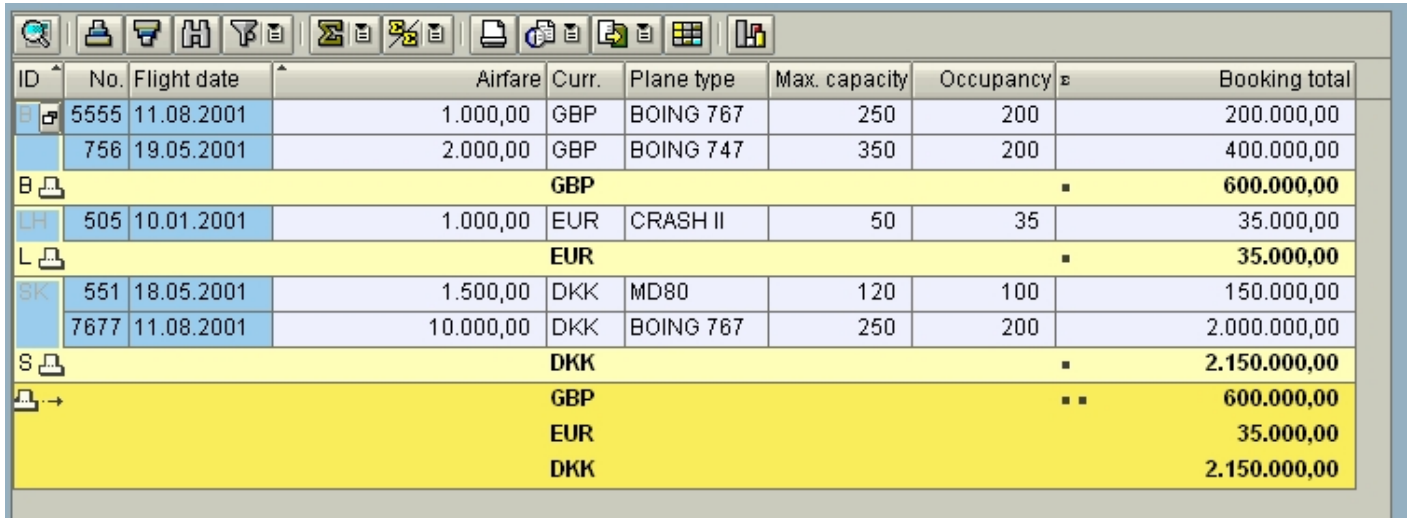
The ALV Grid control

Preface

Note that practical eaxmples of using the ALV grid can be found in development class SLIS.

Example of the fdisplay of an ALV grid:



ID	No.	Flight date	Airfare	Curr.	Plane type	Max. capacity	Occupancy	Booking total
B	5555	11.08.2001	1.000,00	GBP	BOING 767	250	200	200.000,00
	756	19.05.2001	2.000,00	GBP	BOING 747	350	200	400.000,00
B				**GBP**				**600.000,00**
LH	505	10.01.2001	1.000,00	EUR	CRASH II	50	35	35.000,00
L				**EUR**				**35.000,00**
SK	551	18.05.2001	1.500,00	DKK	MD80	120	100	150.000,00
	7677	11.08.2001	10.000,00	DKK	BOING 767	250	200	2.000.000,00
S				**DKK**				**2.150.000,00**
				GBP				**600.000,00**
				EUR				**35.000,00**
				DKK				**2.150.000,00**

Simple example of how to implement an ALV grid

Note that this example uses table ZSFLIGHT. The table is equivalent to the table SFLIGHT.

Steps:

