

Chapter 33: Using Tables in a Screen

Overview

ABAP/4 offers two mechanisms for displaying and using table data in a screen. These mechanisms are *table controls* and *step loops*. Table controls and step loops are types of screen tables you can add to a screen in the Screen Painter. For example, the following screen contains a table control at the bottom:

This chapter describes how to program the screen flow logic and ABAP/4 code that let you use screen tables.

For information on using screen tables, see:

Introduction

Using the LOOP Statement

Using Table Controls

Using Step Loops

Example Transaction: Table Controls

Contents

Introduction.....	33-2
Using the LOOP Statement	33-4
Looping Directly Through a Screen Table.....	33-4
Looping Through an Internal Table.....	33-5
Using Table Controls	33-8
Declaring Table Controls in ABAP/4.....	33-9
Setting Table Control Attributes.....	33-11
Example Transaction: Table Controls	33-13
Using Step Loops	33-16

Introduction

This section provides a quick overview of how table displays are used in a screen.

For an example of the principles outlined here, see transactions TZ60 and TZ61, which demonstrate the use of table controls and step loops, respectively. (TZ60 and TZ61 are sample transactions in development class SDWA, which is delivered with your system).

Table Controls and Step Loops

Table controls and step loops are objects for screen table display that you add to a screen in the Screen Painter. From a programming standpoint, table controls and step loops are almost exactly the same. Table controls are simply enhanced step loops that display data with the Look and Feel of a table widget in a desktop application. With table controls, the user can:

- scroll through the table vertically and horizontally
- re-size the width of a column
- scroll within a field (when field contents are wider than the field)
- select table rows or columns
- re-order the sequence of columns
- save the current display settings for future use

Table controls also offer special formatting features (some automatic, some optional) that make tables easier to look at and use. These are for example:

- automatic table resizing (vertical and horizontal) when the user resizes the window
- separator lines between rows and between columns (vertical and horizontal)
- column header fields for all columns

One 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. (Table control rows are scrollable.)

In general, many of the features provided by the table control are handled locally by your system's SAPGUI frontend. You don't have to program any of these features (except vertical scrolling) in your ABAP/4 transaction.

Screen Table Processing

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. Screen tables are either table controls or step loops. A table control displaying flight data might look as follows:

	Date	FlgtPrice	Curr.	Plane type	Capacity	Occupied
	30.01.1995	689,66	USD	747-400	660	10
	01.02.1995	689,66	USD	747-400	660	20
	01.06.1995	689,66	USD	747-400	660	38
	04.06.1995	689,66	USD	747-400	660	38

Screen Tables and the LOOP Dynpro Statement

You process a screen table by looping through it as you would through the rows of a internal table. To do this, you place a LOOP...ENDLOOP dynpro statement in the screen's flow logic. This loop usually contains a call to an ABAP/4 module, but other flow logic commands are also allowed. The system executes the module with each pass through the loop.

The LOOP dynpro statement has a variety of forms. The two most important forms let you:

- loop through a screen table alone
- loop through a screen table and an internal table in parallel

Screen Tables and Program Fields

You can declare screen table fields as database fields, internal table fields, structure fields, or other program fields. Screen table fields appear once in the field-list for the screen, and once in your program. Thus, all rows in a screen table share the same set of fields (like a "header area") in your program. During a LOOP in the flow logic, the system copies all fields for a screen table row into or out of the relevant program fields.

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. At the very least, an empty LOOP...ENDLOOP must be there.

The LOOP statement also drives scrolling. At PBO, LOOP uses a parameter that tells where in the table to start looping. This parameter thus causes the update of the next screen table display. (For table controls, the parameter is the TOP_LINE field in the table control structure. For step loops, it is the CURSOR parameter to the LOOP statement.) Both the ABAP/4 program and the system can set this parameter.

Note that the number of rows displayed in the screen table can change. This happens when the screen table is resizable and the user changes the height of the window. In this case, the next LOOP at PAI alters the number of table rows passed to the ABAP/4 program at PAI.

Using the LOOP Statement

The LOOP...ENDLOOP dynpro command lets you perform looping operations in the flow logic. You can use this statement to loop through both table controls and step loops. Between a LOOP and its ENDLOOP, you can use the FIELD, MODULE, SELECT, VALUES and CHAIN dynpro keywords. Most often, you use the MODULE statement to call an ABAP/4 module.

