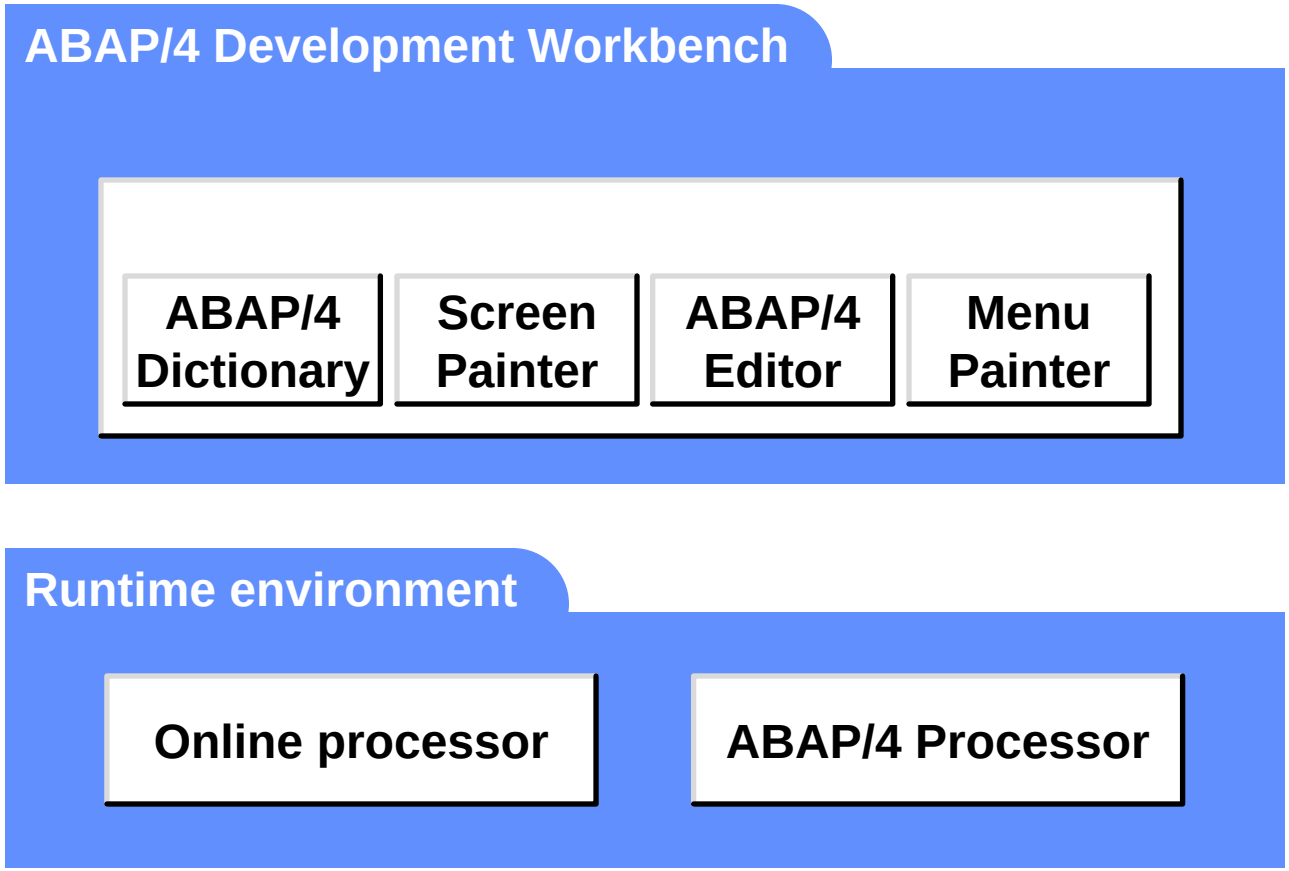
Objective

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**

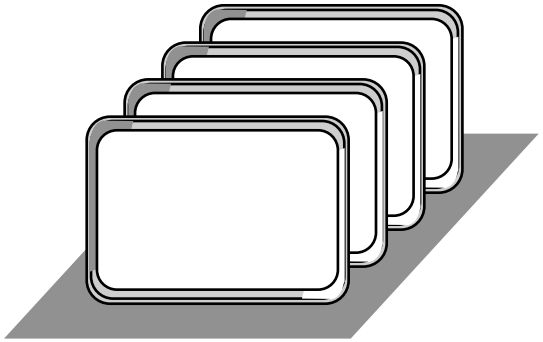
- **A Transaction is a program that conducts a dialog with the User**
- **In a typical dialog, the system displays the screen on which the user can enter or request information.**
- **As per the user input or request, transaction is used to**
  - **Branch to the next Screen**
  - **Display an output**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



Online program

Define screens

Screen Painter

Attributes
Fullscreen
ABAP/4 Dict. fields
Field list
Flow logic

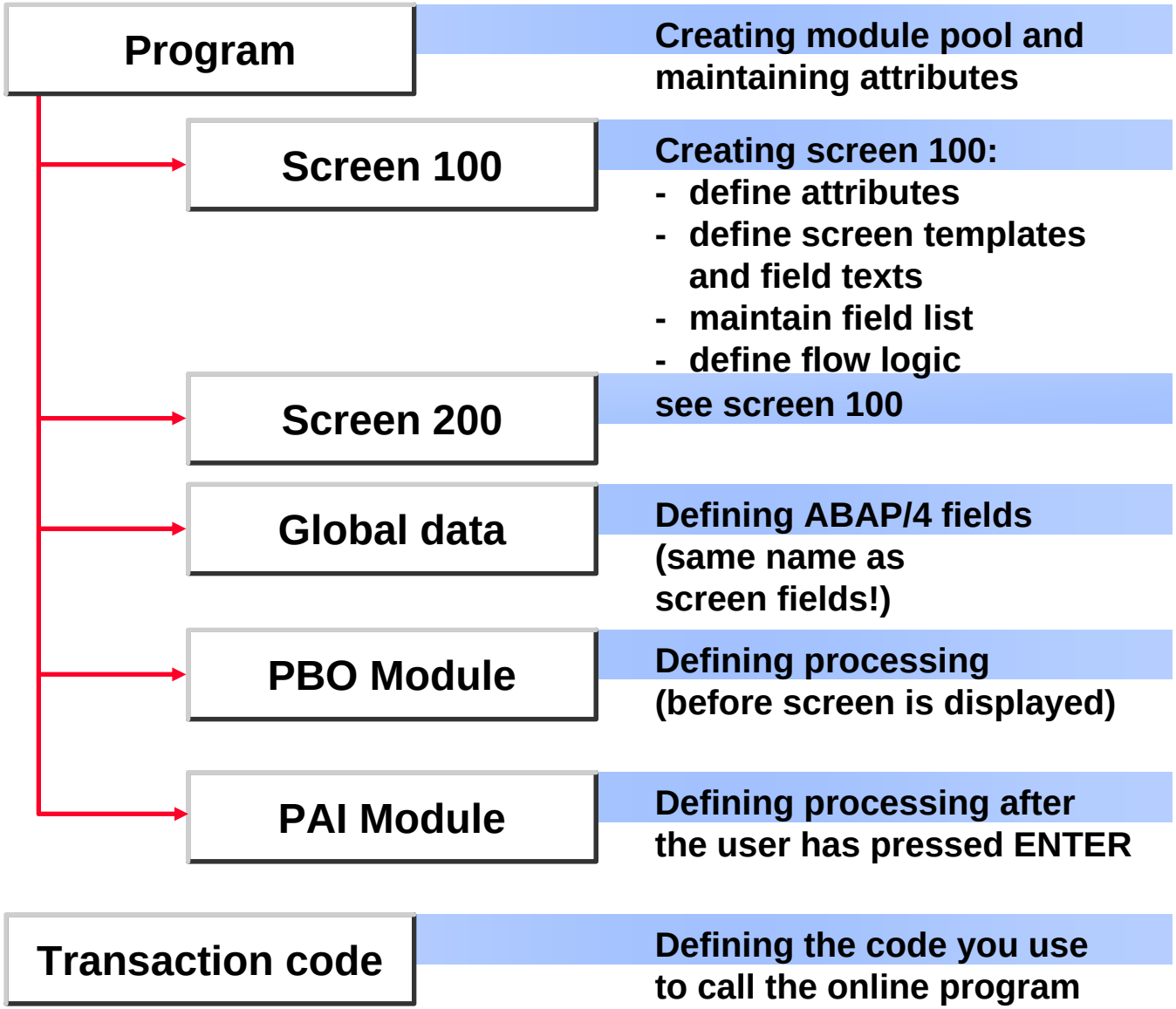
Program module pool

ABAP/4

Global data
PBO modules
PAI modules
Subroutines

Define call

Transaction code


Program
Creating module pool and maintaining attributes
Screen 100
Creating screen 100:
- define attributes
- define screen templates and field texts
- maintain field list
- define flow logic
Screen 200
see screen 100
Global data
Defining ABAP/4 fields (same name as screen fields!)
PBO Module
Defining processing (before screen is displayed)
PAI Module
Defining processing after the user has pressed ENTER
Transaction code
Defining the code you use to call the online program

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.

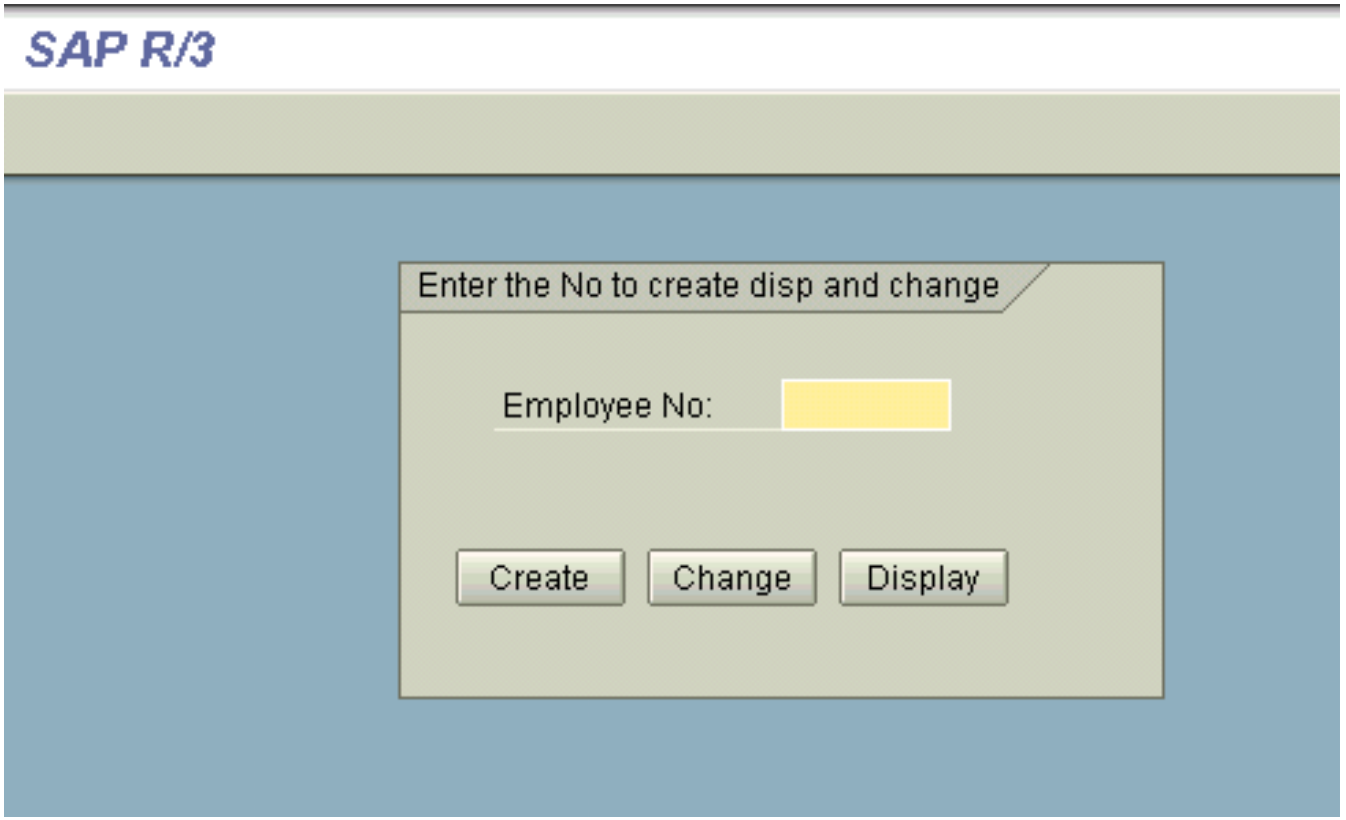
Table Controls: Tabular input/output fields.

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

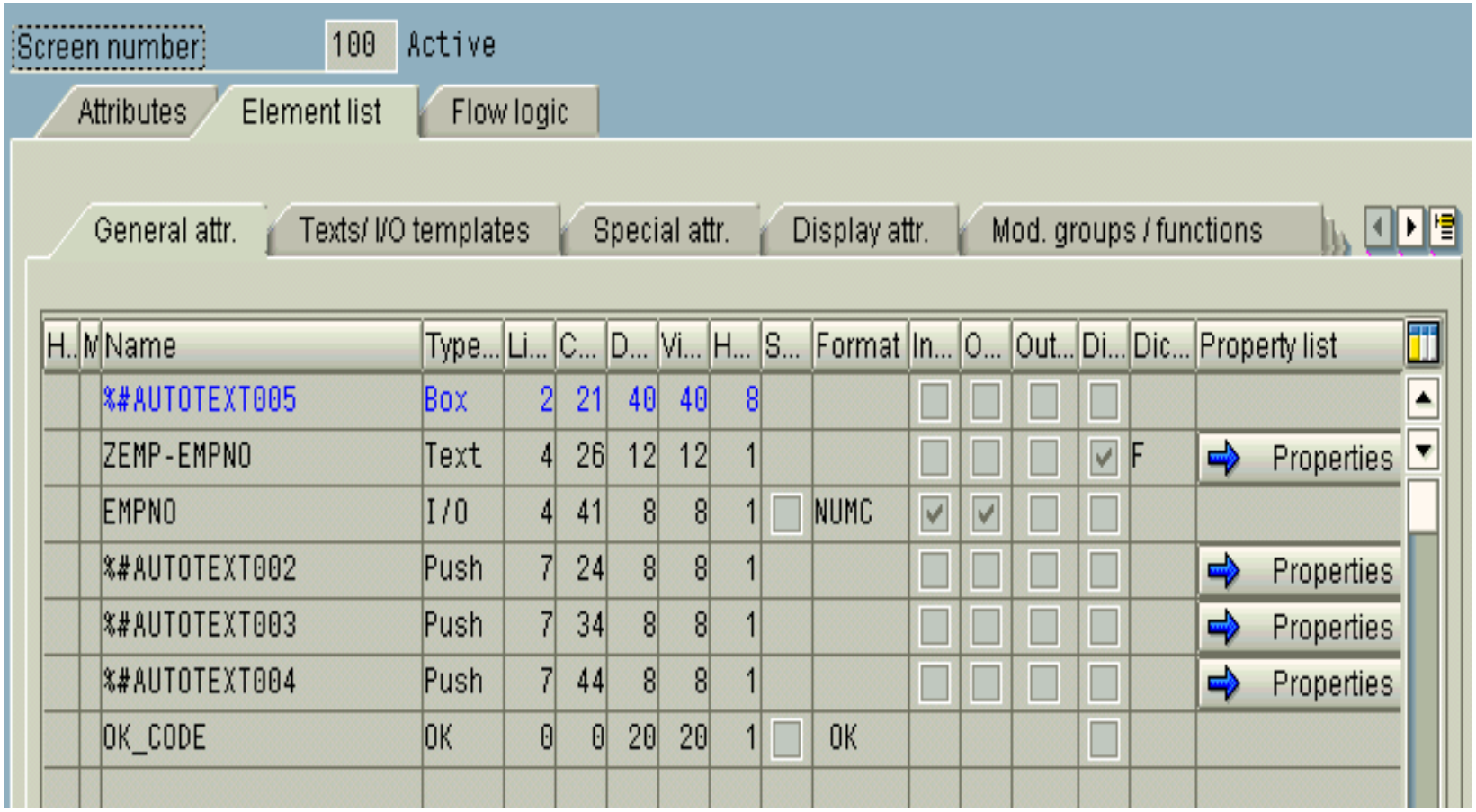
Status Icons: Display elements, indicating the status of the application program.

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.



Cont..


Screen number
100
Active
Attributes
Element list
Flow logic
General attr.
Texts/ I/O templates
Special attr.
Display attr.
Mod. groups / functions
H..
M
Name
Type...
Li...
C...
D...
Vi...
H...
S...
Format
In...
O...
Out...
Di...
Dic...
Property list
%#AUTOTEXT005
Box
2
21
40
40
8
ZEMP-EMPNO
Text
4
26
12
12
1
F
Properties
EMPNO
I/O
4
41
8
8
1
NUMC
%#AUTOTEXT002
Push
7
24
8
8
1
Properties
%#AUTOTEXT003
Push
7
34
8
8
1
Properties
%#AUTOTEXT004
Push
7
44
8
8
1
Properties
OK_CODE
OK
0
0
20
20
1
OK

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.

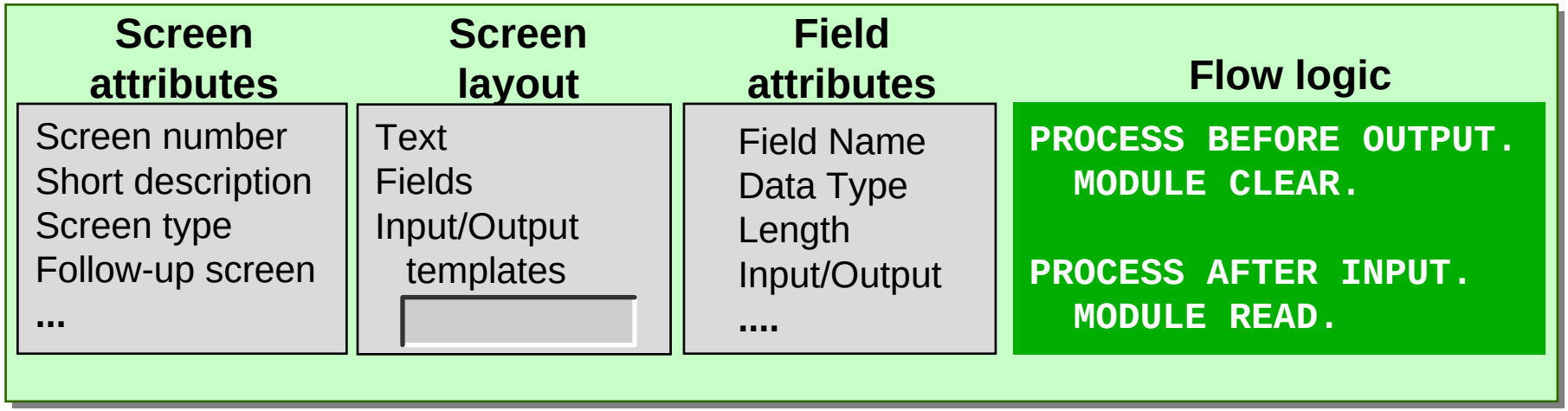
Cont..

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.

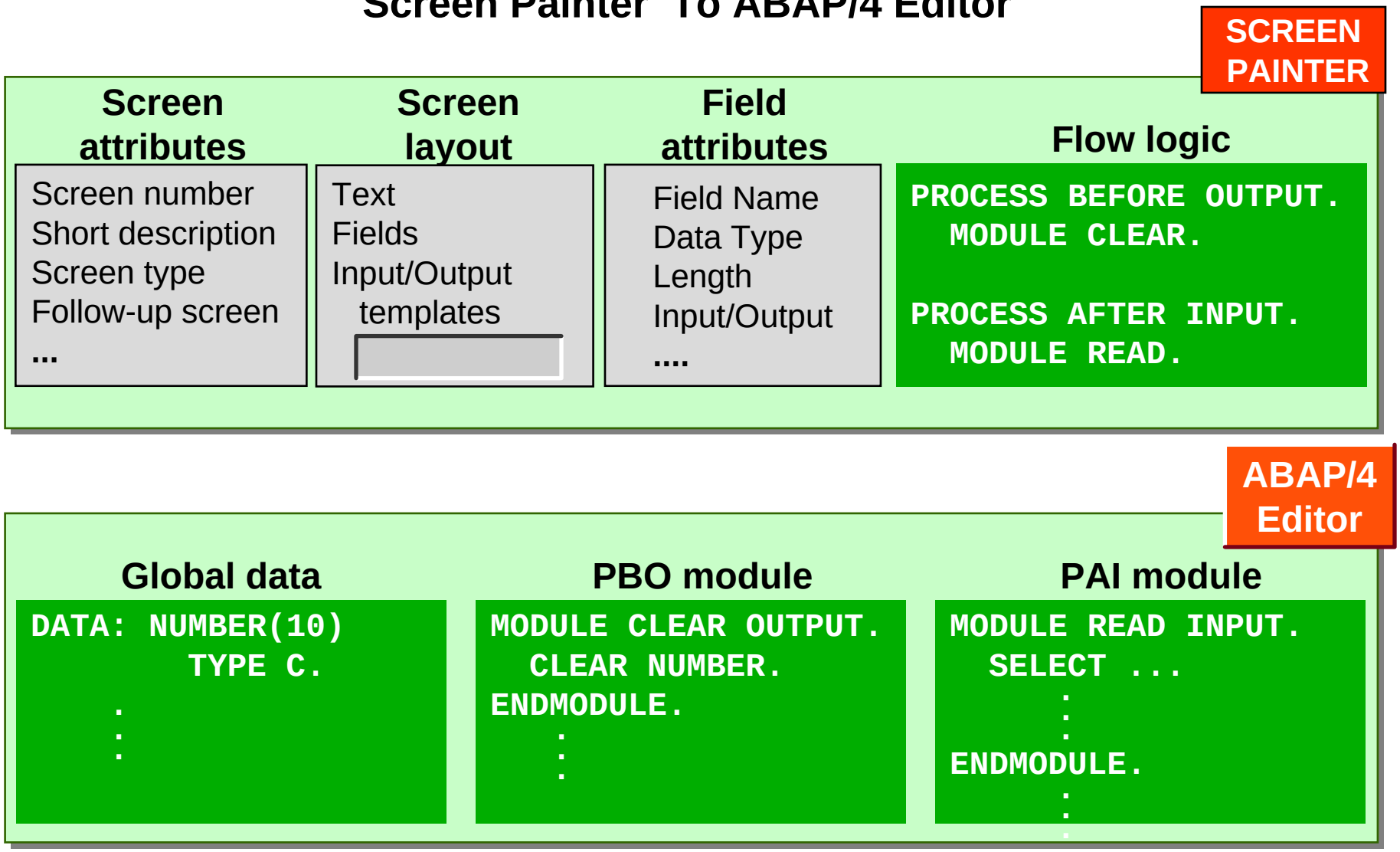
PROCESS AFTER INPUT (PAI) is triggered when the user chooses a function on the screen. You can program the PAI processing of the screen in this block. At the end of the PAI processing, the system either calls the next screen or carries on processing at the point from which the screen was called.

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.

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


Screen attributes
Screen number
Short description
Screen type
Follow-up screen
...
Screen layout
Text
Fields
Input/Output
templates
Field attributes
Field Name
Data Type
Length
Input/Output
....
Flow logic
PROCESS BEFORE OUTPUT.
MODULE CLEAR.
PROCESS AFTER INPUT.
MODULE READ.

Screen Painter To ABAP/4 Editor



To call a module, use the flow logic statement

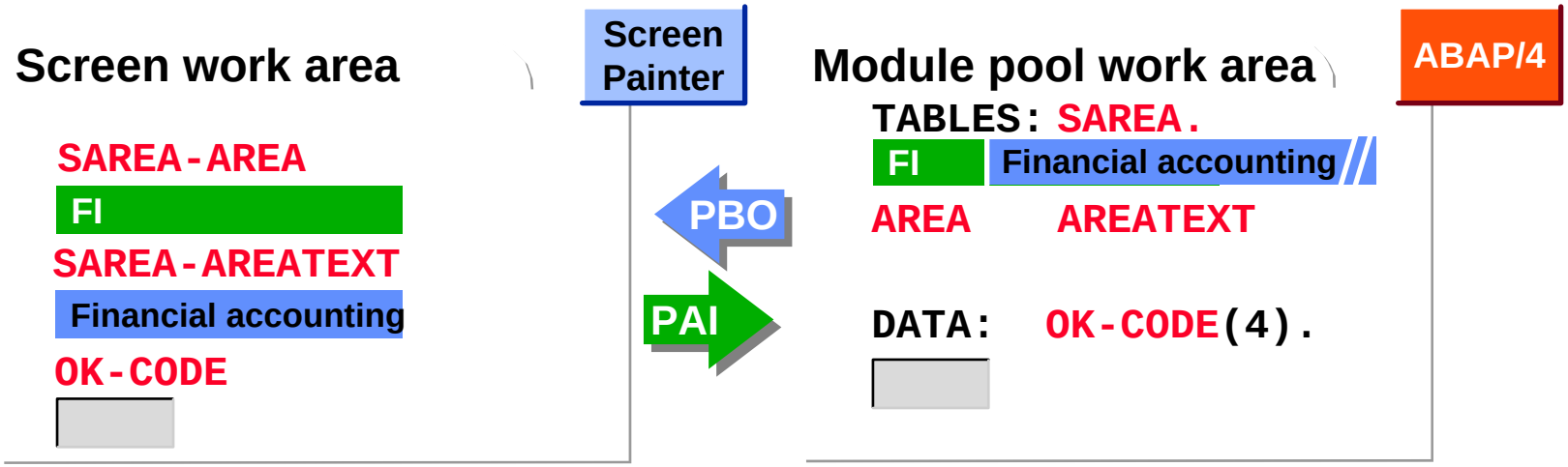
MODULE <mod>.

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.**

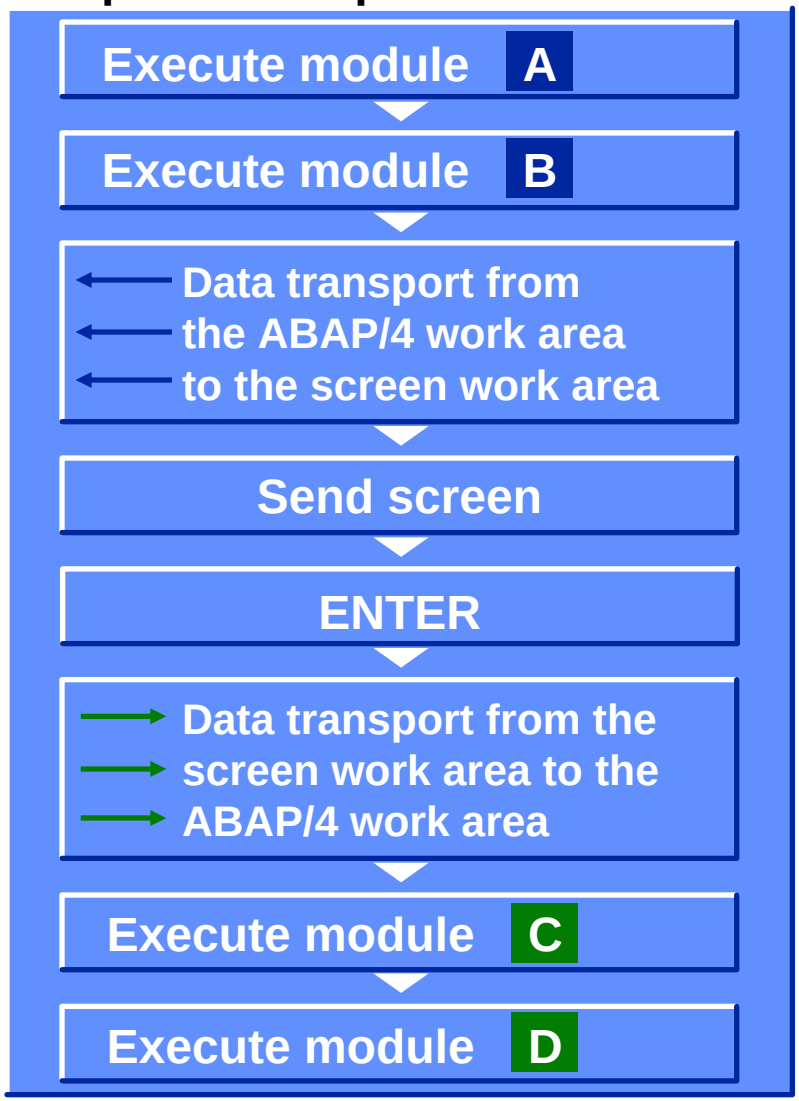
Data Transfer within the Screen and Module Pool work Area .