1. Create an executable program (Report)
2. Create a screen (100) and palce a custom container named ALV_CONTAINER on the screen
3. Create a Pushbutton. Give it the text Exit and the functioncode EXIT

```
REPORT sapmz_hf_alv_grid .

TABLES: zsflight.

*---------------------------------------------------------------------
* G L O B A L   I N T E R N A L   T A B L E S
*---------------------------------------------------------------------
DATA: gi_sflight TYPE STANDARD TABLE OF sflight.

*---------------------------------------------------------------------
* G L O B A L   D A T A
*---------------------------------------------------------------------
DATA: ok_code LIKE sy-ucomm,
      g_wa_sflight LIKE sflight.
```

```
* Declare reference variables to the ALV grid and the container
DATA:
  go_grid             TYPE REF TO cl_gui_alv_grid,
  go_custom_container TYPE REF TO cl_gui_custom_container.

*----------------------------------------------------------------------
* S T A R T - O F - S E L E C T I O N.
*----------------------------------------------------------------------
START-OF-SELECTION.
  SET SCREEN '100'.

*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'EXIT'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.                 " USER_COMMAND_0100  INPUT

*&---------------------------------------------------------------------*
*&      Module  STATUS_0100  OUTPUT
*&---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
* Create objects
  IF go_custom_container IS INITIAL.
    CREATE OBJECT go_custom_container
      EXPORTING container_name = 'ALV_CONTAINER'.

    CREATE OBJECT go_grid
      EXPORTING
        i_parent = go_custom_container.

    PERFORM load_data_into_grid.
  ENDIF.
ENDMODULE.                 " STATUS_0100  OUTPUT
*&---------------------------------------------------------------------*
*&      Form  load_data_into_grid
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM load_data_into_grid.
* Read data from table SFLIGHT
  SELECT *
    FROM zsflight
    INTO TABLE gi_sflight.
```

```
  * Load data into the grid and display them
    CALL METHOD go_grid->set_table_for_first_display
      EXPORTING i_structure_name = 'SFLIGHT'
      CHANGING  it_outtab        = gi_sflight.

  ENDFORM.                    " load_data_into_grid
```

Allow the user to save and reuse the layout

A button can be shown on the grid toolbar, allowing the user to save and reuse a layout. The button looks like this: 

See also example in SAP standard program BCALV_GRID_09.

To do this use the parameters *IS_VARIANT* and *I_SAVE* of the *set_table_for_first_display* method. Note that the IS_VARIANT parameter must have the structure *DISVARIANT*.

The I_SAVE "Options for saving layouts" parameter can have the following values:

- **U** Only user specific layouts can be saved
- **X** Only global layouts can be saved
- **A** Both user specific and global layouts can be saved
- **Space** Layouts can not be saved

Add the following code to the example:

```
FORM load_data_into_grid.
  DATA:
*   For parameter IS_VARIANT
    l_layout TYPE disvariant.


   Code.....

* Load data into the grid and display them
  l_layout-report = sy-repid.

  CALL METHOD go_grid->set_table_for_first_display
    EXPORTING i_structure_name = 'SFLIGHT'
              is_variant       = l_layout
              i_save           = 'A'
    CHANGING  it_outtab        = gi_
```

Integrate user defined functions in the grid toolbar

Posibillities:

- Replace existing functions in the toolbar or context men with user defined functions
- Add new functions to the toolbar or context menu

Note that the whole toolbar can be removed using the IT_TOOLBAR_EXCLUDING parameter of the *set_table_for_first_display* method.

See also example in SAP standard program BCALV_GRID_05

1) To get access to the icons insert the following statement in the top of the program:

TYPE-POOLS: icon.

2) To allow the declaration of o_event_receiver before the lcl_event_receiver class is defined, declare it as deferred in the start of the program

```
* To allow the declaration of o_event_receiver before the lcl_event_receiver class is defined, declare it as deferred in the
* start of the program
CLASS lcl_event_receiver DEFINITION DEFERRED.
```

3) Declare reference to the event handler class

```
DATA:
o_event_receiver TYPE REF TO lcl_event_receiver.
```

