SDN Community Contribution
(This is not an official SAP document.)

Disclaimer & Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.
SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.
SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.

Applies To:

ALV Grid control can be used as of release 4.6C. Besides consider the following official note while utilizing this control:

> *SAP does not guarantee that the methods, events and attributes of this class that are **not** public will remain unchanged or will be available in future releases. This is why you should not derive the class to access protected objects of the class. Using these objects makes it more difficult to upgrade to subsequent releases.*

Summary

Purpose of this tutorial is to provide an easy and quick reference which may be used as a guide while coding to build lists using ALV Grid Control. Actually, there is easy-to-reach information about implementing ALV Grid lists. However, it is generally required to find the information sought in a quicker way. This tutorial handles this, being a condensed source which can be used as a "guide". It will not deal with the technical infrastructure on which ALV lays. Some of the tables are taken from the online SAP Library which is the most trustable source about the topic. Shortly, this will be a booklet summarizing major capabilities of the ALV Grid Control.

To get deep into the ALV Grid control, you can refer to the standard SAP course "*BC412 – ABAP Dialog Programming Using EnjoySAP Controls*" and the book "*Controls Technology*". Also you had better inspect demo programs.

By: Serdar ŞİMŞEKLER

Company and Title: Havelsan Inc., SAP Application Developer

Date: 24 Oct 2004

Updated on 27 June 2005
Updated on 16 August 2005

Prerequisites

To use ALV Grid Control in a simple manner, it will be sufficient just having experience on dialog programming. However, to make use of more capabilities, it is required some knowledge on object-oriented perspective of ABAP programming. A general knowledge on control framework is supposed to exist.

Table of Contents

Introduction

Here is the definition for ALV from SAPHelp:
"The ALV Grid control is a flexible tool for displaying lists. The tool provides common list operations as generic functions and can be enhanced by self-defined options."

The ALV Grid control is used to build non-hierarchical, interactive, and modern-design lists. As a control, it is a component that is installed on the local PC.

The ALV Grid control provides typical list functions as sorting, filtering, summing, etc... while also gives the opportunity to develop user functions where needed. It presents numerous interfaces like Excel Inplace and Crystal Reports.

The wrapper class implemented to encapsulate ALV Grid functionality is "CL_GUI_ALV_GRID". There is another way to display lists with ALV utilizing "REUSE_ALV..." functions. However, that way is not comprised in this tutorial.

Building Blocks

While preparing a list to be displayed via an ALV grid control, we have some basic components to prepare. These are;

i. List data: Obviously, this is the data in an internal table to be listed. Standard ALV functions except sorting makes just read access to the list data. However, sorting changes state of the internal table. The internal table holding list data may be of any flat type. Deep types are only allowed when set for some functionalities of ALV Grid.

ii. Field Catalog: We use another internal table to define specifications on how the fields of our list will be displayed. This internal table is called the "field catalog". The field catalog must comprise some technical and additional information about display options for each column to be displayed. There are three procedures to generate the field catalog as *"Automatic generation", "Semi-automatic generation", and "Manual generation"*. The internal table for the field catalog must be referenced to the dictionary type "LVC_T_FCAT".

iii. Layout Structure: We fill a structure to specify general layout options for the grid. With this structure we can set general display options, grid customizing, totals options, color adjustments etc... The layout structure must be of type "LVC_S_LAY0".

iv. Event Handler: We should define and implement an event handler class if we want to handle events triggered by the ALV Grid instance. After creating ALV Grid instance, we must register an instance of this event handler class to handle ALV Grid events.

v. Additional Data: To trigger some additional features of ALV Grid we can have some additional data to pass as parameters. For example, initial sorting criteria, buttons to be deactivated, etc...

General Scheme

Now, we can figure out a primitive scheme to prepare our ALV Grid. As a control object, ALV Grid instance requires a container to be linked to the screen. Generally, an instance of the class "`cl_gui_custom_container`" is used for this purpose. Instances of some other container classes such as "`cl_gui_docking_container`", "`cl_gui_dialogbox_container`" may also be used. In our example we take a custom container. To create a custom container instance, we need a custom control area on the screen.

Step 1→ Add a custom control on the screen which will be related to the custom container. Let's give it the name '`CC_ALV`'.

Step 2 → Declare global variables to be used for ALV Grid.

```
*-- Global data definitions for ALV
*--- ALV Grid instance reference
DATA gr_alvgrid TYPE REF TO cl_gui_alv_grid .
*--- Name of the custom control added on the screen
DATA gc_custom_control_name TYPE scrfname VALUE 'CC_ALV' .
*--- Custom container instance reference
DATA gr_ccontainer TYPE REF TO cl_gui_custom_container .
*--- Field catalog table
DATA gt_fieldcat TYPE lvc_t_fcat .
*--- Layout structure
DATA gs_layout TYPE lvc_s_layo .
```

Code Part 1 – *Global data definitions for ALV*

Step 3 → Declare your internal table which is supposed to hold the list data. Let's name it "gt_list". Here is an example declaration.

```
*--- Internal table holding list data
DATA BEGIN OF gt_list OCCURS 0 .
INCLUDE STRUCTURE SFLIGHT  .
*--In further sections, some additional fields will added here
*--for some functionality
DATA END OF gt_list .
```

Code Part 2 – *Declaration of the internal table that will hold the list data*

Step 4 → Somewhere in your program before calling list display, fill your list data as you want. Here, it is not our concern what the data are. We assume the internal table is filled reasonably. We will use the data of table SFLIGHT as our list data.

Step 5 → Call the screen which comprises the ALV Grid control. At PBO of this screen we will deal with creating the ALV Grid instance.

```
*--PBO
PROCESS BEFORE OUTPUT .
...
MODULE display_alv .
...
```

Code Part 3 – *PBO of the flow logic for the screen containing ALV Grid control*


```
...
MODULE display_alv OUTPUT .
    PERFORM display_alv .
ENDMODULE .
```

Code Part 4 – *Inside the module*

Step 6 → Now, it is high time we wrote something to play. So, this piece will be the one we will deal mainly. What we do is, checking whether an instance of the container (or ALV Grid) exists. If it exists, refreshing it, and if not, creating and setting ALV for the first display.
FORM display_alv .

```
    IF gr_alvgrid IS INITIAL .
  *----Creating custom container instance
      CREATE OBJECT gr_ccontainer
        EXPORTING
          container_name              = gc_custom_control_name
        EXCEPTIONS
          cntl_error                  = 1
          cntl_system_error           = 2
          create_error                = 3
          lifetime_error              = 4
          lifetime_dynpro_dynpro_link = 5
          others                      = 6 .
      IF sy-subrc <> 0.
  *--Exception handling
      ENDIF.
  *----Creating ALV Grid instance
      CREATE OBJECT gr_alvgrid
        EXPORTING
          i_parent          = gr_ccontainer
        EXCEPTIONS
          error_cntl_create = 1
          error_cntl_init   = 2
          error_cntl_link   = 3
          error_dp_create   = 4
          others            = 5 .
```



```
    IF sy-subrc <> 0.
    *--Exception handling
        ENDIF.

    *----Preparing field catalog.
        PERFORM prepare_field_catalog CHANGING gt_fieldcat .

    *----Preparing layout structure
        PERFORM prepare_layout CHANGING gs_layout .

    *----Here will be additional preparations
    *--e.g. initial sorting criteria, initial filtering criteria, excluding
    *--functions
        CALL METHOD gr_alvgrid->set_table_for_first_display
          EXPORTING
    *     I_BUFFER_ACTIVE               =
    *     I_CONSISTENCY_CHECK           =
    *     I_STRUCTURE_NAME              =
    *     IS_VARIANT                    =
    *     I_SAVE                        =
    *     I_DEFAULT                     = 'X'
            is_layout                     = gs_layout
    *     IS_PRINT                      =
    *     IT_SPECIAL_GROUPS             =
    *     IT_TOOLBAR_EXCLUDING          =
    *     IT_HYPERLINK                  =
          CHANGING
            it_outtab                     = gt_list[]
            it_fieldcatalog               = gt_fieldcat
    *     IT_SORT                       =
    *     IT_FILTER                     =
      EXCEPTIONS
        invalid_parameter_combination = 1
        program_error                 = 2
        too_many_lines                = 3
        OTHERS                        = 4 .
```

```
      IF sy-subrc <> 0.
*--Exception handling
       ENDIF.

ELSE .

       CALL METHOD gr_alvgrid->refresh_table_display
*  EXPORTING
*    IS_STABLE      =
*    I_SOFT_REFRESH =
        EXCEPTIONS
          finished       = 1
          OTHERS         = 2 .

       IF sy-subrc <> 0.
*--Exception handling
       ENDIF.

  ENDIF .

ENDFORM .
```