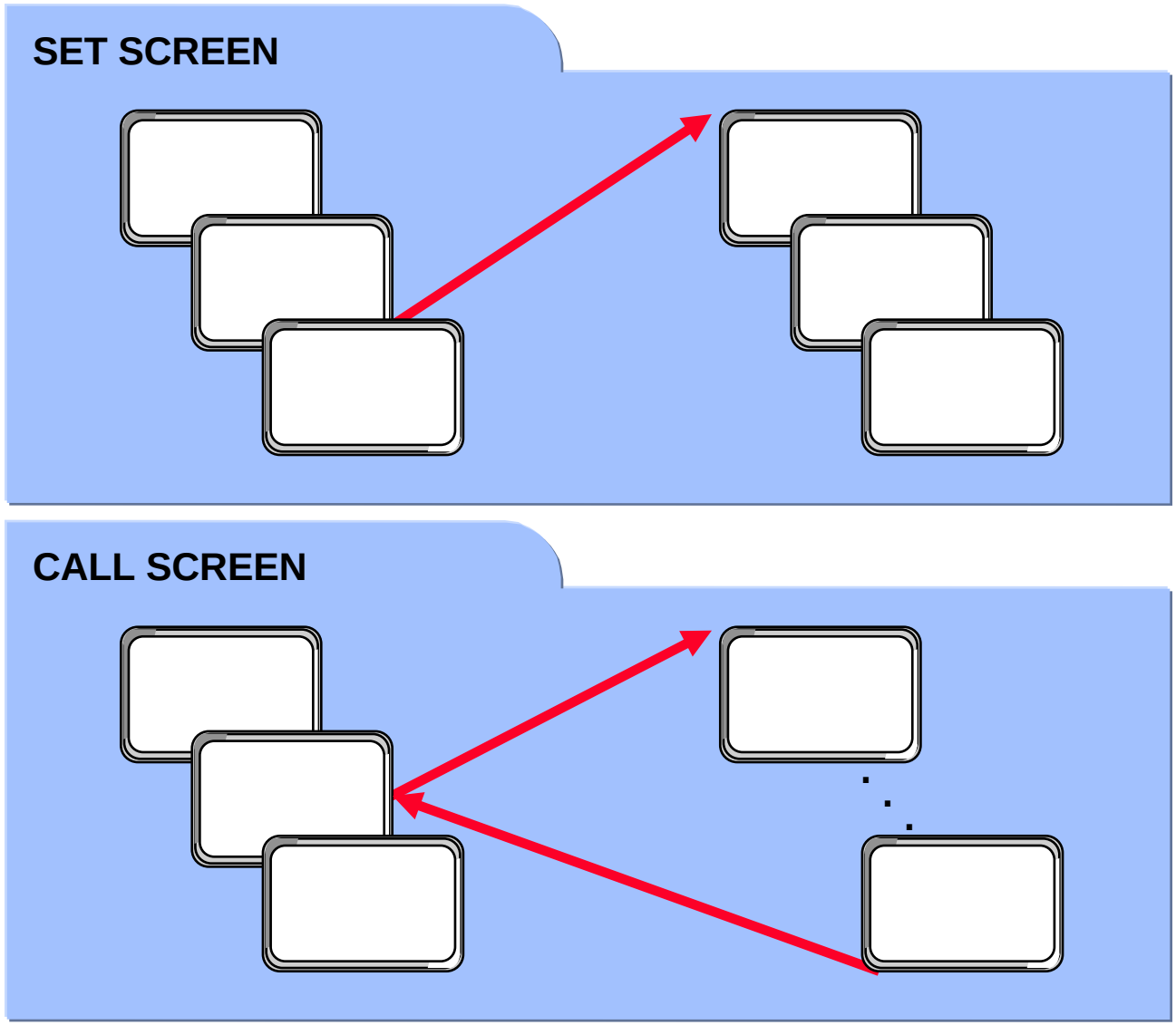


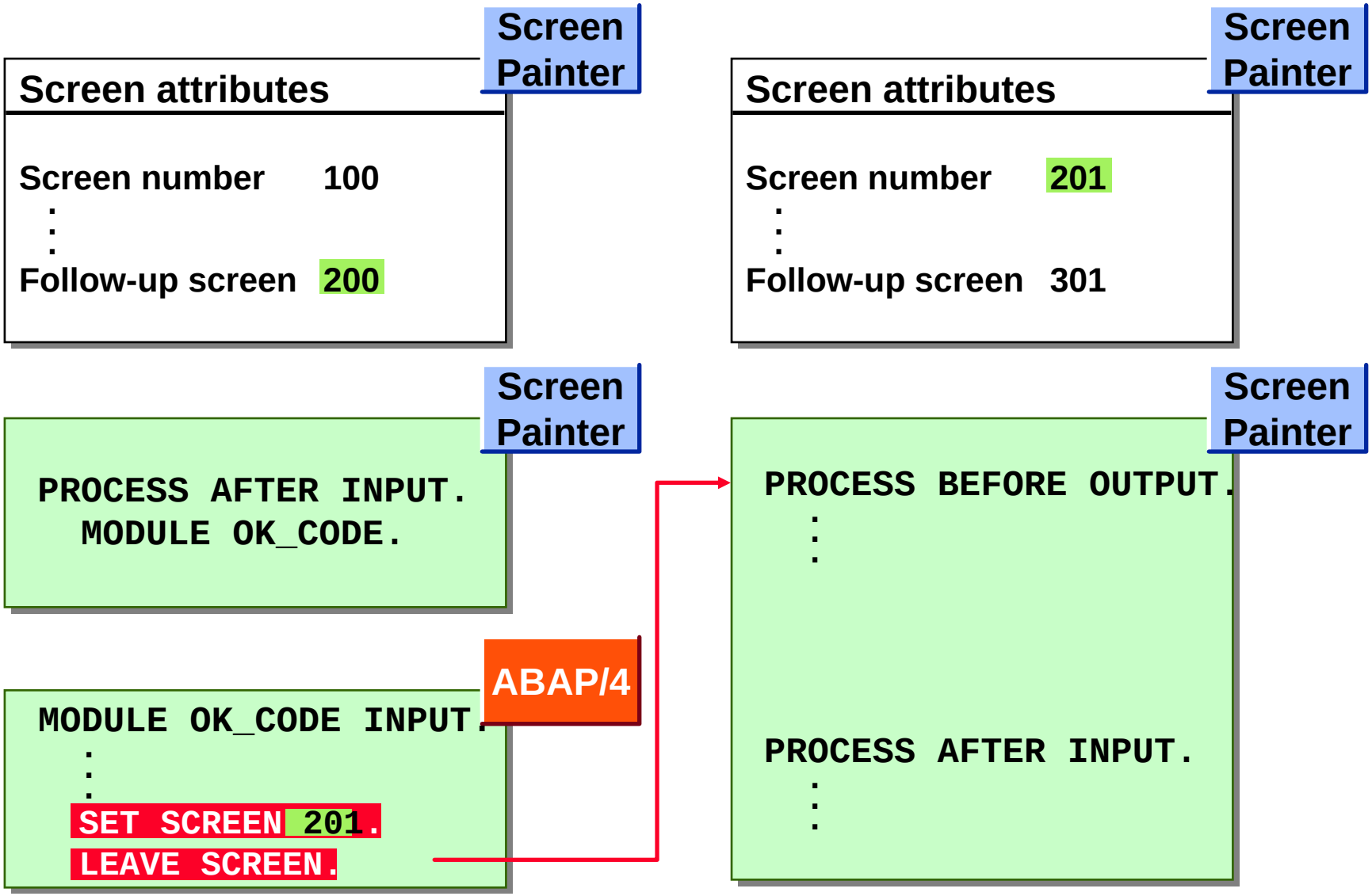
Screen Painter

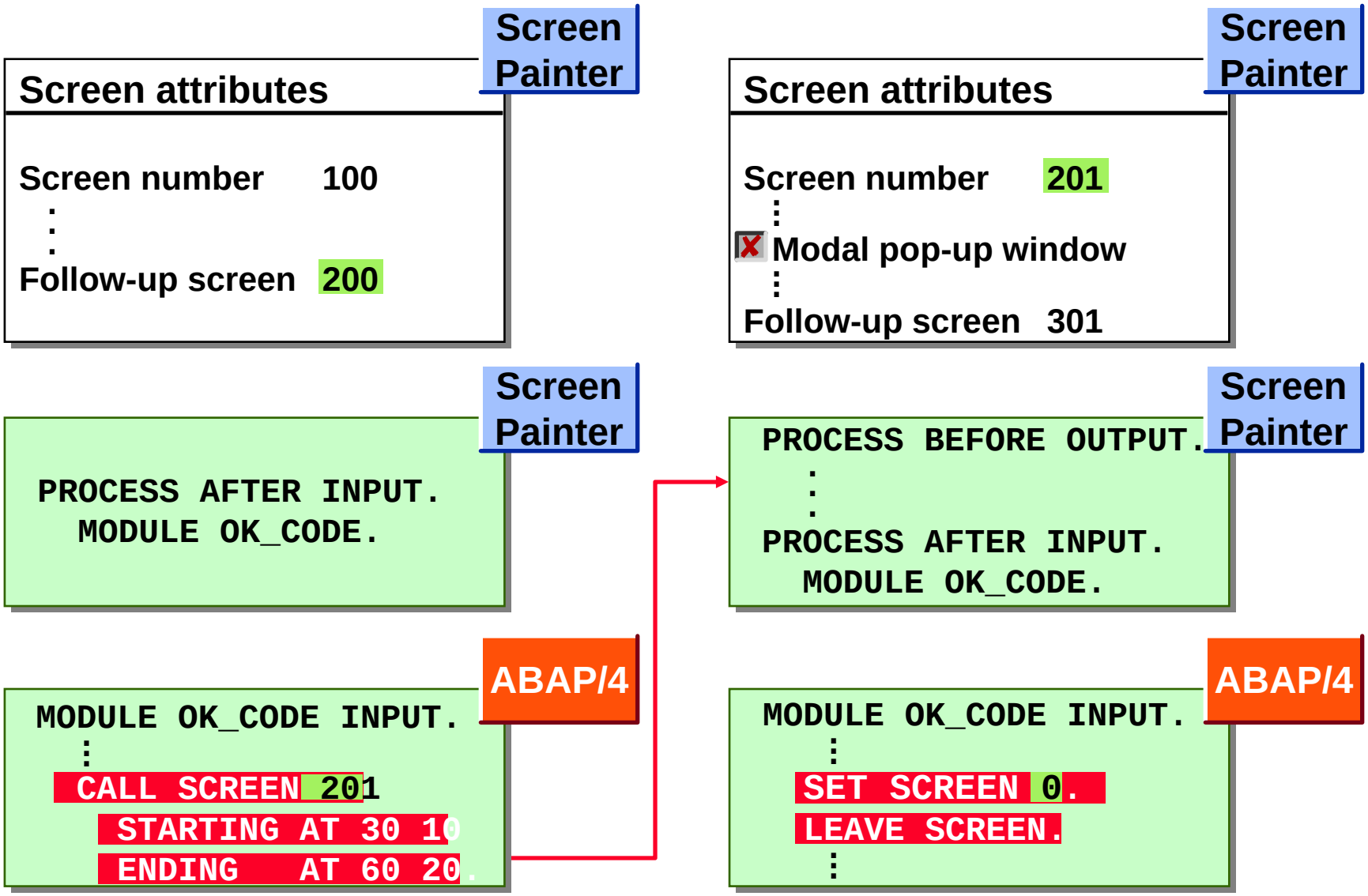
```
PROCESS BEFORE OUTPUT.
  MODULE A.
  MODULE B.

PROCESS AFTER INPUT.
  MODULE C.
  MODULE D.
```

Sequence of operations




SET SCREEN
CALL SCREEN


Screen Painter
Screen attributes
Screen number 100
Follow-up screen 200
Screen Painter
Screen attributes
Screen number 201
Follow-up screen 301
Screen Painter
PROCESS AFTER INPUT.
MODULE OK_CODE.
Screen Painter
PROCESS BEFORE OUTPUT.
ABAP/4
MODULE OK_CODE INPUT.
SET SCREEN 201.
LEAVE SCREEN.
PROCESS AFTER INPUT.


Screen Painter
Screen attributes
Screen number 100
Follow-up screen 200
Screen Painter
Screen attributes
Screen number 201
Modal pop-up window
Follow-up screen 301
Screen Painter
PROCESS AFTER INPUT.
MODULE OK_CODE.
Screen Painter
PROCESS BEFORE OUTPUT.
PROCESS AFTER INPUT.
MODULE OK_CODE.
ABAP/4
MODULE OK_CODE INPUT.
CALL SCREEN 201
STARTING AT 30 10
ENDING AT 60 20.
ABAP/4
MODULE OK_CODE INPUT.
SET SCREEN 0.
LEAVE 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>.

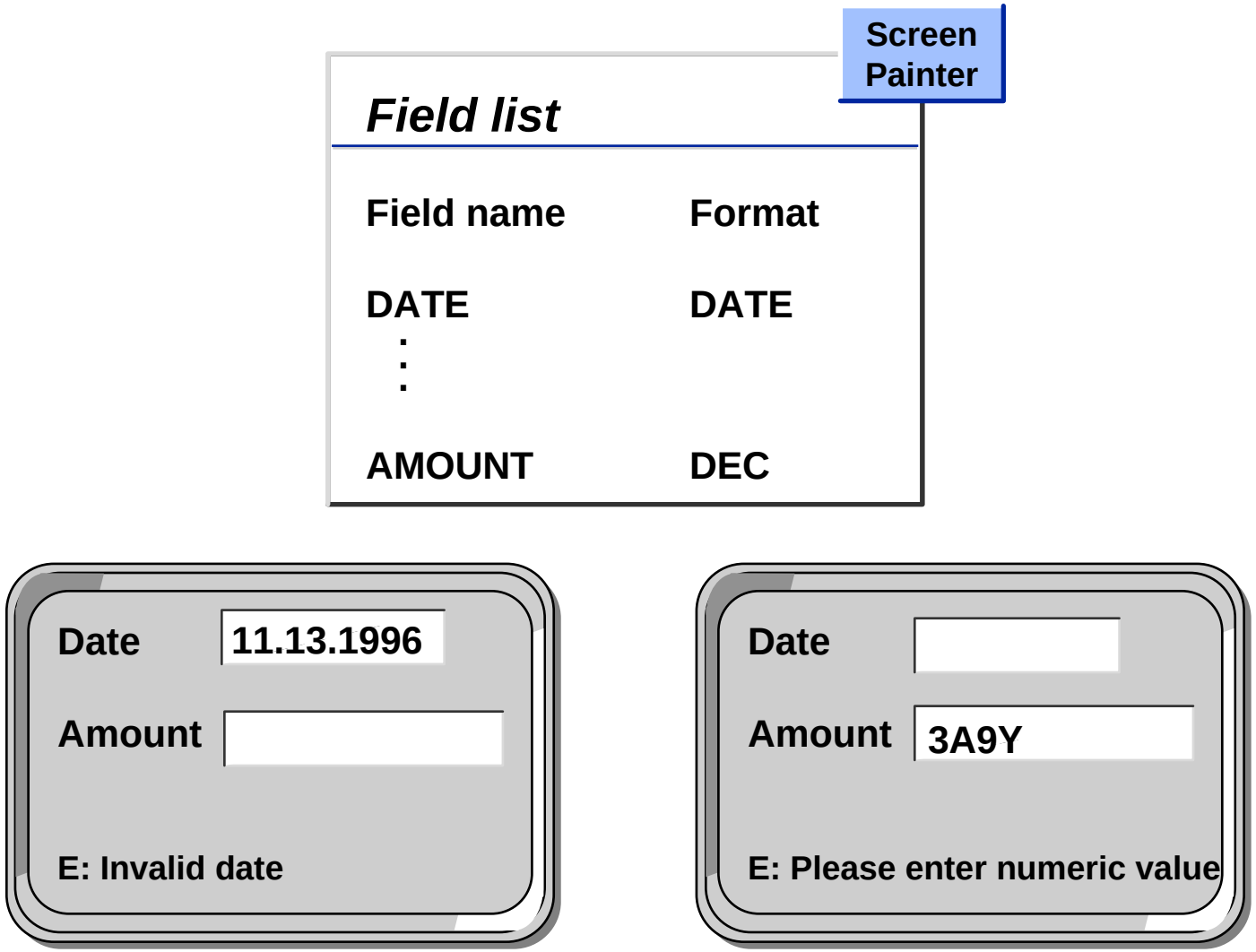
Example:

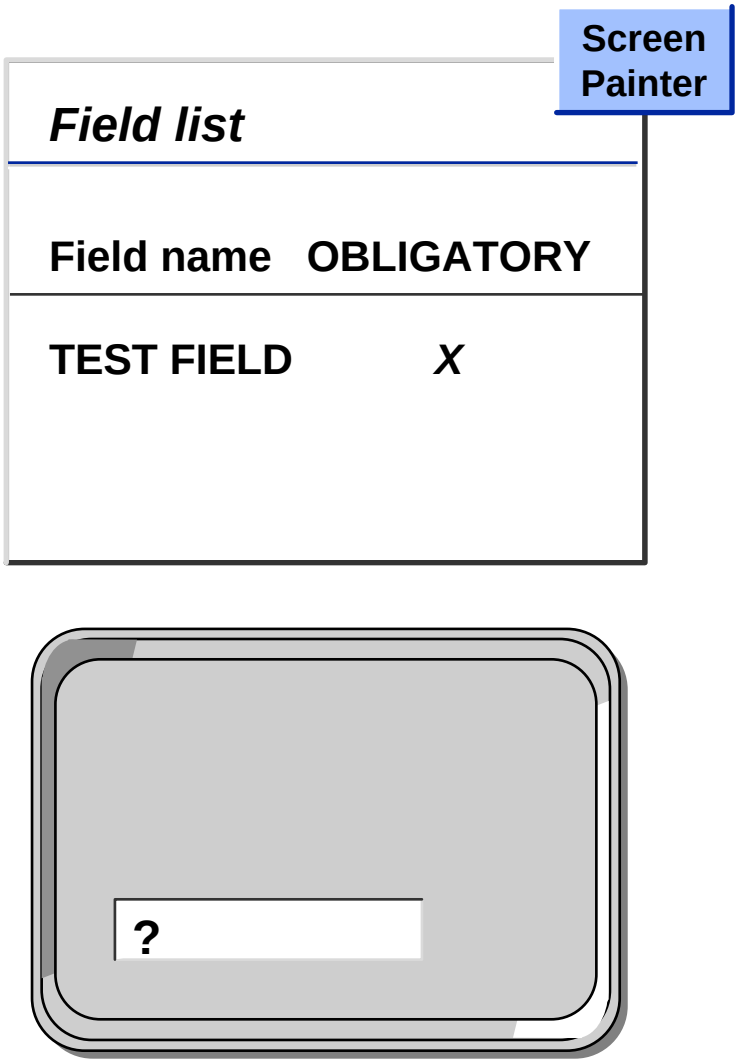
Module Status_100 on input

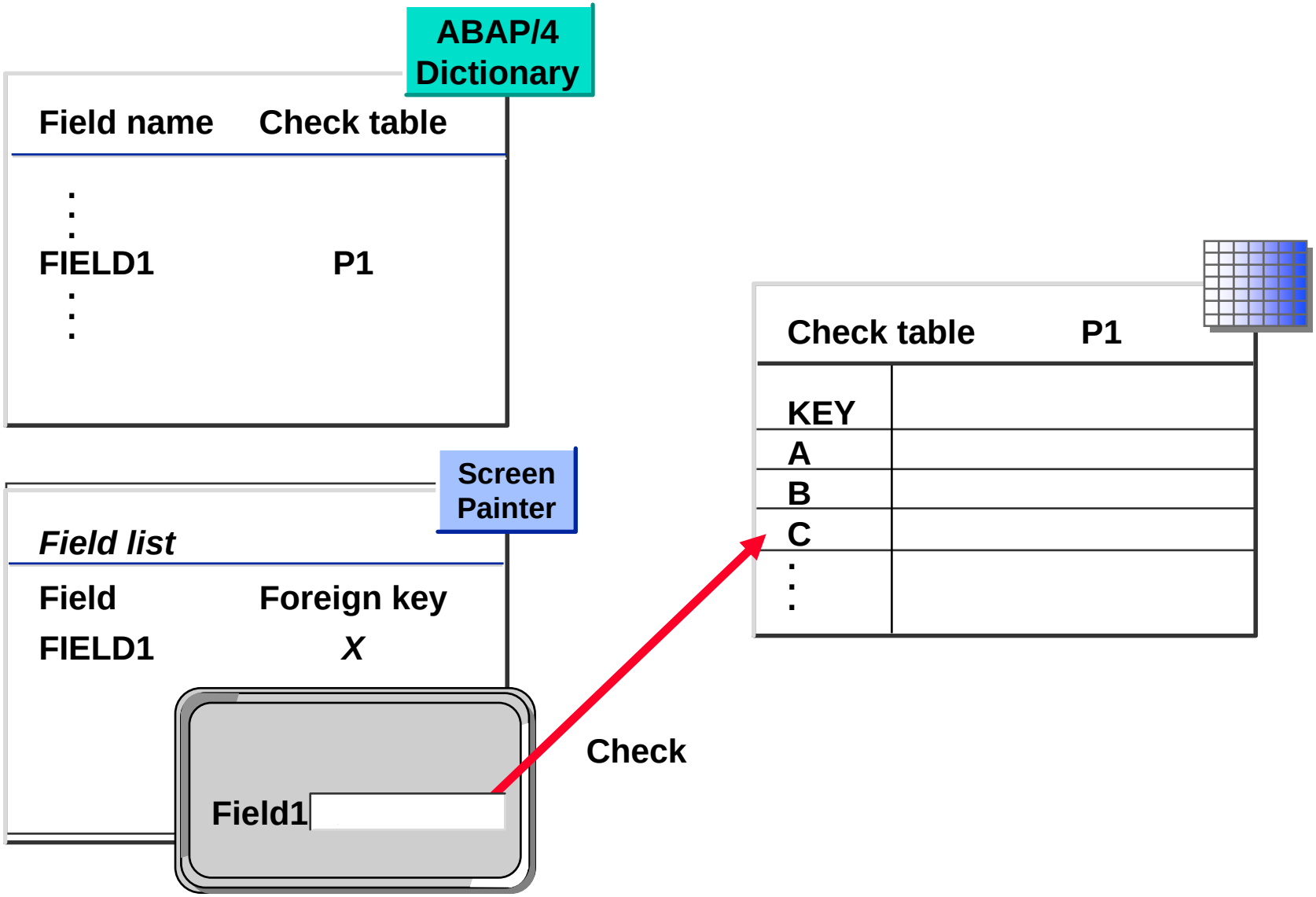
Field format Check :This format limits the kind of input that is valid. For ex. , a DATS field (Date field) is an 8 char string in YYYYMMDD format. All char must be numbers . For the given value entered, the system checks that the day value is valid.

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
11.13.1996
Amount
E: Invalid date
Date
Amount
3A9Y
E: Please enter numeric value


Screen Painter
Field list
Field name OBLIGATORY
TEST FIELD X
?


ABAP/4
Dictionary
Field name
Check table
FIELD1
P1
Check table
P1
KEY
A
B
C
Screen
Painter
Field list
Field
Foreign key
FIELD1
X
Field1
Check

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

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

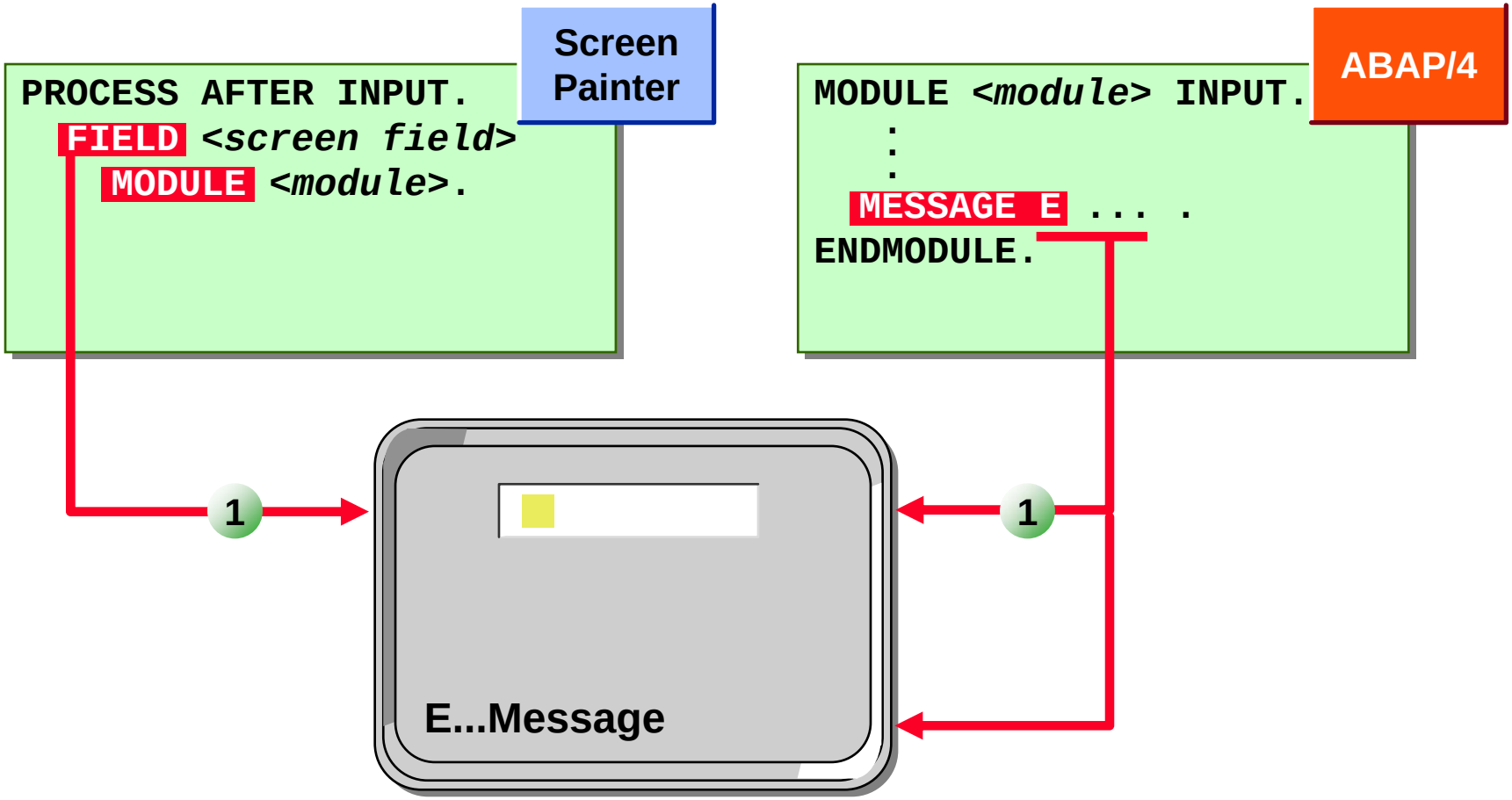
Screen Painter

```
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').
```

The FIELD..MODULEstatement checks the validity for a particular screen field.

