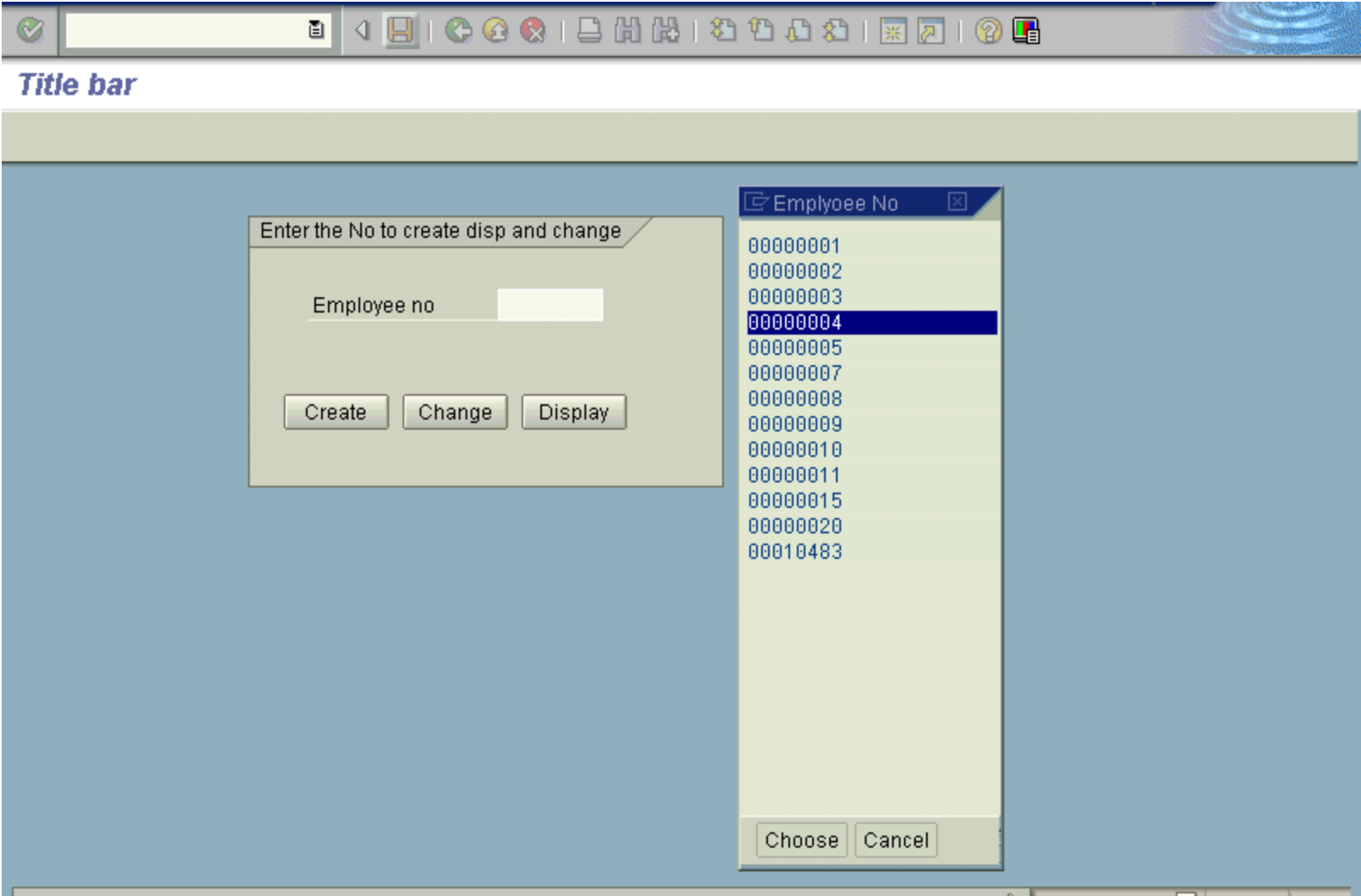
MODULE SEARCH_HLP INPUT.
  DATA: WTABIX LIKE SY-TABIX.
  SELECT EMPNO FROM ZEMP INTO CORRESPONDING FIELDS OF TABLE SELTAB.
  DELETE ADJACENT DUPLICATES FROM SELTAB.

  CALL FUNCTION 'POPUP_WITH_TABLE_DISPLAY'
    EXPORTING
      ENDPOS_COL   = 30
      ENDPOS_ROW   = 30
      STARTPOS_COL = 10
      STARTPOS_ROW = 15
      TITLETEXT    = 'Emplyoee No'
    IMPORTING
      CHOISE       = WTABIX
    TABLES
      VALUETAB     = SELTAB
    EXCEPTIONS
      OTHERS       = 1.