Code Part 5 – *Form checking instance existence, creating instance, setting for first display and refreshing*

From ABAP objects, we are familiar with "`CREATE OBJECT`" statement which instantiate classes. In this snippet of code, we used two instance methods of "`cl_gui_alv_grid`". First is "**`set_table_for_first_display`**" whose name implies for what it is used. After creating the ALV Grid instance we call this method to make our list displayed. We pass list data table, field catalog table, layout structure and additional information. Here are parameter definitions taken from SAP Library.

Parameter	Meaning
I_BUFFER_ACTIVE	Flag to be set by the application if the method call is static. This means the method is always called with the same field catalog. In this case, the field catalog can be held in a special buffer. This accelerates the display of small lists, in particular.
I_STRUCTURE_NAME	Name of the DDIC structure (for example, 'SFLIGHT') for the

	data in the output table. If you specify this parameter, the field catalog is generated automatically.
IS_VARIANT	Determines the layout to be used for displaying the output table. If you use this parameter, you must at least fill field **REPORT** of the structure of type **DISVARIANT**.
I_SAVE	Determines the options available to the user for saving a layout: 'X': global saving only 'U': user-specific saving only 'A': corresponds to 'X' and 'U' SPACE: no saving
I_DEFAULT	This parameter determines if the user is allowed to define default layouts: 'X': Default layouts allowed (default setting) SPACE: Default layouts not allowed If default layouts are allowed and if such a layout exists and no other layout is specified in **IS_VARIANT**, the default layout is automatically loaded when this method is called.
IS_LAYOUT	Determines properties of the grid control. The layout structure has **nothing** to do with the layout for saving filter, sort, and column properties.
IS_PRINT	Parameter for printing on the backend
IT_SPECIAL_GROUPS	If in the field catalog the columns were grouped together with field **SP_GROUP**, you must pass a table with texts for these groups. On the current layout window, it is then possible to use a list box to restrict column selection to one of these groups.
IT_TOOLBAR_EXCLUDING	This table contains function codes of the toolbar that you want to hide for the lifetime of the ALV Grid Control. The function codes are constant attributes and are prefixed with **MC_FC_**.
IT_HYPERLINK	This table assigns a hyperlink address (field **HREF** of **LVC_S_HYPE**) to each handle (field **HANDLE** of **LVC_S_HYPE**). Using this handle, you can then include hyperlinks in the grid.
IT_ALV_GRAPHICS	Settings for displaying the ALV list as a diagram (for example, axis labels). The row type of the table has two fields (variables/value pairs):

	PROP_ID: Assign a constant attribute of the class **CL_ALV_GRAPHICS_CU** with prefix **CO_PROPID_** to this field to determine the changes to be made to the graphic. Use the **CL_ALV_GRAPHICS_CU=>CO_PROPID_TITLE** attribute, for example, to refer to the title of the diagram. **PROP_VAL**: The value of the relevant topic, for example, 'My Title'.
IT_OUTTAB	Output table with the data to be displayed
IT_FIELDCATALOG	Determines the structure of the output table and the format of the data to be displayed
IT_SORT	Table with sort properties for columns that are to be sorted initially
IT_FILTER	Table with filter properties for columns for which a filter is to be set initially

Table 1 – *Interface of the method "set_table_for_first_display"*

The second method we used in our code snippet was "**refresh_table_display**" which, as implied from the name again, is used to refresh the ALV display. We do not want our ALV Grid to be created every time the PBO triggers. The first pass should create it and others should just refresh. However, we may require some changes about the appearance, layout, field catalog etc…. In that case, we will try to use other ALV methods to make changes. The parameter definition table of this method is as follows.

Parameter	Meaning
IS_STABLE	If the row or column field of this structure is set, the position of the scroll bar for the rows or columns remains stable.
I_SOFT_REFRESH	This parameter is used only in exceptional cases. If you set this parameter, any totals created, any sort order defined and any filters set for the data displayed remain unchanged when the grid control is refreshed. This makes sense, for example, if you have not modified the data of the data table and want to refresh the grid control only with regard to layout or field catalog changes.

Table 2 – *Interface of the method "`refresh_table_display`"*

OK! Seeing a simple but whole picture we are now ready to advance to build our basic components which we just point as form calls in the scheme.

Building Field Catalog

As mentioned earlier, there are three procedures to build a field catalog. The simplest way applies if our list structure is similar to a dictionary table. To do this, we simply eliminate the form call and pass the name of dictionary structure (in our example, 'SFLIGHT') to the parameter 'I_STRUCTURE_NAME'. Before explaining other procedures, let's see what a field catalog has in its structure.

Structure Of A Field Catalog

The row structure of a field catalog is of dictionary type 'LVC_S_FCAT'. There are many fields providing adjustment of display options for our list columns. Here are the basic ones.

FIELDNAME	You use this field to assign a field name of your output table to a row of the field catalog. All settings that you make in this row refer to the corresponding column of the output table.
REF_FIELD	You must fill this field if: the output table field described by the current entry in the field catalog has a corresponding field in the Data Dictionary and the field name in the output table is not identical to the field name of the field in the Data Dictionary. If the field names are identical, it is sufficient to specify the DDIC structure or table in field **REF_TABLE** of the field catalog.
REF_TABLE	You must fill this field only if the output table field described by the current entry in the field catalog has a corresponding entry in the Data Dictionary. Using this assignment, the ALV Grid Control can copy the text for the column header from the Dictionary, for example.
CHECKBOX	Outputting a checkbox. The checkbox cannot be modified by the user.
COL_POS	Relevant only if the relative column positions should not be identical to the sequence of fields in the field catalog when the list is displayed for the first time. The parameter determines the relative column position of the field for list output. The user can interactively modify the order of the columns. If this parameter is initial for each field catalog entry, the order of the columns corresponds to the sequence of fields in the field catalog.
DO_SUM	If this field is set, the ALV uses this field to calculate the total (this corresponds to the generic totals function in the toolbar.)
EMPHASIZE	If the field is set to 'X', the ALV uses a pre-defined color for highlighting the column. If it is set to 'Cxyz' (color code), the remaining numbers have the following meaning: x: color number y: intensified display on/off z: inverse display on/off