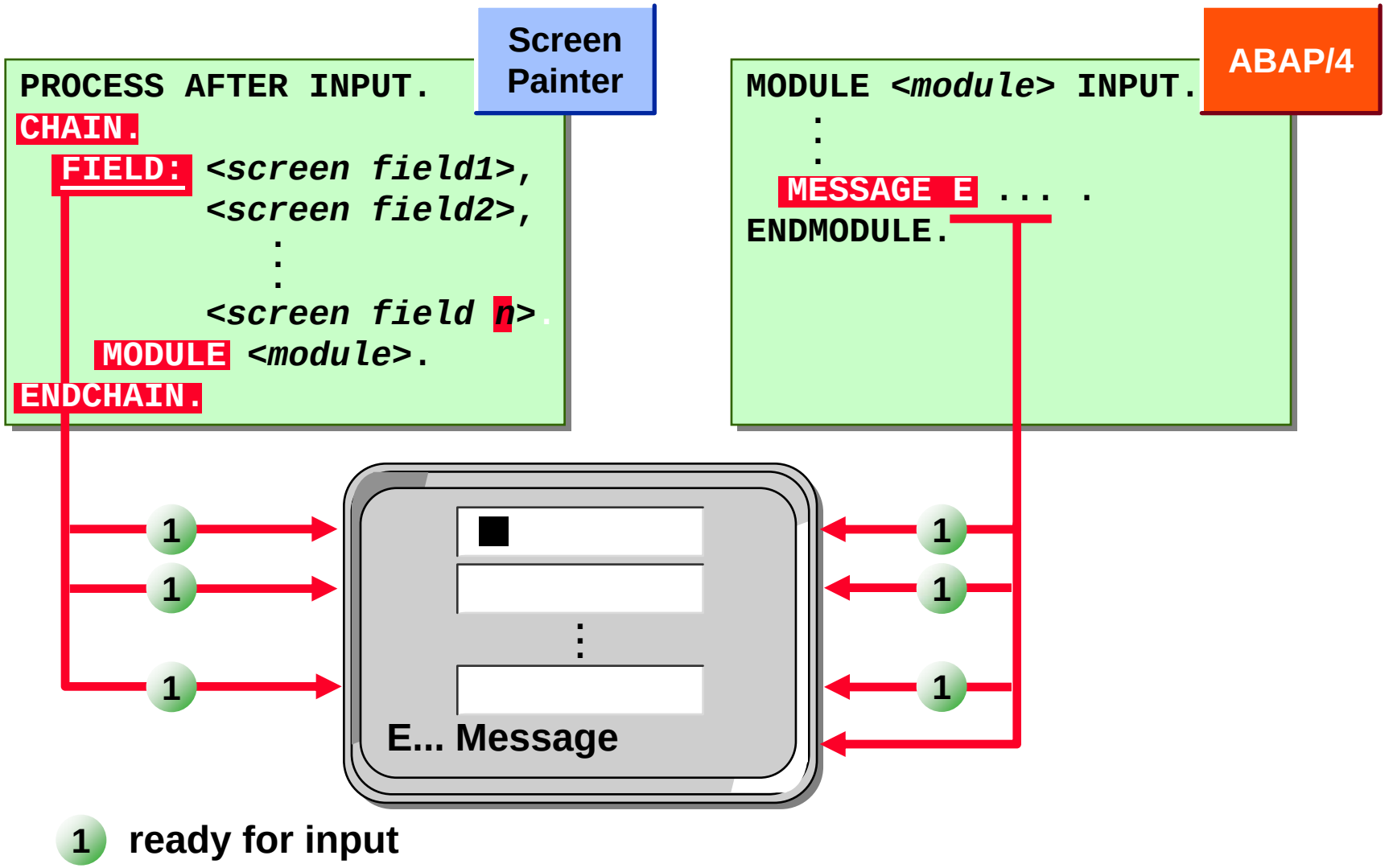


Screen Painter

```
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.
```


Screen Painter
ABAP/4
PROCESS AFTER INPUT.
CHAIN.
FIELD: <screen field1>,
<screen field2>,
:
<screen field n>.
MODULE <module>.
ENDCHAIN.
MODULE <module> INPUT.
:
MESSAGE E ... .
ENDMODULE.
1
1
1
1
1
1
E... Message
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

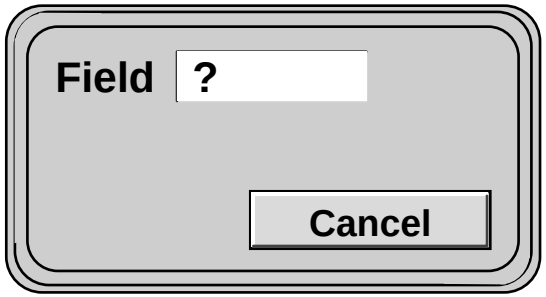
```
PROCESS AFTER INPUT.
  CHAIN.
    FIELD: <screen field 1>,
           <screen field 2>,
              :
              :
           <screen field n>.
    MODULE <module> ON CHAIN-INPUT.
  ENDCHAIN.
    :
    :
```

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.
    :
    :
```

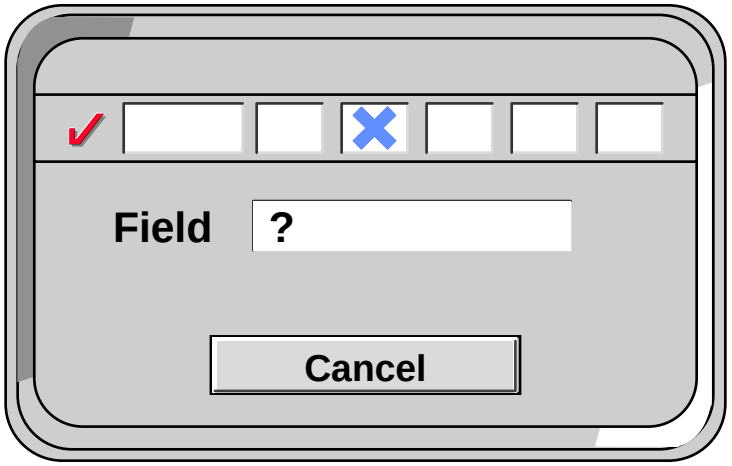


Screen Painter

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

ABAP/4

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



Menu Painter

Function list

Function	Type
ABBR	E

Screen Painter

List of modification groups

Field name	Fcode	Type
ABEND	ABBR	E

- 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

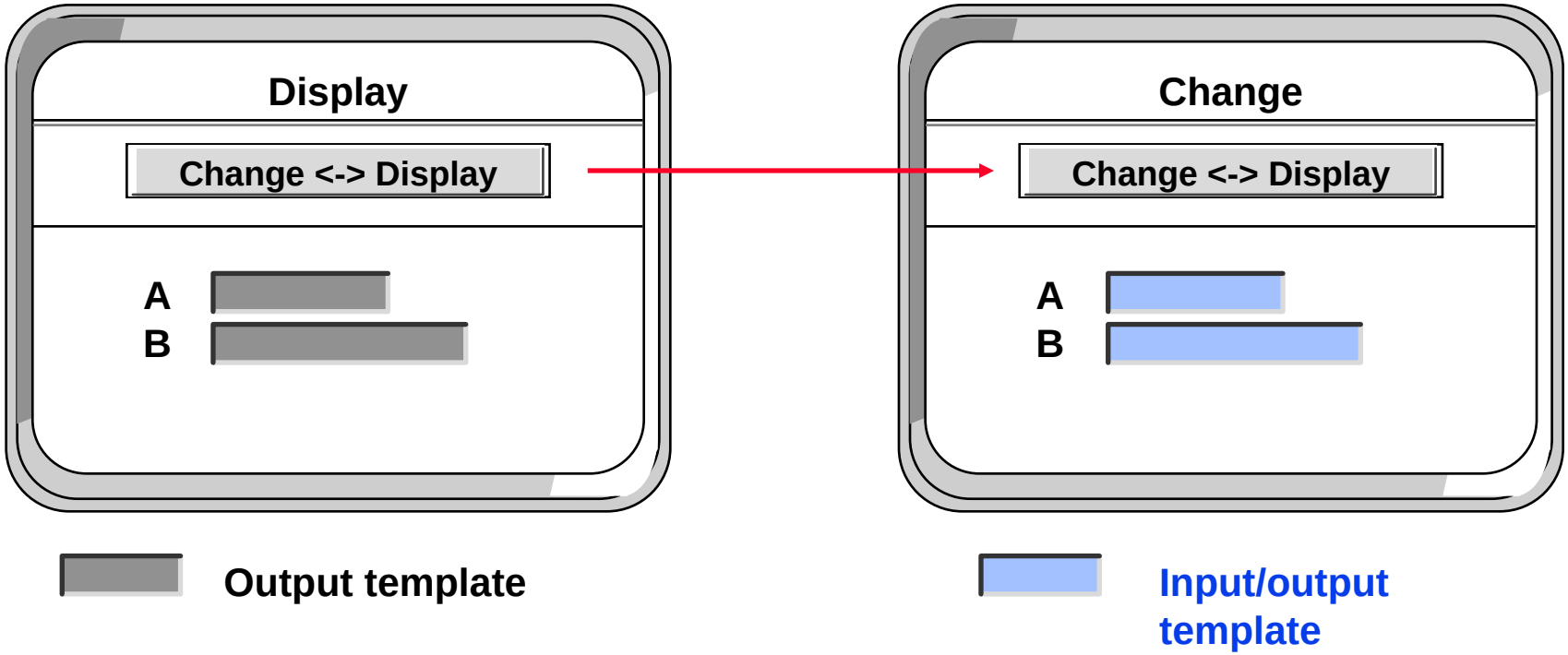
Screen Painter

```
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>..<value 4>.
   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-PROCESSIN

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

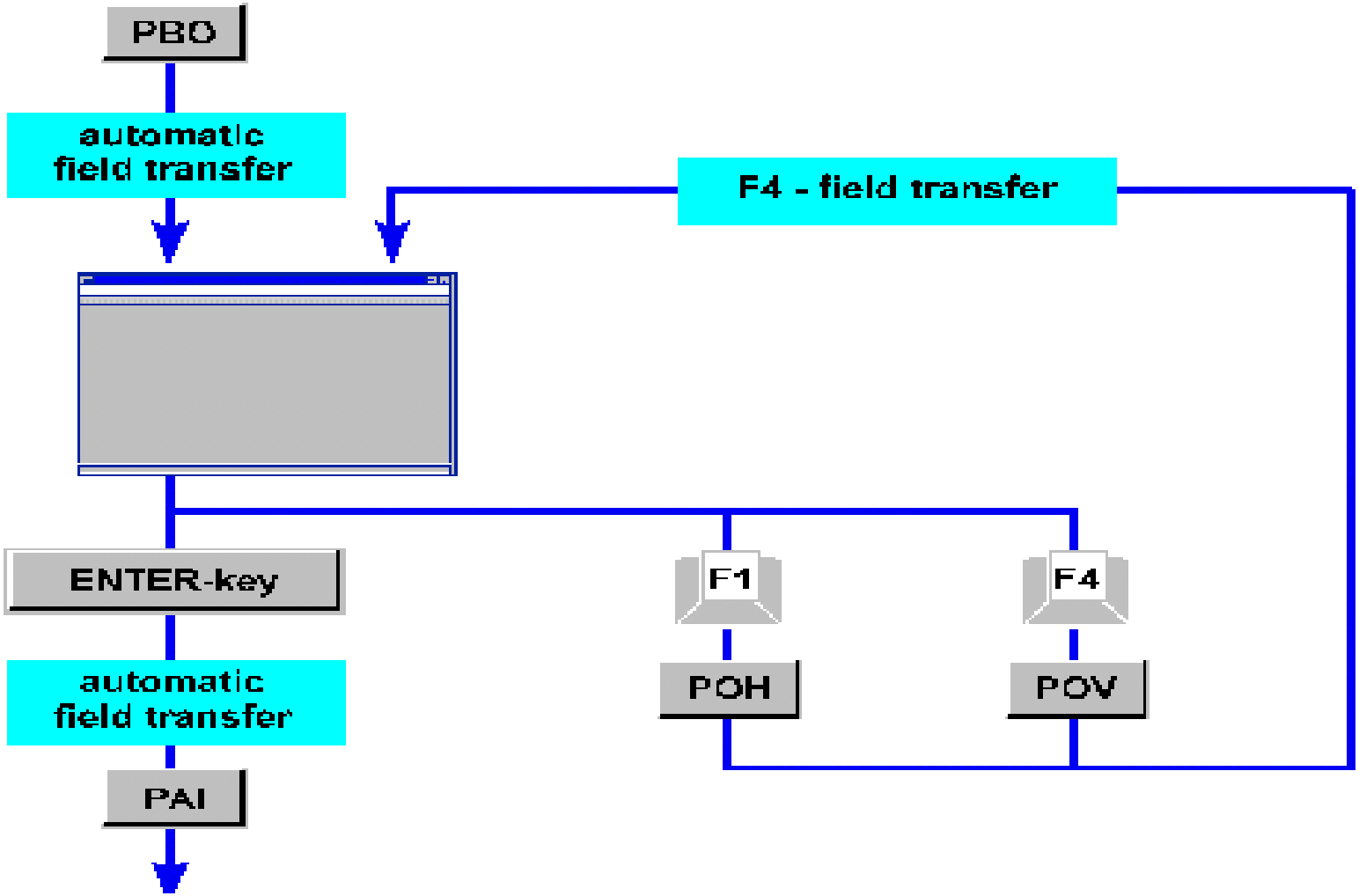
```
LEAVE TO LIST-PROCESSING AND RETURN TO   SCREEN 100.
```

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

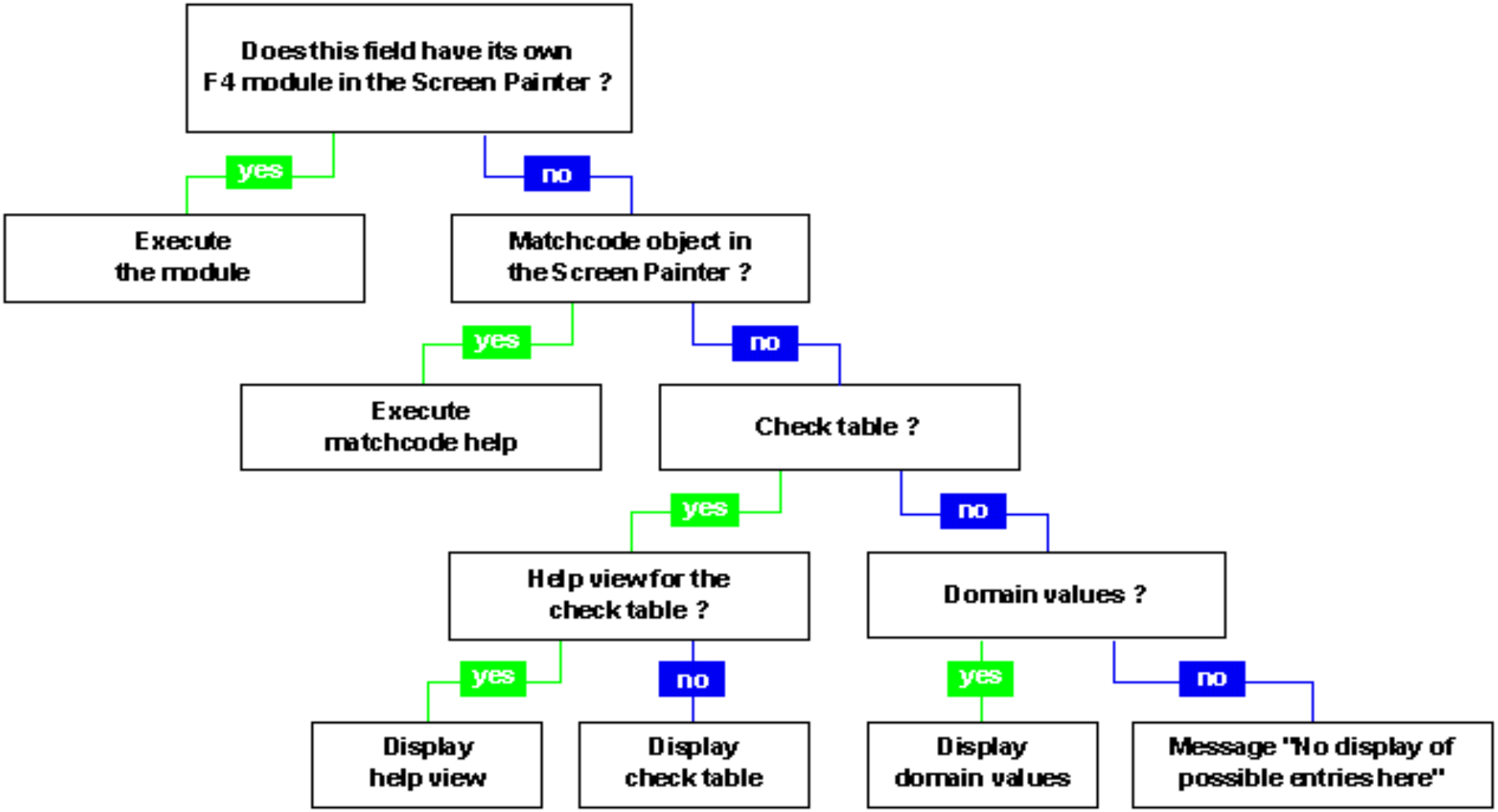
Syntax

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


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

Programming Field- and Value-Help



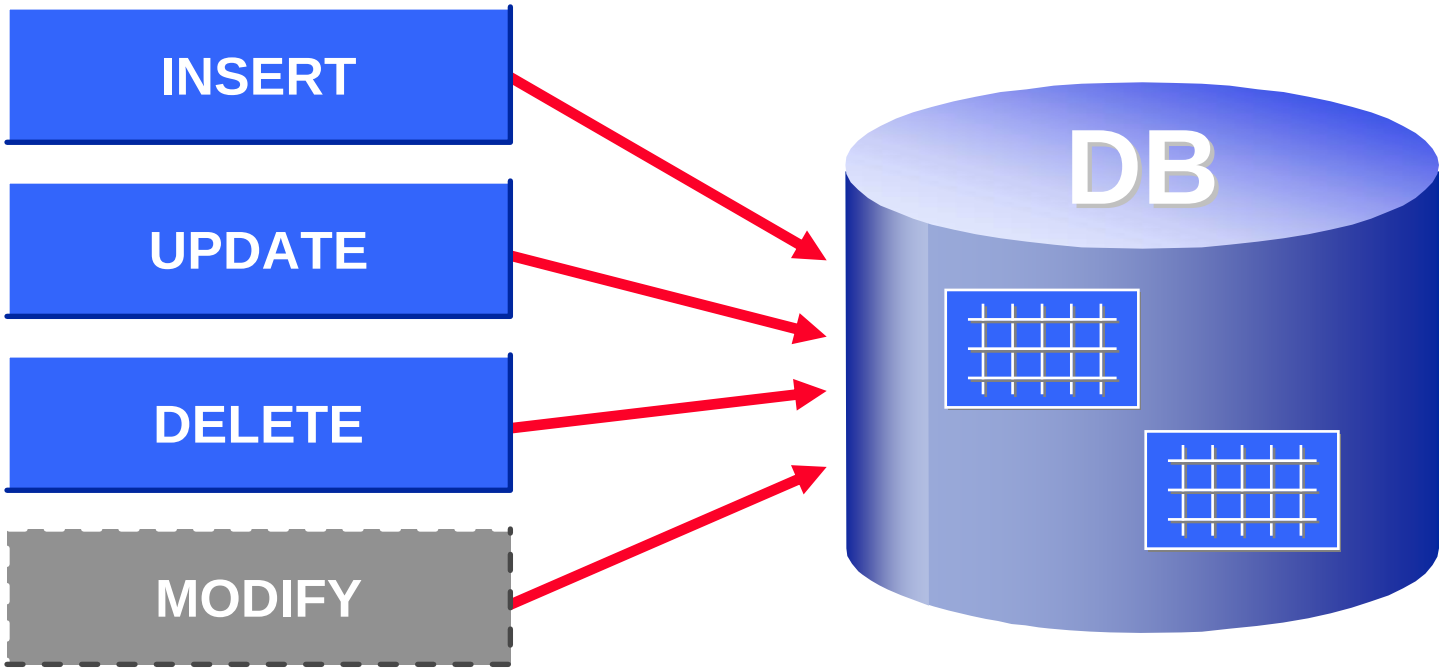
- **Matchcode help**
- **Check tables**
- **Help views**
- **Domain values**


Does this field have its own F4 module in the Screen Painter ?
yes
no
Execute the module
Matchcode object in the Screen Painter ?
yes
no
Execute matchcode help
Check table ?
yes
no
Help view for the check table ?
Domain values ?
yes
no
yes
no
Display help view
Display check table
Display domain values
Message "No display of possible entries here"

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
UPDATE
DELETE
MODIFY
DB

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:

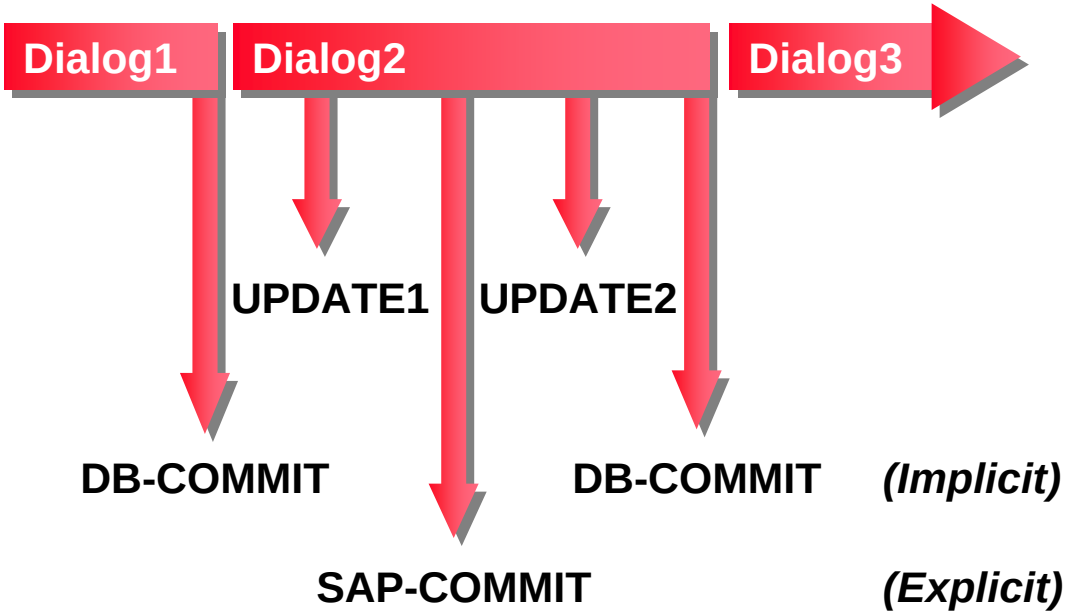
```
INSERT  <table< FROM <itab>.
UPDATE <table< FROM <itab>.
DELETE  <table< FROM <itab>.
```

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.

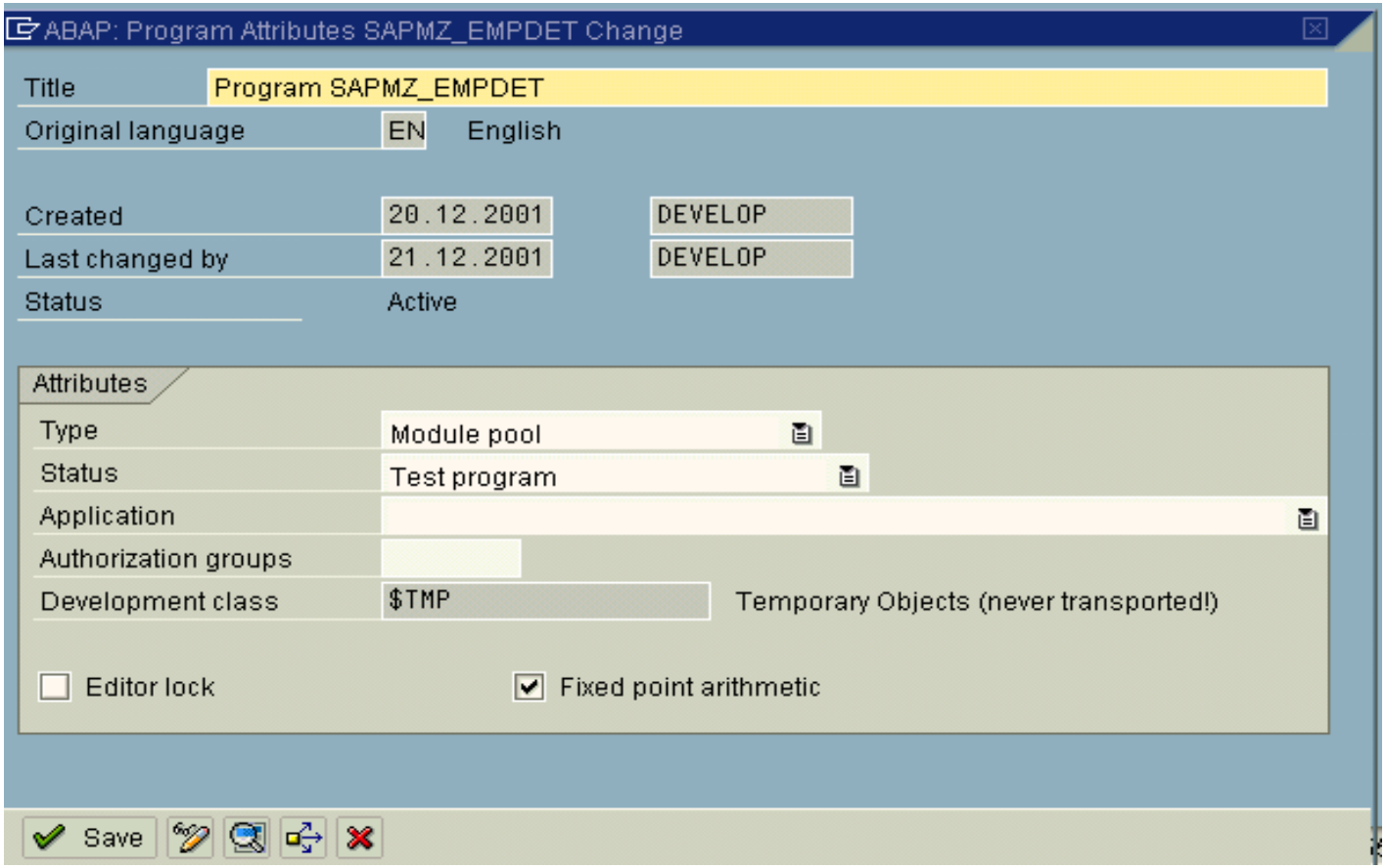
The CALL FUNCTION IN BACKGROUND TASK statement logs a function module to run in a background task. Normally, this statement is used to execute functions on remote hosts (by specifying an additional DESTINATION parameter).