4) Class for event receiver. This class adds the new button to the toolbar and handles the event when the button is pushed

```
CLASS lcl_event_receiver DEFINITION.
  PUBLIC SECTION.
    METHODS:
     handle_toolbar FOR EVENT toolbar OF cl_gui_alv_grid
       IMPORTING
         e_object e_interactive,

     handle_user_command FOR EVENT user_command OF cl_gui_alv_grid
       IMPORTING e_ucomm.
ENDCLASS.

*----------------------------------------------------------------------*
*       CLASS lcl_event_receiver IMPLEMENTATION
*----------------------------------------------------------------------*
CLASS lcl_event_receiver IMPLEMENTATION.
  METHOD handle_toolbar.
*  Event handler method for event toolbar.

    CONSTANTS:
*  Constants for button type
```

```
      c_button_normal           TYPE i VALUE 0,
      c_menu_and_default_button TYPE i VALUE 1,
      c_menu                    TYPE i VALUE 2,
      c_separator               TYPE i VALUE 3,
      c_radio_button            TYPE i VALUE 4,
      c_checkbox                TYPE i VALUE 5,
      c_menu_entry              TYPE i VALUE 6.

    DATA:
       ls_toolbar  TYPE stb_button.

*   Append seperator to the normal toolbar
    CLEAR ls_toolbar.
    MOVE c_separator TO ls_toolbar-butn_type..
    APPEND ls_toolbar TO e_object->mt_toolbar.

*   Append a new button that to the toolbar. Use E_OBJECT of
*   event toolbar. E_OBJECT is of type CL_ALV_EVENT_TOOLBAR_SET.
*   This class has one attribute MT_TOOLBAR which is of table type
*   TTB_BUTTON. The structure is STB_BUTTON
    CLEAR ls_toolbar.
    MOVE 'CHANGE'        TO ls_toolbar-function.
    MOVE  icon_change    TO ls_toolbar-icon.
    MOVE 'Change flight' TO ls_toolbar-quickinfo.
    MOVE 'Change'        TO ls_toolbar-text.
    MOVE ' '             TO ls_toolbar-disabled.
    APPEND ls_toolbar    TO e_object->mt_toolbar.
  ENDMETHOD.

  METHOD handle_user_command.
*   Handle own functions defined in the toolbar
    CASE e_ucomm.
      WHEN 'CHANGE'.
        LEAVE TO SCREEN 0.
    ENDCASE.

  ENDMETHOD.
```

ENDCLASS.

5) In the PBO module, crate object for event handler and set handler

```
CREATE OBJECT o_event_receiver.
SET HANDLER o_event_receiver->handle_user_command FOR go_grid.
SET HANDLER o_event_receiver->handle_toolbar FOR go_grid.
```

6) In the PBO module after `the CALL METHOD go_grid->set_table_for_first_display, raise event toolbar to show the modified toolbar`

```
CALL METHOD go_grid->set_toolbar_interactive.
```

Set focus to the grid

After *CALL METHOD go_grid->set_table_for_first_display* insert the following stament:

CALL METHOD cl_gui_control=>set_focus EXPORTING control = go_grid.

Set the title of the grid

Fill the *grid_title* field of structure *lvc_s_layo*.

Note that the structure *lvc_s_layo* can be used for to customize the grid appearance in many ways.

```
DATA:
* ALV control: Layout structure
gs_layout TYPE lvc_s_layo.

* Set grid title
gs_layout-grid_title = 'Flights'.

CALL METHOD go_grid->set_table_for_first_display
        EXPORTING i_structure_name = 'SFLIGHT'
                  is_layout        = gs_layout
        CHANGING  it_outtab        = gi_sflight.
```

Customize the appearence of the grid

The structure *lvc_s_layo* contains fields for setting graphical properties, displaying exceptions, calculating totals and enabling specific interaction options.

Fill the apporpiate fields of structure *lvc_s_layo* and insert it as a parametrer in the CALL METHOD go_grid->set_table_for_first_display. See the example under Set the title of the grid.

If you want to change apperance after list output, use the methods *get_frontend_layout* and *set_frontend_layout*.

Examples of fields in structure *lvc_s_layo:*

GRID_TITLE Setting the title of the grid

SEL_MODE. Selection mode, determines how rows can be selected. Can have the follwing values:

- **A** Multiple columns, multiple rows with selection buttons.
- **B** Simple selection, listbox, Single row/column
- **C** Multiple rows without buttons
- **D** Multiple rows with buttons and select all ICON

Setting and getting selected rows (Columns) and read line contents

You can read which rows of the grid that has been selected, and dynamic select rows of the grid using methods *get_selected_rows* and *set_selected_rows*. There are similiar methods for columns.

Note that the grid table always has the rows in the same sequence as displayed in the grid, thus you can use the index of the selected row(s) to read the information in the rows fronm the table. In the examples below the grid table is named gi_sflight.

Data declaratrion:

```
DATA:
* Internal table for indexes of selected rows
gi_index_rows TYPE lvc_t_row,
* Information about 1 row
g_selected_row LIKE lvc_s_row.
```

Example 1: Reading index of selected row(s) and using it to read the grid table

```
CALL METHOD go_grid->get_selected_rows
  IMPORTING
    et_index_rows = gi_index_rows.

DESCRIBE TABLE gi_index_rows LINES l_lines.

IF l_lines = 0.
  CALL FUNCTION 'POPUP_TO_DISPLAY_TEXT'
      EXPORTING
          textline1 = 'You must choose a valid line'.
  EXIT.
ENDIF.
```

```
  LOOP AT gi_index_rows INTO g_selected_row.
    READ TABLE gi_sflight INDEX g_selected_row-index INTO g_wa_sflight.
   ENDIF.
  ENDLOOP.
```

