

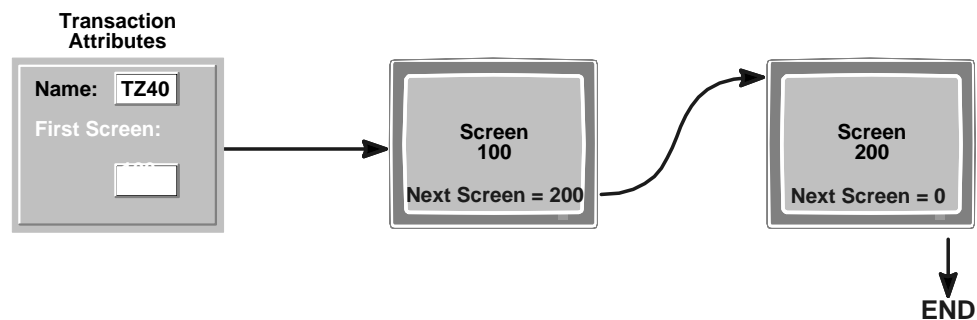
Chapter 31: Controlling the Screen Flow

Overview

For the user, a transaction is a series of screens that appear one after another. In the transaction program, screens are chained together by a series of “next-screen” numbers. When you define the transaction, you specify the number of the first screen. Then, for each screen in the transaction, you can specify a “next screen” statically or dynamically:

- Static screen pointers

When you define a screen, you specify a *Next screen* attribute for it. This attribute gives the name of the screen that is to follow the current screen by default. However, the static attribute is overridden whenever a “next-screen” is set dynamically.



Static Screen Chain

- Dynamic screen sequence

Any screen can set its own “next screen” as part of screen processing. The ABAP/4 commands for doing this are SET SCREEN and CALL SCREEN. When you set screens dynamically, you can string them together one after the other (as in a chain), or insert groups of them into the current chain.

The following topics provide information on handling screens in a transaction:

Introduction to Screen Flow Control

Setting the Next Screen

Calling a New Screen Sequence

Leaving the Current Screen

Example Transaction: Setting and Calling Screens

Processing Screens in Background

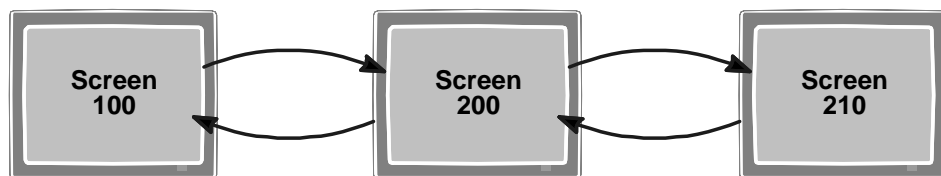
Contents

Introduction to Screen Flow Control	31-3
Setting the Next Screen.....	31-6
Calling a New Screen Sequence	31-7
Leaving the Current Screen	31-8
Processing Screens in Background	31-11

Introduction to Screen Flow Control

As an example of controlling screen flow in a transaction, look at transaction TZ40. (This transaction is in development class SDWA and is delivered with the system.) TZ40 lets users display flight information and enter updates into the display.

TZ40 uses two screens and a dialog box (popup window) for getting user updates. The transaction always displays the first two screens (numbers 100 and 200). The third (210) however only appears under certain conditions. The possible flow of screens looks as follows:



Possible Screen Flow

In practice, the user sees the following sequence:

- **Screen 100:** The user enters flight information and presses **ENTER** to request a display of flight details.