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

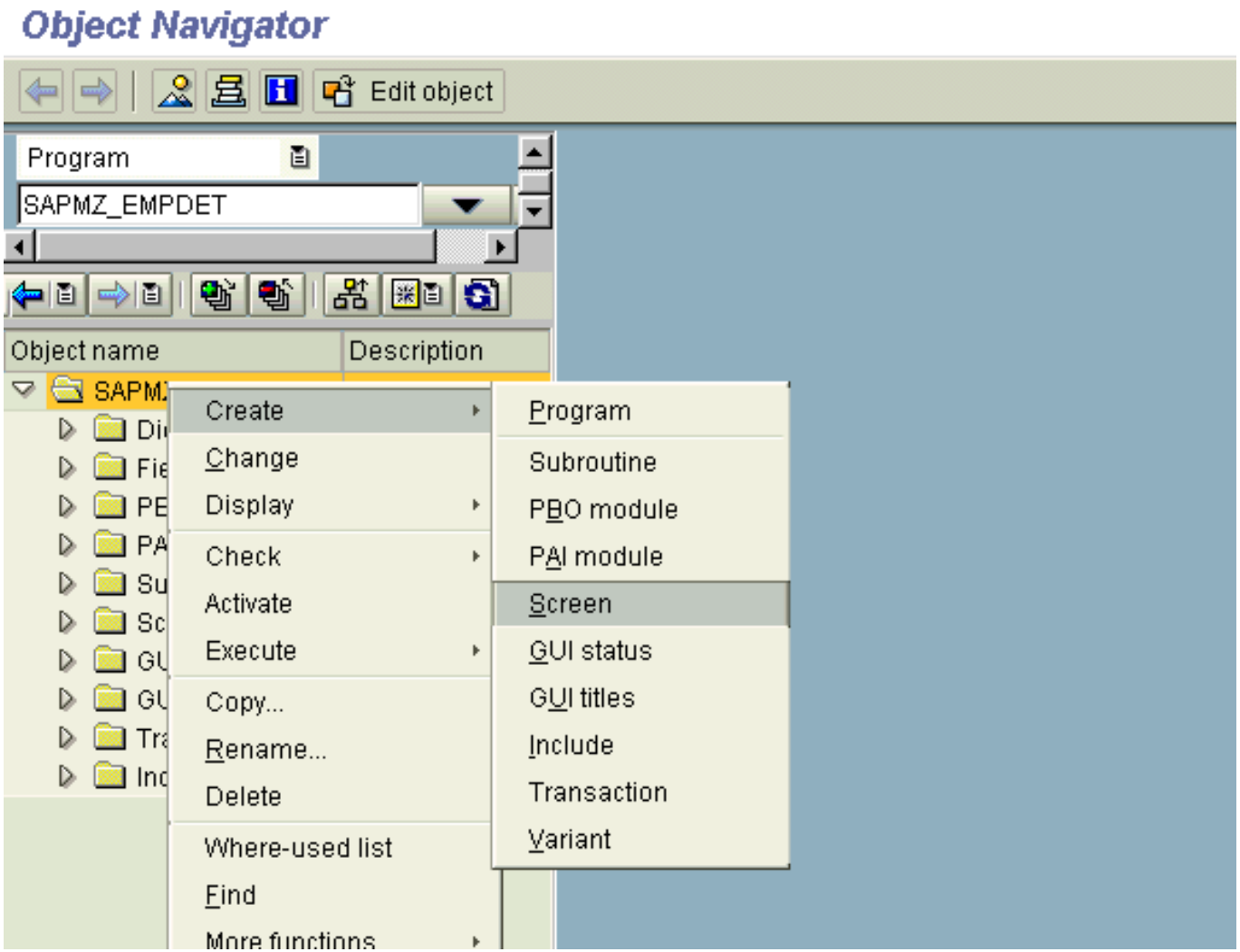
In this Exercise we see how to write a simple Transaction

First create a program with naming convention SAPMZ<initials>.

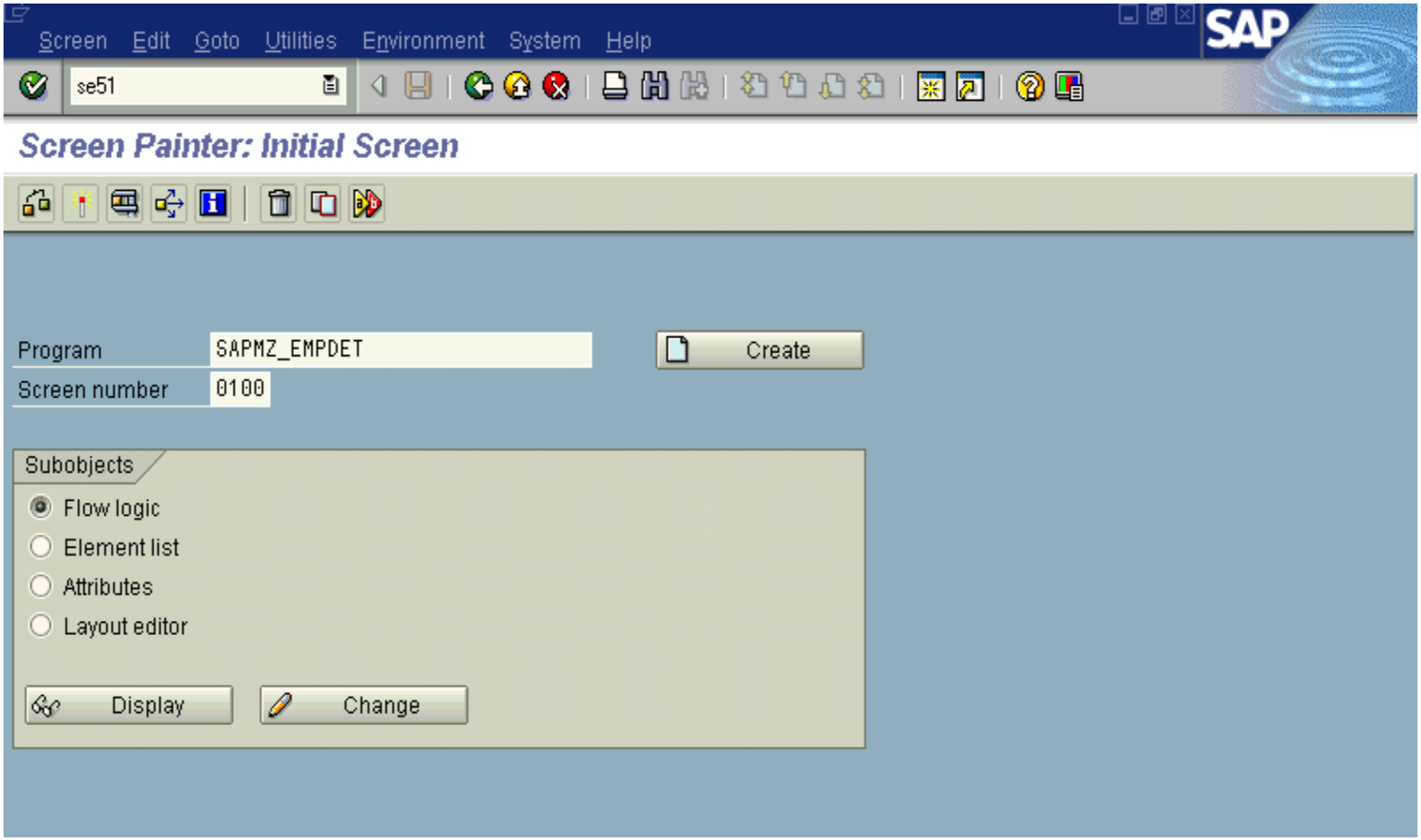
Eg. SAPMZ_EMPDET



Now create a screen using Transaction SE80

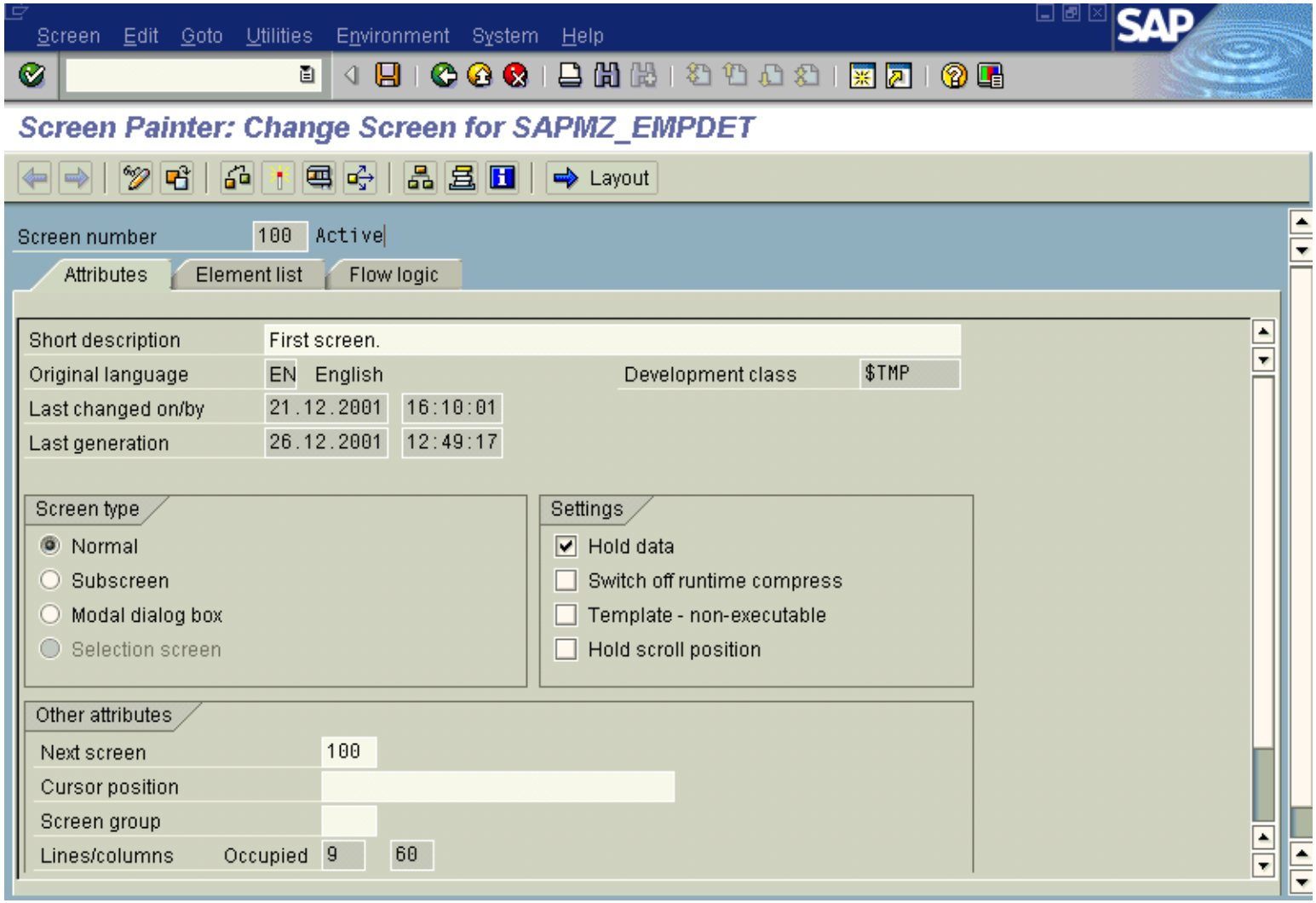


We can also create a screen using Transaction SE51



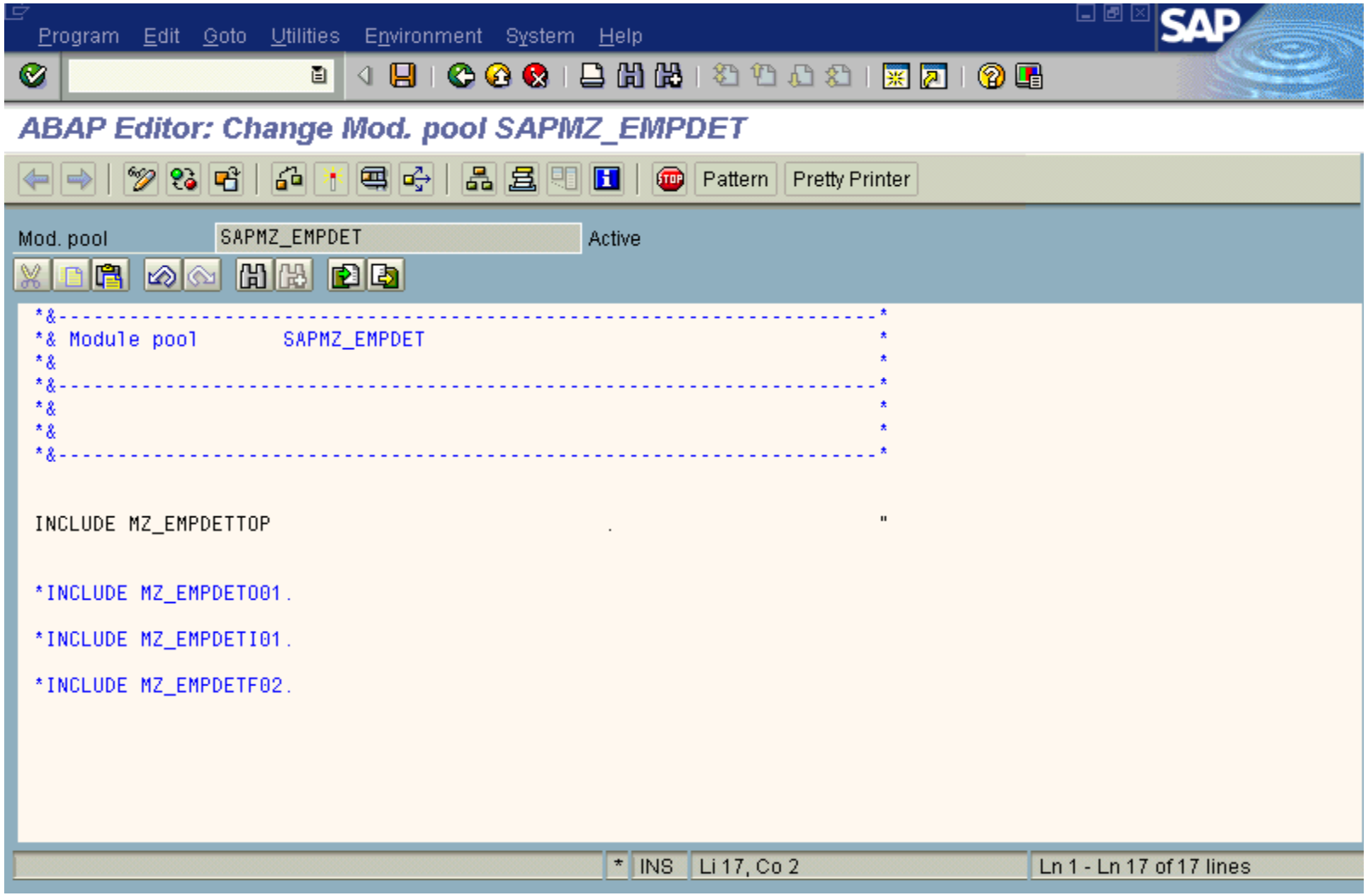
Click the create button

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



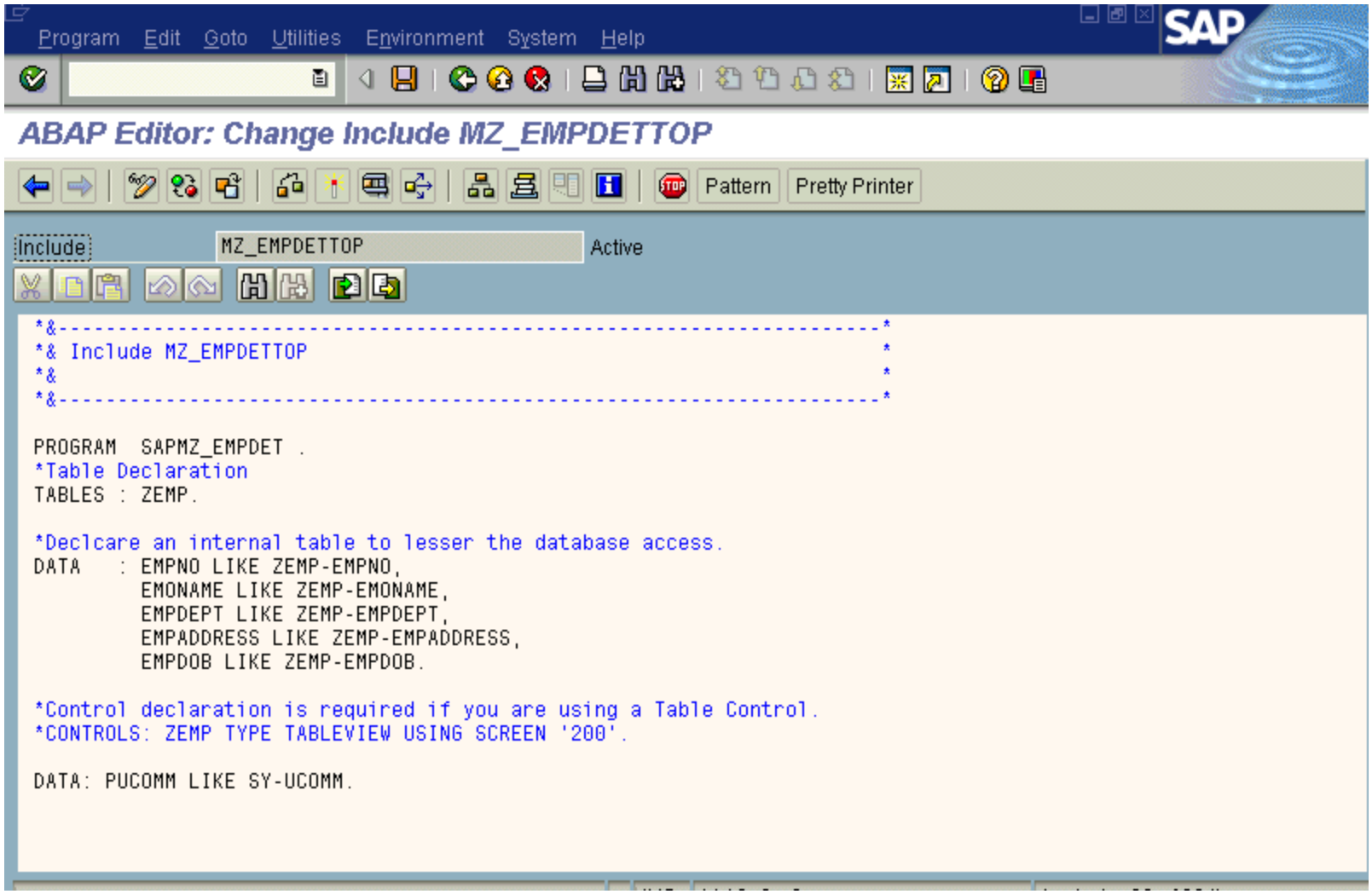
Click on the save button to save the entries

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



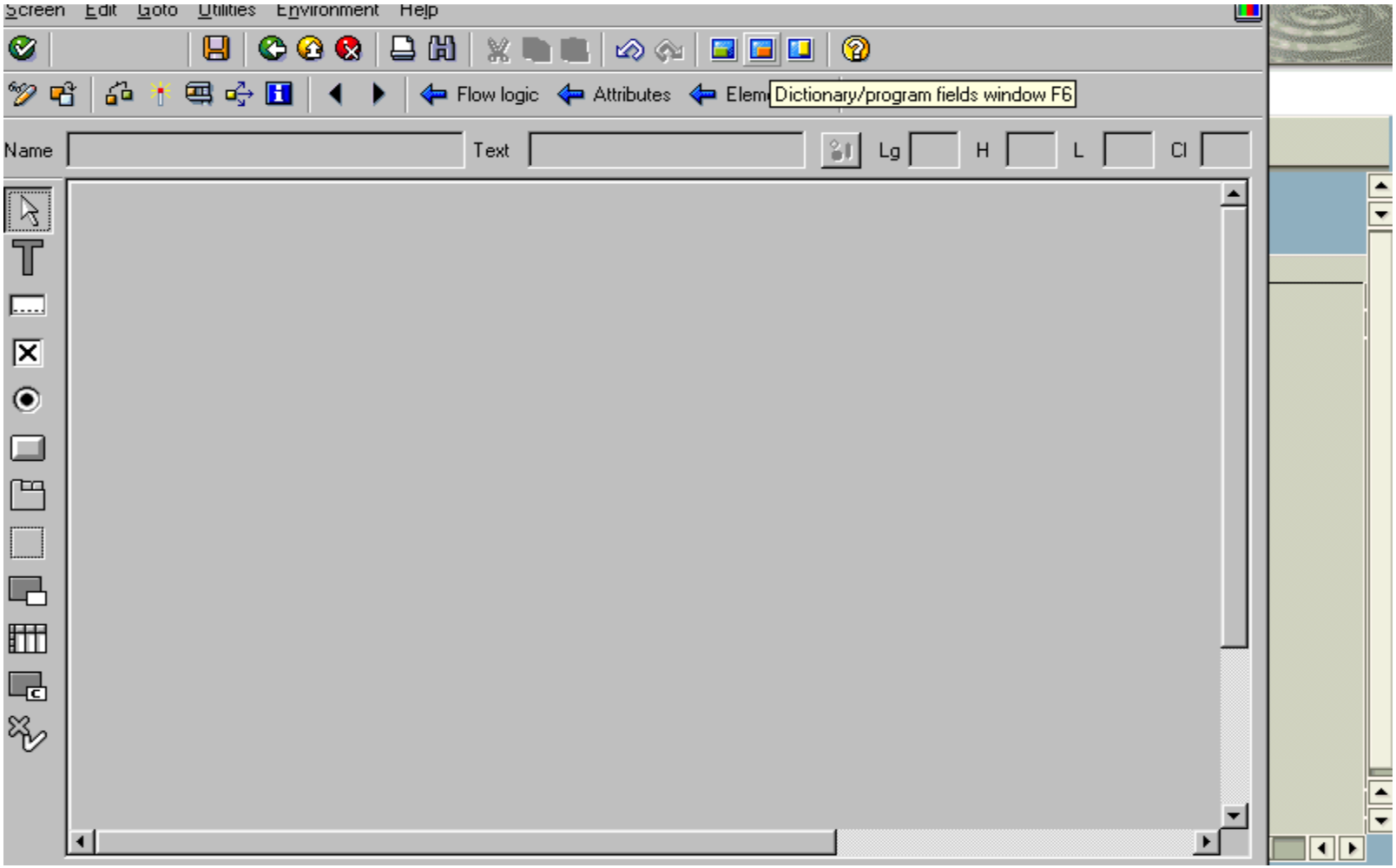
Cont. Of the previous screen

Click on the SAVE icon to save the code.

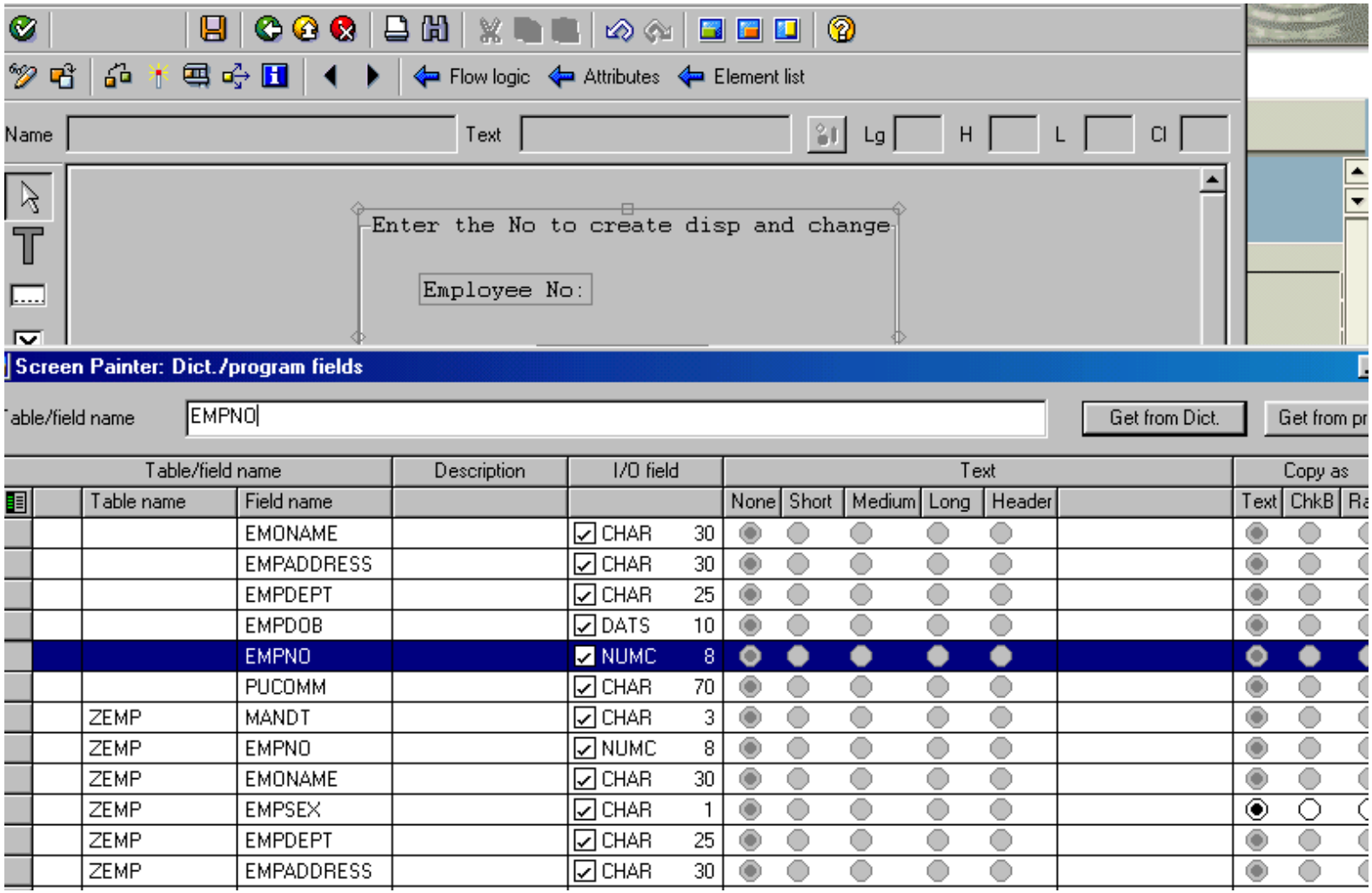
Click the BACK icon to come out.