Example 2: Set selected row(s).

```
  DESCRIBE TABLE gi_index_rows LINES l_lines.
  IF l_lines > 0.
    CALL METHOD go_grid->set_selected_rows
       exporting
         it_index_rows = gi_index_rows.

  ENDIF.
```

Make an Exception field (= Traffic lights)

There can be defined a column in the grid for display of traffic lights. This field is of type Char 1, and canb contain the following values:

- **1** Red
- **2** Yellow
- **3** Green

The name of the traffic light field is supplied inh the *gs_layout-excp_fname* used by method*set_table_for_first_display*.

Example

```
TYPES: BEGIN OF st_sflight.
       INCLUDE STRUCTURE zsflight.
TYPES:  traffic_light TYPE c.

´TYPES: END OF st_sflight.
TYPES: tt_sflight TYPE STANDARD TABLE OF st_sflight.

DATA: gi_sflight TYPE tt_sflight.



*   Set the exception field of the table
    LOOP AT gi_sflight INTO g_wa_sflight.
      IF g_wa_sflight-paymentsum < 100000.
```

```
      g_wa_sflight-traffic_light = '1'.
    ELSEIF g_wa_sflight-paymentsum => 100000 AND
           g_wa_sflight-paymentsum < 1000000.
      g_wa_sflight-traffic_light = '2'.
    ELSE.
      g_wa_sflight-traffic_light = '3'.
    ENDIF.
    MODIFY gi_sflight FROM g_wa_sflight.
  ENDLOOP.

*   Name of the exception field (Traffic light field)
  gs_layout-excp_fname = 'TRAFFIC_LIGHT'.

*   Grid setup for first display
  CALL METHOD go_grid->set_table_for_first_display
    EXPORTING i_structure_name = 'SFLIGHT'
                        is_layout              = gs_layout
    CHANGING  it_outtab                = gi_sflight.
```

Color a line

The steps for coloring a line i the grid is much the same as making a traffic light.

```
* To color a line the structure of the  table must include a  Char 4 field  for color properties
TYPES: BEGIN OF st_sflight.
       INCLUDE STRUCTURE zsflight.
*      Field for line color
types:  line_color(4) type c.
TYPES: END OF st_sflight.

TYPES: tt_sflight TYPE STANDARD TABLE OF st_sflight.

DATA: gi_sflight TYPE tt_sflight.


* Loop trough the table to set the color properties of each line. The color properties field is
* Char 4 and the characters is set as follows:
* Char 1 = C = This is a color property
* Char 2 = 6 = Color code (1 - 7)
```

```
* Char 3 = Intensified on/of = 1 = on
* Char 4 = Inverse display = 0 = of
    LOOP AT gi_sflight INTO g_wa_sflight.
     IF g_wa_sflight-paymentsum < 100000.
       g_wa_sflight-line_color    = 'C610'.
     ENDIF.
     MODIFY gi_sflight FROM g_wa_sflight.
   ENDLOOP.


* Name of the color field
gs_layout-info_fname = 'LINE_COLOR'.

* Grid setup for first display
 CALL METHOD go_grid->set_table_for_first_display
     EXPORTING i_structure_name = 'SFLIGHT'
                        is_layout              = gs_layout
     CHANGING  it_outtab                = gi_sflight.
```

Refresh grid display

Use the grid method *REFRESH_TABLE_DISPLAY*

Example:

```
CALL METHOD go_grid->refresh_table_display.
```

Complete code for the ALV grid example

This example shows and ALV grid with flights. After selecting a line a change button can be pushed to display a change screen. After the changes have been saved, the ALV grid screen is displayed again, and the grid is updated with the changes.