Change Flight Data

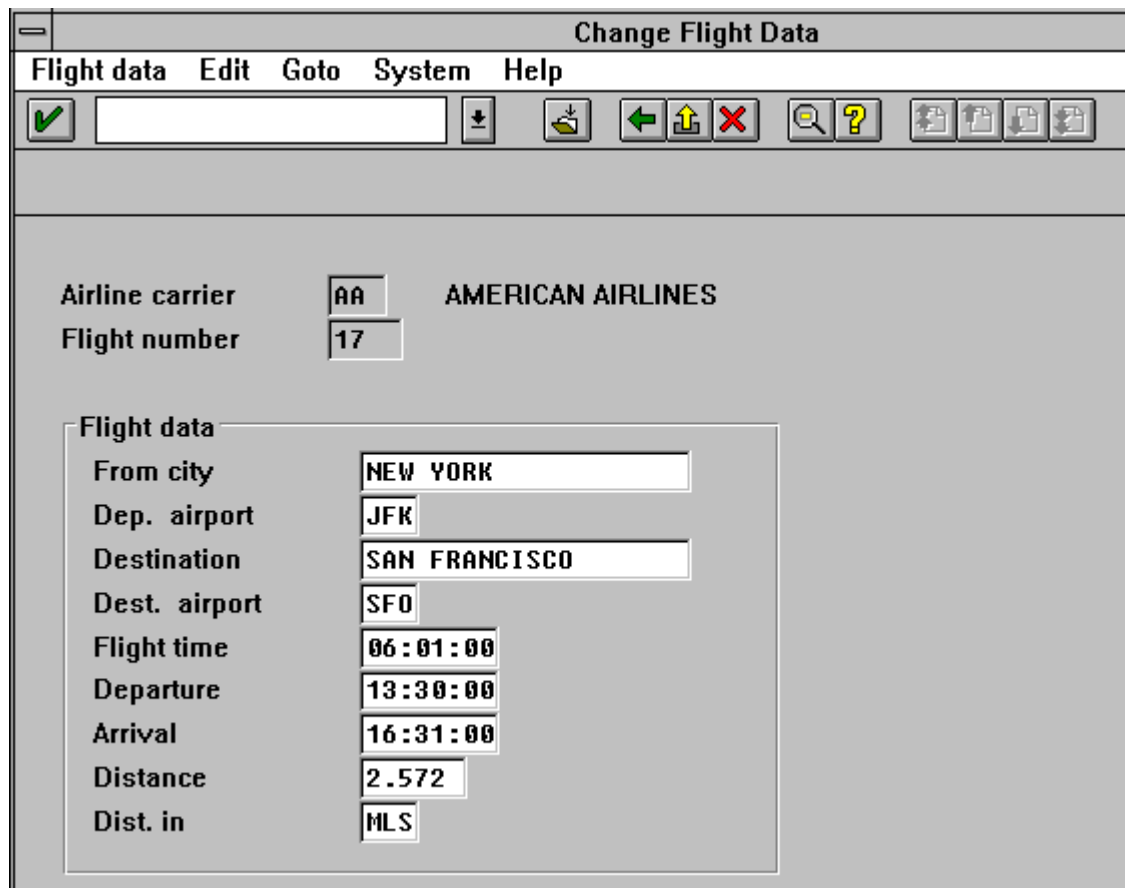
Flight data Edit Goto System Help

✓ [] [] [] [] [] [] [] []

Airline carrier ?

Flight number ?

- **Screen 200:** The system displays complete details about the flight, in update mode. The user types over the display to enter the changes.