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

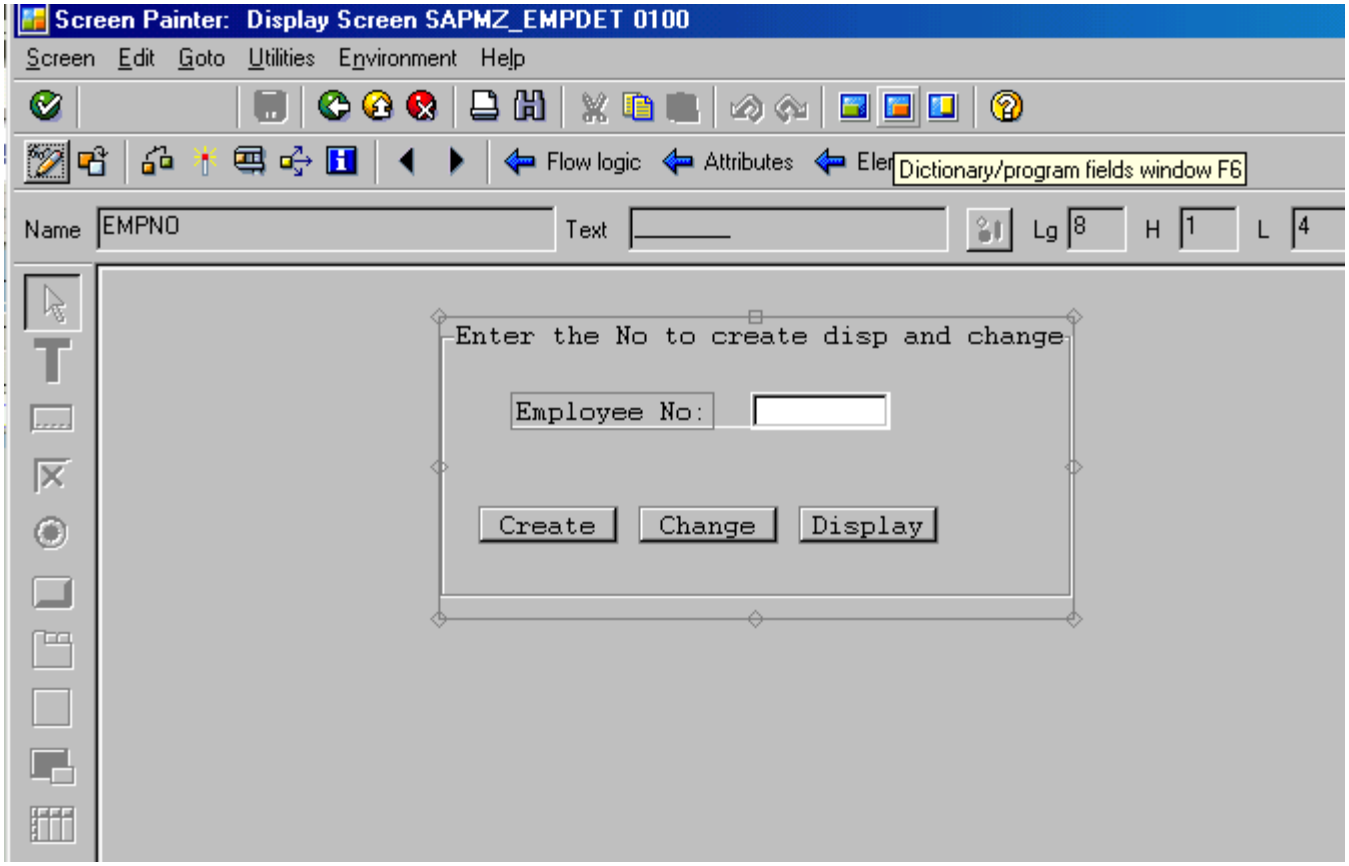
Click on the DICT/PROGRAM fields button on the application toolbar to include the fields on the screen from the dictionary tables or internal tables or other fields declared in the program



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

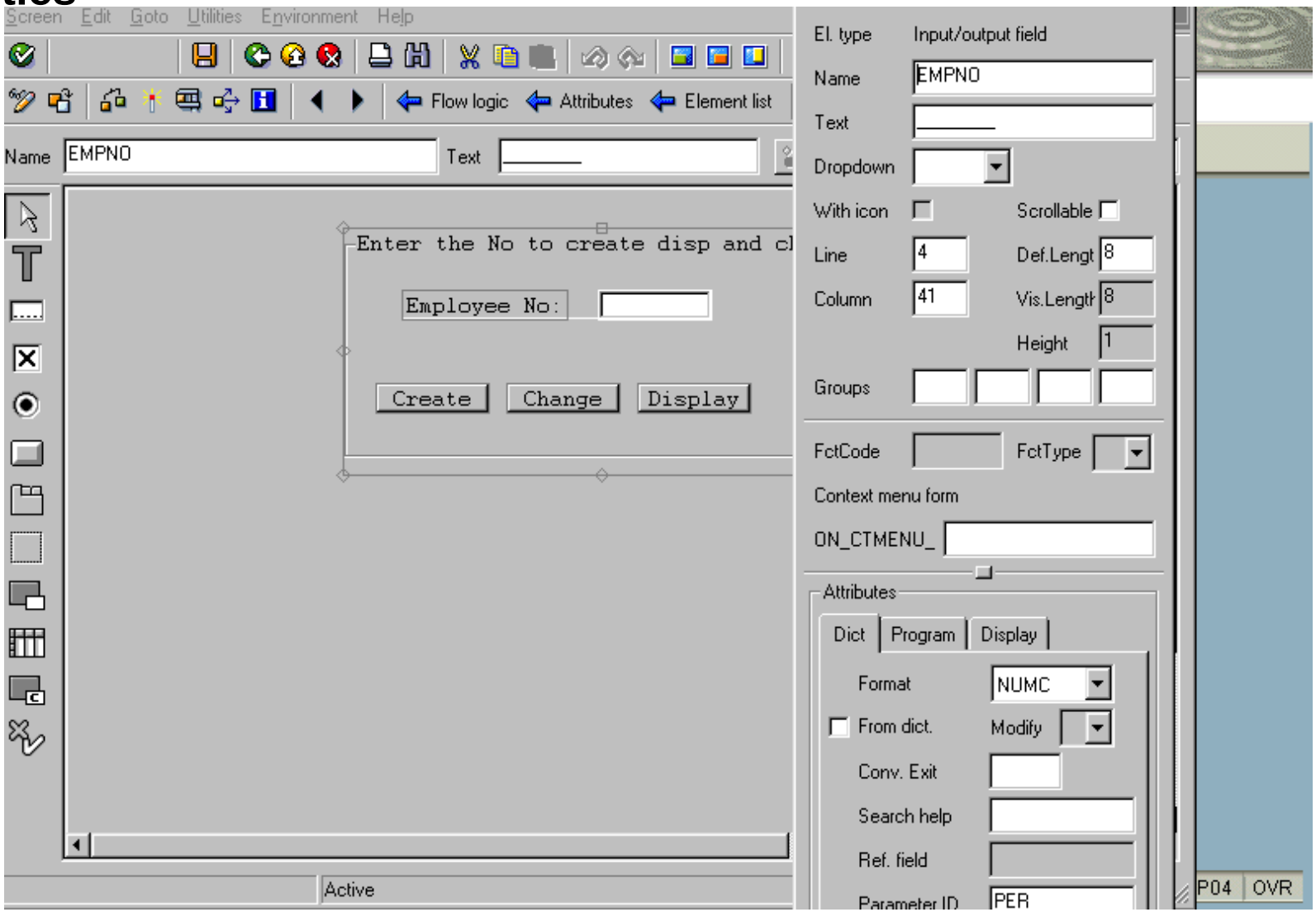


Using the corresponding icon (Text, Entry, Check and so on) on the object bar and drag and place the object on he screen.

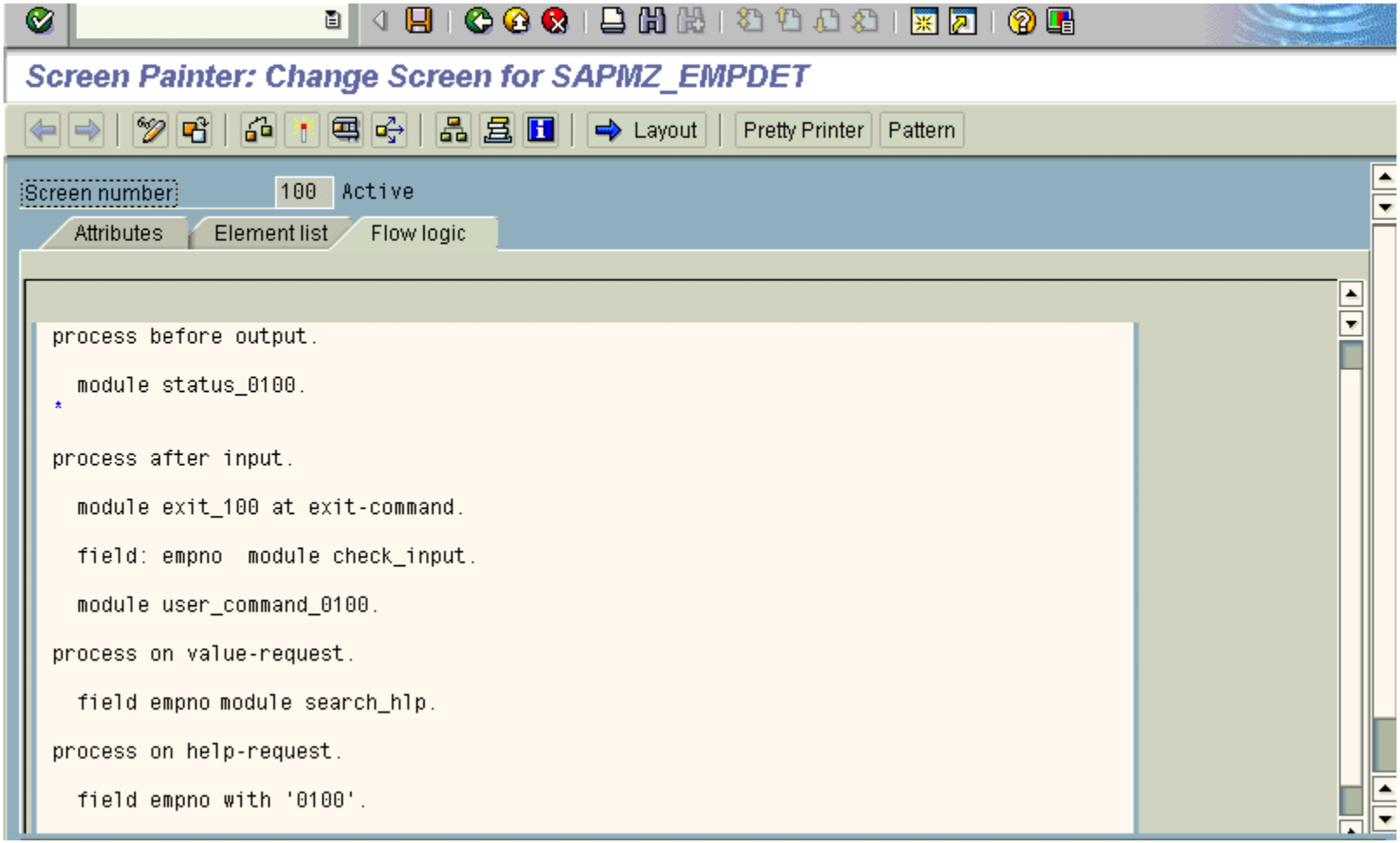


Click on the SAVE icon to save the code .

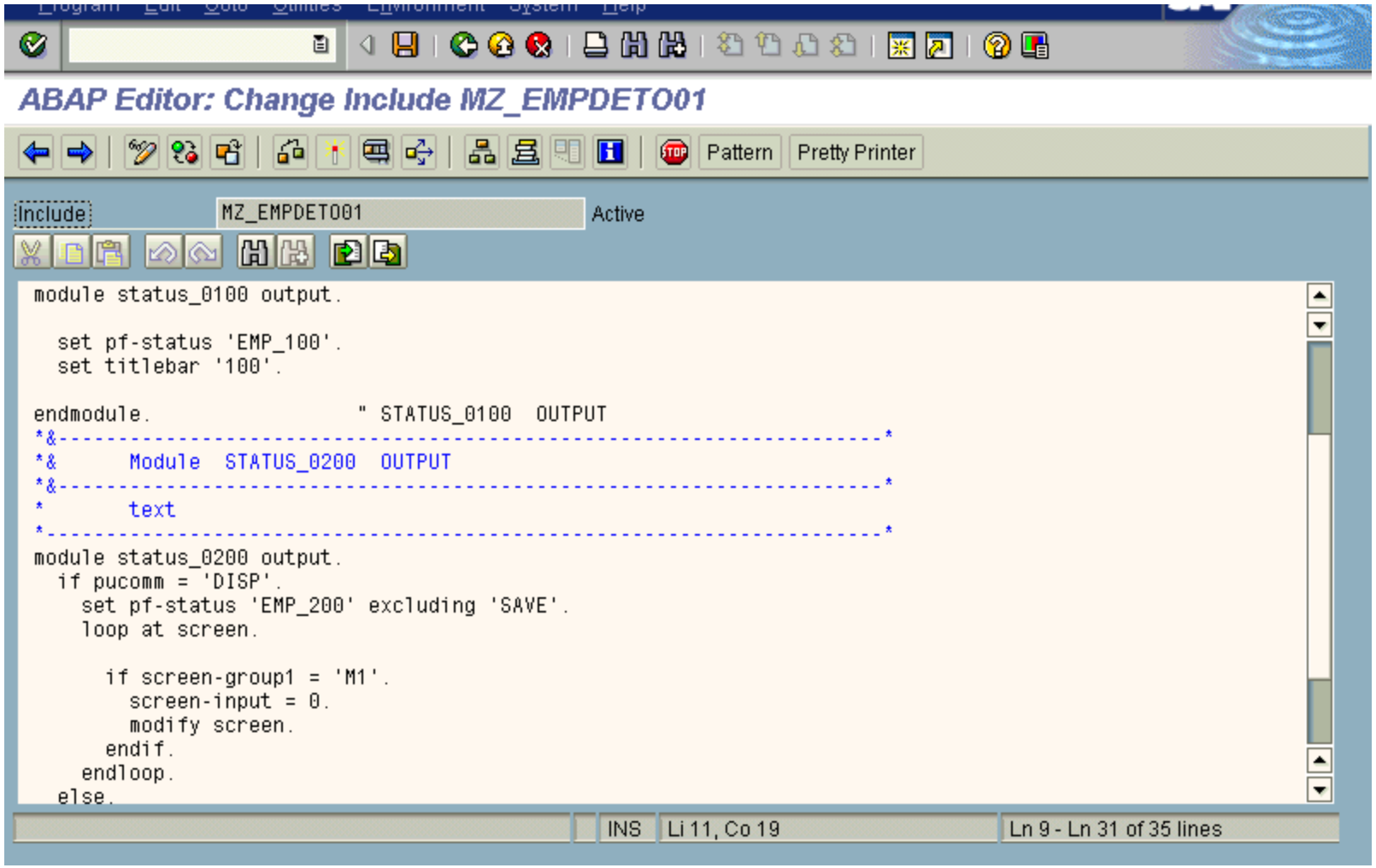
Double Click on the table Control object to view the attributes or properties



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

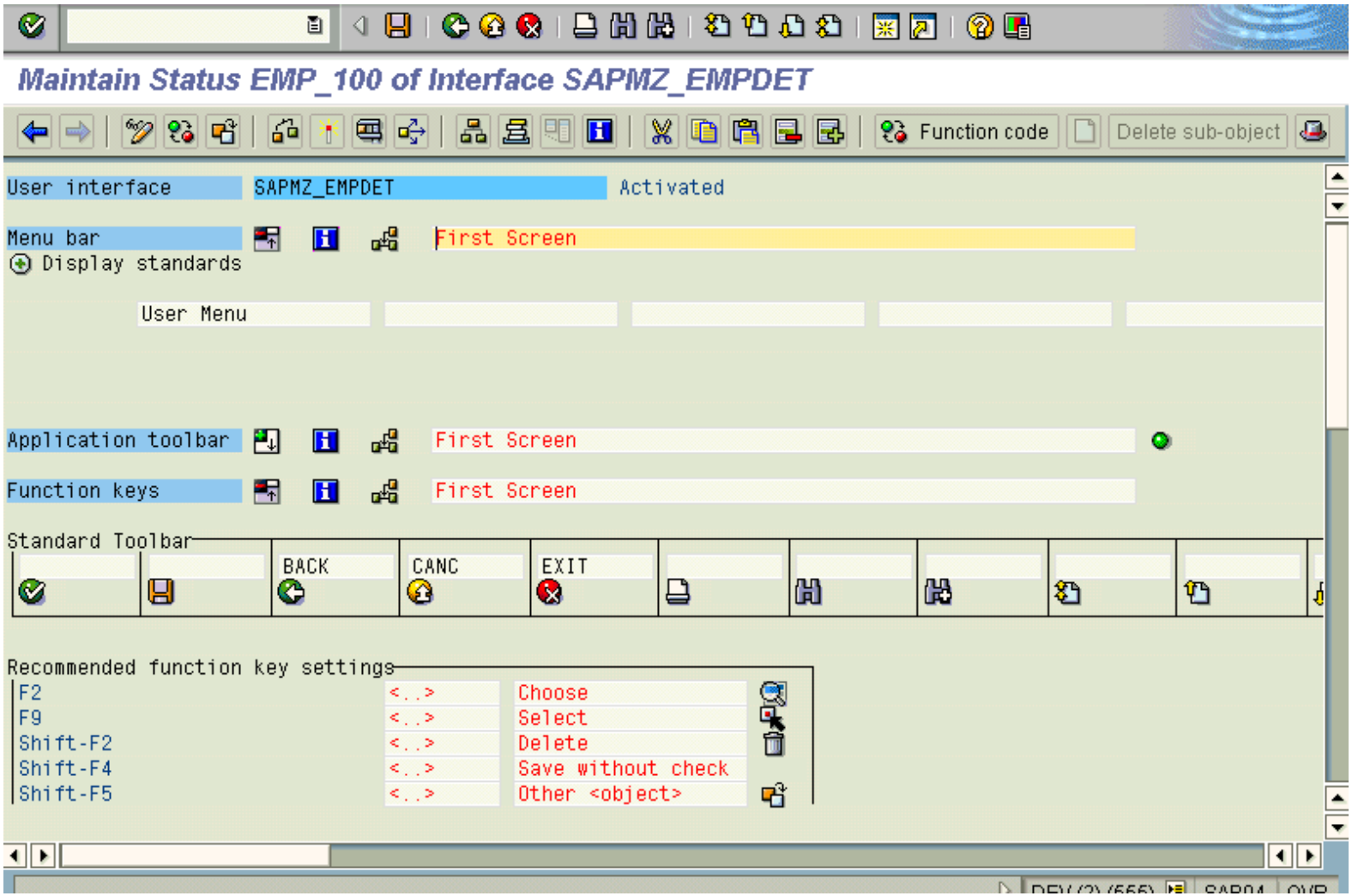


In the PBO write the relevant code for PF-STATUS and TITLE BAR

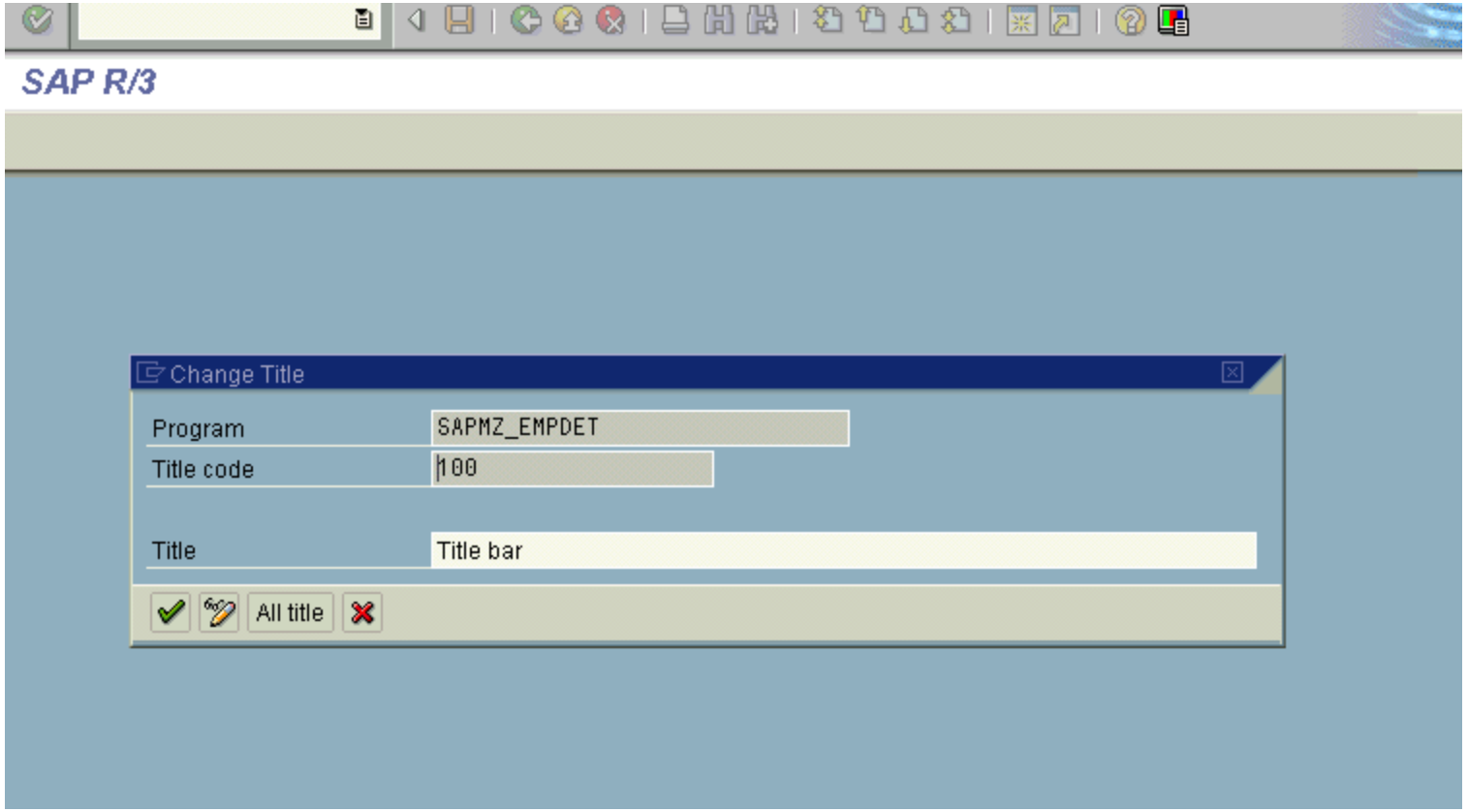


Click on the SAVE icon to save the code .

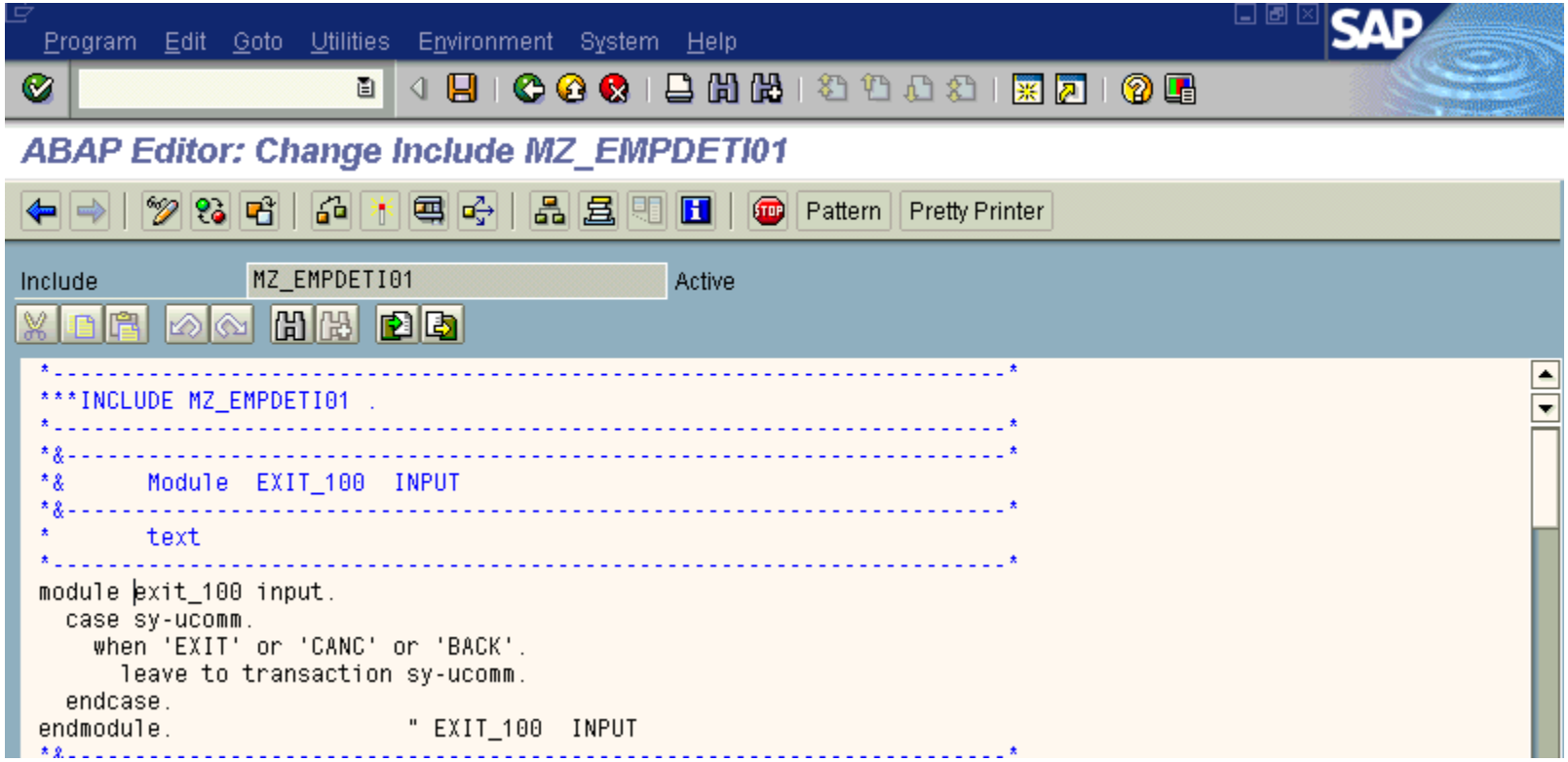
This is a GUI status containing buttons and menus for the screen. All ABAP programs will have a default GUI status.



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

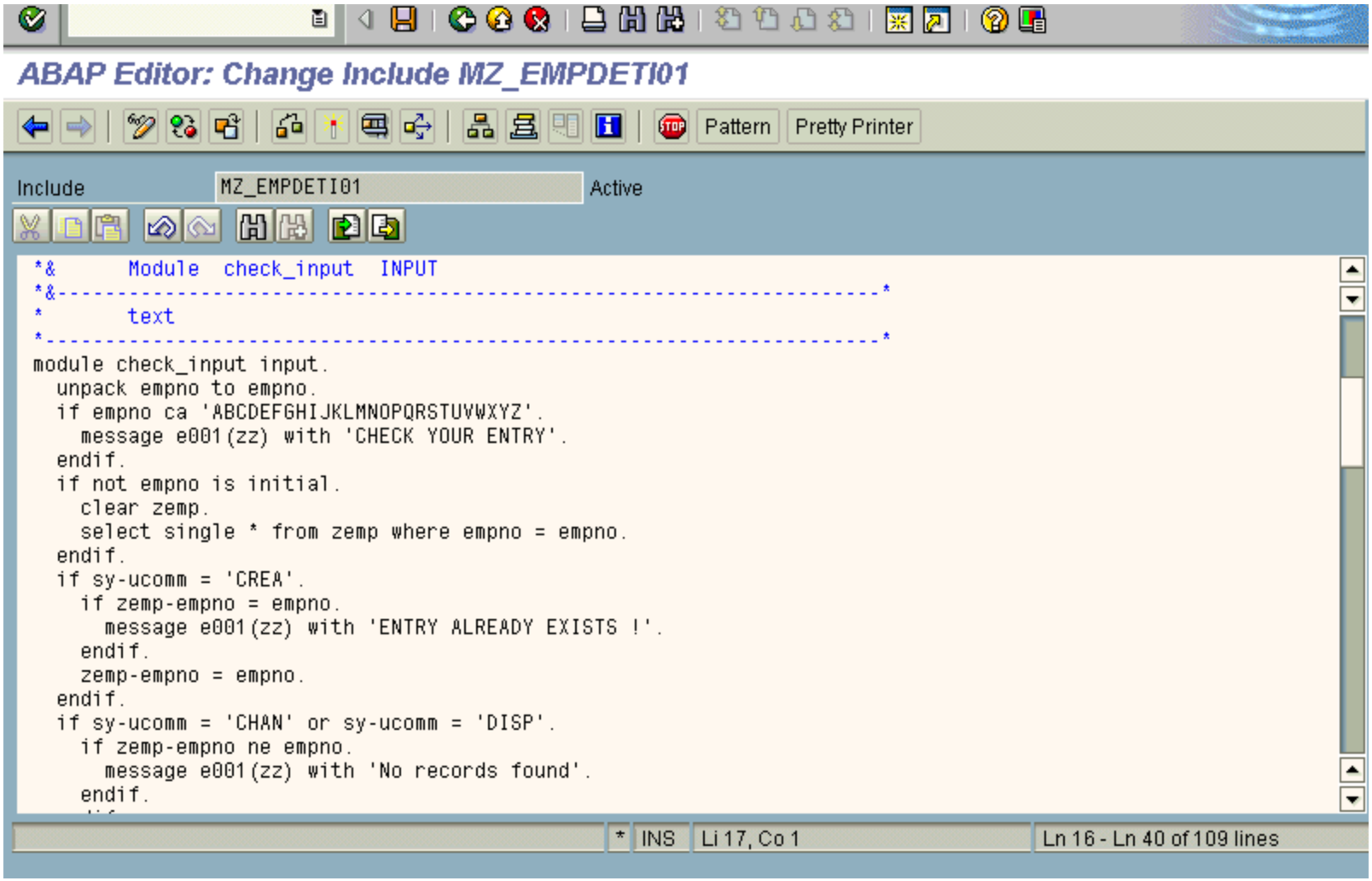


In the PAI of the flow logic we can use the conditional statement AT EXIT-COMMAND as below.