HOTSPOT	If this field is set, all cells of this column are hotspot-sensitive.
HREF_HNDL	Handle to which an URL is assigned. The ALV Grid Control displays all cells of the column as hyperlinks. You must maintain the target address of the hyperlink in a table of type **LVC_T_HYPE** and pass it using set_table_for_first_display.
KEY	If this field is set, the ALV Grid Control color-codes the column as a key field and fixes this column during horizontal scrolling. The order of the key columns in the ALV Grid Control can be modified interactively. In contrast to the SAP List Viewer, the ALV Grid Control allows you to directly hide key columns with **NO_OUT**
LOWERCASE	If this field is set, the ALV Grid Control recognizes upper/lower case in the output table. This affects the sorting of fields, for example.
NO_OUT	If you set this field, you hide the relevant column in the list. Nevertheless, the column is available in the field selection and can be interactively selected by the user as a display field. The ALV displays the contents of hidden fields on the detail screen for a row in the grid control.
NO_MERGING	If this field is set, cells with the same value are not merged into a single cell when this column is sorted.
NO_SUM	If you set this field, you lock totals calculation for the relevant field.
OUTPUTLEN	Determines the column width of the field: If the field has a reference to the Data Dictionary, you can leave the field set to its initial value. In this case, the ALV adopts the output length of the relevant domain. For fields without reference to the DDIC, you must specify the desired field output length.
STYLE	Displays all cells of this column with a style e.g. as pushbuttons. Constants “MC_STYLE…” of the class “`cl_gui_alv_grid`” can be passed to this field.
TECH	If this field is set, the relevant field is not displayed on the list and cannot be shown interactively. The field is only known in the field catalog. (For example, it must not be specified as a sorting criterion).
DECIMALS_O	If a field has no currency, then you can use this field to determine the number of decimal places to be displayed. This setting is kept even if you afterwards assign a currency field to this field or assign a currency to the **CURRENCY** field of the field catalog.
DECMFIELD	Defining the digits after the comma on a row-by-row basis. You can use an additional field in the output table to determine how many digits are to be displayed after the comma in each row.

EDIT_MASK	If you set a conversion exit (for example, conv = ' **==ALPHA** ' for function module **CONVERSION_EXIT_ALPHA_OUTPUT**), you enforce output conversion for the associated output field.
ICON	If this field is set, the column contents of the output table are output as an icon. The column contents must consist of valid icon strings (**@xx@** or **@xx\Q** <Quickinfo> **@**). You should consider the problem of printing icons.
JUST	Relevant only to fields of data type **CHAR** or **NUMC**. Justifications: 'R': right justified 'L': left justified 'C': centered How the column header is justified, depends on how the column contents are justified. You cannot justify the column header separately.
LZERO	Relevant only to fields of data type **NUMC**. In the default setting, the ALV Grid Control displays these fields right justified without leading zeros. If you set **LZERO**, leading zeros are displayed.
NO_SIGN	Relevant only to value fields. If you set **NO-SIGN**, values are displayed without signs.
NO_ZERO	If **NO_ZERO** is set, no zeros are displayed for initial value fields. The cell remains empty.
COLDDICTXT	Relevant only to fields with reference to the Data Dictionary. You use values 'L', 'M', 'S' or 'R' to determine if **SCRTEXT_L**, **SCRTEXT_M, SCRTEXT_S** or **REPTEXT** is used as the column header.
COLTEXT	Determines the column header of the column. You should assign a value to this field if it does not have a Data Dictionary reference.
REPTEXT	Relevant only to fields with reference to the Data Dictionary. For such fields, the ALV Grid Control copies the field label for the header of the corresponding data element into this field.
SCRTEXT_L	Relevant only to fields with reference to the Data Dictionary. For such fields, the ALV Grid Control copies the **long** field label of the corresponding data element into this field.
SCRTEXT_M	Relevant only to fields with reference to the Data Dictionary. For such fields, the ALV Grid Control copies the **medium** field label of the corresponding data element into this field.
SCRTEXT_S	Relevant only to fields with reference to the Data Dictionary. For such fields, the ALV Grid Control copies the **short** field label of the corresponding data element into this field.
SELDDICTXT	Relevant only to fields with reference to the Data Dictionary. You use values 'L', 'M', 'S' or 'R' to determine if **SCRTEXT_L**,

	SCRTEXT_M, **SCRTEXT_S** or **REPTEXT** is used as the text for column selection.
SELTEXT	Determines the text to be used in the column selection for the column. You should assign a value to this field if it does not have a Data Dictionary reference.
TIPDDICTXT	Relevant only to fields with reference to the Data Dictionary. You use values 'L', 'M', 'S' or 'R' to determine if **SCRTEXT_L**, **SCRTEXT_M**, **SCRTEXT_S** or **REPTEXT** is used as the tool tip.
TOOLTIP	Determines the text to be used as the tool tip for the column. You should assign a value to this field if it does not have a Data Dictionary reference.
INTTYPE	ABAP data type
SP_GROUP	You use the group key (char(4)) to group several fields together. On the dialog box for defining a layout, the user can then limit the list of hidden columns to this group.

Table 3 – *Field Catalog Fields*

Building Field Catalog Manually

The work in this procedure is just filling the internal table for the field catalog. We have seen the structure of a field catalog above. To achieve filling the field catalog correctly, one must at least fill the following fields of the field catalog structure for each column of the list:

Output table fields *with* DDIC reference	**Output table fields *without* DDIC reference**	**Explanation**
FIELDNAME	FIELDNAME	Name of the field of the internal output table
REF_TABLE		Name of the DDIC reference structure
REF_FIELD		Name of the DDIC reference field (only needed if other than **`FIELDNAME`**)
	INTTYPE	ABAP data type of the field of the internal output table
	OUTPUTLEN	Column width
	COLTEXT	Column header
	SELTEXT	Column description in column selection for layout

Table 4 – *Mandatory fields to be filled to build the field catalog manually*

```
FORM prepare_field_catalog CHANGING pt_fieldcat TYPE lvc_t_fcat .
   DATA ls_fcat type lvc_s_fcat .
   ls_fcat-fieldname = 'CARRID' .
   ls_fcat-inttype   = 'C' .
   ls_fcat-outputlen = '3' .
   ls_fcat-coltext   = 'Carrier ID' .
   ls_fcat-seltext   = 'Carrier ID' .
   APPEND ls_fcat to pt_fieldcat .

   CLEAR ls_fcat .
   ls_fcat-fieldname = 'CONNID' .
   ls_fcat-ref_table = 'SFLIGHT' .
   ls_fcat-ref_table = 'CONNID' .
   ls_fcat-outputlen = '3' .
   ls_fcat-coltext   = 'Connection ID' .
   ls_fcat-seltext   = 'Connection ID' .
   APPEND ls_fcat to pt_fieldcat .
   ENDFORM .
```

Code Part 6 – *Preparing field catalog manually*

Building Field Catalog Semi-Automatically

It is a boring work to fill and append rows for all columns of our list. And it is not flexible to proceed with automatically generating of the field catalog. Fortunately, there is a middle ground as generating the field catalog semi-automatically.