Change Flight Data

Flight data Edit Goto System Help

Airline carrier **AA** AMERICAN AIRLINES

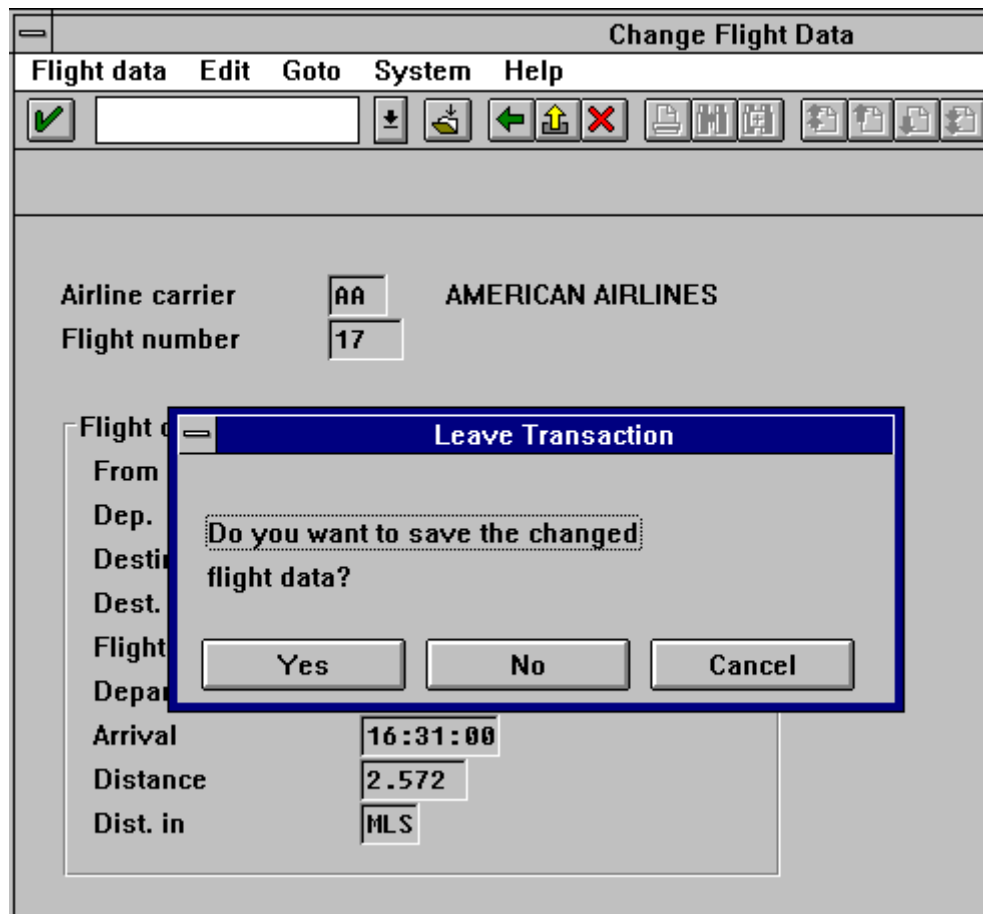
Flight number **17**

Flight data

From city	NEW YORK
Dep. airport	JFK
Destination	SAN FRANCISCO
Dest. airport	SFO
Flight time	06:01:00
Departure	13:30:00
Arrival	16:31:00
Distance	2.572
Dist. in	MLS

- **Screen 210**

Screen 210 appears only if the user tries to exit screen 200 without saving. The popup reminds the user to save the changes or cancel them (by specifying *Yes* or *No*).



To make this sequence of screens possible, transaction TZ40 must be able to call the dialog box screen conditionally.

An ABAP/4 module can “branch to” or “call” the next screen. The difference lies in where you want control to go after processing the next screen. The relevant ABAP/4 commands are:

```
SET SCREEN <screen-number>.
CALL SCREEN <screen-number>.
LEAVE SCREEN.
LEAVE TO SCREEN <screen-number>.
```

With SET SCREEN, the current screen simply specifies the next screen in the chain. Control branches to this next screen as soon as the current screen has been processed. Return from next screen to current screen is not automatic.

With CALL SCREEN, the current (calling) chain is suspended, and a next screen (or screen chain) is called in. The called screen can then return to the suspended chain with the statement LEAVE SCREEN TO SCREEN 0.

For complete information, see:

Setting the Next Screen

Calling a New Screen Sequence

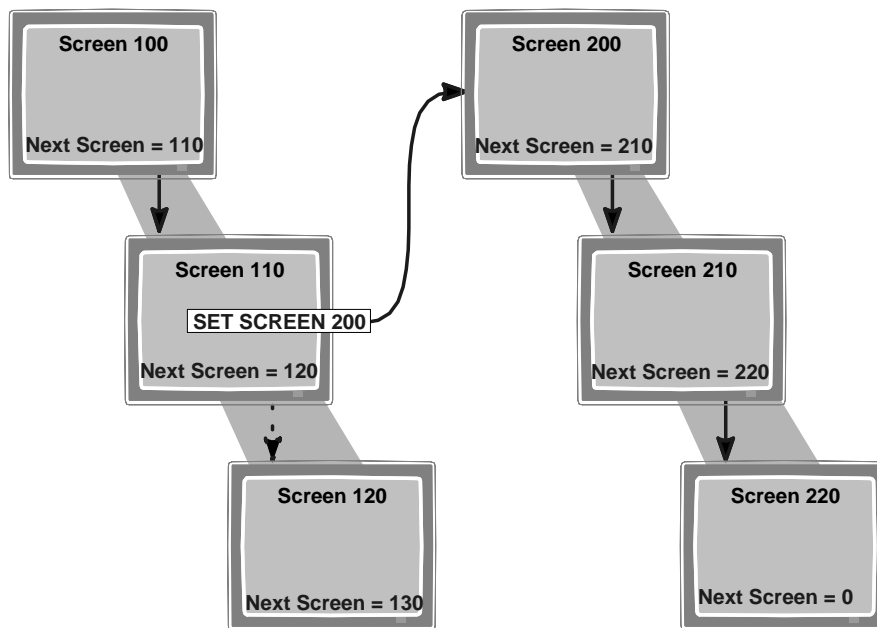
Leaving the Current Screen

Setting the Next Screen

Every screen has a static *Next screen* attribute that specifies the next screen to follow the current one. You can override this specification by using the SET SCREEN statement:

```
SET SCREEN <screen number>.
```

SET SCREEN tells the system to ignore the statically defined *Next screen* and use <screen number> as the next screen instead.



SET SCREEN

This override is temporary and has no effect on the attribute values stored in the Screen Painter.

A SET SCREEN statement merely specifies the next screen: it does not interrupt processing of the current screen. If you want to branch to the next screen without finishing the current one, use LEAVE SCREEN.



Note

Note that you can specify the next-screen number with a variable:

```
DATA: REQSCRN LIKE SY-DYNNR VALUE '100'.  
MODULE SET_NEXT_SCREEN.  
  SET SCREEN REQSCRN.  
ENDMODULE.
```

The system field SY-DYNNR always contains the number of the current screen.

Calling a New Screen Sequence

Sometimes you want to insert a screen, or a whole sequence of screens, into the course of a transaction. For instance, you might want to let an user call a popup screen from the main application screen to let them enter secondary information. After they have completed their entries, the users should be able to close the popup and return directly to the place where they left off in the main screen. There are two methods for doing this:

- use the CALL SCREEN statement

The CALL SCREEN statement lets you insert such a sequence into the current one. Using this statement is described here.