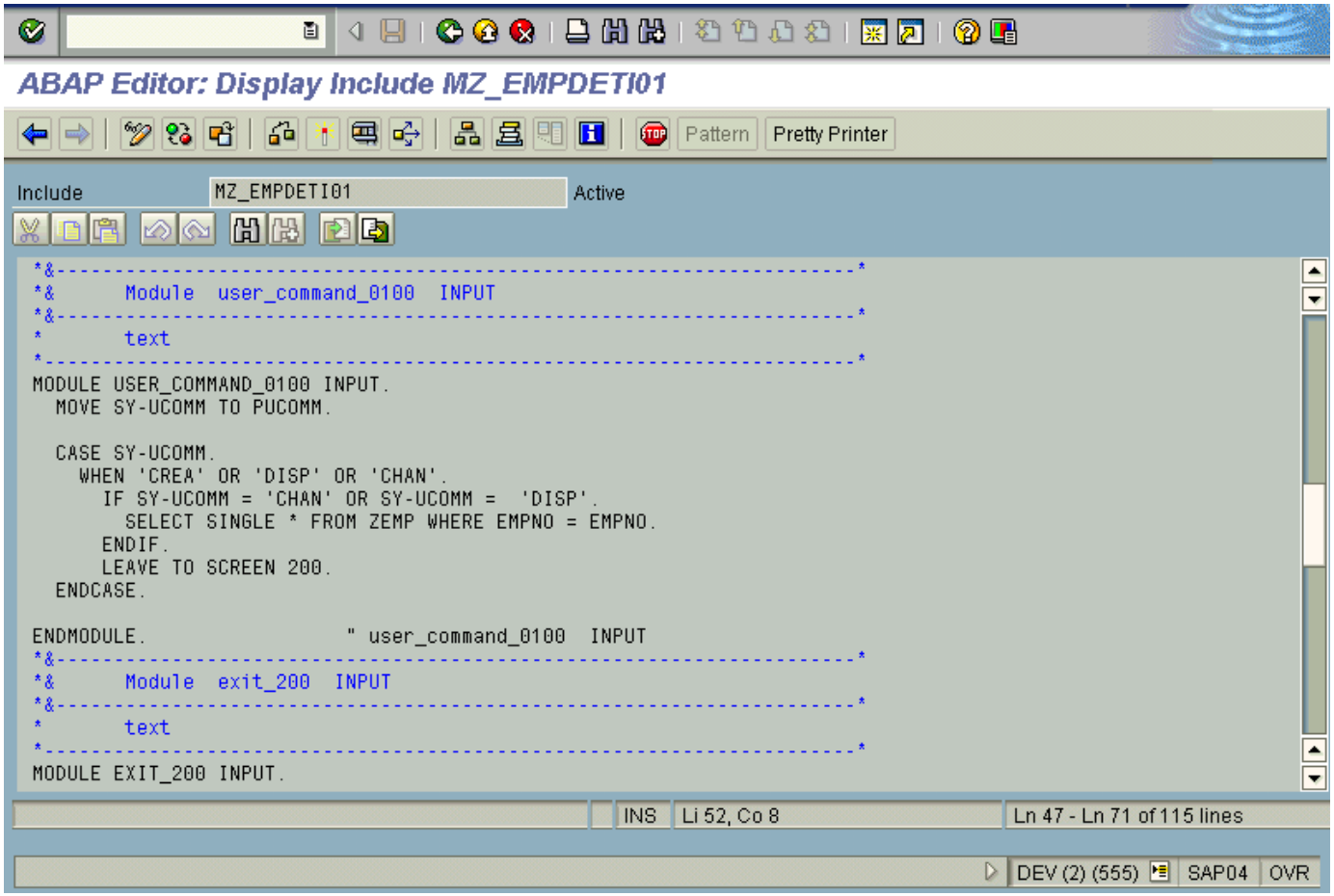
Click on the SAVE icon to save the code .

Code for the Field check in the PAI module.

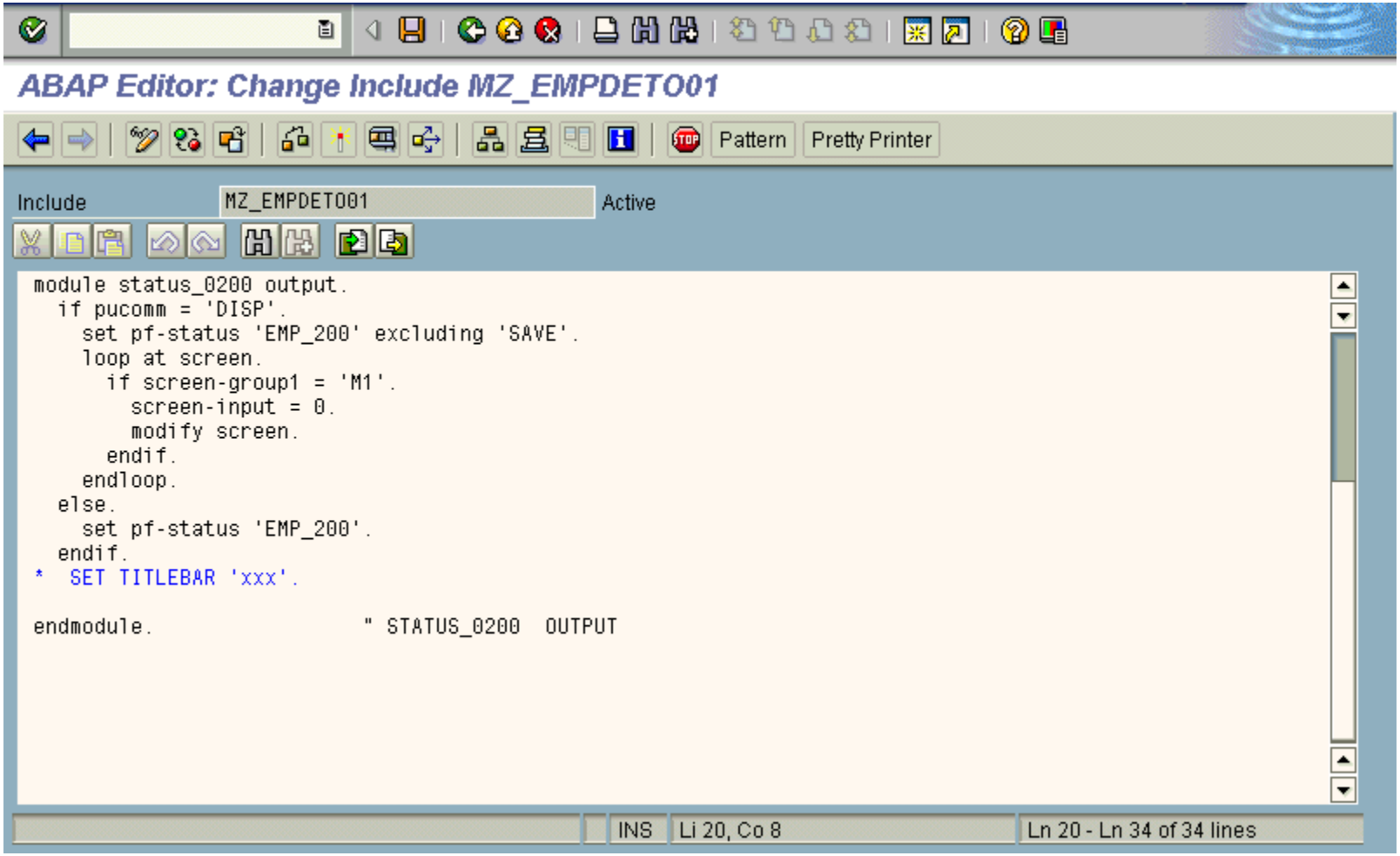


Click on the SAVE icon to save the code .

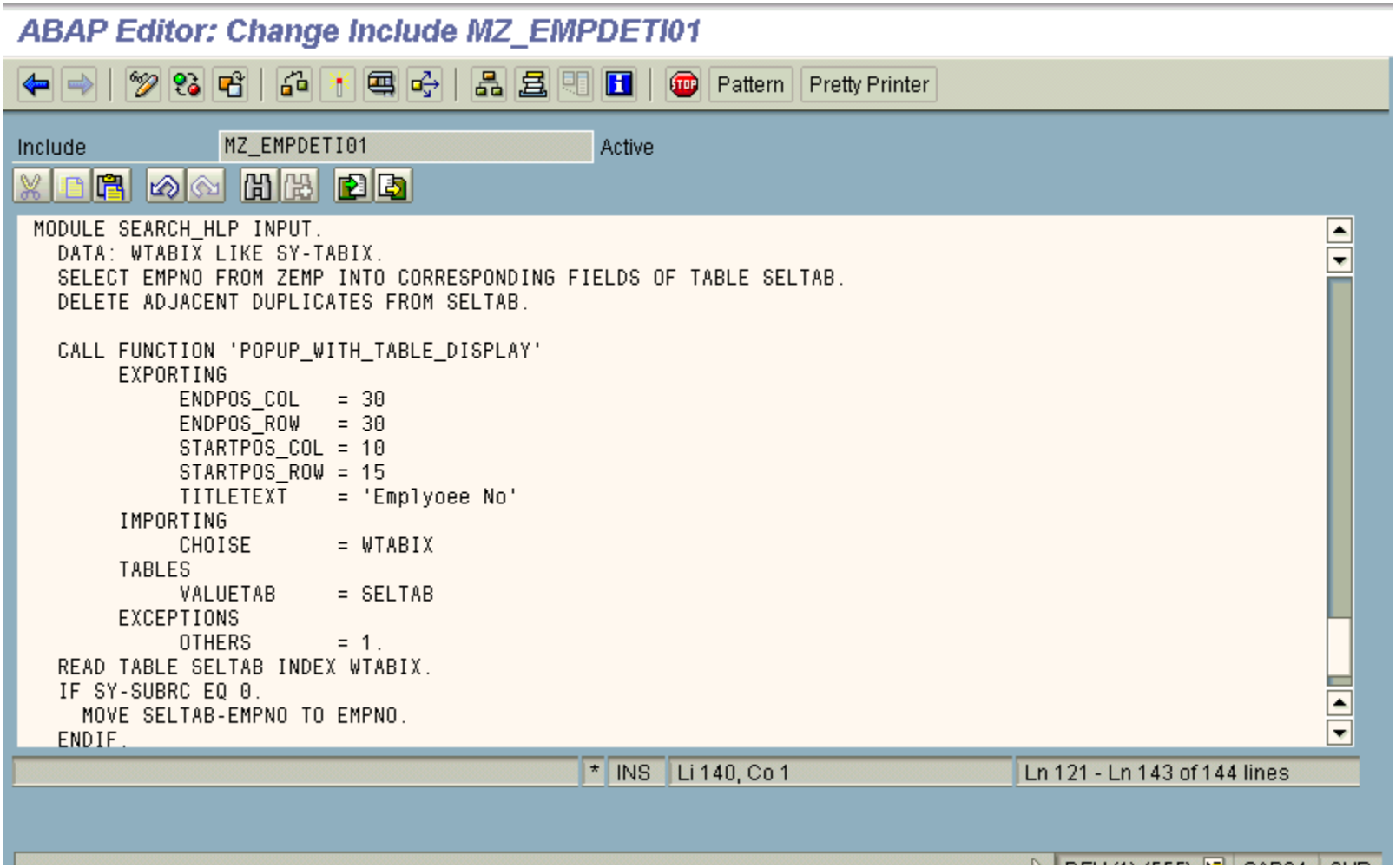
In this module we can even write the code to Retrieve the Data, Branching to Different Screen and also use function codes .



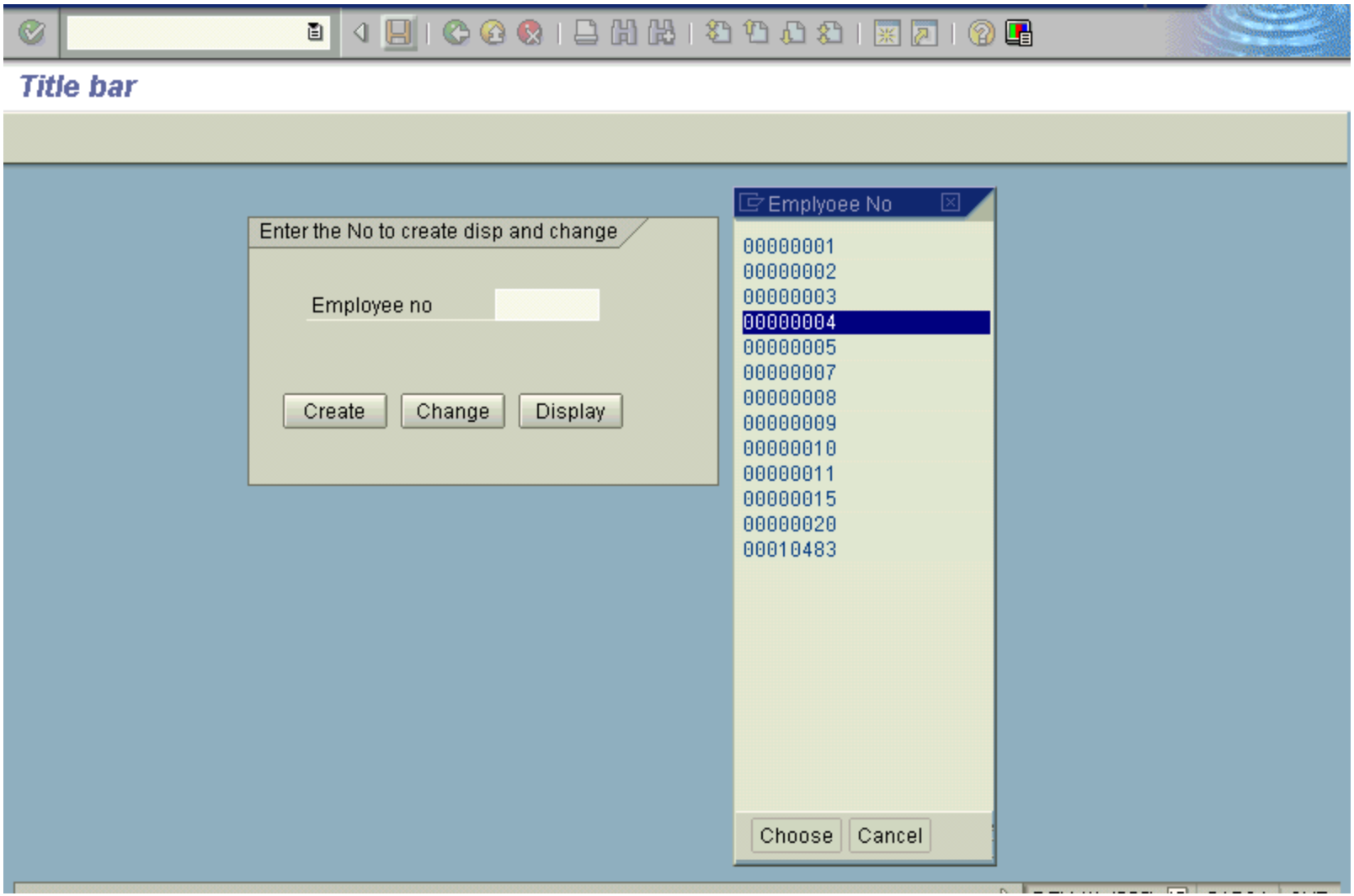
Depending on the user requirements in the previous screen 100, you can dynamically change the Attributes of the object using SCREEN Attributes in the screen 200.



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

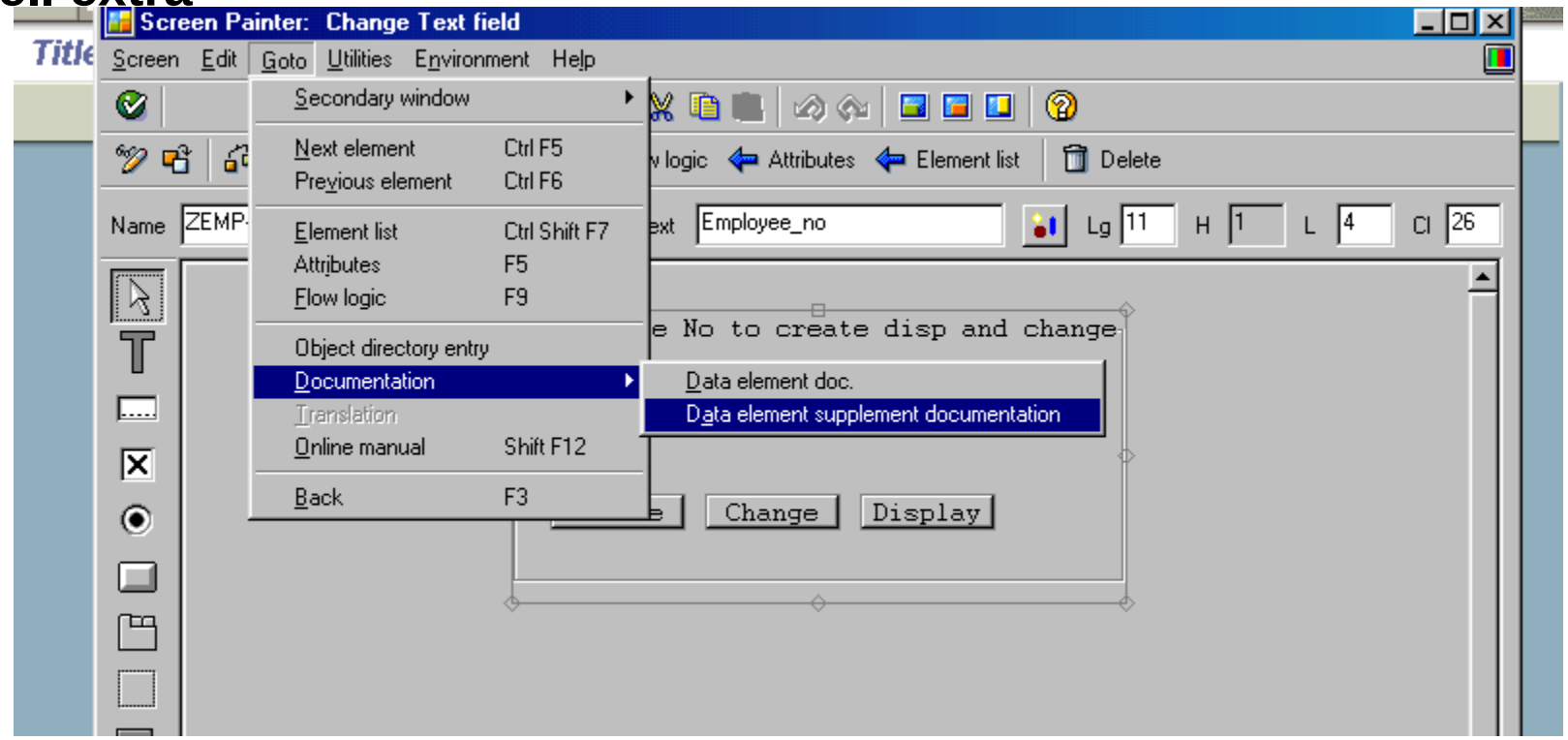


You Can Use the F4 to see the search help

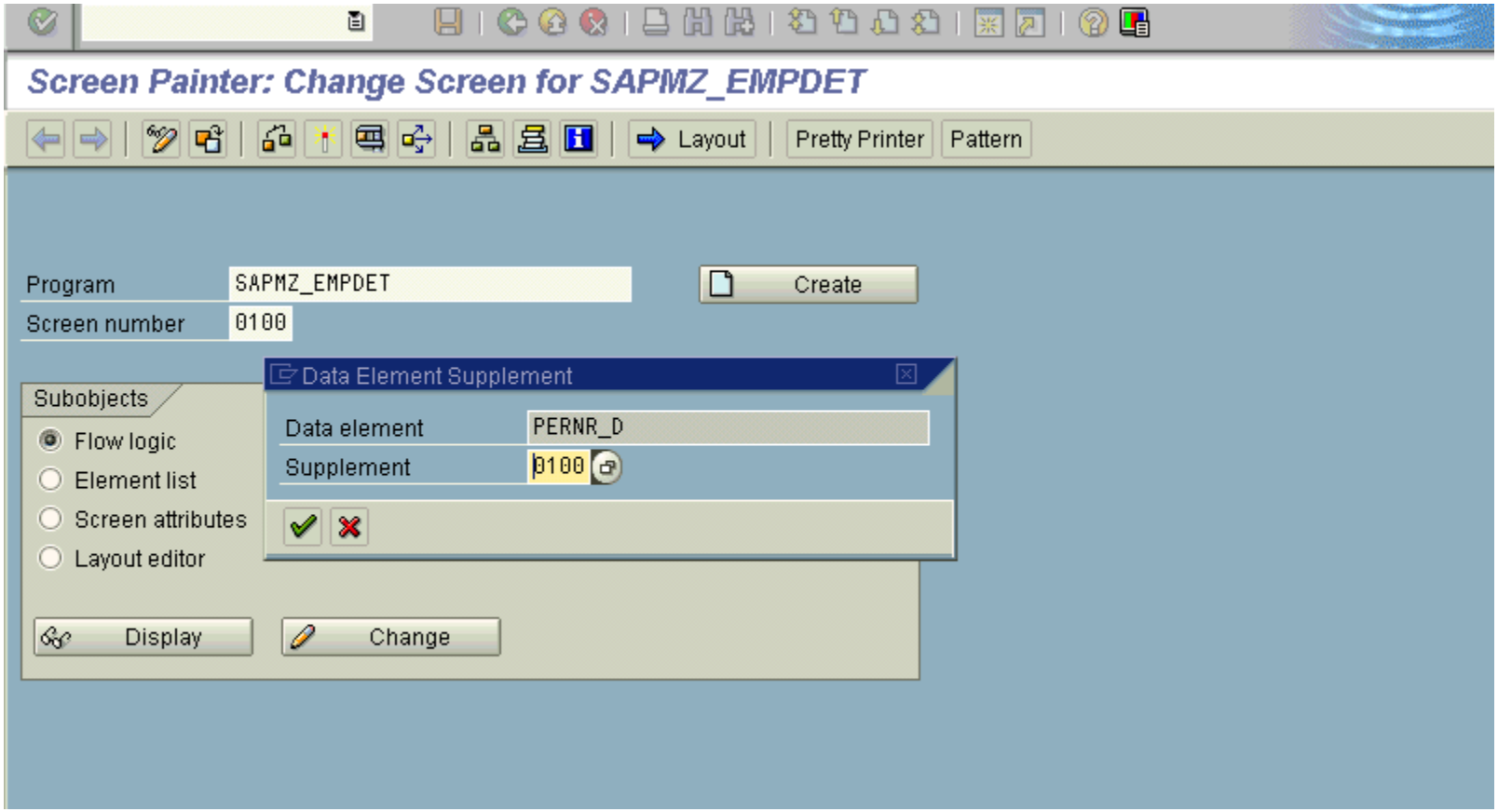


In the Process on value Help (POH) of the flow logic we can use the additional documentation descriptive text for the data element in the ABAP/4 Dict.

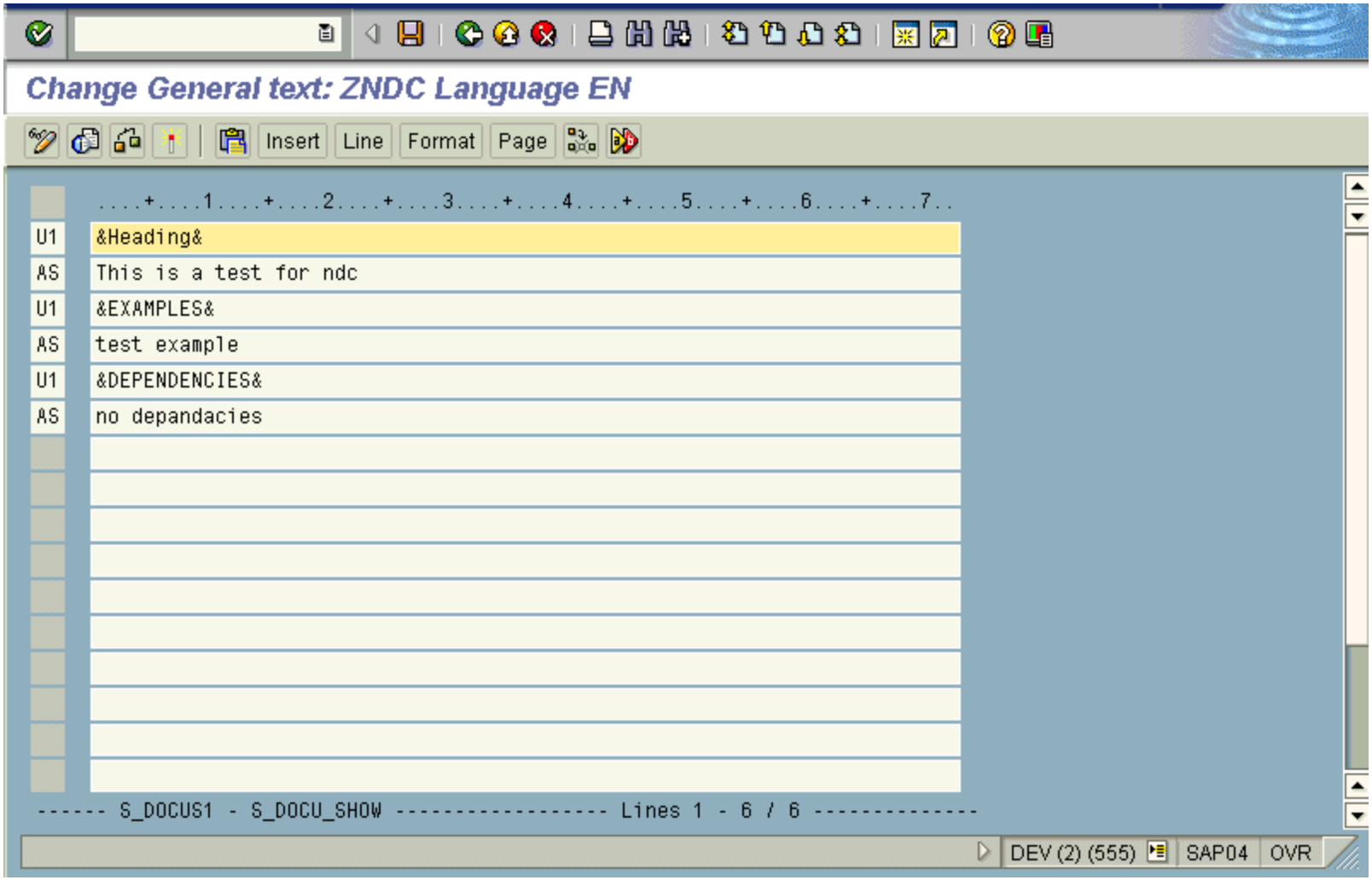
In the screen painter you place the cursor in the field string of a screen on the field to be documented and select the menu goto > documentation > data el. extra