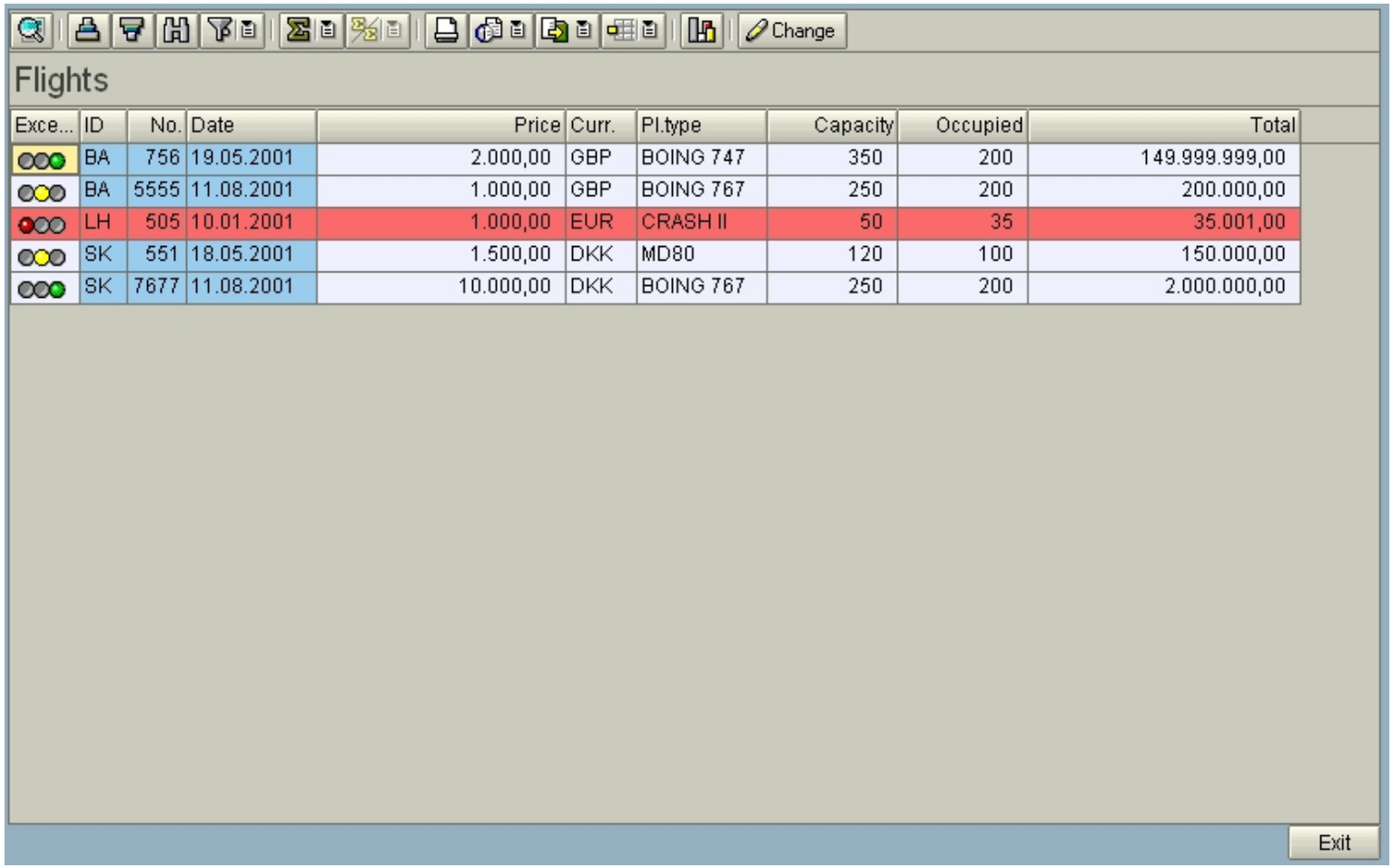
The example shows:

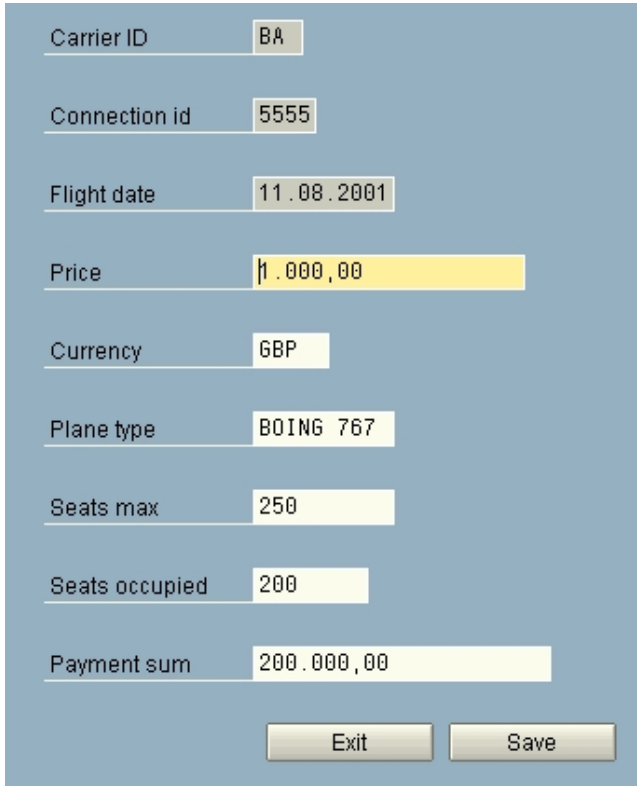
- How to setup the ALV grid
- How to ste focus to the grid
- How to set the title of the grid
- How to allow a user to save and resue a grid layout (Variant)
- How to customize the ALV grid toolbar
- Refresh the grid
- Set and get row selection and read ine contents
- Make and exception field (Traffic light)
- Coloring a line