This procedure requires a function module to call. We pass the name of the structure to be the template and the function module generates a field catalog for us. After getting the generated field catalog, we loop at it and change whatever we want. The name of the function module is "**LVC_FIELDCATALOG_MERGE**". Here is an example coding to illustrate semi-automatic field catalog generation procedure.


```
FORM prepare_field_catalog CHANGING pt_fieldcat TYPE lvc_t_fcat .

   DATA ls_fcat type lvc_s_fcat .

  CALL FUNCTION 'LVC_FIELDCATALOG_MERGE'
       EXPORTING
            i_structure_name       = 'SFLIGHT'
       CHANGING
            ct_fieldcat            = pt_fieldcat[]
       EXCEPTIONS
            inconsistent_interface = 1
            program_error          = 2
            OTHERS                 = 3.

  IF sy-subrc <> 0.
*--Exception handling
  ENDIF.

  LOOP AT pt_fieldcat INTO ls_fcat .

    CASE pt_fieldcat-fieldname .
      WHEN 'CARRID'  .
        ls_fcat-outpulen = '10' .
        ls_fcat-coltext  = 'Airline Carrier ID' .
        MODIFY pt_fieldcat FROM ls_fcat .
      WHEN 'PAYMENTSUM' .
        ls_fcat-no_out   = 'X' .
        MODIFY pt_fieldcat FROM ls_fcat .
    ENDCASE .

  ENDLOOP .

ENDFORM .
```

Code Part 7 – *Preparing field catalog semi-automatically*

In the sample code above, firstly a field catalog is generated. It is filled with respect to <structure_name> (e.g. 'SFLIGHT'). After filling, we have changed the output length and column text for a column (here 'CARRID') and make another column ('PAYMENTSUM') not to be displayed.

Now that, I can hear the question "What would happen if I both pass a structure name to the parameter 'I_STRUCTURE_NAME' and a table to the parameter 'IT_FIELDCATALOG'?". Of course, the method gives a priority to one and it is the name of the structure passed to the parameter 'I_STRUCTURE_NAME' which has the priority.

More about what you can do with the field catalog will be discussed in "Additional Functionalities" sections.

Layout Adjustments

It comes now painting our ALV Grid in a general aspect. To define general appearance of our ALV Grid we fill a structure of type "LVC_S_LAYO". Here is the table containing fields and their functionalities serviced by this adjustment.

Field name	Description	Value range
CWIDTH_OPT	If this field is set, the ALV Grid Control optimizes the column width. You can then see the column header and the contents of the cells of this column.	SPACE, 'X'
SMALLTITLE	If this field is set, the title size in the grid control is set to the font size of the column header.	SPACE, 'X'
GRID_TITLE	Title between grid control and toolbar	Char string of max. 70
NO_HEADERS	If this field is set, column headers are hidden.	SPACE, 'X'
NO_HGRIDLN	If this field is set, columns are displayed without horizontal grid lines.	SPACE, 'X'
NO_MERGING	If this field is set, cells are not merged when a column is sorted.	SPACE, 'X'
NO_ROWMARK	If this field is set, the button at the beginning of a row is hidden in selection modes cell selection (SEL_MODE = 'D') and column/row selection (SEL_MODE = 'A').	SPACE, 'X'
NO_TOOLBAR	If this field is set, the toolbar is hidden.	SPACE, 'X'
NO_VGRIDLN	If this field is set, columns are displayed without vertical grid lines.	SPACE, 'X'

SEL_MODE	Set the selection mode (see table at C.4.).	SPACE, 'A', 'B', 'C', 'D'
EXCP_CONDS	If this field is set, the ALV also shows an exception in the (sub)totals line. As the color for this exception, the ALV uses the smallest exception value ('1': red, '2': yellow, '3' green) of the rows to which the (sub)total refers.	SPACE, 'X'
EXCP_FNAME	Field name of the output table for displaying an exception	Char string of max. 30
EXCP_LED	The exception is not displayed as a traffic light, but as an LED.	SPACE, 'X'
EXCP_ROLLN	Name of a data element. The F1 help for this data element is then called for the exception column. In addition, the long field label of the element is displayed as the tool tip for this column.	Char string of max. 30
CTAB_FNAME	Field name in output table for coloring cells	Char string of max. 30
INFO_FNAME	Field name in output table for coloring rows	Char string of max. 30
ZEBRA	If this field is set, the list shows a striped pattern in the print preview and when it is printed.	SPACE, 'X'
NO_TOTARR	The ALV Grid Control displays arrows in the totals line and the subtotals line that additionally indicate the totaling area. Set this parameter to suppress these arrows.	SPACE, 'X'
NO_TOTEXP	An icon displayed at the beginning of a (sub)totals line indicates whether the line has been expanded or not. Set this parameter to suppress this icon.	SPACE, 'X'
NO_TOTLINE	If this field is set, only subtotals, but no totals, are displayed.	SPACE, 'X'
NUMC_TOTAL	If this field is set, the user can calculate totals for fields of data type NUMC (normally, users are not allowed to do this).	SPACE, 'X'
TOTALS_BEF	If this field is set, the ALV displays totals calculated as the first rows in the grid control. Subtotals are displayed before a new value of the subtotals criterion.	SPACE, 'X'

DETAILINIT	If this field is set, the detail screen also shows columns with initial values.	SPACE, 'X'
DETAILTITL	Title in the title bar of the detail screen.	Char string of max. 30
S_DRAGDROP	Structure for Drag & Drop settings	
KEYHOT	If this field is set, all key fields are hotspot-sensitive. If a key field is clicked once, event hotspot_click is triggered.	SPACE, 'X'
SGL_CLK_HD	Enables the single click on column header function. This function sorts the list in ascending order when the column is clicked for the first time, and then in descending order when the column is clicked a second time.	SPACE, 'X'
STYLEFNAME	You use this field to pass the name of the cell table for displaying cells as pushbuttons.	Char string of max. 30

Table 5 – *Fields of layout structure*

```
FORM prepare_layout CHANGING ps_layout TYPE lvc_s_layo.

ps_layout-zebra = 'X' .
ps_layout-grid_title = 'Flights' .
ps_layout-smalltitle = 'X' .

ENDFORM.                    " prepare_layout
```

Code Part 8 - *Filling layout structure*

Printing Adjustments

We handle printing adjustments via a structure to be passed to the parameter "`is_print`" of the method "**`set_table_for_first_display`**". The structure is as follows:

Field name	Description	Value range
GRPCHGEDIT	Enables user-definable group change editing for the print preview mode. If this field is set, the jump to the SAP List Viewer is configured accordingly. On the sort dialog box, the user can then determine how a sorting criterion value change is indicated graphically: as a page break or as an underline. Using the sort table you can dynamically set this formatting.	SPACE, 'X'
NO_COLWOPT	The ALV Grid Control sets all columns to their optimum width before the list is printed or displayed in the print preview. If you set this parameter, this default setting is overridden.	SPACE, 'X'
PRNTLSTINF	Prints list information. If this field is set, information on sorting, subtotals and filters defined as well as data statistics are printed at the beginning of the list.	SPACE, 'X'
PRNT_TITLE	Specifies the time at which the grid title is to be printed.	0-3 with the following meaning: 0: Before the event **PRINT_TOP_OF_LIST** 1: After the event **PRINT_TOP_OF_LIST** 2: Before the event **PRINT_TOP_OF_PAGE** 3: After the event **PRINT_TOP_OF_PAGE**
RESERVELNS	Number of reserved rows for event print_end_of_page. If no number is specified, the text specified there is overwritten by the list.	Natural number

Table 6 – *Fields of the structure to handle printing properties*

The print output of the field "`PRNTLSTINF`" is not visible in the print preview of the ALV Grid. If you create a spool request first, you can check the final list layout in transaction `SP01`.

Excluding Unwanted Standard Function Buttons

In your list, you may want to exclude some of the standard function buttons since they are not useful for your list. To exclude those buttons, you fill a table of type "UI_FUNCTIONS" and pass it to the parameter "IT_TOOLBAR_EXCLUDING" of the method "**set_table_for_first_display**". The function codes for the buttons may be acquired by inspecting the constant attributes of the class "cl_gui_alv_grid" or putting a break point into a method, like the event-handling method of the event "**after_user_command**", which deals with the ALV command.