  READ TABLE SELTAB INDEX WTABIX.
  IF SY-SUBRC EQ 0.
    MOVE SELTAB-EMPNO TO EMPNO.
  ENDIF.
```

\* INS Li 140, Co 1 Ln 121 - Ln 143 of 144 lines

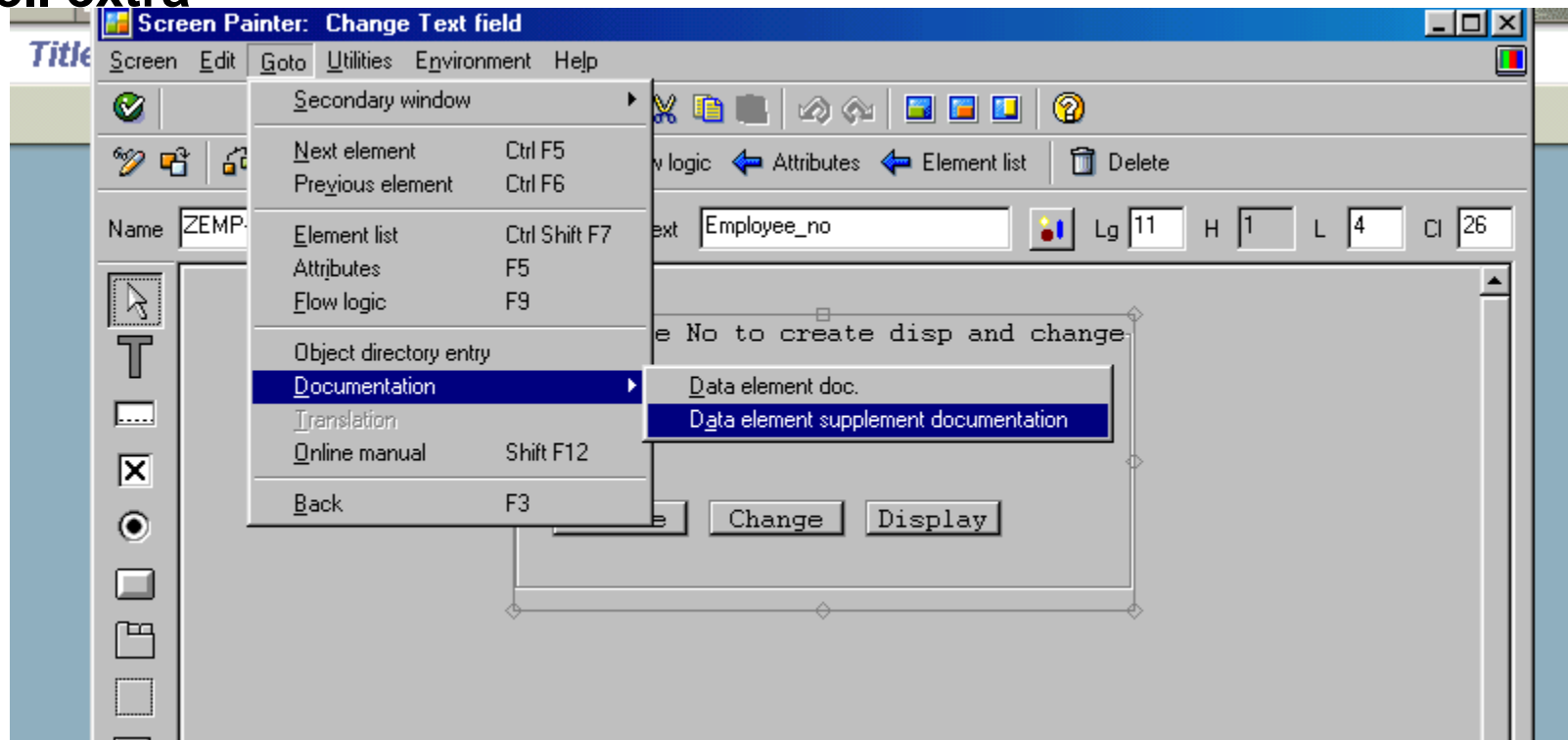


# You Can Use the F4 to see the search help

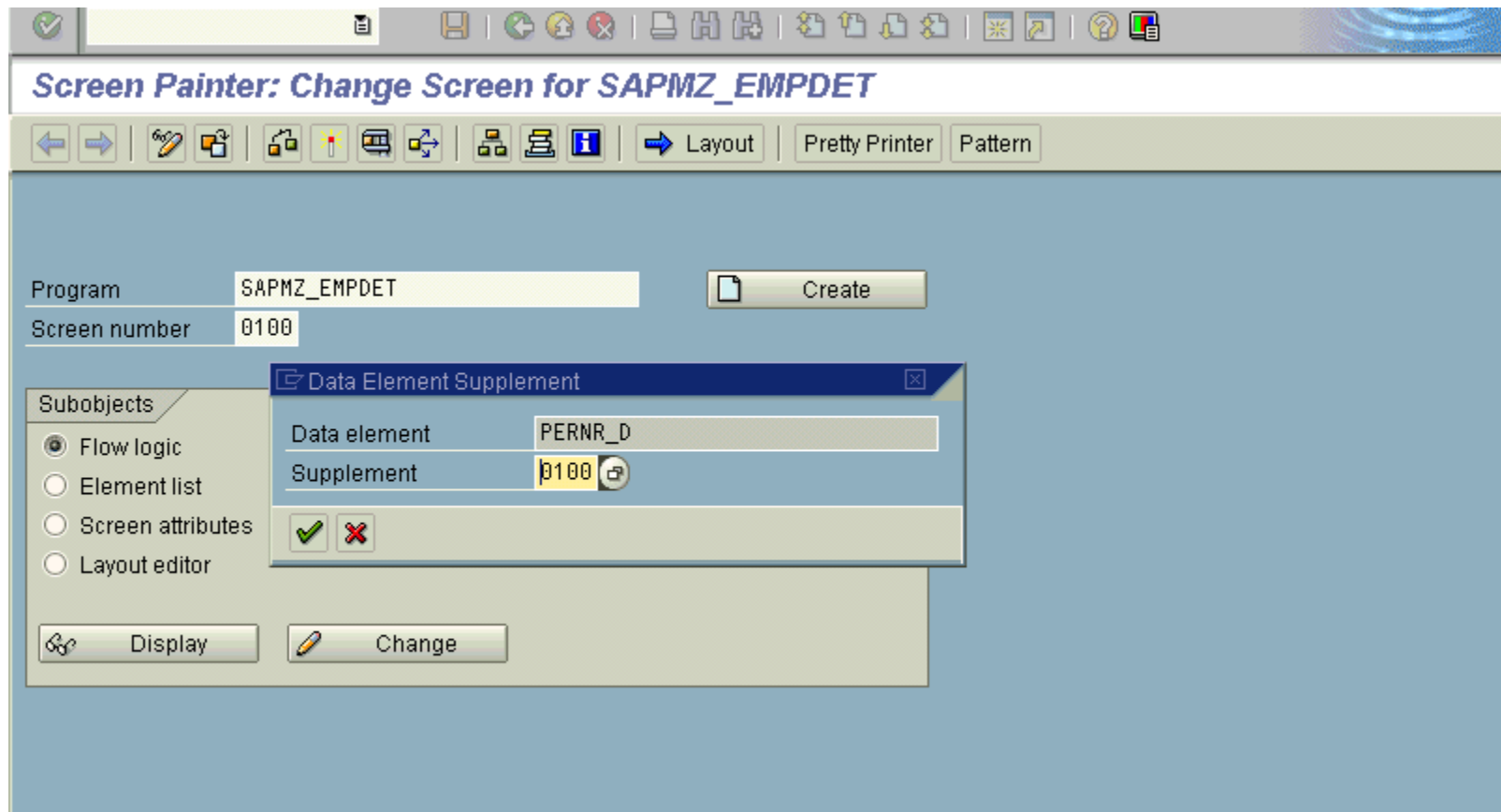


In the Process on value Help (POH) of the flow logic we can use the additional documentation descriptive text for the data element in the ABAP/4 Dict.

In the screen painter you place the cursor in the field string of a screen on