Steps:

- Create screen 100 with the ALV grid. Remeber to include an exit button
- Add a change button to the ALV grid toolbar
- Create screen 200 the Change screen

The screens:





The code:

```
REPORT sapmz_hf_alv_grid .

* Type pool for icons - used in the toolbar
TYPE-POOLS: icon.

TABLES: zsflight.

* To allow the declaration of o_event_receiver before the
* lcl_event_receiver class is defined, decale it as deferred in the
* start of the program
CLASS lcl_event_receiver DEFINITION DEFERRED.



*---------------------------------------------------------------------
* G L O B A L   I N T E R N  A L   T A B L E S
*---------------------------------------------------------------------
*DATA: gi_sflight TYPE STANDARD TABLE OF sflight.
```

```
* To include a traffic light and/or color a line the structure of the
* table must include fields for the traffic light and/or the color
TYPES: BEGIN OF st_sflight.
        INCLUDE STRUCTURE zsflight.
*       Field for traffic light
TYPES:  traffic_light TYPE c.
*       Field for line color
types:  line_color(4) type c.
TYPES: END OF st_sflight.

TYPES: tt_sflight TYPE STANDARD TABLE OF st_sflight.

DATA: gi_sflight TYPE tt_sflight.

*---------------------------------------------------------------------
* G L O B A L   D A T A
*---------------------------------------------------------------------
DATA: ok_code         LIKE sy-ucomm,
*     Work area for internal table
      g_wa_sflight    TYPE st_sflight,
*     ALV control: Layout structure
      gs_layout       TYPE lvc_s_layo.

* Declare reference variables to the ALV grid and the container
DATA:
  go_grid             TYPE REF TO cl_gui_alv_grid,
  go_custom_container TYPE REF TO cl_gui_custom_container,
  o_event_receiver    TYPE REF TO lcl_event_receiver.

DATA:
* Work area for screen 200
  g_screen200 LIKE zsflight.

* Data for storing information about selected rows in the grid
DATA:
* Internal table
  gi_index_rows TYPE lvc_t_row,
* Information about 1 row
  g_selected_row LIKE lvc_s_row.


*---------------------------------------------------------------------
* C L A S S E S
*---------------------------------------------------------------------
CLASS lcl_event_receiver DEFINITION.
  PUBLIC SECTION.
    METHODS:
     handle_toolbar FOR EVENT toolbar OF cl_gui_alv_grid
       IMPORTING
         e_object e_interactive,
```

```
     handle_user_command FOR EVENT user_command OF cl_gui_alv_grid
       IMPORTING e_ucomm.


ENDCLASS.

*----------------------------------------------------------------------*
*       CLASS lcl_event_receiver IMPLEMENTATION
*----------------------------------------------------------------------*
CLASS lcl_event_receiver IMPLEMENTATION.

  METHOD handle_toolbar.
*  Event handler method for event toolbar.

    CONSTANTS:
*  Constants for button type
      c_button_normal           TYPE i VALUE 0,
      c_menu_and_default_button TYPE i VALUE 1,
      c_menu                    TYPE i VALUE 2,
      c_separator               TYPE i VALUE 3,
      c_radio_button            TYPE i VALUE 4,
      c_checkbox                TYPE i VALUE 5,
      c_menu_entry              TYPE i VALUE 6.

    DATA:
        ls_toolbar  TYPE stb_button.

*   Append seperator to the normal toolbar
    CLEAR ls_toolbar.
    MOVE c_separator TO ls_toolbar-butn_type..
    APPEND ls_toolbar TO e_object->mt_toolbar.

*   Append a new button that to the toolbar. Use E_OBJECT of
*   event toolbar. E_OBJECT is of type CL_ALV_EVENT_TOOLBAR_SET.
*   This class has one attribute MT_TOOLBAR which is of table type
*   TTB_BUTTON. The structure is STB_BUTTON
    CLEAR ls_toolbar.
    MOVE 'CHANGE'        TO ls_toolbar-function.
    MOVE  icon_change    TO ls_toolbar-icon.
    MOVE 'Change flight' TO ls_toolbar-quickinfo.
    MOVE 'Change'        TO ls_toolbar-text.
    MOVE ' '             TO ls_toolbar-disabled.