To hide the entire toolbar, you can set the field "NO_TOOLBAR" of the layout structure to 'X'.

```
FORM exclude_tb_functions CHANGING pt_exclude TYPE ui_functions .

  DATA ls_exclude TYPE ui_func.

  ls_exclude = cl_gui_alv_grid=>mc_fc_maximum .
  APPEND ls_exclude TO pt_exclude.
  ls_exclude = cl_gui_alv_grid=>mc_fc_minimum .
  APPEND ls_exclude TO pt_exclude.
  ls_exclude = cl_gui_alv_grid=>mc_fc_subtot .
  APPEND ls_exclude TO pt_exclude.
  ls_exclude = cl_gui_alv_grid=>mc_fc_sum .
  APPEND ls_exclude TO pt_exclude.
  ls_exclude = cl_gui_alv_grid=>mc_fc_average .
  APPEND ls_exclude TO pt_exclude.
  ls_exclude = cl_gui_alv_grid=>mc_mb_sum .
  APPEND ls_exclude TO pt_exclude.
  ls_exclude = cl_gui_alv_grid=>mc_mb_subtot .

ENDFORM .
```

Code Part 9 – *Filling the table to exclude unwanted standard functions*

Here, names beginning with "MC_FC_" are names for functions directly and the names beginning with "MC_MB_" are for the function menus including some subfunctions as menu entries. By excluding one from the latter type, you exclude all of the functions under it.

Non-Event Based Additional Functionalities

Up to this point, we are able to display our list in ALV Grid format. However, since requirements never end, we will want it to do more. In this section, we will go beyond the basics and get much deeper to specifically deal with additional functionalities that ALV Grid control can handle.

Changing Field Catalog or Layout after First Display

During runtime, it is possible to set a new layout or a new field catalog after first displaying of the list. These components have set/get methods to accomplish this.

```
For the field catalog   :     get_frontend_fieldcatalog
                              set_frontend_fieldcatalog
For the layout          :     get_frontend_layout
                              set_frontend_layout
```

Using these methods, anytime at execution, you can get the contents and modify them.

```
.. ..
DATA ls_fcat TYPE lvc_s_fcat .
DATA lt_fcat TYPE lvc_t_fcat .
DATA ls_layout TYPE lvc_s_layo .

      CALL METHOD gr_alvgrid->get_frontend_fieldcatalog
        IMPORTING
          et_fieldcatalog = lt_fcat[]    .

      LOOP AT lt_fcat INTO ls_fcat .
        IF ls_fcat-fieldname = 'PAYMENTSUM' .
          ls_fcat-no_out = space .
          MODIFY lt_fcat FROM ls_fcat .
        ENDIF .
      ENDLOOP .

      CALL METHOD gr_alvgrid->set_frontend_fieldcatalog
        EXPORTING
          it_fieldcatalog = lt_fcat[]    .
```



```
        CALL METHOD gr_alvgrid->get_frontend_layout
          IMPORTING
            es_layout = ls_layout    .

        ls_layout-grid_title = 'Flights (with Payment Sums)' .

        CALL METHOD gr_alvgrid->set_frontend_layout
          EXPORTING
            is_layout = ls_layout .
  .. ..
```

Code Part 10 – *Changing field catalog and layout after first display*

Setting Sort Conditions

It is possible to set sort conditions for the table data. This is achieved by filling an internal table of structure "LVC_T_SORT" which consists of the sort criteria. To have an initial sorting, pass it to the parameter "IT_SORT" of the method "**set_table_for_first_display**".

```
  FORM prepare_sort_table CHANGING pt_sort TYPE lvc_t_sort .

    DATA ls_sort TYPE lvc_s_sort .
    ls_sort-spos = '1' .
    ls_sort-fieldname = 'CARRID' .
    ls_sort-up = 'X' .  "A to Z
    ls_sort-down = space .
    APPEND ls_sort TO pt_sort .

    ls_sort-spos = '2' .
    ls_sort-fieldname = 'SEATSOCC' .
    ls_sort-up = space .
    ls_sort-down = 'X' .  "Z to A
    APPEND ls_sort TO pt_sort .

  ENDFORM.                    " prepare_sort_table
```

Code Part 11 – *Preparing the table for sorting settings*

We have two important points to tell about this topic. First one is that, be ready for a short dump if any one of the fields given to be sorted is not in the content of the field catalog. Secondly, when you make ALV Grid to sort data, by default it vertically merges fields having the same content. To avoid from this for all of the columns, you can set "no_merging" field of the layout structure to 'X'. If you want to disable merging for just some columns, set "no_merging" field of the field catalog row corresponding to that column.

You can get and set sort criteria applied whenever you want by using methods "**get_sort_criteria**" and "**set_sort_criteria**", respectively.

Filtering

The procedure is like the one in sorting. Here, the type of the table you must fill is "LVC_T_FILT". Filling this table is similar to filling a RANGES variable.

```
FORM prepare_filter_table CHANGING pt_filt TYPE lvc_t_filt .

  DATA ls_filt TYPE lvc_s_filt .

  ls_filt-fieldname = 'FLDATE' .
  ls_filt-sign = 'E' .
  ls_filt-option = 'BT' .
  ls_filt-low = '20030101' .
  ls_filt-high = '20031231' .

  APPEND ls_filt TO pt_filt .

ENDFORM.                    " prepare filter table
```

Code Part 12 – *Preparing the table for filter setting*

You can get and set filtering criteria applied whenever you want by using methods "**get_filter_criteria**" and "**set_filter_criteria**", respectively.

Making Selections

It is generally required to select some cells, rows or columns in ALV Grid. The structure of the control gives the opportunity to set different selection modes through the value of the field "SEL_MODE" in the layout structure. Here are those modes and their functionalities:

Value	Mode	Possible selections	Comment
SPACE	same as 'B'	see 'B'	Default setting
'A'	Column and row selection	Multiple columns Multiple rows	The user selects the rows through pushbuttons at the left border of the grid control.
'B'	Simple selection, list box	Multiple columns Multiple rows	
'C'	Multiple selection, list box	Multiple columns Multiple rows	
'D'	Cell selection	Multiple columns Multiple rows Any cells	The user selects the rows through pushbuttons at the left border of the grid control.

Table 7 – *Values for adjusting selection property of the ALV Grid control*

Beyond setting this option, you can set "NO_ROWMARK" option to hide the mark column which is normally visible when the selection mode allows multiple row selection.

One point to notice here is that if you set your ALV Grid as to be editable, it may override your selection mode regardless of your layout settings.

After a selection is made, the rest being important about the developer is to figure out what is selected. This will be essential if you implement your functions as interacting with selections made. Certainly, ALV Grid tells this information. You use methods:

GET_SELECTED_CELLS: This method returns the exact addresses of selected cells in a table of type "LVC_T_CELL" via the output parameter "et_cell". The ALV Grid Control returns only the indexes of cells that are selected individually. If an entire row or column is selected, the table has no information about that row. For individually selected cells, you will get the name of the column and the index of the row for each one.

GET_SELECTED_CELLS_ID: This method also returns the addresses of selected cells. The difference is, first its output type is "LVC_T_CENO" via the output parameter "et_cells" and it returns the IDs for columns and rows of selected cells.

GET_SELECTED_ROWS: For the selection mode 'A', 'C' or 'D' you can select rows. To get information about selected rows, this method is utilized. It has two output table parameters. However, the parameter "et_index_rows" is obsolete. The other parameter "et_row_no" is of type "LVC_T_ROID" and returns row indexes for selected rows (but not cells or columns) in it.