the field to be documented and select the menu goto > documentation > data el. extra

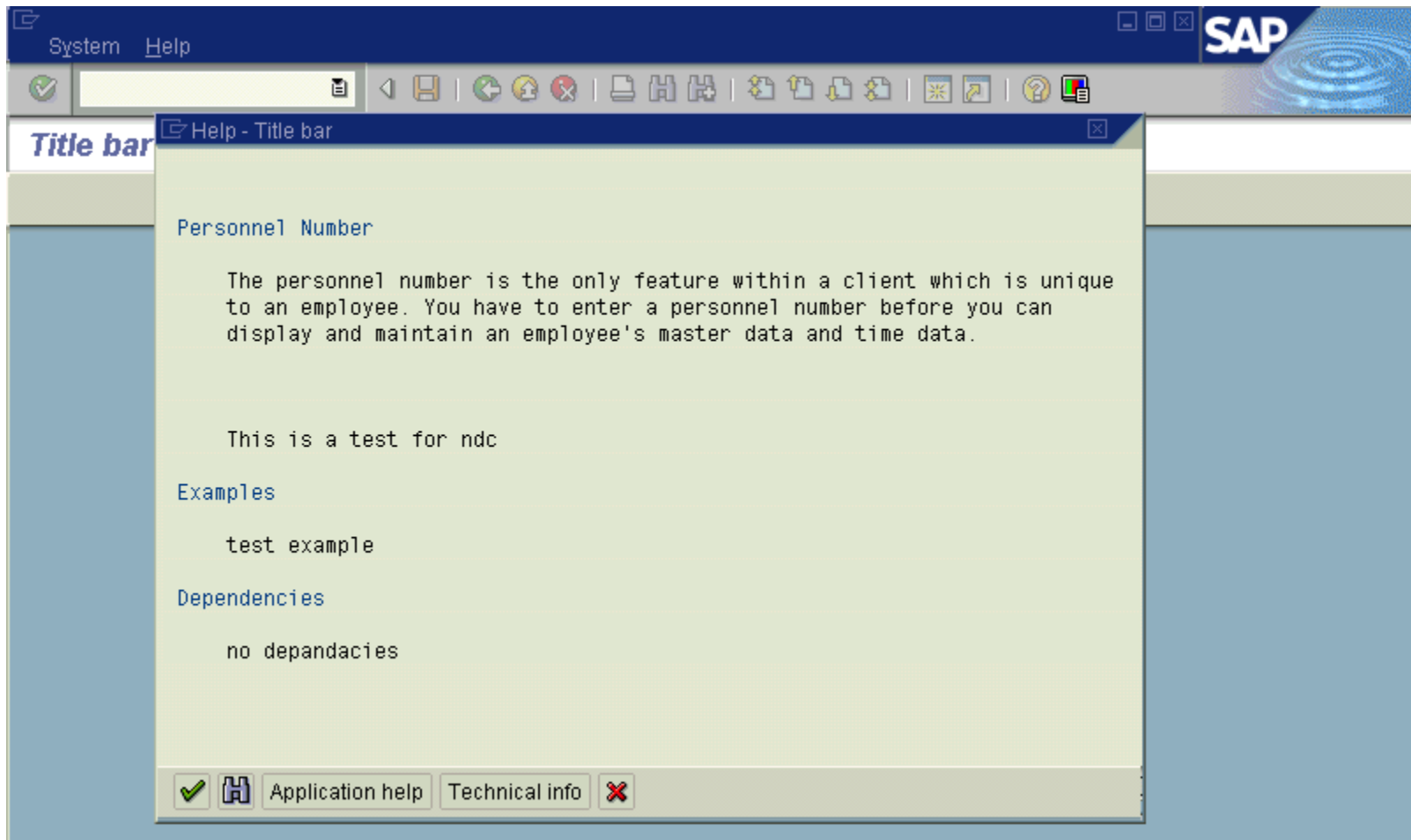


**Now you will get a popup with the data element and the number of the current screen as an identifier for the additional text. In addition to the help contents in the ABAP/4 Dict you can enter your own Description**

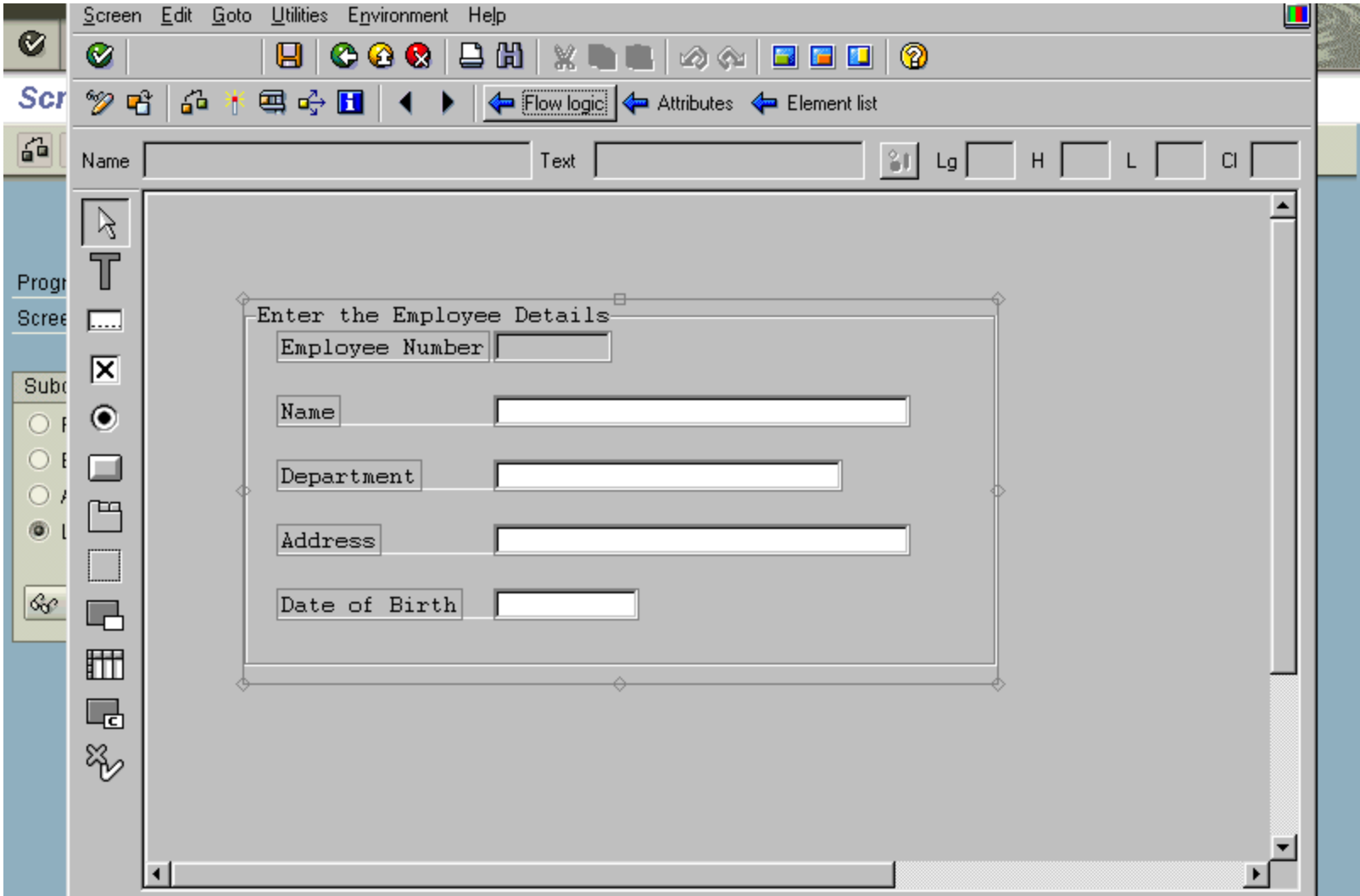




**Now while Running transaction place the cursor on the field and press F1**

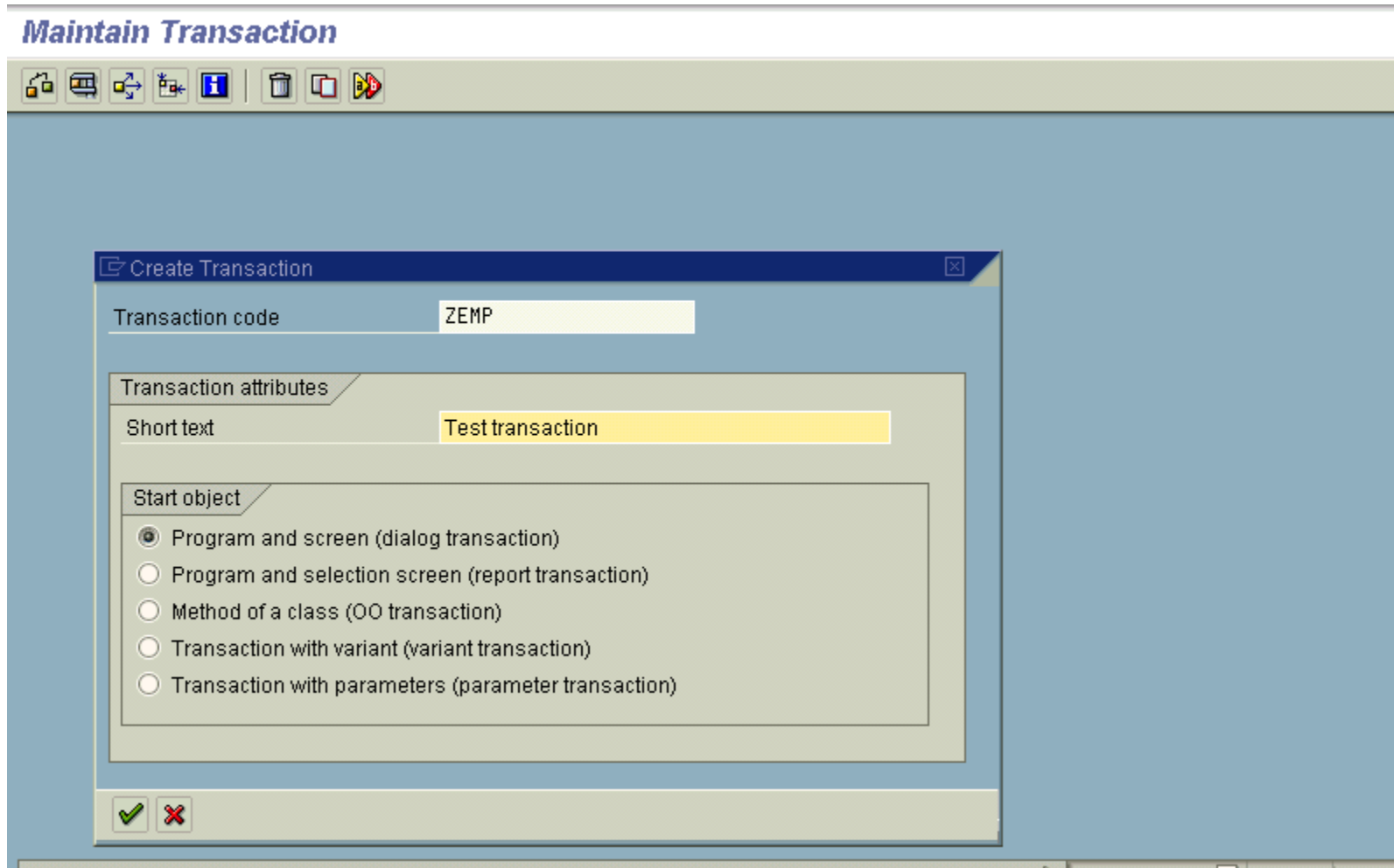