    APPEND ls_toolbar    TO e_object->mt_toolbar.

  ENDMETHOD.


  METHOD handle_user_command.
*   Handle own functions defined in the toolbar
    CASE e_ucomm.
```

```
      WHEN 'CHANGE'.
        PERFORM change_flight.
*         LEAVE TO SCREEN 0.
    ENDCASE.

  ENDMETHOD.

ENDCLASS.


*----------------------------------------------------------------------
* S T A R T - O F - S E L E C T I O N.
*----------------------------------------------------------------------
START-OF-SELECTION.
  SET SCREEN '100'.

*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'EXIT'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.                 " USER_COMMAND_0100  INPUT

*&---------------------------------------------------------------------*
*&      Module  STATUS_0100  OUTPUT
*&---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  DATA:
*   For parameter IS_VARIANT that is sued to set up options for storing
*   the grid layout as a variant in method set_table_for_first_display
    l_layout TYPE disvariant,
*   Utillity field
    l_lines TYPE i.

* After returning from screen 200 the line that was selected before
* going to screen 200, should be selected again. The table gi_index_rows
* was the output table from the GET_SELECTED_ROWS method in form
* CHANGE_FLIGHT
  DESCRIBE TABLE gi_index_rows LINES l_lines.
  IF l_lines > 0.
    CALL METHOD go_grid->set_selected_rows
        EXPORTING
          it_index_rows = gi_index_rows.

    CALL METHOD cl_gui_cfw=>flush.
    REFRESH gi_index_rows.
  ENDIF.
```

```
* Read data and create objects
  IF go_custom_container IS INITIAL.
*   Read data from datbase table
    PERFORM get_data.


*   Create objects for container and ALV grid
    CREATE OBJECT go_custom_container
      EXPORTING container_name = 'ALV_CONTAINER'.

    CREATE OBJECT go_grid
      EXPORTING
        i_parent = go_custom_container.

*   Create object for event_receiver class
*   and set handlers
    CREATE OBJECT o_event_receiver.
    SET HANDLER o_event_receiver->handle_user_command FOR go_grid.
    SET HANDLER o_event_receiver->handle_toolbar FOR go_grid.

*   Layout (Variant) for ALV grid
    l_layout-report = sy-repid. "Layout fo report


*----------------------------------------------------------------------
* Setup the grid layout using a variable of structure lvc_s_layo
*----------------------------------------------------------------------

*   Set grid title
    gs_layout-grid_title = 'Flights'.

*   Selection mode - Single row without buttons
*   (This is the default  mode
    gs_layout-sel_mode = 'B'.

*   Name of the exception field (Traffic light field) and the color
*   field + set the exception and color field of the table
    gs_layout-excp_fname = 'TRAFFIC_LIGHT'.
    gs_layout-info_fname = 'LINE_COLOR'.

    LOOP AT gi_sflight INTO g_wa_sflight.
      IF g_wa_sflight-paymentsum < 100000.
*       Value of traffic light field
        g_wa_sflight-traffic_light = '1'.
*       Value of color field:
*       C = Color, 6=Color 1=Intesified on, 0: Inverse display off
        g_wa_sflight-line_color    = 'C610'.
      ELSEIF g_wa_sflight-paymentsum => 100000 AND
             g_wa_sflight-paymentsum < 1000000.
        g_wa_sflight-traffic_light = '2'.
```

```
      ELSE.
        g_wa_sflight-traffic_light = '3'.
      ENDIF.
      MODIFY gi_sflight FROM g_wa_sflight.
    ENDLOOP.

*   Grid setup for first display
    CALL METHOD go_grid->set_table_for_first_display
      EXPORTING i_structure_name = 'SFLIGHT'
                is_variant       = l_layout
                i_save           = 'A'
                is_layout        = gs_layout
      CHANGING  it_outtab        = gi_sflight.

*-- End of grid setup ---------------------------------------

*   Raise event toolbar to show the modified toolbar
    CALL METHOD go_grid->set_toolbar_interactive.