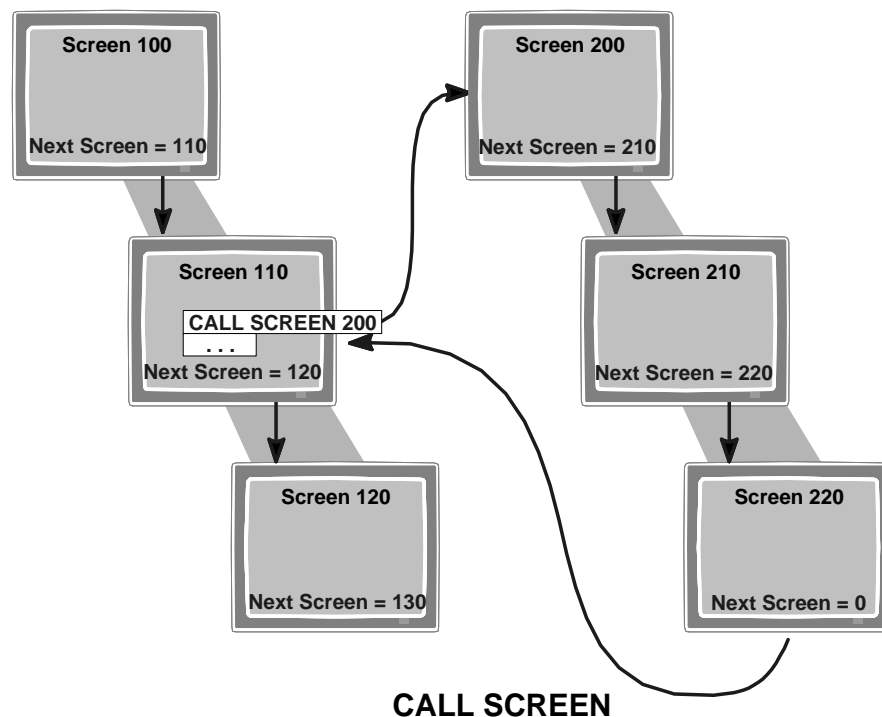
- call a dialog module

A dialog module is a callable sequence of screens that does not belong to a particular transaction. Dialog modules have their own module pools, and can be called by any transaction. For information on using dialog modules, see *Calling Dialog Modules*.

The syntax for calling up a new screen sequence is:

```
CALL SCREEN <screen number>.
```

You can think of CALL SCREEN as “stacking” a sequence, since the statement actually suspends the current sequence and starts a new one. The system continues with the new sequence until it is finished, at which point the suspended sequence is resumed. (Processing resumes with the statement directly after the CALL SCREEN.)



Leaving the Current Screen

To call a screen as a dialog box (popup), use CALL SCREEN with the options STARTING AT, ENDING AT:

```
CALL SCREEN <screen number>  
  STARTING AT <start column> <start line>  
  ENDING AT   <end column>   <end line>
```

The STARTING AT and ENDING AT options tell the system where to position the popup screen. The screen itself must be smaller than a regular screen.

In the ABAP/4 world, each stackable sequence of screens is a “call mode”. This is important because of the way you return from a given current sequence. To terminate a call mode and return to a suspended chain, set the “next screen” to 0 and leave to it:

```
LEAVE TO SCREEN 0.           or   SET SCREEN 0.  
                             LEAVE SCREEN.
```

When you return to the suspended chain, execution resumes with the statement directly following the original CALL SCREEN statement.

The original sequence of screens in a transaction is itself a calling mode. If you LEAVE TO SCREEN 0 in this sequence (that is, without having stacked any additional call modes), you return from the transaction altogether.



Note

You can have up to 9 calling modes stacked at one time.

Leaving the Current Screen

To discontinue processing for the current screen, use:

```
LEAVE TO SCREEN <screen number>.
```

or

```
SET SCREEN <number>.  
LEAVE SCREEN.
```

Both of these statements terminate processing for the current screen and go directly to <screen number>. If you use SET SCREEN without LEAVE SCREEN, the program finishes processing for the current screen before branching to <screen number>.

If you use LEAVE SCREEN without a SET SCREEN before it, you terminate the current screen and branch directly to the screen specified as the default next-screen in the screen attributes.

In “calling mode”, the special screen number 0 (LEAVE TO SCREEN 0) causes the system to jump back to the previous call level. That is, if you have called a screen sequence with CALL SCREEN, leaving to screen 0 terminates the sequence and returns to the calling screen. If you have not called a screen sequence, LEAVE TO SCREEN 0 terminates the transaction. For information on CALL SCREEN, see *Calling a New Screen Sequence*.

Example Transaction: Setting and Calling Screens

To see a complete implementation of screen flow control, it is useful to look at how transaction TZ40 (development class SDWA) is organized.

Screen Flow Logic

To demonstrate how a transaction branches to or calls a screen, look at processing for screen 200. The handling of the exit commands (function codes BACK and EXIT) shows how. When handling a BACK or EXIT function code, the PAI module must check whether flight details have changed since the screen display or last save. If so, screen 200 must call up the popup 210 to prompt about saving. The relevant parts of the screen 200's flow logic are:

```
*-----*
*   Screen 200: Flow Logic                               *
*&-----*
PROCESS AFTER INPUT.
  MODULE EXIT_0200 AT EXIT-COMMAND.
*   (...<Field checks here>...)
  MODULE USER_COMMAND_0200.
```

ABAP/4 Code

The PAI modules for screen 200 follow. Transaction TZ40 offers all the return functions (*Back*, *Exit* and *Cancel*) as exit-commands. In Screen 200, however, only the Cancel function allows immediate exit from the screen. To effect a cancel, standard exit logic is used to tell the system to go back to screen 100:

Leaving the Current Screen

```

*&-----*
*&      Module  EXIT_0200  INPUT
*&-----*
MODULE EXIT_0200 INPUT.
  CASE OK_CODE.
    WHEN 'CANC'.
      CLEAR OK_CODE.
      SET SCREEN 100.
      LEAVE SCREEN.
    ENDCASE.
  ENDMODULE.