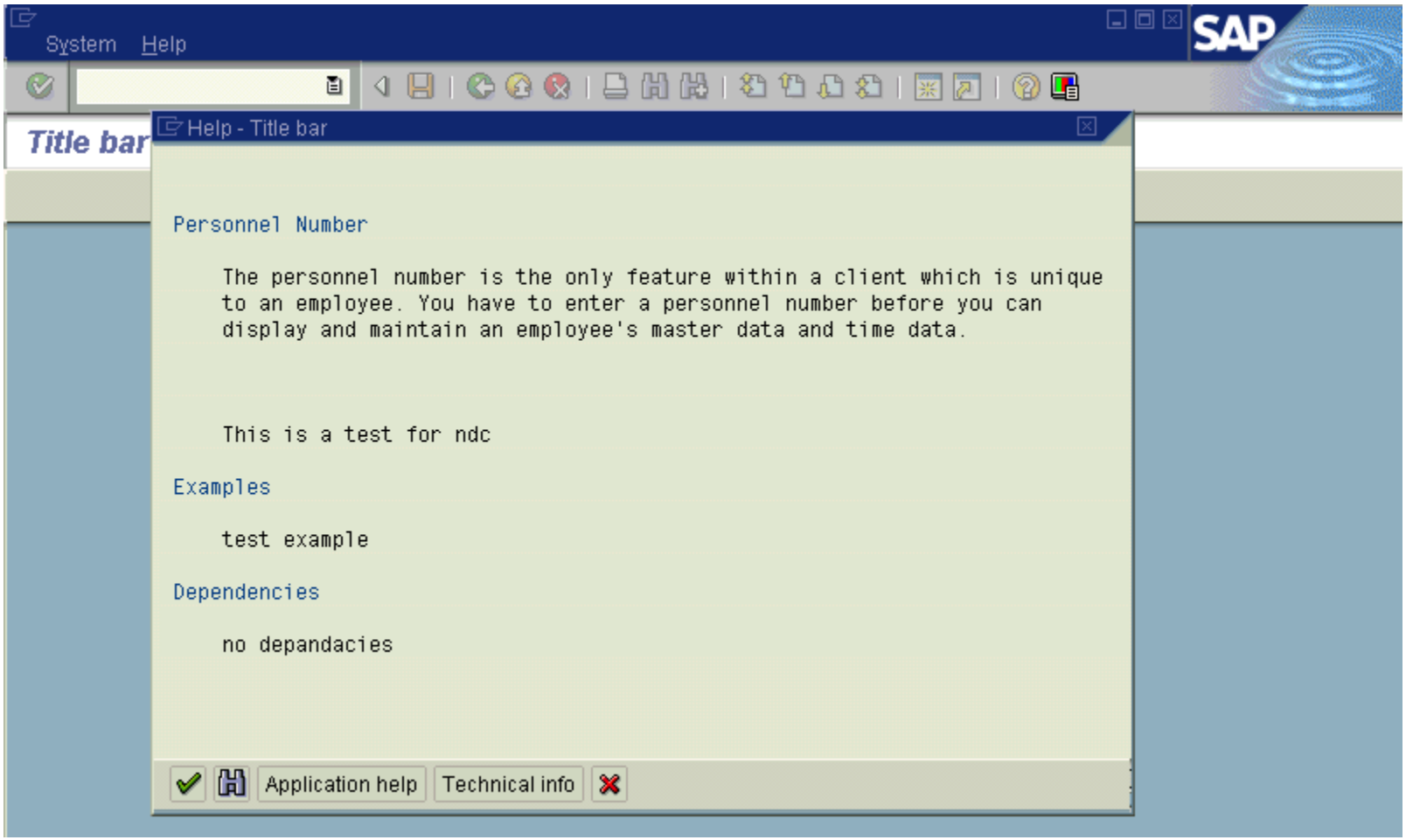
Now you will get a popup with the data element and the number of the current screen as an identifier for the additional text. In addition to the help contents in the ABAP/4 Dict you can enter your own Description



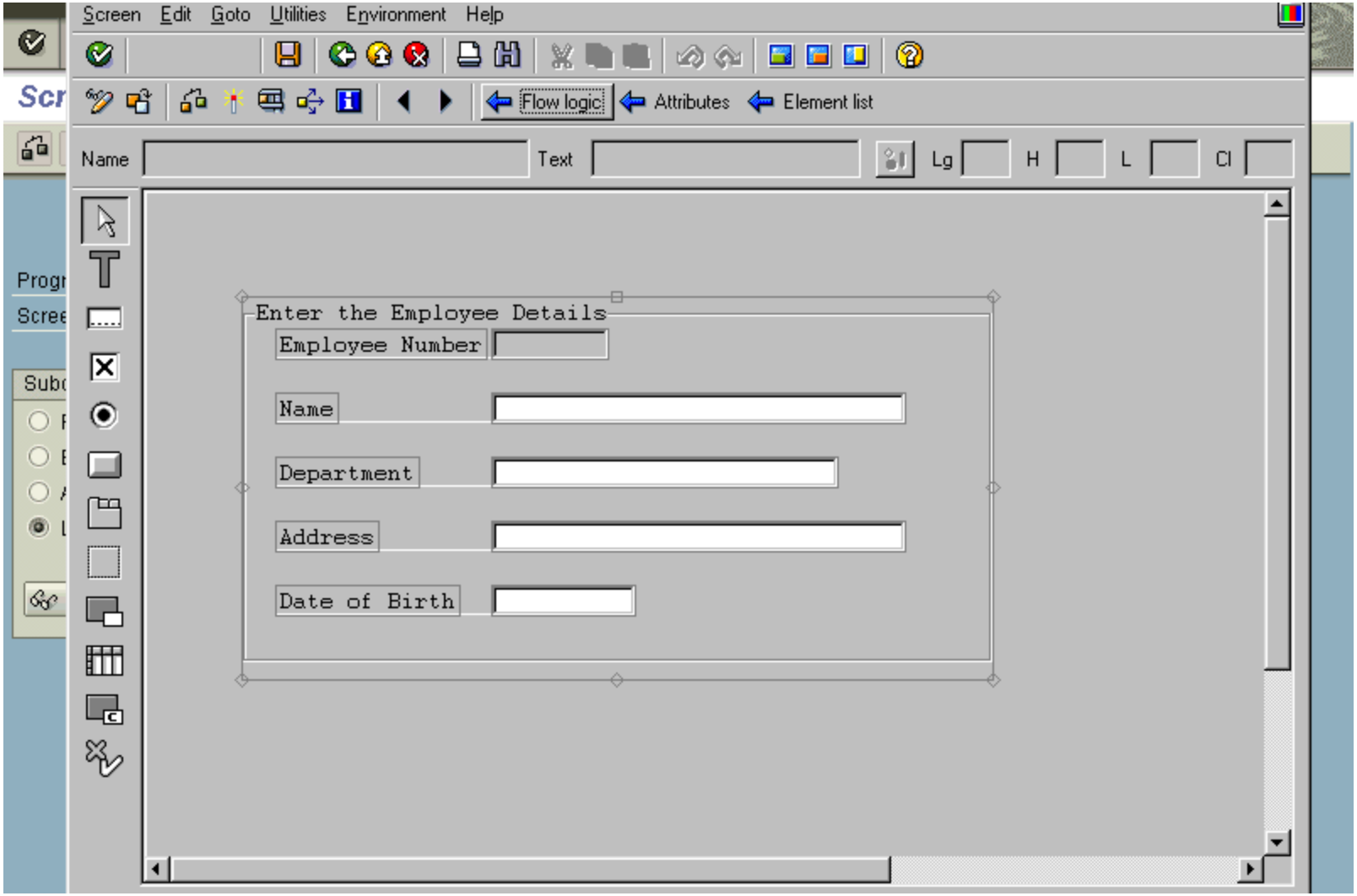
Here you enter your own description and SAVE



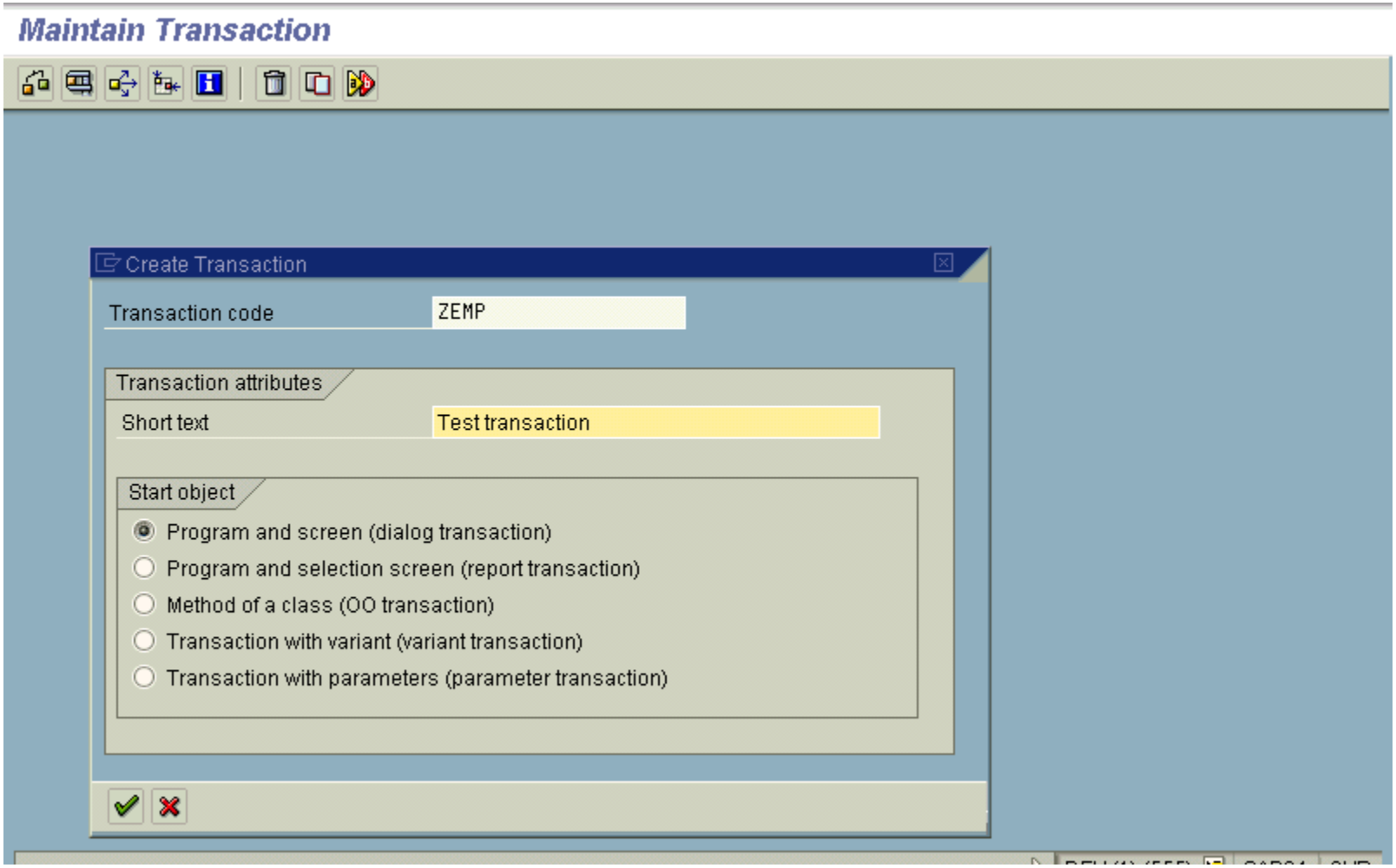
Now while Running transaction place the cursor on the field and press F1



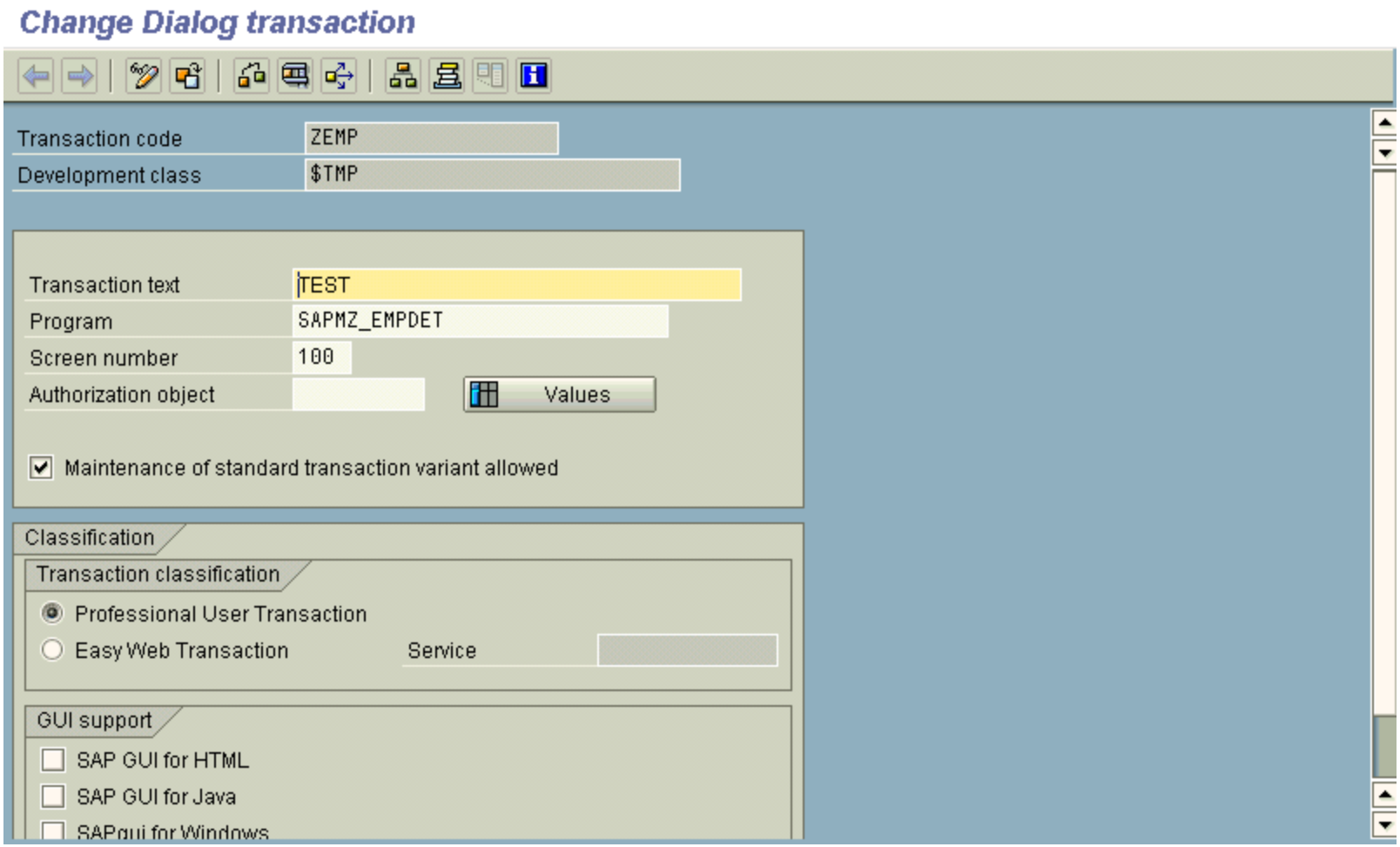
Layout slide for screen 200.



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

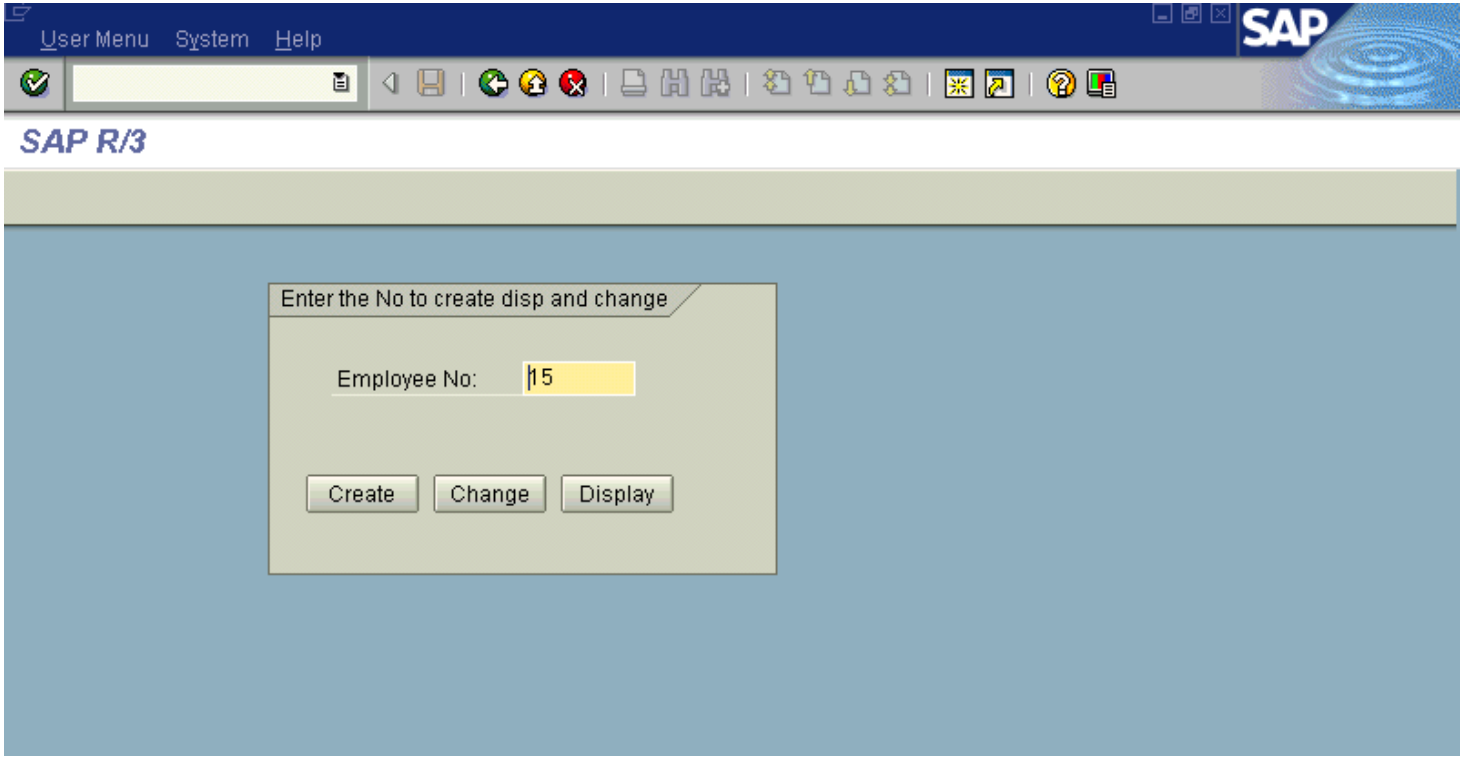


Click on the enter button

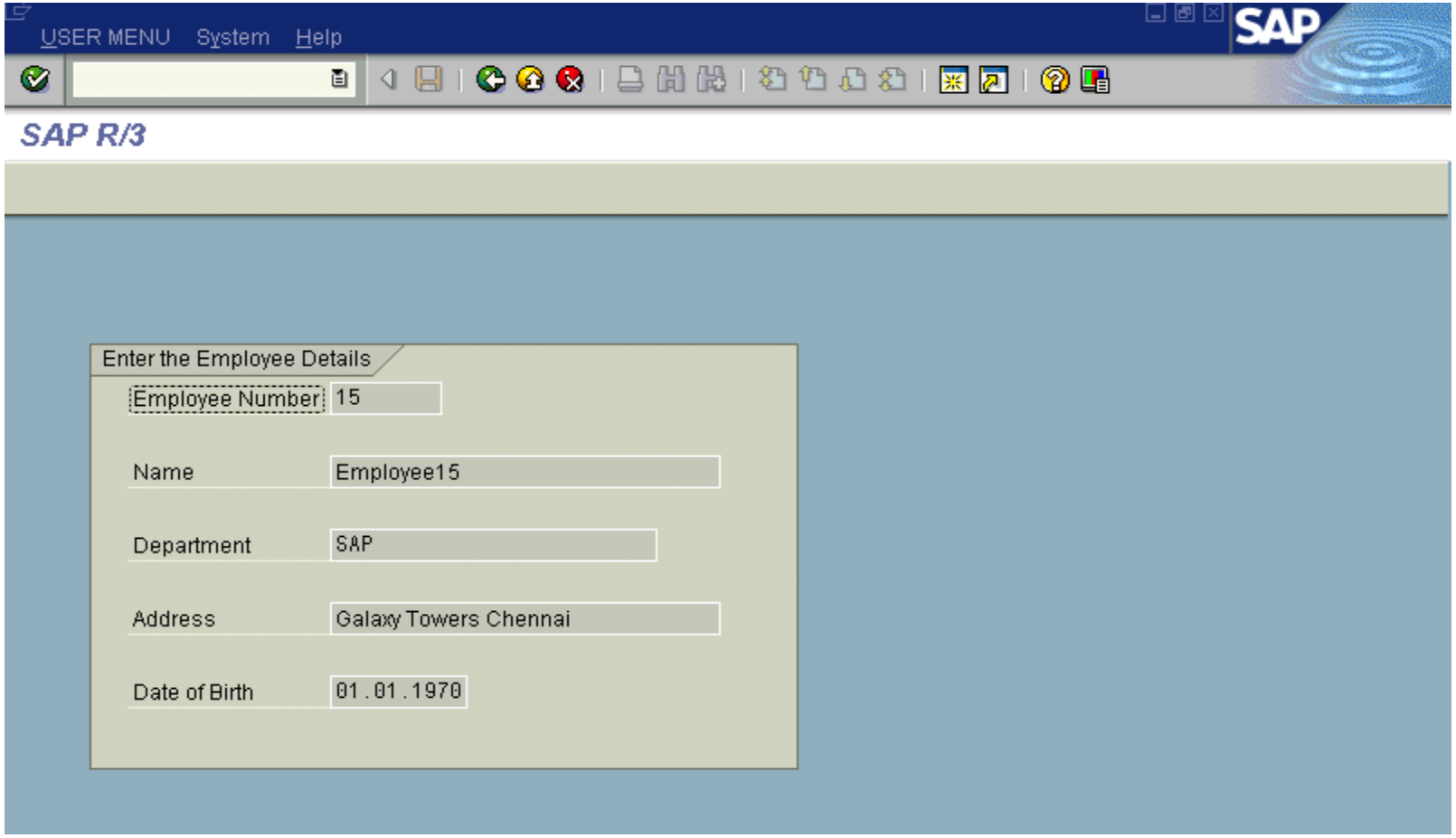


Enter the information in the fields and SAVE

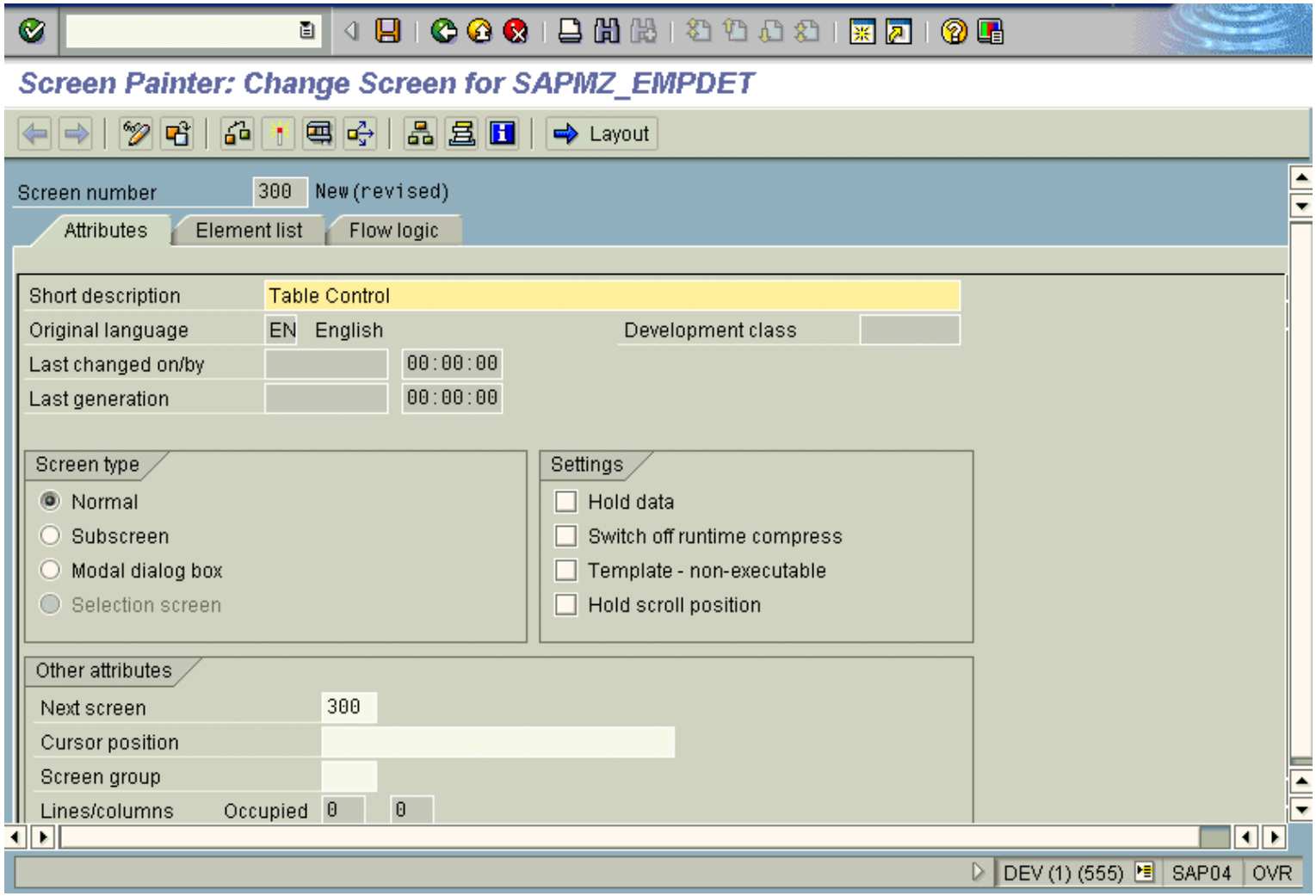
In Screen 100 we give the required input(Employee No) and retrieve the Data from the Data Dictionary and display the details on to the screen 200.



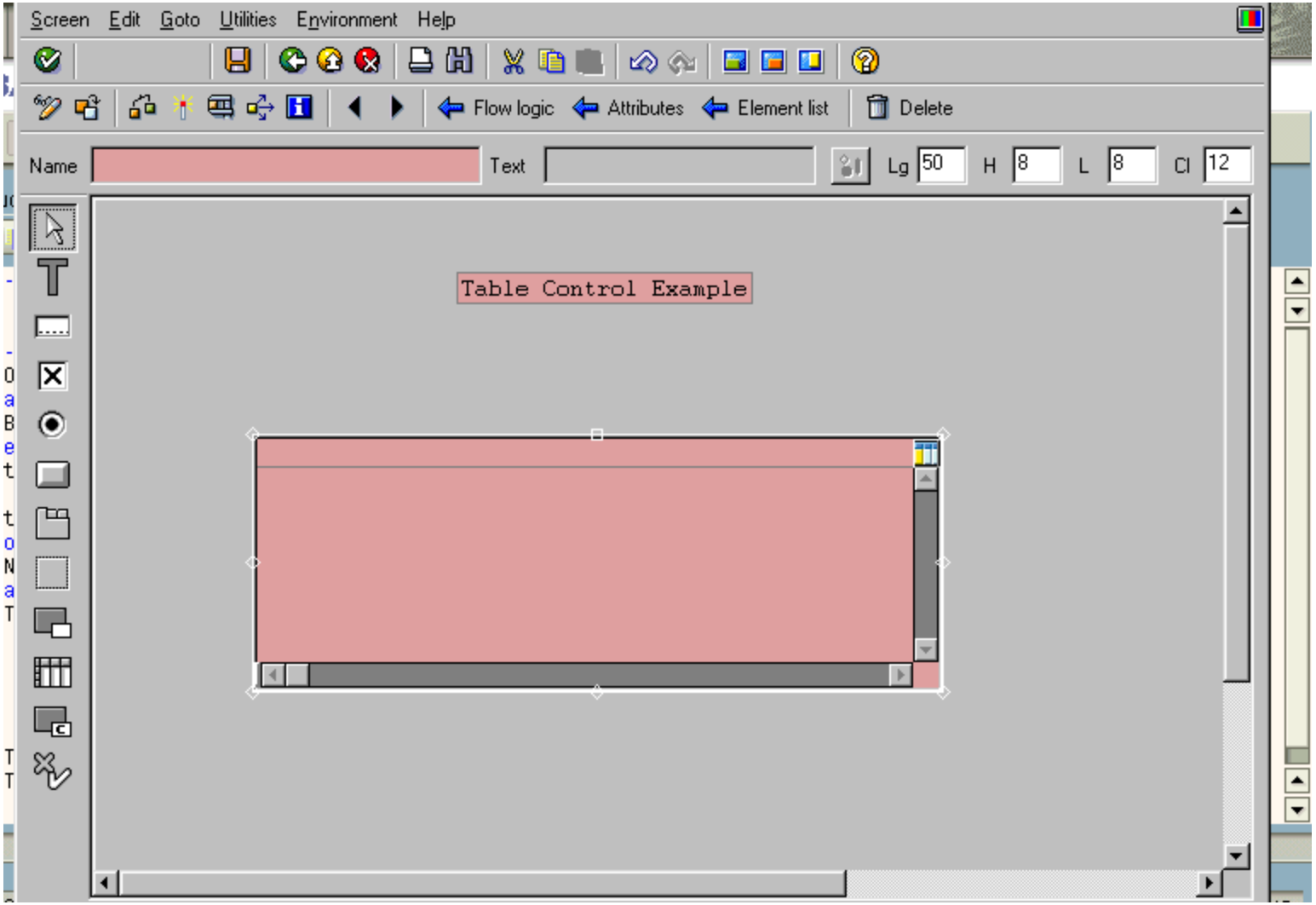
The details of the employee in screen 200.