*   Set focus to the grid. This is not necessary in this
*   example as there is only one control on the screen
    CALL METHOD cl_gui_control=>set_focus EXPORTING control = go_grid.


  ENDIF.
ENDMODULE.                 " STATUS_0100  OUTPUT

*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0200  INPUT
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
MODULE user_command_0200 INPUT.
  CASE ok_code.
    WHEN 'EXIT200'.
      LEAVE TO SCREEN 100.
      WHEN'SAVE'.
      PERFORM save_changes.
  ENDCASE.


ENDMODULE.                 " USER_COMMAND_0200  INPUT

*&---------------------------------------------------------------------*
*&      Form  get_data
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
FORM get_data.
* Read data from table SFLIGHT
  SELECT *
```

```
    FROM zsflight
    INTO TABLE gi_sflight.
ENDFORM.                    " load_data_into_grid
*&---------------------------------------------------------------------*
*&      Form  change_flight
*&---------------------------------------------------------------------*
*  Reads the contents of the selected row in the grid, ans transfers
*  the data to screen 200, where it can be changed and saved.
*----------------------------------------------------------------------*
FORM change_flight.
  DATA:l_lines TYPE i.

  REFRESH gi_index_rows.
  CLEAR   g_selected_row.

* Read index of selected rows
  CALL METHOD go_grid->get_selected_rows
    IMPORTING
      et_index_rows = gi_index_rows.

* Check if any row are selected at all. If not
* table  gi_index_rows will be empty
  DESCRIBE TABLE gi_index_rows LINES l_lines.
  IF l_lines = 0.
    CALL FUNCTION 'POPUP_TO_DISPLAY_TEXT'
         EXPORTING
              textline1 = 'You must choose a line'.
    EXIT.
  ENDIF.

* Read indexes of selected rows. In this example only one
* row can be selected as we are using gs_layout-sel_mode = 'B',
* so it is only ncessary to read the first entry in
* table gi_index_rows
  LOOP AT gi_index_rows INTO g_selected_row.
    IF sy-tabix = 1.
     READ TABLE gi_sflight INDEX g_selected_row-index INTO g_wa_sflight.
    ENDIF.
  ENDLOOP.

* Transfer data from the selected row to screenm 200 and show
* screen 200
  CLEAR g_screen200.
  MOVE-CORRESPONDING g_wa_sflight TO g_screen200.
  LEAVE TO SCREEN '200'.


ENDFORM.                    " change_flight
*&---------------------------------------------------------------------*
*&      Form  save_changes
*&---------------------------------------------------------------------*
```

```
* Changes made in screen 200 are written to the datbase table
* zsflight, and to the grid table gi_sflight, and the grid is
* updated with method refresh_table_display to display the changes
*---------------------------------------------------------------------*
FORM save_changes.
  DATA: l_traffic_light TYPE c.
* Update traffic light field


* Update database table
  MODIFY zsflight FROM g_screen200.

* Update grid table , traffic light field and color field.
* Note that it is necessary to use structure g_wa_sflight
* for the update, as the screen structure does not have a
* traffic light field
  MOVE-CORRESPONDING g_screen200 TO g_wa_sflight.
  IF g_wa_sflight-paymentsum < 100000.
    g_wa_sflight-traffic_light = '1'.
*   C = Color, 6=Color 1=Intesified on, 0: Inverse display off
    g_wa_sflight-line_color    = 'C610'.
  ELSEIF g_wa_sflight-paymentsum => 100000 AND
         g_wa_sflight-paymentsum < 1000000.
    g_wa_sflight-traffic_light = '2'.
    clear g_wa_sflight-line_color.
  ELSE.
    g_wa_sflight-traffic_light = '3'.
    clear g_wa_sflight-line_color.
  ENDIF.

  MODIFY gi_sflight INDEX g_selected_row-index FROM g_wa_sflight.

* Refresh grid
  CALL METHOD go_grid->refresh_table_display.
  CALL METHOD cl_gui_cfw=>flush.

  LEAVE TO SCREEN '100'.
ENDFORM.                    " save_changes
```