# Layout slide for screen 200.



**Now Create a transaction code for your dialog program using Txn SE93**

*Maintain Transaction*



Transaction code: ZEMP

Transaction attributes

Short text: Test transaction

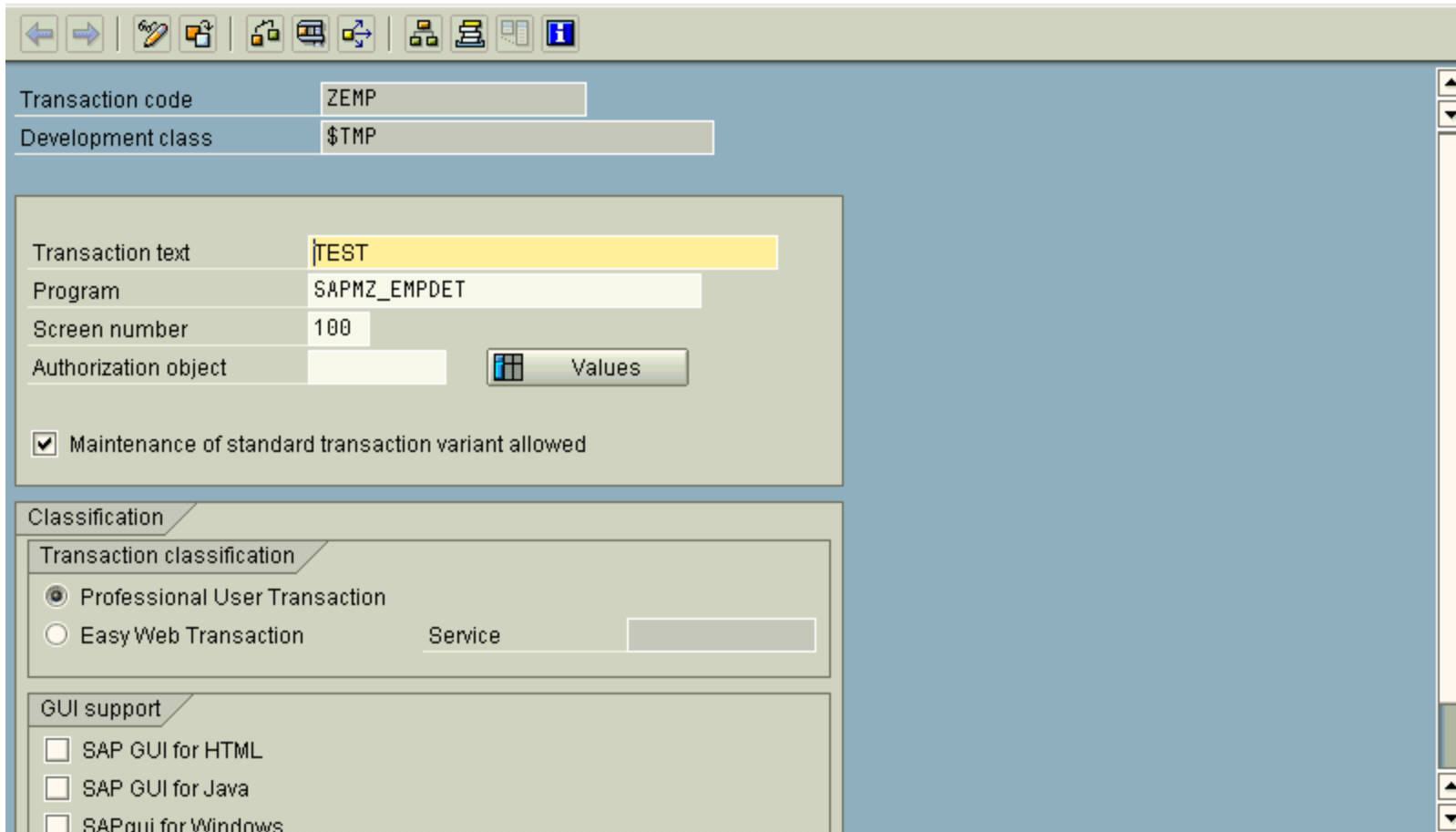
Start object

- Program and screen (dialog transaction)
- Program and selection screen (report transaction)
- Method of a class (OO transaction)
- Transaction with variant (variant transaction)
- Transaction with parameters (parameter transaction)

✓ ✗