You *must* code a LOOP statement in both the PBO and PAI events for each table in your screen. This is because the LOOP statement causes the screen fields to be copied back and forth between the ABAP/4 program and the screen field. For this reason, at least an empty LOOP...ENDLOOP must be there.

There are two important forms of the LOOP statement:

- **LOOP.**

This statement loops through screen table rows, transferring the data in each block to and from the corresponding ABAP/4 fields in your program. The screen table fields may be declared in ABAP/4 as anything (database table, structure or individual fields) except as internal table fields.

With step loops, if you are implementing your own scrolling (for example, with **F21-F24**) you must use this statement.

- **LOOP AT <internal table>.**

This statement loops through an internal table and the screen table rows in parallel. The screen table fields often are, but need not be, declared as internal table fields.

With this LOOP, step loop displays appears with scroll bars. This scrolling is handled automatically by the system.

For more details on the different LOOP statements, see:

Looping Directly Through a Screen Table

Looping Through an Internal Table

Looping Directly Through a Screen Table

Use the simple form of the LOOP statement

```
LOOP.  
...<actions>...  
ENDLOOP.
```

to loop through the currently displayed rows of a screen table. If you are using a table control, include the additional WITH CONTROL parameter:

```
LOOP WITH CONTROL <table-control>.  
...<actions>...  
ENDLOOP.
```

This simple LOOP is the most general form of the LOOP statement. If you use this LOOP, you can declare the screen table fields as any type (internal table, database table, structure or individual fields). The simple LOOP copies the screen table fields back and forth to the relevant ABAP/4 fields. If you want to manipulate the screen values in a different structure, you must explicitly move them to where you want them.

Each pass through the loop places the next table row in the ABAP/4 fields, and executes the LOOP <actions> (usually ABAP/4 module calls) for it. In a PBO event, the LOOP statement causes loop fields in the program to be copied row by row to the screen. In a PAI event, the fields are copied row by row to the relevant program fields.

In an ABAP/4 module, use the system variable SY-STEPL to find out the index of the screen table row that is currently being processed. The system sets this variable each time through the loop. SY-STEPL always has values from 1 to the number of rows currently displayed. This is useful when you are transferring field values back and forth between a screen table and an internal table. You can declare a table-offset variable in your program (often called BASE, and usually initialized with SY-LOOPC) and use it with SY-STEPL to get the internal table row that corresponds to the current screen table row.

(In the example below, the screen fields are declared as an internal table. The program reads or modifies the internal table to get table fields passed back and forth to the screen.)



Example

```
***SCREEN FLOW LOGIC***
PROCESS BEFORE OUTPUT.
  LOOP.
    MODULE READ_INTTAB.
  ENDLOOP.

PROCESS AFTER INPUT.
  LOOP.
    MODULE MODIFY_INTTAB.
  ENDLOOP.

***ABAP/4 MODULES***
MODULE READ_INTTAB.
  IND = BASE + SY-STEPL - 1.
  READ TABLE INTTAB INDEX IND.
ENDMODULE.

MODULE MODIFY_INTTAB.
  IND = BASE + SY-STEPL - 1.
  MODIFY INTTAB INDEX IND.
ENDMODULE.
```



Note

Remember that the system variable SY-STEPL only has a meaning within the confines of LOOP...ENDLOOP processing. Outside the loop, it has no valid value.

Looping Through an Internal Table

The statement

```
LOOP AT <internal table>.
```

loops through an internal table and a screen table in parallel. In particular, LOOP AT loops through the portion of the internal table that is currently visible in the screen. You can use this form of the LOOP statement for both table controls and step loops.

The complete syntax for this form of the LOOP statement is:

```

LOOP AT <internal table> CURSOR <scroll-var>
                                [WITH CONTROL <table-control>]
                                [FROM <line1>] [TO <line2>].
...<actions>...
ENDLOOP.

```

This form of LOOP loops through the internal table, performing <actions> for each row. For each internal table row, the system transfers the relevant program fields to or from the corresponding screen table row.

When using step-loops, omit the CURSOR parameter in the PAI event. The FROM and TO parameters are only possible with step loops. (See *Using Step Loops* .) The WITH CONTROL parameter is only for use with table controls.

The following topics provide more information:

How the System Transfers Data Values

Scrolling and the Scroll Variables

How the System Transfers Data Values

With the LOOP AT <internal table> statement, the screen table need not be declared as the internal table. Screen tables can also be database tables, structures or other program fields. If you don't define the screen table as an internal table, you must make sure that the correct fields are moved to and from the internal table header as needed during looping.