GET_SELECTED_COLUMNS: As understood so far, this method returns information about selected columns. Its output table parameter "et_index_columns" is of type "LVC_T_COL" and consist of the names of selected columns.

In your program, you may want to make some cells, rows or columns to be selected during execution. For those purposes, you can use “SET” versions of methods above whose interfaces are similar but the direction is reverse.

After a screen transition, when you come back to the screen with your ALV, your selections may be lost. You can utilize “GET” methods before transition, to backup those information and after returning to the screen, you can use “SET” methods to restore them.

Retrieving and Setting Scroll Status Info

Besides setting some parts selected, you may also want to get and set scroll status information. We have a pair of get and set methods for this purpose.
`GET_SCROLL_INFO_VIA_ID:` This method is used to retrieve scroll info. It has three output parameters which are “`es_col_info`” having column name being displayed first on the left, “`es_row_no`” having index of the row displayed first at the top, and “`es_row_info`” which is obsolete.
`SET_SCROLL_INFO_VIA_ID:` This method is used for setting the scroll status of the list. It has the same interface with the “get” version. That’s why they can be used correspondingly.

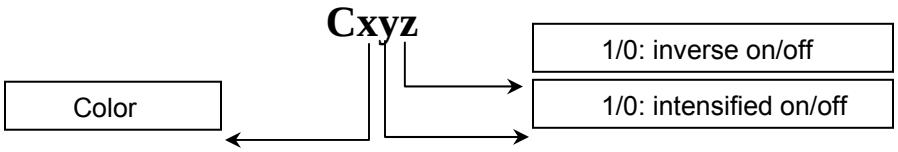
Like in selections, after a screen transition, when you come back to the screen with your ALV, the scroll information may be lost. You can utilize “GET” method before transition, to backup the scroll information and after returning to the screen, you can use “SET” method to restore it.

Coloring

It is possible to paint some cells, rows, and columns through the ALV Grid control. Basically, with no additional effort, if you set a column to be a key column it is automatically colored. To paint we have the following procedures.

Coloring an Entire Column

To make an entire column be painted with the color you want, you can use the “`emphasize`” option of the field catalog. Simply assign a color code to this field of the row added for your column. Color codes are constructed as follows:



x	Color	Intended for
1	gray-blue	headers
2	light gray	list bodies
3	yellow	totals
4	blue-green	key columns
5	green	positive threshold value
6	red	negative threshold value
7	orange	Control levels

Table 8 – *Color numbers*

The “key setting” made via the field “`key`” of the field catalog overrides this setting. So if you want this color to be colored different than the key color, you should set the “key” field to space while generating

the field catalog. However, then there may be some side effects on column orders. You can set the column order as you want at the frontend. But if this is not suitable for you, then unset all key settings and do all coloring and ordering as you want. Be careful that the function module generating the field catalog will always set the key properties of key fields.

Coloring An Entire Row

Coloring a row is a bit (really a bit) more complicated. To enable row coloring, you should add an additional field to your *list data table*. It should be of character type and length at least 4. This field will contain the color code for the row. So, let's modify declaration of our list data table "`gt_list`".

```
*--- Internal table holding list data
DATA BEGIN OF gt_list OCCURS 0 .
INCLUDE STRUCTURE SFLIGHT  .
DATA   rowcolor(4) TYPE c .
DATA END OF gt_list .
```

Code Part 13 – *Adding the field that will contain row color data*

As you guess, you should fill the color code to this field. Its format will be the same as explained before at *section C.6.3.* But how will ALV Grid know that you have loaded the color data for the row to this field. So, you make it know this by passing the name of the field containing color codes to the field "`INFO_FNAME`" of the layout structure.

e.g.

```
ps_layout-info_fname = <field_name_containing_color_codes>. "e.g. 'ROWCOLOR'
```

You can fill that field anytime during execution. But, of course, due to the flow logic of screens, it will be reflected to your list display as soon as an ALV refresh occurs.

You can color an entire row as described in the next section. However, this method is less time consuming.

Coloring Individual Cells

This is the last point about coloring procedures for the ALV Grid. The procedure is similar to coloring an entire row. However, since an individual cell can be addressed with two parameters we will need something more. What is meant by "more" is a table type structure to be included into the structure of the list data table. It seems strange, because including it will make our list data structure deep. But anyhow ALV Grid control handles this.

The structure that should be included must be of type "`LVC_T_SCOL`". If you want to color the entire row, this inner table should contain only one row with field "`fname`" is set to space, some color value at field "`col`", "0" or "1" at fields "int" (intensified) and "`inv`" (inverse).

If you want to color individual cells, then for each cell column, append a line to this inner table which also contains the column name at field "`fname`". It is obvious that you can color an entire column by filling this inner table with a row for that column for each row in the list data table. But, it is also obvious that, this will be more time consuming than the method at *section C.6.1*.

Again key field coloring will override your settings. That's why, we have another field in this inner table called "`nokeycol`". For each field represented in the inner table, set this field to 'X' to prevent overriding of key color settings.

In this procedure, again we must tell the control the name of the inner table containing color data. The field "CTAB_FNAME" of the layout structure is used for this purpose.

```
*--- Internal table holding list data
DATA BEGIN OF gt_list OCCURS 0 .
INCLUDE STRUCTURE SFLIGHT  .
DATA   rowcolor(4) TYPE c .
DATA   cellcolors TYPE lvc_t_scol .
DATA END OF gt_list .
```

Code Part 14 – *Adding inner table that will contain cell color data*

```
DATA ls_cellcolor TYPE lvc_s_scol .
...
READ TABLE gt_list INDEX 5 .
ls_cellcolor-fname = 'SEATSOCC' .
ls_cellcolor-color-col = '7' .
ls_cellcolor-color-int = '1' .
APPEND ls_cellcolor TO gt_list-cellcolors .
MODIFY gt_list INDEX 5 .
```

Code Part 15 – *A sample code to make the cell at row 5 and column 'SEATSOCC' colored*

A juicy-brained guy may ask what happens if all these three procedures applied for coloring at the same time. And again the same answer is given as there is a priority among them. The priority order is: cell setting - row setting - column setting. Beside these, key field setting must be handled.

Inserting Hyperlinks

Inserting hyperlink fields is achieved by a source table containing hyperlinks and handles to relate it to the list data. The hyperlinks table should be of the type "LVC_T_HYPE". For each field which will include hyperlinks you must add an "int4" type field to your list data table. Those new fields will contain the handle to get information from the hyperlink table. You state the field name which contains handle for each field that will contain hyperlink, in the field catalog by putting the handle name to the "WEB_FIELD" field for each column. OK, all this logic may be confusing; all are fields, which one is the field that will contain the field with…. It will be now good to add a sample code here.