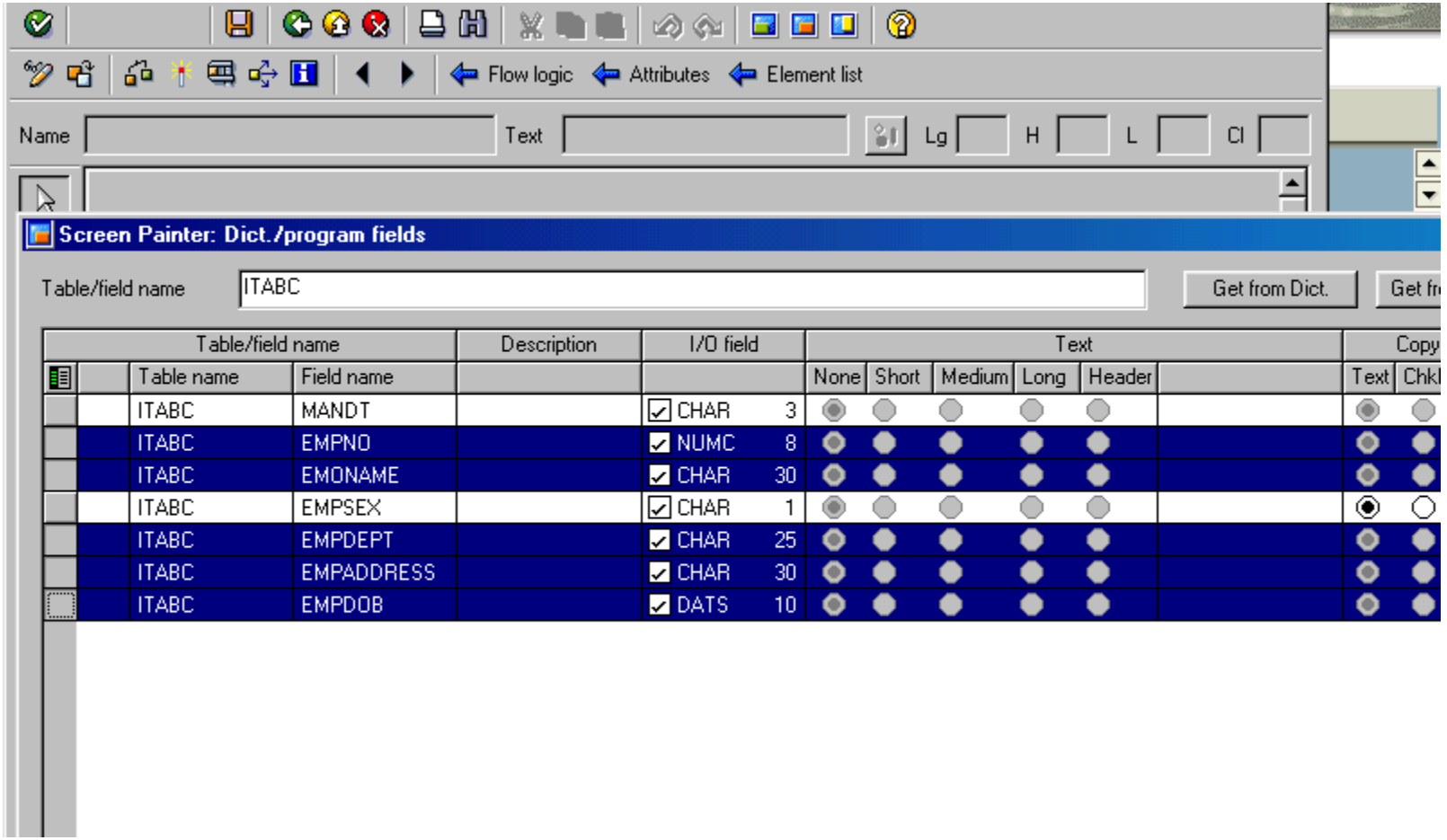
In this example we will create a screen and use the table control object to display data. Use SE51 to Create a screen (In our case 300)



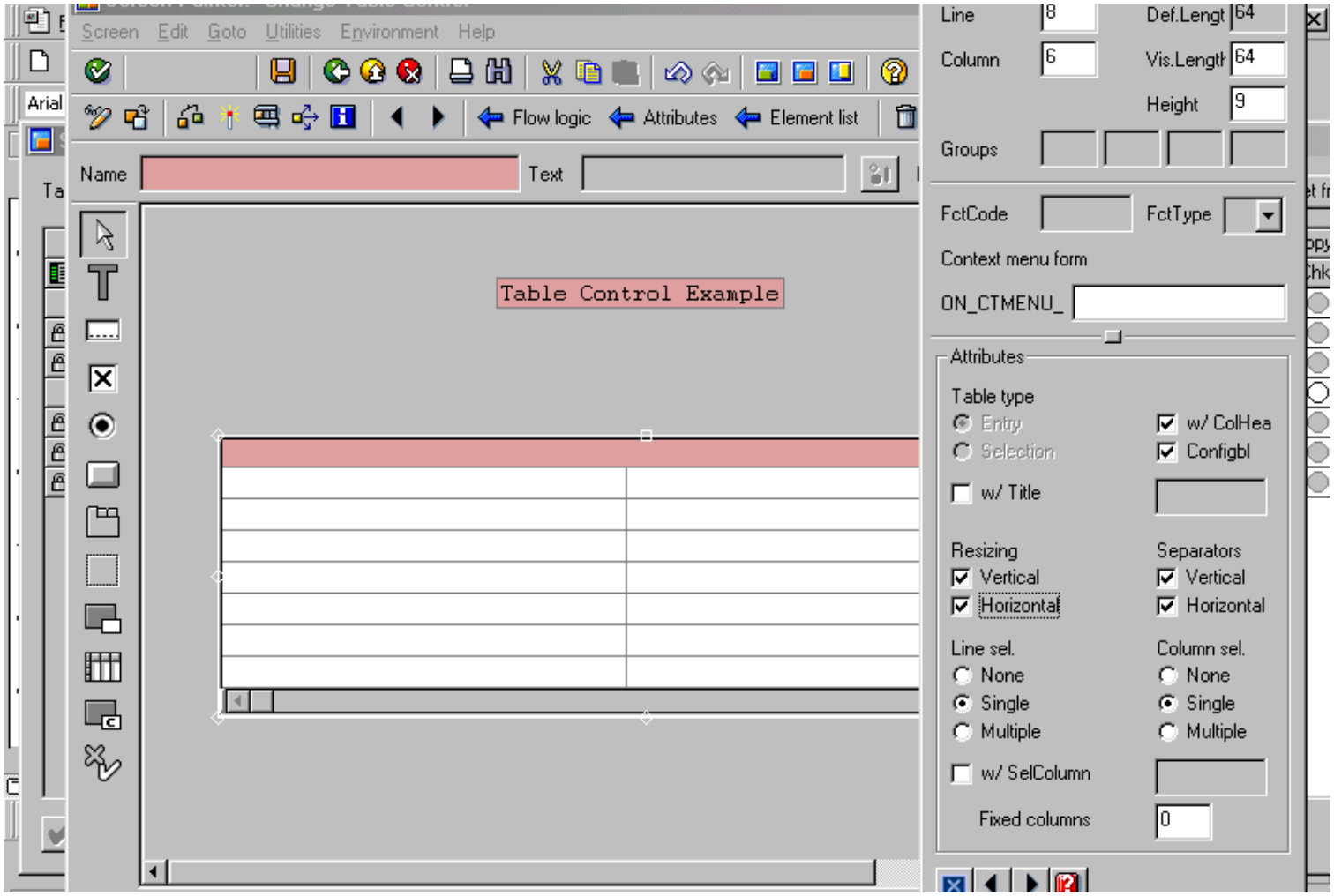
We will design the screen with a text object and the table control object



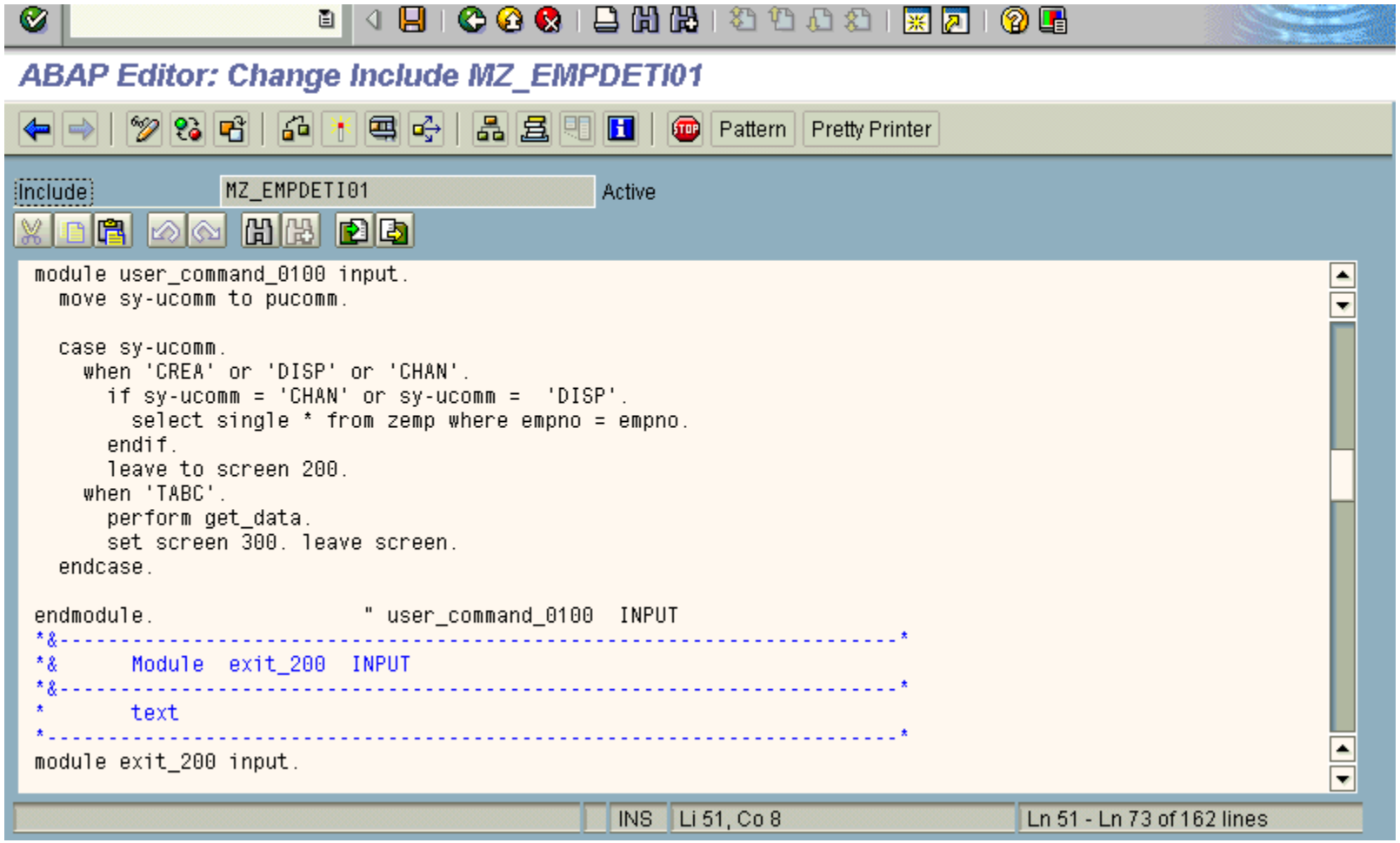
Enter the Internal or Table or field name on the Table/Field name and click on the <u>Get from program</u>



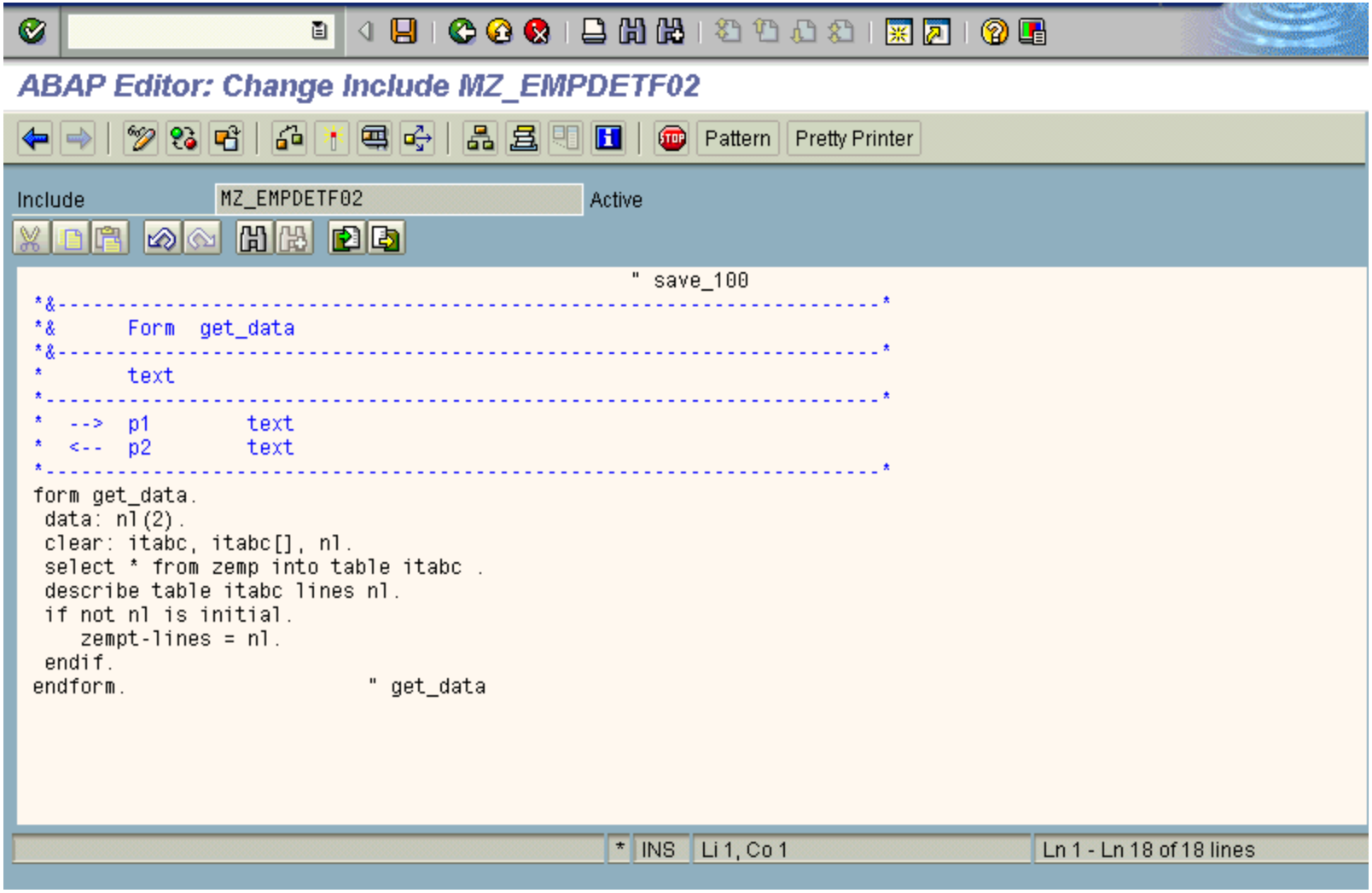
Using the attribute button of the table control you can put vertical and horizontal separators an resizing if required and name the table control



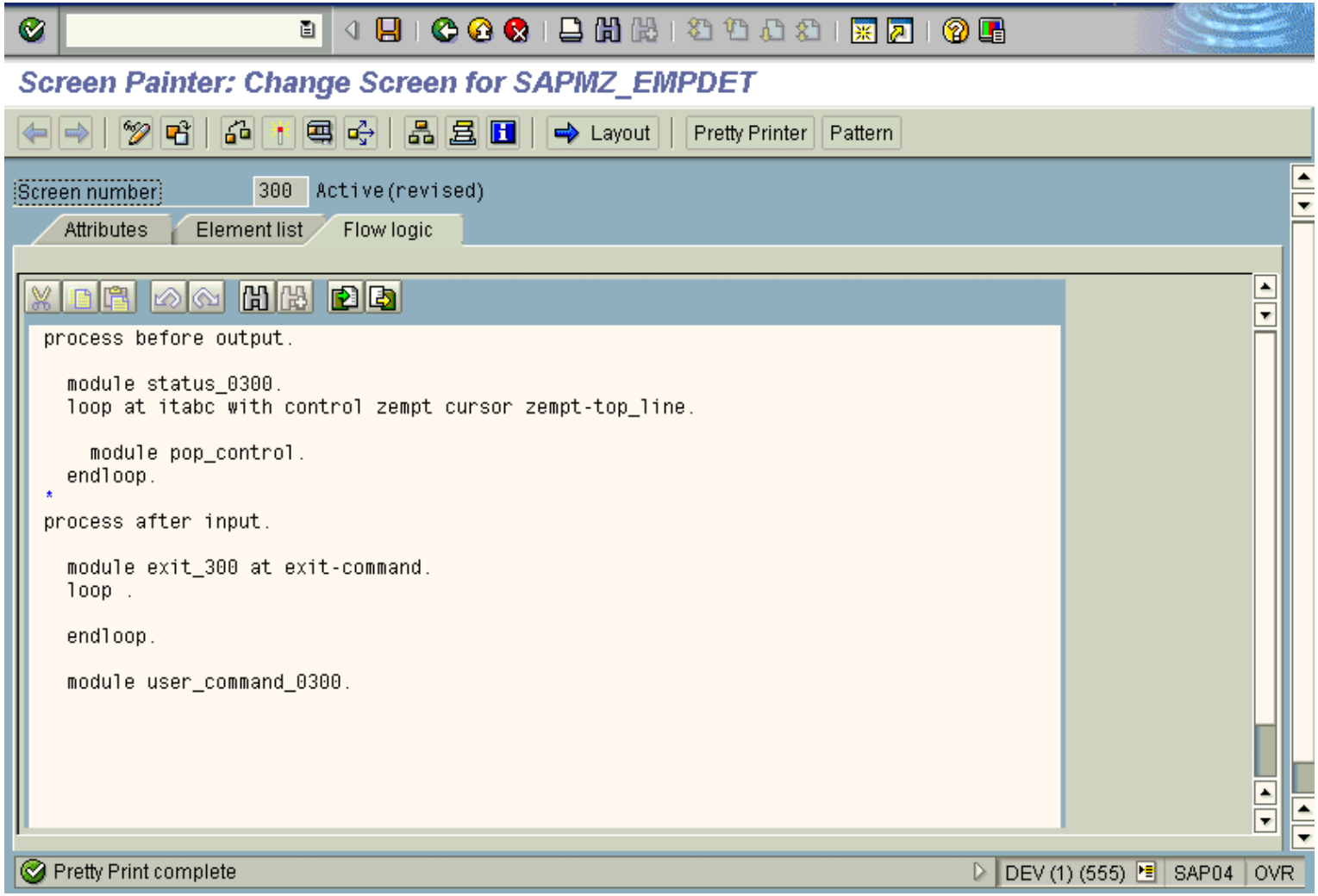
Now we have to write code for the data retrieval and populate the internal table. In the PAI of the screen 100 we are branching it to our table control screen i.e. 300.



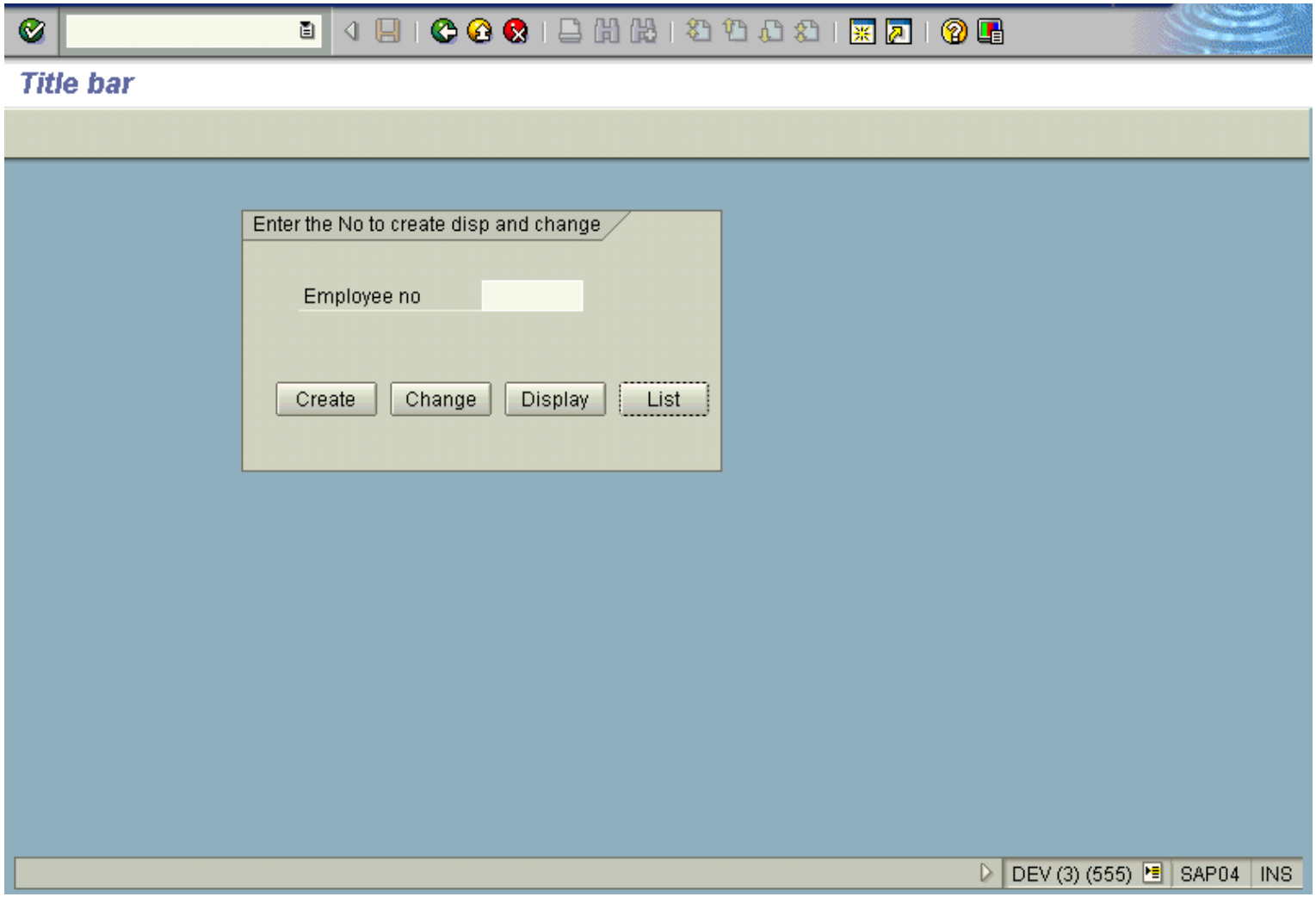
We write code to retrieve Data.



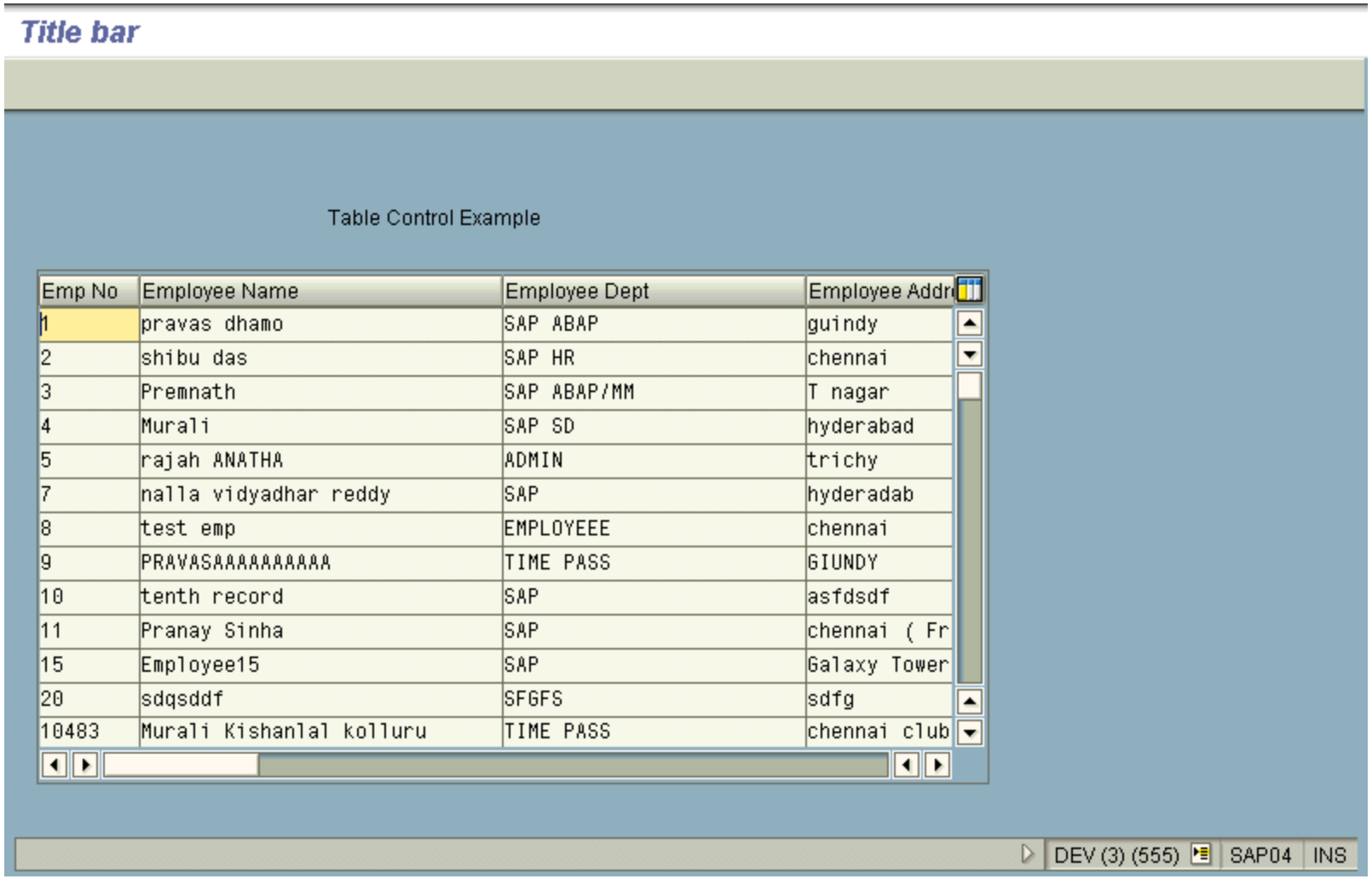
Now using the PBO of the screen 300 flow logic we populate the table control



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.



Summary

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