Note also that the fields you use in the screen table may be only a subset of the corresponding dictionary or internal table fields. The system transfers fields as needed in the screen table. Processing (in detail) is as follows:

- **PBO processing**

1. A row of the internal table is placed in the header area.
2. All loop statements are executed for that row.

Loop statements are most often calls to ABAP/4 modules. Where necessary, these modules should move the internal table fields to the relevant program fields (dictionary table or other fields).

Example transaction TZ60 does this in the DISPLAY_FLIGHTS routine:

```

MODULE DISPLAY_FLIGHTS OUTPUT.
  SFLIGHT-FLDATE      = INT_FLIGHTS-FLDATE.
  SFLIGHT-PRICE       = INT_FLIGHTS-PRICE.
  SFLIGHT-CURRENCY    = INT_FLIGHTS-CURRENCY.
  SFLIGHT-PLANETYPE   = INT_FLIGHTS-PLANETYPE.
  SFLIGHT-SEATSMAX    = INT_FLIGHTS-SEATSMAX.
  SFLIGHT-SEATSOCC    = INT_FLIGHTS-SEATSOCC.
ENDMODULE.

```

3. The system transfers values from the program fields to the screen fields with the same names.

- **PAI processing**

1. A row of the internal table is placed to the internal table header area.

2. All screen table fields are transferred to the ABAP/4 program fields with the same names.
3. All loop <actions> are executed for the current internal table row.
Again, <actions> is usually a call to an ABAP/4 module. Where necessary, this module should first move the program field values to the header area for the internal table. This step over-writes data values from the internal table with those from the screen. Step 1 is necessary for cases where the screen table fields are only a subset of the fields defined for the internal table. If you want to update the internal table with new screen values, you must have the original table row as a basis in the header area.

Remember: If you want to update the internal table with the contents of the header area, use the MODIFY statement in the ABAP/4 module. The system does not automatically make this update for you.

**Note**

Note that there is a dynpro-language MODIFY statement and an ABAP/4 MODIFY statement. Do not use the dynpro-language MODIFY to update internal tables.

Scrolling and the Scroll Variables

Scrolling with the LOOP AT <internal table> statement can be either automatic or programmer-implemented. Scrolling with scroll bars is automatic, and managed by the system. Scrolling with function keys or other pushbuttons must be implemented by the programmer. (The example transaction TZ60 gives an example of both.)

Both for table controls and step loops, vertical scrolling is controlled by the <scroll-var> parameter. This section describes how to use this variable. Horizontal scrolling (available only with table controls) is handled locally by the SAPGUI front end. Your ABAP/4 code need not support horizontal scrolling in any way.

The first time a PBO event is processed, your program should set <scroll-var> to tell the system where to start displaying. For table controls, <scroll-var> is the TOP_LINE field in the TABLEVIEW structure. For step loops, declare a local program variable for use as the CURSOR parameter.

Thereafter, the system sets <scroll-var> in two situations:

- updates <scroll-var> whenever the user scrolls with the scroll bar
- increments <scroll-var> with each pass through a LOOP AT block

This works because when the user scrolls with the scroll bar, the system takes the following actions:

1. Triggers PAI for the current screen display.
Scrolling triggers a PAI event just as if the user had performed an action that transmitted a function code. During the PAI for scrolling with the scroll bars, the OK_CODE field has no value. (If you offer scrolling with function keys, the OK_CODE contains the function code for the function key.)
 - a. Transfers screen values for the *current* display to the ABAP/4 program.

- b. Performs PAI processing for the current screen values, using the current value of <scroll-var>. Each time through the LOOP, the system increments <scroll-var> by one, so your program can access the relevant internal table entries automatically.
(As usual, if your program detects any errors here, it should send a warning or error message. PAI processing is then repeated until no more errors are found.)
2. Triggers PBO for the new screen display (as requested by the user scrolling).
For this PBO, the system updates <scroll-var> to the new value (as set by the user scrolling) and repeats LOOP AT processing for the screen. As a result, a new segment of the internal table is transferred to the screen.