**Click on the enter button**

## Change Dialog transaction



Transaction code: ZEMP

Development class: \$TMP

Transaction text: TEST

Program: SAPMZ\_EMPDET

Screen number: 100

Authorization object:  Values

Maintenance of standard transaction variant allowed

**Classification**

Transaction classification

Professional User Transaction

Easy Web Transaction Service

**GUI support**

SAP GUI for HTML

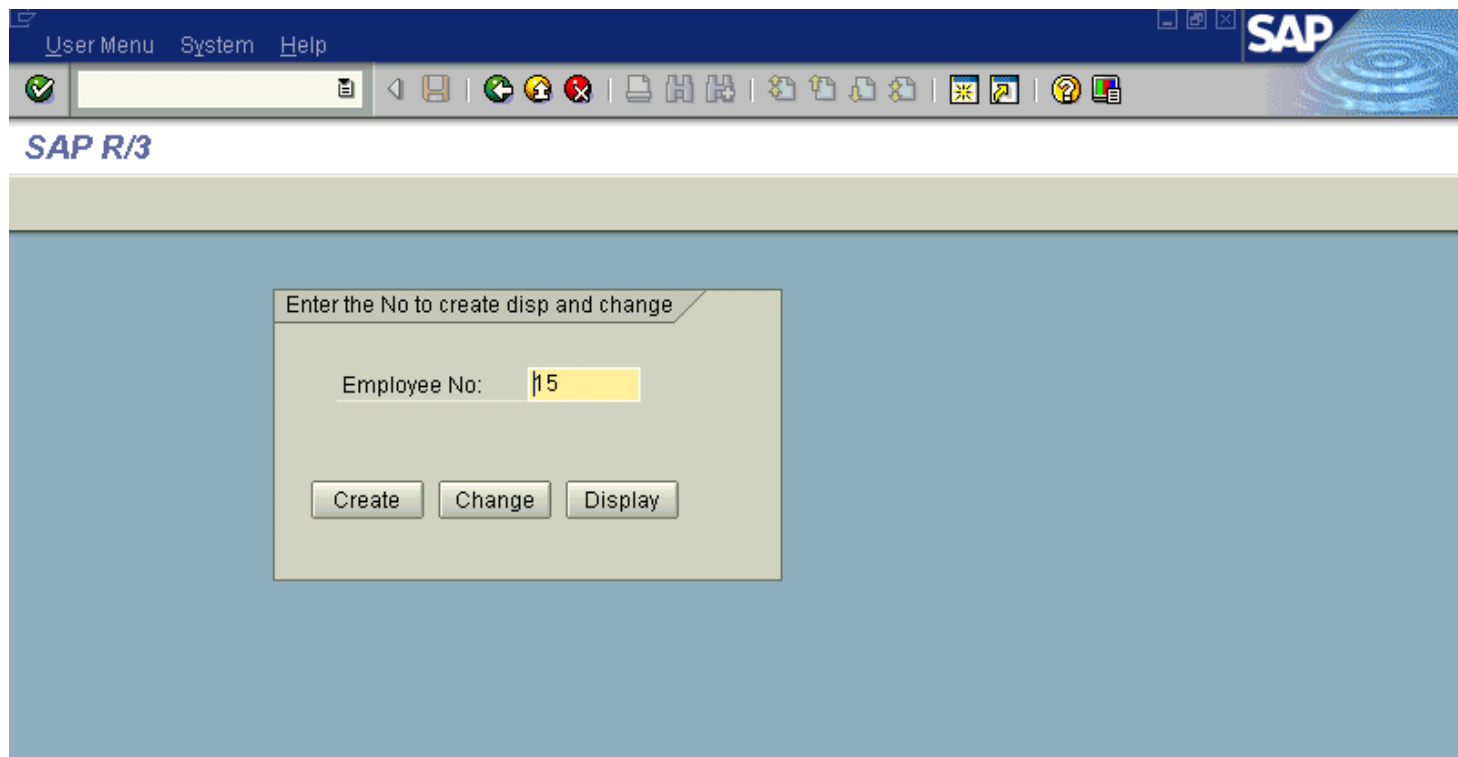
SAP GUI for Java

SAPgui for Windows

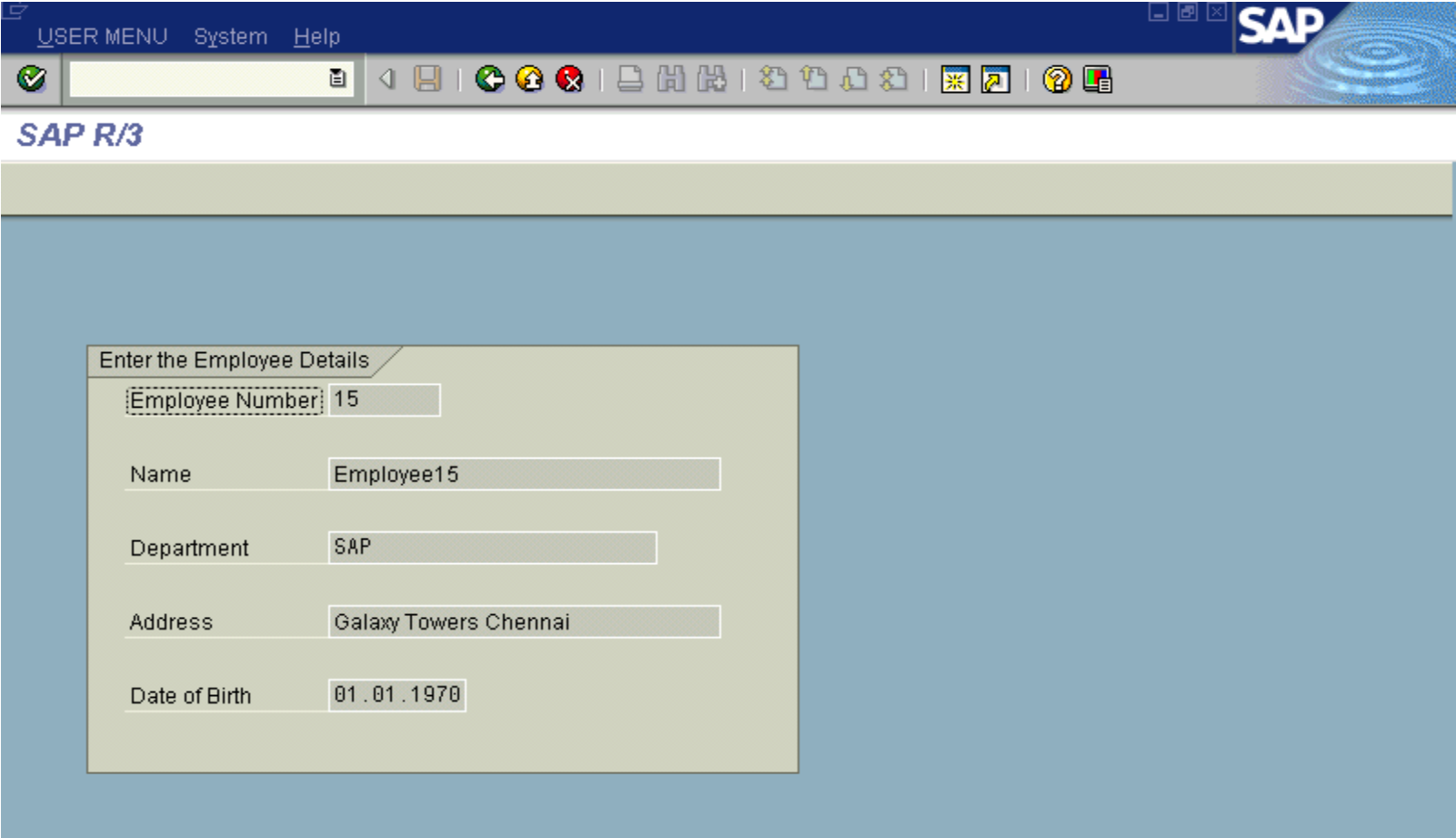