```

All other function codes for screen 200 are handled as follows:

- The SAVE function triggers an update of the database.
- The EXIT and BACK functions trigger calls to the SAFETY_CHECK routine. This routine checks for unsaved data in the screen, and reminds the user to save if necessary.

Note the return technique. For the EXIT function, control returns from the transaction altogether (SET SCREEN 0). For the BACK function, the previous screen is set as the following screen. (SET SCREEN 100).

```

*&-----*
*&      Module  USER_COMMAND_0200  INPUT
*&-----*
MODULE USER_COMMAND_0200 INPUT.
  CASE OK_CODE.
    WHEN 'SAVE'.
      UPDATE SPFLI.
      IF SY-SUBRC = 0.
        MESSAGE S001 WITH SPFLI-CARRID SPFLI-CONNID.
      ELSE.
        MESSAGE A002 WITH SPFLI-CARRID SPFLI-CONNID.
      ENDIF.
      CLEAR OK_CODE.
    WHEN 'EXIT'.
      CLEAR OK_CODE.
      PERFORM SAFETY_CHECK USING RCODE.
      IF RCODE = 'EXIT'. SET SCREEN 0. LEAVE SCREEN. ENDIF.
    WHEN 'BACK'.
      CLEAR OK_CODE.
      PERFORM SAFETY_CHECK USING RCODE.
      IF RCODE = 'EXIT'. SET SCREEN 100. LEAVE SCREEN. ENDIF.
    ENDCASE.
  ENDMODULE.

```

Code for the SAFETY_CHECK routine follows. The CHECK statement compares current screen values to the saved screen values. If the values match, no save is needed, and the routine terminates.

If the values differ, SAFETY_CHECK calls the popup screen 210. The popup asks the user if he wants to save, and returns the answer (SAVE, EXIT and CANC) in the OK_CODE.

```

*-----*

```

```

*      Subroutine SAFETY_CHECK                                     *
*-----*
FORM SAFETY_CHECK USING RCODE.
  LOCAL OK_CODE.
  RCODE = 'EXIT'.
  CHECK SPFLI NE OLD_SPFLI.
  CLEAR OK_CODE.
  CALL SCREEN 210 STARTING AT 10 5.
  CASE OK_CODE.
    WHEN 'SAVE'. UPDATE SPFLI.
    WHEN 'EXIT'.
    WHEN 'CANC'. CLEAR SPFLI.
  ENDCASE.
ENDFORM.

```

Processing Screens in Background

You can suppress entire screens using SUPPRESS DIALOG. This command allows you to perform screen processing "in the background". The system carries out all PBO and PAI logic, but does not display the screen to the user.

Suppressing screens is useful when you are branching to list-mode from a transaction dialog step.

Use the SUPPRESS DIALOG command in the first module called from the screen's PBO logic. For example:

```

**** ABAP/4 module processing for Screen 100
      CALL SCREEN 110 STARTING AT 10 5

**** Screen 110 flow logic
PROCESS BEFORE OUTPUT.
  MODULE DIALOG_WINDOW.

**** ABAP/4 module processing
MODULE DIALOG_WINDOW OUTPUT.
  SUPPRESS DIALOG.
  LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.
  WRITE: /
  WRITE: /
ENDMODULE.

```

The SUPPRESS DIALOG statement lets you use the processing context for screen 110 as a framework for displaying the standard list output. If you don't use SUPPRESS DIALOG here, screen 110 is displayed, and is empty. When the user presses **ENTER**, the standard list output is displayed.