Example

```

***SCREEN FLOW LOGIC***
PROCESS BEFORE OUTPUT:
  LOOP AT INTTAB CURSOR TABCNTL-CURRENT_LINE.
  ENDLOOP.

PROCESS AFTER INPUT.
  LOOP AT INTTAB.
    MODULE MODIFY_INTTAB.
  ENDLOOP.

***ABAP/4 MODULE***
MODULE MODIFY_INTTAB.
  MODIFY INTTAB INDEX COUNT.
ENDMODULE.

```

If you want to offer other scrolling options besides scroll bars (for example, the F21-F24 scroll buttons in the standard tool bar), you must program these yourself. To do this, you can set <scroll-var> yourself where necessary. The LOOP statement in PBO then updates the screen table accordingly at the next screen display.

Using Table Controls

The structure of table controls is different from step loops. A step loop, as a screen object, is simply a series of field rows that appear as a repeating block. A table control, as a screen object, consists of

- table fields (displayed in the screen)
- a control structure that governs the table display and what the user can do with it

The field list for screen 200 (in transaction TZ60) shows both:

Screen Painter: Display Screen SAPMTZ60 0200 Field List													
Screen Edit Goto Utilities Settings System Help													
Hi	Field name	FType	Ln	Cl	Lg	VLg	Ht	Roll	Fmt.	I	O	OutO	Dic
+	FLIGHTS	Table	11	3	73	73	9	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
-	SFLIGHT-FLDATE	Text	1	1	5	5	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
-	SFLIGHT-PRICE	Text	1	2	9	9	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
-	SFLIGHT-CURRENCY	Text	1	3	5	5	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
-	SFLIGHT-PLANETYPE	Text	1	4	11	11	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
-	SFLIGHT-SEATSMAX	Text	1	5	9	9	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
-	SFLIGHT-SEATSOCC	Text	1	6	10	10	1			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Here you see the control structure (the FLIGHTS field with *Ftyp* = *Table*) and the actual table fields (the SFLIGHT fields, with *Ftyp* = *Text* or *I/O*). The attributes for the FLIGHTS field (available when you press *Attribs. for 1 Field*) show the control structure fields you can set statically. Other control structure fields can be set dynamically from the ABAP/4 program.

The following topics provide more information:

Declaring Table Controls in ABAP/4

Setting Table Control Attributes

Declaring Table Controls in ABAP/4

When you use a table control in a screen, you must declare both the table control structure and table control fields in your ABAP/4 program. For example, transaction TZ60 has:

```
TABLES:    SFLIGHT.
CONTROLS: FLIGHTS TYPE TABLEVIEW USING SCREEN 200.
```

The CONTROLS statement defines a control structure of type TABLEVIEW. The system takes the initial values for the structure from the Screen Painter attributes for the given screen.

A TABLEVIEW structure contains the following fields:

Field name	Type	Purpose
FIXED_COLS	integer	Number of immovable columns at the left end of the table. All columns after the fixed columns are movable and can be reordered in the table.
LINES	integer	The number of displayable rows in a

Using Table Controls

		table.
TOP_LINE	integer	The row of the table where the screen display starts.
CURRENT_LINE	integer	The row currently being processed inside a loop. This field is an absolute (non-relative) index, with value TOP_LINE + SY_STEPL.-1
LEFT_COL	integer	The left-most non-fixed column. Since the user can scroll the non-fixed part of the display, this field holds the number of the first column to appear after the fixed columns. LEFT_COL gives the absolute (non-relative) value of the column, regardless of whether the user has re-ordered the sequence of the columns.
LINE_SEL_MODE	integer	Enables line selection. Values: 0=None, 1=Single-selection only, 2=Multiple-selection allowed.

COL_SEL_MODE	integer	Enables column selection. Values: 0=None, 1=Single-selection only, 2=Multiple-selection allowed
LINE_SELECTOR	char 1	Indicator: displays line-selection column. This is a column of ordinary check boxes that can be examined in the ABAP/4 program. The system sets a box to X when the user clicks on it.
H_GRID	char 1	Indicator: displays horizontal gridlines
V_GRID	char 1	Indicator: displays vertical gridlines
COLS (OCCURS 10)	TAB_COLUMN	Embedded internal table: one table entry for each column in the table.

The fields in TAB_COLUMN structure describe a single field and its column in the screen table:

Field name	Type	Purpose
SCREEN	SCREEN	Embedded SCREEN structure: all the fields from a single row of the SCREEN system table.
INDEX	integer	Current position of the column in the display (in case the user has re-order the sequence of columns).
SELECTED	char 1	Set (by the system) to X when the user clicks on the column.
VISLENGTH	int1	The visible length (in characters) of the field. Lengths of up to 255 characters are allowed.