**Enter the information in the fields and SAVE**



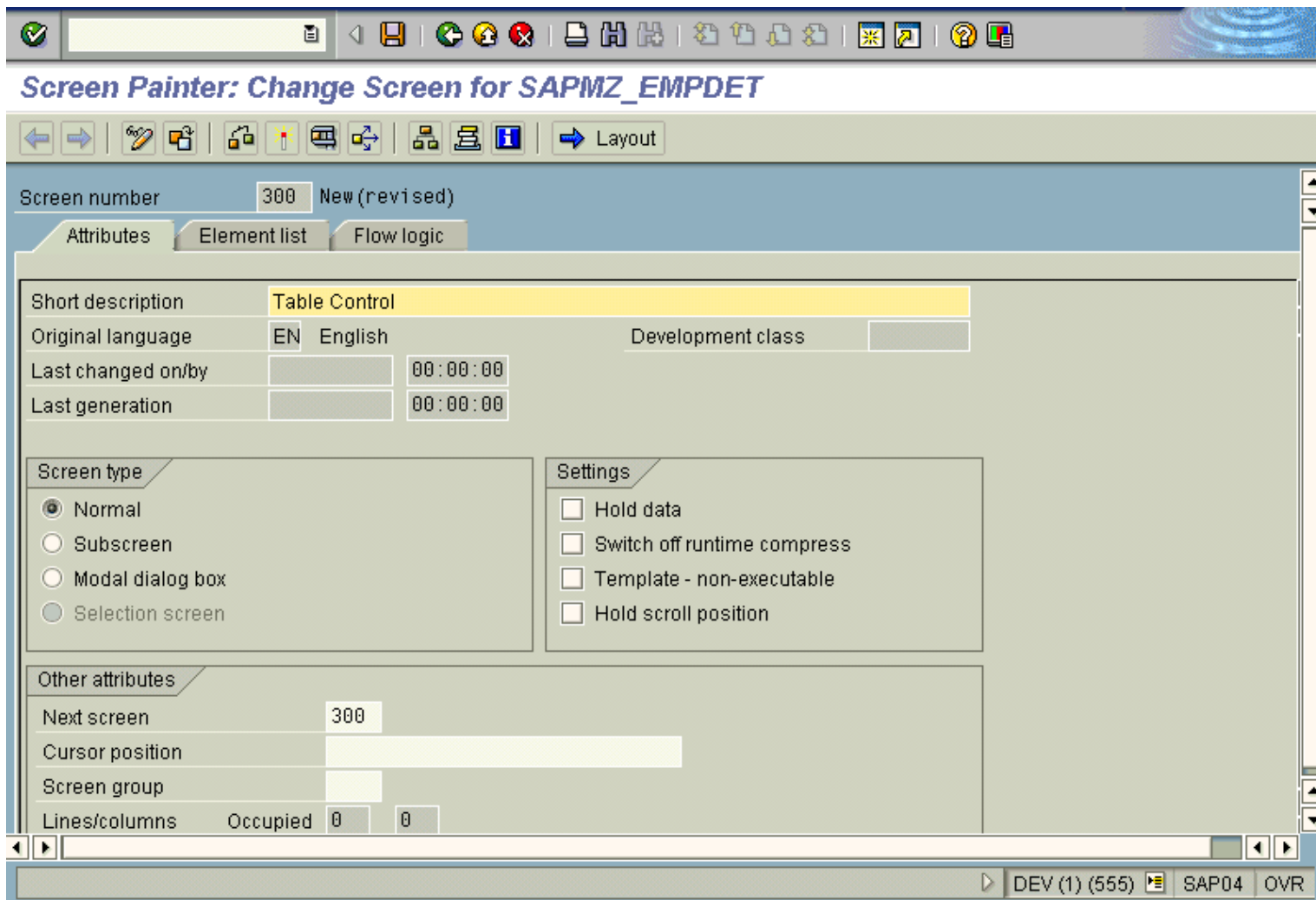
**In Screen 100 we give the required input(Employee No) and retrieve the Data from the Data Dictionary and display the details on to the screen 200.**



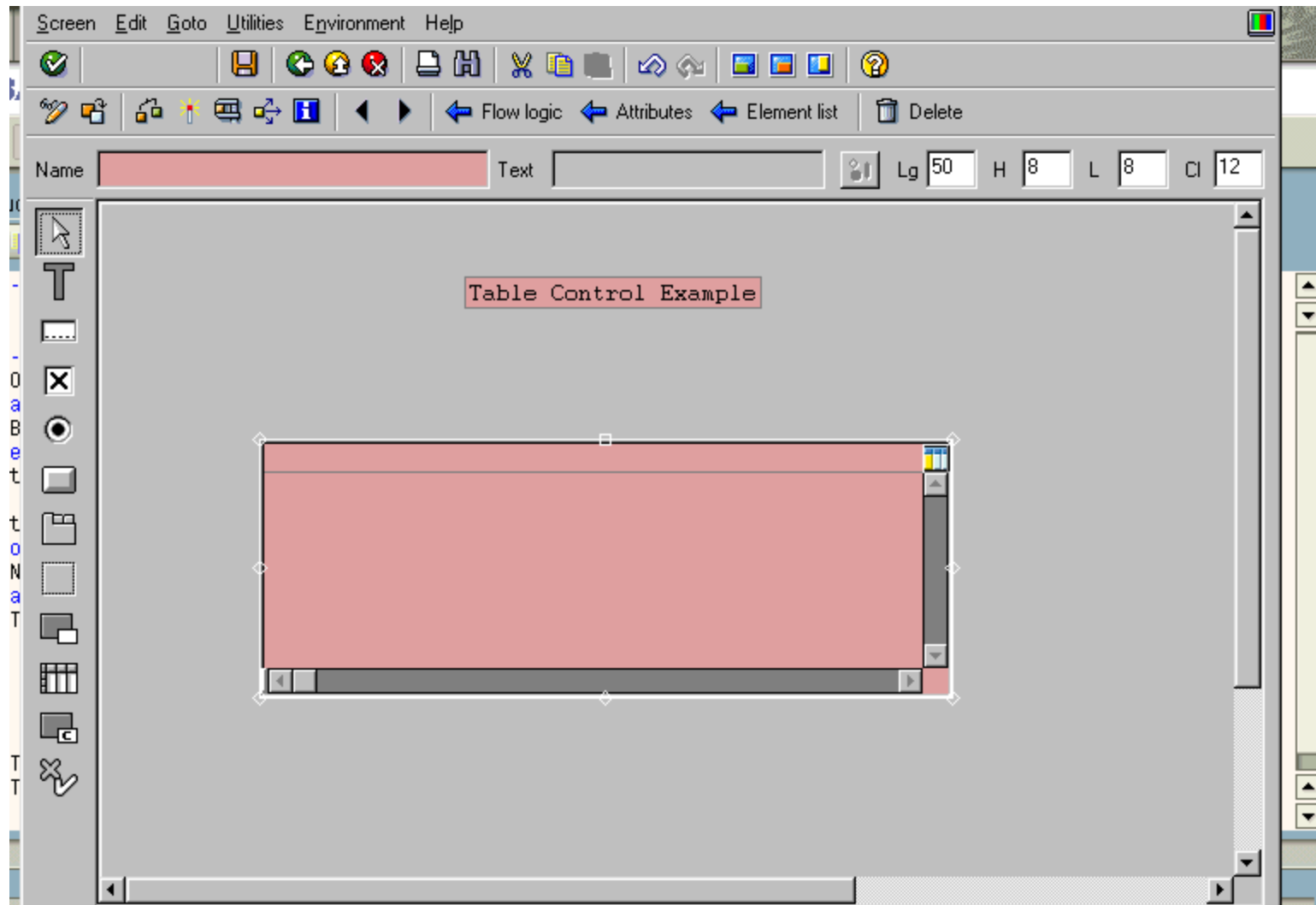
# The details of the employee in screen 200.



**In this example we will create a screen and use the table control object to display data. Use SE51 to Create a screen (In our case 300)**

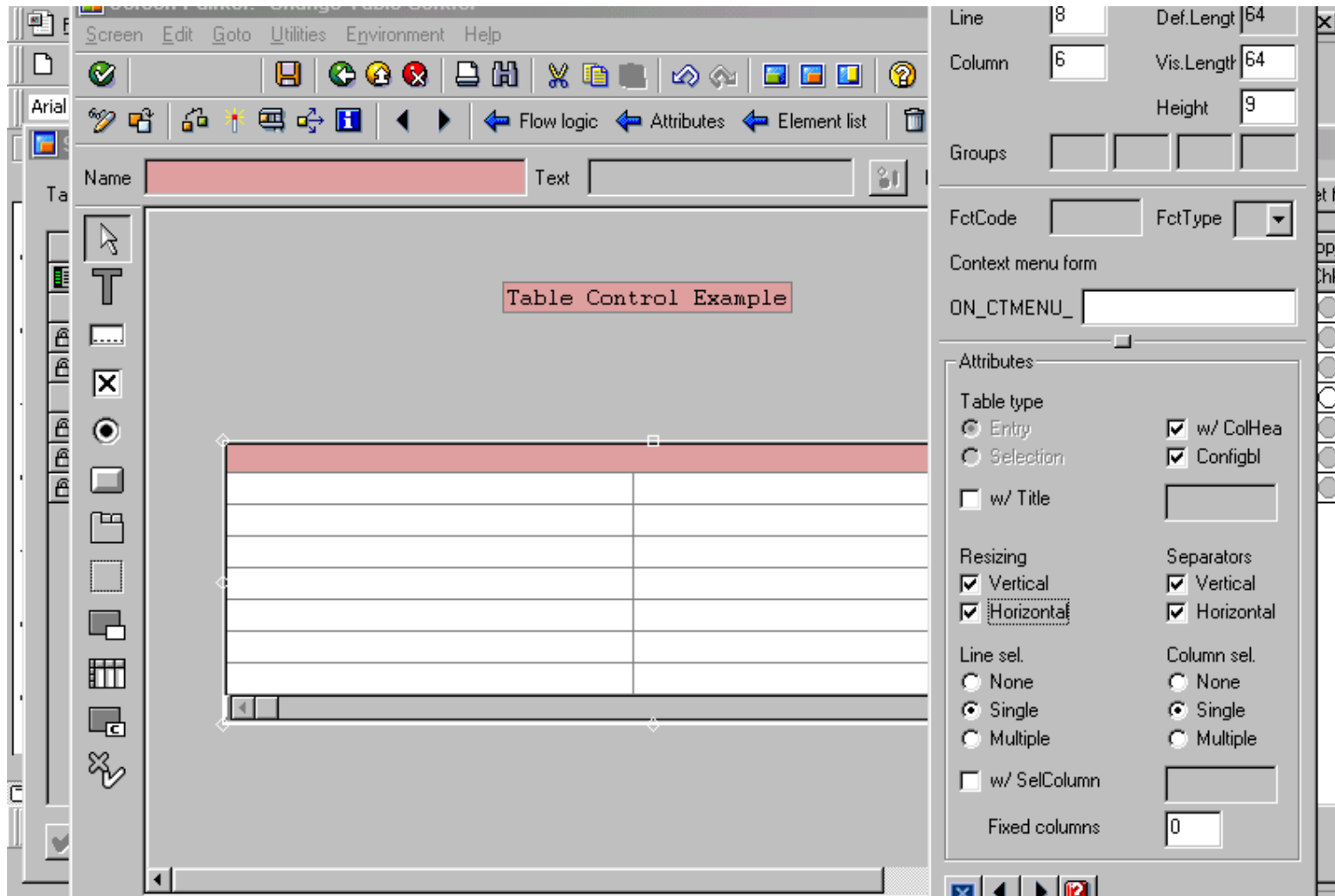