Assume we want to bring in hyperlinks in columns 'CARRID' and 'CONNID'. So we should add two more fields to our list data table:


```
*--- Internal table holding list data
DATA BEGIN OF gt_list OCCURS 0 .
INCLUDE STRUCTURE SFLIGHT  .
DATA   rowcolor(4) TYPE c .
DATA   cellcolors TYPE lvc_t_scol .
DATA   carrid_handle TYPE int4 .
DATA   connid_handle TYPE int4 .
DATA END OF gt_list .
```

Code Part 16 – *Two new fields to hold handle data each for the field that will contain hyperlink*

Build your hyperlinks table. Remember that, its type must be "LVC_T_HYPE".

```
*--- Hyperlinks table
FORM prepare_hyperlinks_table CHANGING pt_hype TYPE lvc_t_hype .

  DATA ls_hype TYPE lvc_s_hype .

  ls_hype-handle = '1' .
  ls_hype-href = 'http://www.company.com/carrids/car1' .
  APPEND ls_hype TO pt_hype .
  ls_hype-handle = '2' .
  ls_hype-href = 'http://www.company.com/carrids/car1' .
  APPEND ls_hype TO pt_hype .
  ls_hype-handle = '3' .
  ls_hype-href = 'http://www.company.com/carrids/car1' .
  APPEND ls_hype TO pt_hype .
  ls_hype-handle = '4' .
  ls_hype-href = 'http://www.company.com/connids/con11' .
  APPEND ls_hype TO pt_hype .
  ls_hype-handle = '5' .
  ls_hype-href = 'http://www.company.com/connids/con12' .
  APPEND ls_hype TO pt_hype .

.. ..
ENDFORM .
```

Code Part 17 – *Preparing hyperlinks with their handles*

We must state the field containing handle for each field. This is done while generating your field catalog.

e.g.
While modifying field catalog entry for 'CARRID':
```
ls_fieldcat-web_field = 'CARRID_HANDLE'.
```
And for 'CONNID':
```
ls_fieldcat-web_field = 'CONNID_HANDLE'.
```

When calling the method "**set_table_for_first_display**" pass the name of the handle data table to the parameter "it_hyperlink".

While you are filling your list data, you should put handles in related fields. Following sample code says that when clicked on a CARRID field containing 'XX' go to the URL 'http://www.company.com/carrids/car1' and if it is a CONNID field fulfilling the former condition and containing '01', go to the URL 'http://www.company.com/connids/con11'. Here, the hyperlink table considered is the table prepared in *Code Part 15*.

```
LOOP AT gt_list .
  IF gt_list-carrid = 'XX'.
    gt_list-carrid_handle = '1' .
    IF gt_list-connid = '01' .
      gt_list-connid_handle = '4' .
    ENDIF .
    MODIFY gt_list .
  ENDIF .
ENDLOOP .
```

Code Part 18 – *Assigning handles to fields*

Making Fields As Dropdown Lists

Certainly, it will be nice to make some fields as dropdown menus. The procedure for making a field as a dropdown menu is similar to making it to contain a hyperlink. However, in this case, we do not give our table filled with handles directly to the method "**set_table_for_first_display**". The table for values must be of type "LVC_T_DROP" and we will use the method "**set_drop_down_table**" to register our handles table. To make an entire column as dropdown, just while generating the field catalog; pass the handle number to field "DRDN_HNDL" for that field.
e.g. `ps_fcat-drdn_hndl = '1' .`

To make individual cells as dropdown, you must add a new field to contain the handle, for each column field to be as dropdown. While filling your list data or somewhere separate where you modify it, you fill these new fields with handles. To match the fields containing handle information with the columns, we use again the field catalog. We pass the name of our new field to the field "DRDN_FIELD" of the field catalog structure for the column.
e.g.

For the example at the Code Part 18 – ***Assigning handles to fields***:

```
ps_fcat-drdn_field = 'PTYP_DD_HNDL' .
```

```
*--- Internal table holding list data
DATA BEGIN OF gt_list OCCURS 0 .
INCLUDE STRUCTURE SFLIGHT  .
DATA   rowcolor(4) TYPE c .
DATA   cellcolors TYPE lvc_t_scol .
DATA   carrid_handle TYPE int4 .
DATA   connid_handle TYPE int4 .
DATA   ptype_dd_hndl TYPE int4 .
DATA END OF gt_list .
```

Code Part 19 – *Adding a new field to contain handle for drilldown values*

```
*--- Drilldown values
FORM prepare_drilldown_values.
  DATA lt_ddval TYPE lvc_t_drop .
  DATA ls_ddval TYPE lvc_s_drop .
  ls_ddval-handle = '1' .
  ls_ddval-value = 'JFK-12' .
  APPEND ls_ddval TO lt_ddval .
  ls_ddval-handle = '1' .
  ls_ddval-value = 'JSF-44' .
  APPEND ls_ddval TO lt_ddval .
  ls_ddval-handle = '1' .
  ls_ddval-value = 'KMDA-53' .
  APPEND ls_ddval TO lt_ddval .
  ls_ddval-handle = '1' .
  ls_ddval-value = 'SS3O/N' .
  APPEND ls_ddval TO lt_ddval .
  CALL METHOD gr_alvgrid->set_drop_down_table
    EXPORTING
      it_drop_down = lt_ddval .
ENDFORM.                    " prepare_drilldown_values
```

Code Part 20 – *Preparing values for the dropdown menu with the handle '1'*

As you examine at *Code Part 19*, after preparing the values table, we register our table by the method "**`set_drop_down_table`**".

Managing Display Variants

You can manage display variants by using parameters, "`is_variant`" and "`i_save`" of "**`set_table_for_first_display`**". Here are options for variant handling. <structure_name> is the variant-defining structure of type "`DISVARIANT`". The field "`report`" in this structure should contain the value of "`sy-repid`".

Mode	is_variant	i_save
Change current display variant	SPACE	SPACE
Select and change current display variant	<structure_name>	SPACE
Select, change and save current display variant	<structure_name>	'U': Only user-specific 'X': Only global 'A': Both

Table 9 – *Display variant modes*

Event Based Additional Functionalities

As being developed by object-oriented methodology, ALV Grid control has some events that are triggered during interaction between the user. These events are used to utilize some additional functionalities of the ALV Grid. For these types of functionalities, we require a class to be implemented (generally local in our program) to be the event handler for the ALV Grid instance. It is assumed in this tutorial that the object- oriented perspective of ABAP programming is known. We will not deal with the syntax and logic on how to define and implement a local class, why to register an event handler method to handle an event, etc…

Here are some of the events of ALV Grid control. This table is taken from the online SAP Library. The column "HTML" shows that whether the related event is supported at SAP GUI for HTML. "✓" means that it is applicable, "X" is for non-applicable ones and "(✓)" is used for restricted-applicability.

User-defined Text Output

Event	Application	HTML
`print_end_of_list`	Define output text to be printed at the end of the entire list	✓
`print_top_of_list`	Define output text to be printed at the beginning of the entire list	✓
`print_end_of_page`	Define output text to be printed at the end of each page	✓
`print_top_of_page`	Define output text to be printed at the beginning of each page	✓
`subtotal_text`	Define self-defined subtotals texts	✓

Table 10 – *Events for user-defined text output*

Mouse-controlled Actions in the Grid Control

Event	Application	HTML
`button_click`	Query a click on a pushbutton in the ALV Grid Control	✓
`double_click`	Query a double-click on a cell of the ALV Grid control	✓
`hotspot_click`	Query a hotspot click on columns defined for this purpose in advance	✓
`onDrag`	Collect information when elements of the ALV Grid Control are dragged	X

`onDrop`	Process information when elements of the ALV Grid Control are dropped	X
`onDropComplete`	Perform final actions after successful Drag&Drop	X
`onDropGetFlavor`	Distinguish between options for Drag&Drop behavior	X

Table 11 – *Events for mouse-controlled Actions*

Processing of Self-defined and Standard Functions

Event	Application	HTML
`before_user_command`	Query self-defined and standard function codes	✓
`user_command`	Query self-defined function codes	✓
`after_user_command`	Query self-defined and standard function codes	✓

Table 12 – *Events for processing of self-defined and standard functions*

Definition of Self-defined Functions

Event	Application	HTML
`toolbar`	Change, delete or add GUI elements in the toolbar	✓
`menu_button`	Define menus for menu buttons in the toolbar	✓
`context_menu_request`	Change context menu	X
`onf1`	Define self-defined F1 help	(✓)