The *screen* type refers to the SCREEN table, a system table that describes all the objects in a screen. For complete information on using the SCREEN table, see: *Modifying the Screen*.

Setting Table Control Attributes

Use the fields in the TABLEVIEW control structure to set attributes for the screen table at runtime. For example, to set the scrolling variable TOP_LINE, transaction TZ60 uses the statement:

```
FLIGHTS-TOP_LINE = 1.
```

Using Table Controls

Initially, table control fields get their values from the attribute you specify for the control in the Screen Painter. Thereafter, most table control fields can be set in a variety of ways: by the system, by the ABAP/4 program, or by the user customizing a display. The following table specifies exactly how each field may be set:

Field name	Initialized by	Later values can be set
FIXED_COLS	In ABAP/4 or default=0	In ABAP/4.
LINES	In ABAP/4 or default=0	In ABAP/4. Also, set automatically by the system if you use LOOP AT <int-table>.
TOP_LINE	In ABAP/4 or default=1	In ABAP/4. Also by the system when the user moves the slider on the vertical scroll bar.
CURRENT_LINE	Default=0	No modifications in ABAP/4 allowed. Set by the system during LOOPing. (CURRENT_LINE=TOP_LINE + SY_STEPL-1)
LEFT_COL	In ABAP/4 or default=0.	In ABAP/4. Also by the system when the user moves the slider on the horizontal scroll bar.
LINE_SEL_MODE	Screen Painter or ABAP/4 program.	In ABAP/4.
COL_SEL_MODE	Screen Painter or ABAP/4 program.	In ABAP/4.
LINE_SELECTOR	Screen Painter or ABAP/4 program.	In ABAP/4.
H_GRID	Screen Painter or ABAP/4 program.	In ABAP/4.
V_GRID	Screen Painter or ABAP/4 program.	In ABAP/4.
COLS_SCREEN	Screen Painter, ABAP/4 program, or user customization	In ABAP/4.
COLS_INDEX	Screen Painter, ABAP/4 program, or user customization	In ABAP/4.
COL_SELECTED	In ABAP/4.	In ABAP/4.

If needed, you can always reset a table control to the initial values specified in the Screen Painter. To do this, use the statement `REFRESH CONTROL <table-control> FROM SCREEN <screen>`. See the online ABAP/4 keyword help for more information.

Example Transaction: Table Controls

As an example of programming with table controls, look at the code for transaction TZ60. TZ60 (development class SDWA) displays flight information using two screens. In the first, the user specifies a flight connection and requests a display. In the second (screen 200), the transaction displays all flights scheduled for the connection:

The screenshot shows the TZ60 transaction interface. At the top, there are input fields for 'Airline carrier' (AA) and 'Flight number' (17), with the text 'AMERICAN AIRLINES' displayed next to the carrier code. Below these are fields for 'From city' (NEW YORK), 'Destination' (SAN FRANCISCO), 'Flight time' (06:01:00), and 'Departure' (13:30:00). To the right, there are fields for 'Distance' (2.572) and 'Arrival' (16:31:00), with the unit 'MLS' next to the distance. At the bottom, there is a table control displaying flight data.

	Date	FlgtPrice	Curr.	Plane type	Capacity	Occupied
<input type="checkbox"/>	30.01.1995	689,66	USD	747-400	660	10
<input type="checkbox"/>	01.02.1995	689,66	USD	747-400	660	20
<input type="checkbox"/>	01.06.1995	689,66	USD	747-400	660	38
<input type="checkbox"/>	04.06.1995	689,66	USD	747-400	660	38
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						

In looking at the code, the main things to notice here are:

- how table data is fetched and passed around

The fields in the table control are declared as database fields (table SFLIGHTS) in the ABAP/4 program. To store several SFLIGHTS rows at a time, the program uses the internal table INT_FLIGHTS.

1. During PAI for screen 100, the program selects all flights for the given connection and stores them in internal table INT_FLIGHTS.
2. AT PBO for screen 200, the LOOP statement loops through INT_FLIGHTS, calling module DISPLAY_FLIGHTS for each row. DISPLAY_FLIGHTS transfers an INT_FLIGHTS row to the SFLIGHTS work area. The SFLIGHTS

Example Transaction: Table Controls

fields match the screen table names, so at the end of each loop pass, the system transfers the ABAP/4 values to the screen table fields.