**We will design the screen with a text object and the table control object**

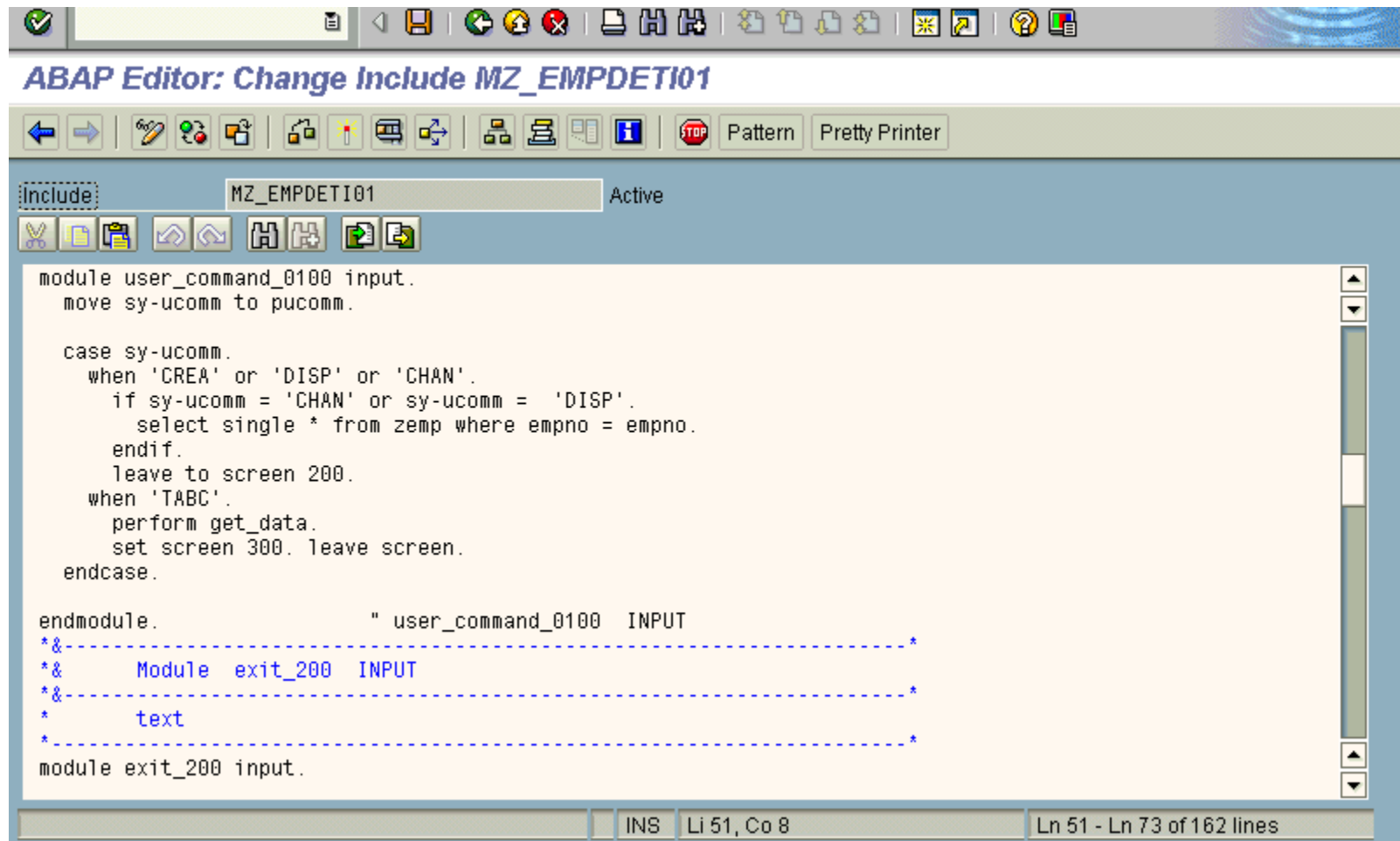




**Using the attribute button of the table control you can put vertical and horizontal separators and resizing if required and name the table control**



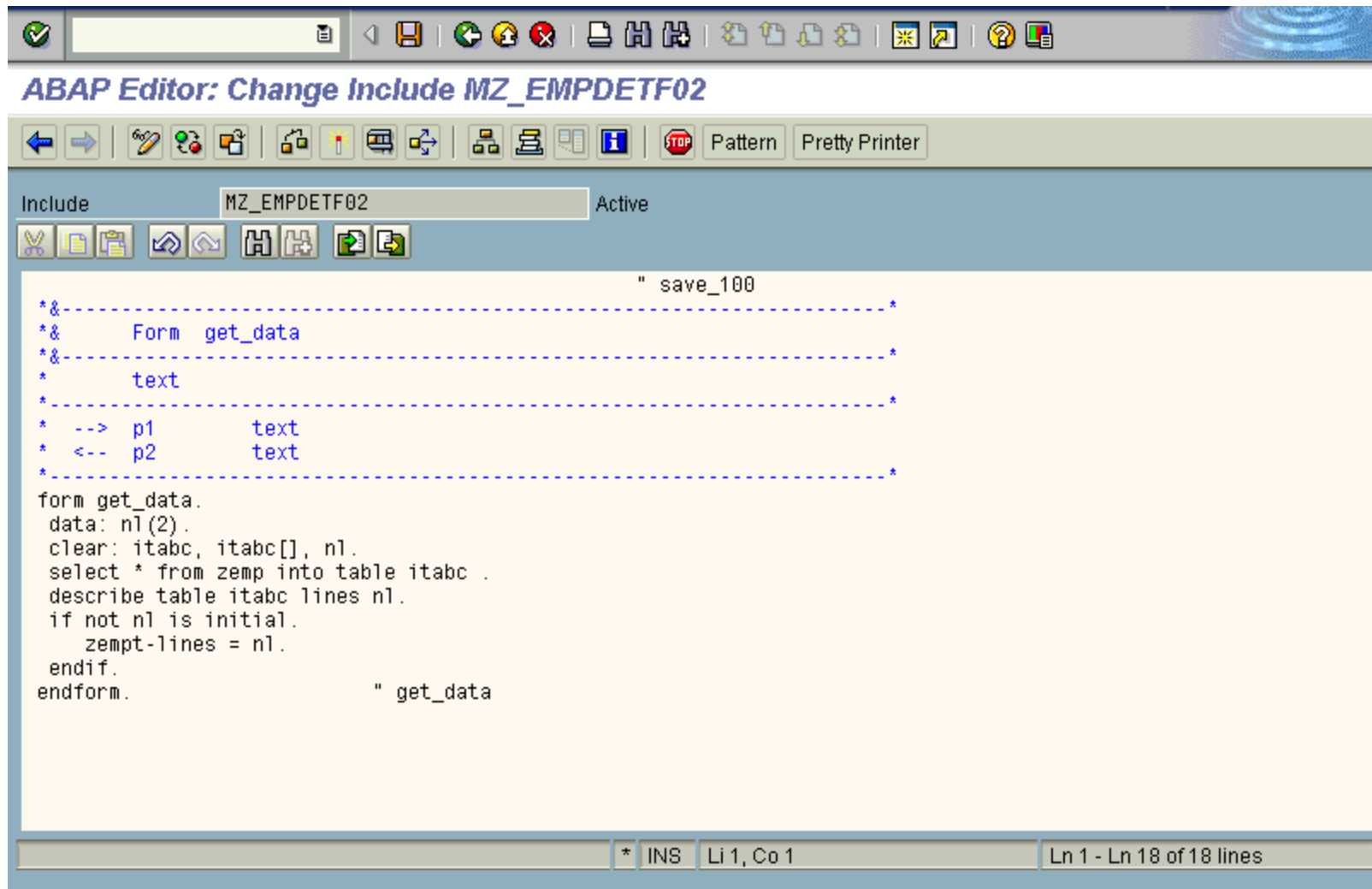
Now we have to write code for the data retrieval and populate the internal table. In the PAI of the screen 100 we are branching it to our table control screen i.e. 300.