- use of table control fields to direct scrolling

A table control contains information for managing scrolling. This includes fields telling how many table rows are filled and which table row should be first in the screen display into the table. For both step loops and table controls, the system manages scrolling with the scroll bars automatically. This includes scrolling the actual screen display, as well as resetting scrolling variables. However, the example program offers scrolling with the **F21-F24** keys. In this case, the program must implement these functions explicitly.

1. During PAI for screen 100, the program initializes the “first-table-row” variable (field `TOP_LINE`) to 1.
2. At PAI for screen 200, the program resets scrolling variables, especially the `TOP_LINE` field, if the user has scrolled with the **F21-F24** function keys.

Screen Flow Logic

The screen flow logic (PAI only) for screen 200 in transaction TZ60 looks as follows.

```
*-----*
*   Screen 200: Flow Logic                               *
*&-----*
PROCESS BEFORE OUTPUT.
  MODULE STATUS_0200.
  LOOP AT INT_FLIGHTS WITH CONTROL FLIGHTS
                                         CURSOR
FLIGHTS-CURRENT_LINE.
  MODULE DISPLAY_FLIGHTS.
  ENDLOOP.
*
*
PROCESS AFTER INPUT.
  LOOP AT INT_FLIGHTS.
    MODULE SET_LINE_COUNT.
  ENDLOOP.
  MODULE USER_COMMAND_0200.
```

ABAP/4 Code

The following section of the TOP-module code shows the definitions relevant to the following code.

```
*-----*
*   INCLUDE MTD40TOP                                     *
*-----*
PROGRAM SAPMTZ60 MESSAGE-ID A&.
  TABLES: SFLIGHT.
  <....Other declarations....>
  DATA INT_FLIGHTS LIKE SFLIGHT OCCURS 1 WITH HEADER LINE.

  DATA: LINE_COUNT TYPE I.

  CONTROLS: FLIGHTS TYPE TABLEVIEW USING SCREEN 200.
```

The following PBO module transfers internal table fields to the proper screen table fields at PBO.

```
*&-----*
*&      Module  DISPLAY_FLIGHTS  OUTPUT
*&-----*
MODULE DISPLAY_FLIGHTS OUTPUT.
  SFLIGHT-FLDATE      = INT_FLIGHTS-FLDATE.
  SFLIGHT-PRICE       = INT_FLIGHTS-PRICE.
  SFLIGHT-CURRENCY    = INT_FLIGHTS-CURRENCY.
  SFLIGHT-PLANETYPE   = INT_FLIGHTS-PLANETYPE.
  SFLIGHT-SEATSMAX    = INT_FLIGHTS-SEATSMAX.
  SFLIGHT-SEATSOCC    = INT_FLIGHTS-SEATSOCC.
ENDMODULE.
```

At PAI, the program must loop again, to transfer screen table fields back to the program. During this looping, the program can use SY_LOOPC to find out how many rows were transferred. SY_LOOPC is a system variable telling the total number of rows currently showing in the display.

```
*&-----*
*&      Module  SET_LINE_COUNT  INPUT
*&-----*
MODULE SET_LINE_COUNT INPUT.
  LINE-COUNT = SY-LOOPC.
ENDMODULE
```

Transaction TZ60 lets the user press function keys (F21-F24) to scroll the display. The system handles scrolling with the scroll bars automatically, but not scrolling with function keys. So PAI for screen 200 must implement function-key scrolling explicitly:

```
MODULE USER_COMMAND_0200 INPUT.
  CASE OK_CODE.
    * WHEN 'CANC'...
    * WHEN 'EXIT'...
    * WHEN 'BACK'...
    * WHEN 'NEW'...
    WHEN 'P--' .

                                                                CLEAR OK_CODE.
                                                                PERFORM PAGING USING

    'P--' .
      WHEN 'P-' .

                                                                CLEAR OK_CODE.
                                                                PERFORM PAGING USING

    'P-' .
      WHEN 'P+' .

                                                                CLEAR OK_CODE.
                                                                PERFORM PAGING USING

    'P+' .
      WHEN 'P++' .

                                                                CLEAR OK_CODE.
                                                                PERFORM PAGING USING

    'P++' .
  ENDCASE.
ENDMODULE.
```

The PAGING routine causes the system to scroll the display by re-setting the table control field TOP_LINE to a new value. The TOP_LINE field tells the LOOP statement where to start looping at PBO.