```
module user_command_0100 input.  
  move sy-ucomm to pucomm.  
  
  case sy-ucomm.  
    when 'CREA' or 'DISP' or 'CHAN'.  
      if sy-ucomm = 'CHAN' or sy-ucomm = 'DISP'.  
        select single * from zemp where empno = empno.  
      endif.  
      leave to screen 200.  
    when 'TABC'.  
      perform get_data.  
      set screen 300. leave screen.  
  endcase.  
  
endmodule.          " user_command_0100  INPUT  
*&-----*  
*&   Module exit_200  INPUT  
*&-----*  
*   text  
*&-----*  
module exit_200 input.
```

INS Li 51, Co 8 Ln 51 - Ln 73 of 162 lines

## We write code to retrieve Data.



The screenshot shows the ABAP Editor interface. The title bar reads "ABAP Editor: Change Include MZ\_EMPDET02". The editor window displays the following code:

```
                                " save_100
*&-----*
*&   Form get_data
*&-----*
*   text
*&-----*
* --> p1      text
* <-- p2      text
*&-----*
form get_data.
  data: n1(2).
  clear: itabc, itabc[], n1.
  select * from zemp into table itabc .
  describe table itabc lines n1.
  if not n1 is initial.
    zempt-lines = n1.
  endif.
endform.                                " get_data
```

The status bar at the bottom indicates the cursor is at line 1, column 1, and the document contains 18 lines.



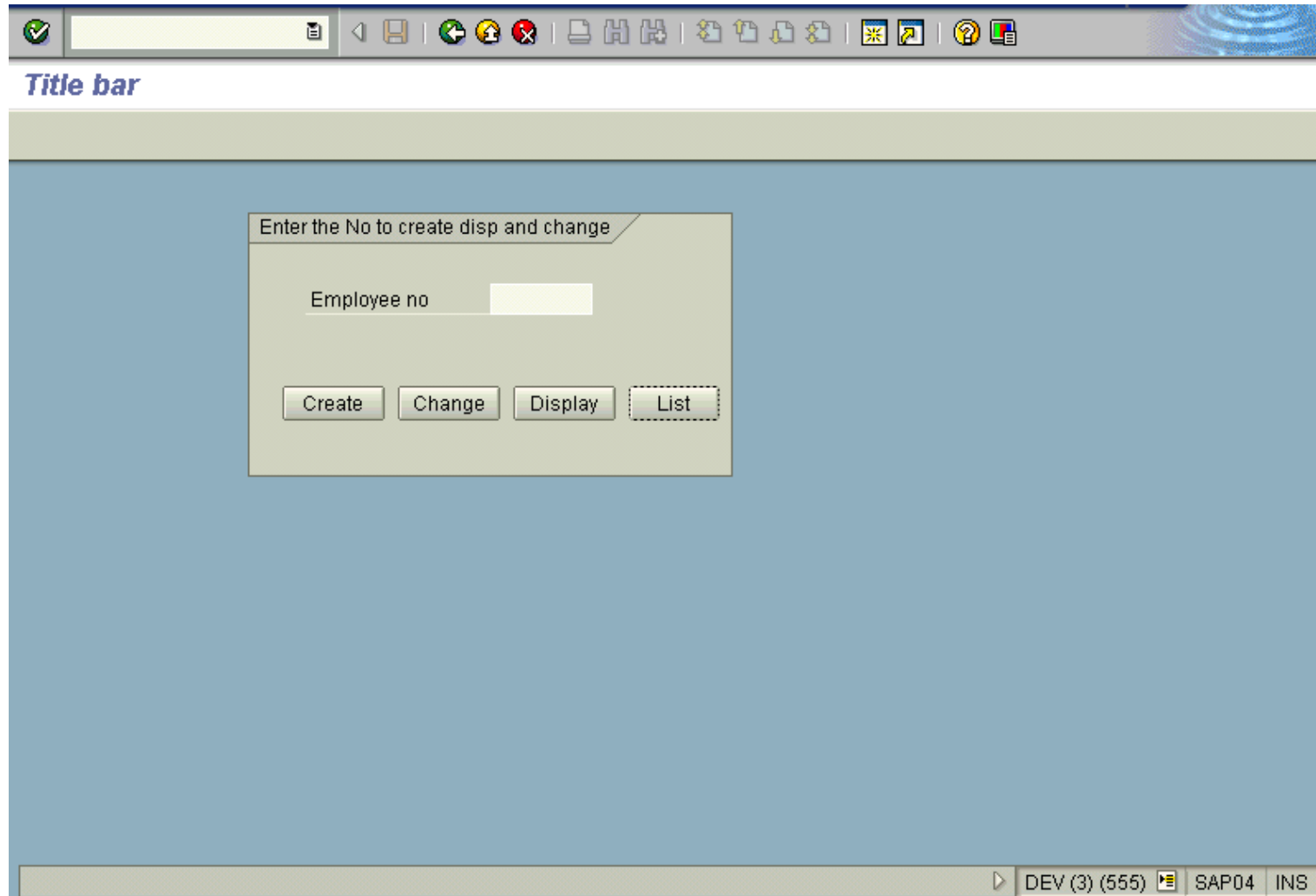
## Now using the PBO of the screen 300 flow logic we populate the table control

The screenshot displays the SAP Screen Painter interface for screen 300. The title bar reads "Screen Painter: Change Screen for SAPMZ\_EMPDET". The "Screen number" is 300, and it is marked as "Active (revised)". The "Flow logic" tab is selected, showing the following code:

```
process before output.  
  
  module status_0300.  
    loop at itabc with control zempt cursor zempt-top_line.  
  
      module pop_control.  
        endloop.  
      *  
process after input.  
  
  module exit_300 at exit-command.  
    loop .  
  
  endloop.  
  
  module user_command_0300.
```

The status bar at the bottom indicates "Pretty Print complete" and shows the current session as "DEV (1) (555) SAP04 OVR".

**In our example when the user clicks the LIST Button in the initial screen, the complete list of employees will be displayed in the second screen.**



Here we get the list of all employees.

*Title bar*

Table Control Example

Emp No	Employee Name	Employee Dept	Employee Addr
1	pravas dhano	SAP ABAP	guindy
2	shibu das	SAP HR	chennai
3	Premnath	SAP ABAP/MM	T nagar
4	Murali	SAP SD	hyderabad
5	rajah ANATHA	ADMIN	trichy
7	nalla vidyadhar reddy	SAP	hyderabad
8	test emp	EMPLOYEEE	chennai
9	PRAVASAAAAAAAAA	TIME PASS	GIUNDY
10	tenth record	SAP	asfdsdf
11	Pranay Sinha	SAP	chennai ( Fr
15	Employee15	SAP	Galaxy Tower
20	sdqsddf	SFGFS	sdfg
10483	Murali Kishanlal kolluru	TIME PASS	chennai club

## **Summary**

**The slides explained the structure of transaction, screen and menu painter and all the main features associated with dialog programming.**