Using Step Loops

```

*&-----*
*&      Form  PAGING
*&-----*
FORM PAGING USING CODE.
  DATA: I TYPE I.

                                         J TYPE I.

  CASE CODE.
    WHEN 'P--'. FLIGHTS-TOP_LINE = 1.
    WHEN 'P-'.  FLIGHTS-TOP_LINE = FLIGHTS-TOP_LINE - LINE_COUNT.
                                         IF FLIGHTS-
TOP_LINE LE 0.
                                         FLIGHTS-
TOP_LINE = 1. ENDIF.
                                         I = FLIGHTS-TOP_LINE +
    WHEN 'P+'.
LINE_COUNT.
                                         J = FLIGHTS-
LINES - LINE_COUNT + 1.
                                         IF J LE 0.
                                         J = 1.
ENDIF.
                                         IF I LE J.
                                         FLIGHTS-
TOP_LINE = I.
                                         ELSE. FLIGHTS-
TOP_LINE = J.
                                         ENDIF.
    WHEN 'P++'.
                                         FLIGHTS-
TOP_LINE = FLIGHTS-LINES - LINE_COUNT + 1.
                                         IF FLIGHTS-
TOP_LINE LE 0.
                                         FLIGHTS-
TOP_LINE = 1. ENDIF.
  ENDCASE.
ENDFORM.

```

Using Step Loops

A step loop is a repeated series of field-blocks in a screen. Each block can contain one or more fields, and can extend over more than one line on the screen.

Step-loops as structures in a screen do not have individual names. The screen can contain more than one step-loop, but if so, you must program the LOOP...ENDLOOPS in the flow logic accordingly. The ordering of the LOOP...ENDLOOPS must exactly parallel the order of the step-loops in the screen. The ordering tells the system which loop processing to apply to which loop. Step-loops in a screen are ordered primarily by screen row, and secondarily by screen column.

Transaction TZ61 (development class SDWA) implements a step-loop version of the table you saw in transaction TZ60. Screen 200 for TZ61, shows the following step loop:

Airline carrier	AA	AMERICAN AIRLINES			
Flight number	64				
From city	SAN FRANCISCO	SFO			
Destination	NEW YORK	JFK			
Flight time	05:21:00	Distance	2.572	MLS	
Departure	09:00:00	Arrival	17:21:00		

Flgt date	FlgtPrice	Curr.	Plane type	Max. capacit	Occupied
30.01.1995	689,66	USD	A319	220	10
01.02.1995	689,66	USD	A319	220	20
01.06.1995	689,66	USD	A319	220	38
04.06.1995	689,66	USD	A319	220	38

See the TZ61 code for a demonstration of how to use step loops.

Static and Dynamic Step Loops

Step-loops fall into two classes: static and dynamic. Static step-loops have a fixed size that cannot be changed at runtime. Dynamic step-loops are variable in size. If the user re-sizes the window, the system automatically increases or decreases the number of step-loop blocks displayed. In any given screen, you can define any number of static step-loops, but only a single dynamic one.

You specify the class for a step-loop in the Screen Painter. Each loop in a screen has the attributes *Looptype* (*fixed*=static, *variable*=dynamic) and *Loopcount*. If a loop is fixed, the *Loopcount* tells the number of loop-blocks displayed for the loop. This number can never change.

Programming with static and dynamic step-loops is essentially the same. You can use both the LOOP and LOOP AT statements for both types.

Looping in a Step Loop

When you use LOOP AT <internal-table> with a step loop, the system automatically displays the step loop with vertical scroll bars. The scroll bars, and the updated (scrolled) table display, are managed by the system.

Use the following additional parameters if desired:

- FROM <line1> and TO <line2>

These parameters limit the parts of the internal table that can be displayed or processed in the step loop. If you use one or both of these, declare <line1> and

Using Step Loops

<line2> in ABAP/4 with LIKE SY-TABIX. If you don't use them, the system simply uses the beginning and/or end of the internal table as the limit.

- **CURSOR <scroll-var>**

The CURSOR parameter tells which internal table row should be the first in the screen display. <Scroll-var> is a local program variable that can be set either by your program or automatically by the system. The value of <scroll-var> is absolute with respect to the internal table (that is, not relative to the FROM or TO values).

The CURSOR <scroll-var> parameter is required in the PBO event to tell the system where to start displaying.