Table 13 – *Events for definition of self-defined functions*

General Scheme for the Event Handler Class

Here is a sample code to illustrate a general scheme for our local class.

```
CLASS lcl_event_handler DEFINITION .
  PUBLIC SECTION .
    METHODS:
*--To add new functional buttons to the ALV toolbar
     handle_toolbar FOR EVENT toolbar OF cl_gui_alv_grid
                IMPORTING e_object e_interactive          ,
*--To implement user commands
     handle_user_command
            FOR EVENT user_command OF cl_gui_alv_grid
                IMPORTING e_ucomm                         ,
*--Hotspot click control
     handle_hotspot_click
            FOR EVENT hotspot_click OF cl_gui_alv_grid
                IMPORTING e_row_id e_column_id es_row_no  ,
*--Double-click control
     handle_double_click
        FOR EVENT double_click OF cl_gui_alv_grid
            IMPORTING e_row e_column                      ,
*--To be triggered before user commands
     handle_before_user_command
            FOR EVENT before_user_command OF cl_gui_alv_grid
                IMPORTING e_ucomm                         ,
*--To be triggered after user commands
     handle_after_user_command
            FOR EVENT context_menu_request OF cl_gui_alv_grid
                IMPORTING e_object
*--Controlling data changes when ALV Grid is editable
     handle_data_changed
            FOR EVENT data_changed OF cl_gui_alv_grid
                IMPORTING er_data_changed                 ,
```

```
*--To be triggered after data changing is finished
     handle_data_changed_finished
            FOR EVENT data_changed_finished OF cl_gui_alv_grid
                IMPORTING e_modified                        ,
*--To control menu buttons
     handle_menu_button
            FOR EVENT menu_button OF cl_gui_alv_grid
                IMPORTING e_oject e_ucomm                   ,
*--To control button clicks
     handle_button_click
            FOR EVENT button_click OF cl_gui_alv_grid
                IMPORTING e_oject e_ucomm                   .
  PRIVATE SECTION.
ENDCLASS.

CLASS lcl_event_handler IMPLEMENTATION .
*--Handle Toolbar
  METHOD handle_toolbar.
   PERFORM handle_toolbar USING e_object e_interactive .
  ENDMETHOD .
*--Handle Hotspot Click
  METHOD handle_hotspot_click .
   PERFORM handle_hotspot_click USING e_row_id e_column_id es_row_no .
  ENDMETHOD .
*--Handle Double Click
  METHOD handle_double_click .
   PERFORM handle_double_click USING e_row e_column es_row_no .
  ENDMETHOD .
*--Handle User Command
  METHOD handle_user_command .
   PERFORM handle_user_command USING e_ucomm .
  ENDMETHOD.
```

```
  *--Handle After User Command
    METHOD handle_context_menu_request .
     PERFORM handle_context_menu_request USING e_object .
    ENDMETHOD.

  *--Handle Before User Command
    METHOD handle_before_user_command .
     PERFORM handle_before_user_command USING e_ucomm .
    ENDMETHOD .
  *--Handle Data Changed
    METHOD handle_data_changed .
     PERFORM handle_data_changed USING er_data_changed .
    ENDMETHOD.
  *--Handle Data Changed Finished
    METHOD handle_data_changed_finished .
     PERFORM handle_data_changed_finished USING e_modified .
    ENDMETHOD .
  *--Handle Menu Buttons
    METHOD handle_menu_button .
      PERFORM handle_menu_button USING e_object e_ucomm .
    ENDMETHOD .
  *--Handle Button Click
    METHOD handle_button_click .
      PERFORM handle_button_click USING e_object e_ucomm .
    ENDMETHOD .


  ENDCLASS .
```

Code Part 21 – *Local event handler class definition and implementation*

Here in the implementation part, we are branching to forms to get rid of restrictions driven by the OO context. It is your choice to branch or directly write your codes into the method.

Here in local class coding, just implement the methods which will handle events that you want to be triggered. In this tutorial, we will prefer to deal with topics as functionalities rather than explaining events one-by-one. Having an event handler class we are now able to instantiate it and register its methods to handle ALV Grid instance events.


```
DATA gr_event_handler TYPE REF TO  lcl_event_handler .
   .. ..

*--Creating an instance for the event handler
    CREATE OBJECT gr_event_handler .

*--Registering handler methods to handle ALV Grid events
    SET HANDLER gr_event_handler->handle_user_command FOR gr_alvgrid  .
    SET HANDLER gr_event_handler->handle_toolbar FOR gr_alvgrid       .
    SET HANDLER gr_event_handler->handle_menu_button FOR gr_alvgrid   .
    SET HANDLER gr_event_handler->handle_double_click FOR gr_alvgrid  .
    SET HANDLER gr_event_handler->handle_hotspot_click FOR gr_alvgrid .
    SET HANDLER gr_event_handler->handle_button_click FOR gr_alvgrid  .
    SET HANDLER gr_event_handler->handle_before_user_command
           FOR gr_alvgrid .
    SET HANDLER gr_event_handler->handle_context_menu_request
           FOR gr_alvgrid .
    SET HANDLER gr_event_handler->handle_data_changed FOR gr_alvgrid  .
    SET HANDLER gr_event_handler->handle_data_changed_finished
          FOR gr_alvgrid .
```

Code Part 22 - *Creating event handler instance and registering handler methods*

Hotspot Clicking

From field catalog field definitions, we know that we can make some columns to respond to single clicks as hotspots by setting the value for the field "HOTSPOT" to 'X' while generating the field catalog. After clicking, the event "**hotspot_click**" is triggered. This event has three parameters in its interface. The parameter "e_row_id" is obsolete. Other two parameters are "es_row_no" which is of type "LVC_S_ROID" and passes information about the row index at "es_row_no-row_id", and "e_column_id" of type "LVC_S_COL" which returns the column fieldname at "e_column_id-fieldname". Utilizing these parameters you know where the user clicked and trigger your action.


```
FORM handle_hotspot_click USING i_row_id TYPE lvc_s_row
                          i_column_id TYPE lvc_s_col
                          is_row_no TYPE lvc_s_roid.

    READ TABLE gt_list INDEX is_row_no-row_id .
    IF sy-subrc = 0 AND i_column_id-fieldname = 'SEATSOCC' .
      CALL SCREEN 200 . "Details about passenger-seat matching
    ENDIF .

  ENDFORM .
```

Code Part 23 – *An example implementation for the method, handling the event "hotspot_click"*

Double Clicking

As you can guess, handling a double-click is very similar to handle a hotspot click. You do nothing additional to make a field double-click intensive; that is you need not set some option in the field catalog. After double-click event occurs, the method handling it will return again three parameters and again one of them, "e_row", is obsolete. The parameters "e_column" and "es_row_no" are similar to the parameters of the event "**hotpot_click**".

```
  FORM handle_double_click USING i_row TYPE lvc_s_row
                          i_column TYPE lvc_s_col
                          is_row_no TYPE lvc_s_roid.

    READ TABLE gt_list INDEX is_row_no-row_id .
    IF sy-subrc = 0 AND i_column-fieldname = 'SEATSOCC' .
      CALL SCREEN 200 . "Details about passenger-seat matching
    ENDIF .

  ENDFORM .
```

Code Part 24 - *An example implementation for the method, handling the event "double_click"*

Pushbuttons On The List

To make a cell to be displayed as a pushbutton, we have two steps. Firstly, insert a new inner table of type "LVC_T_STYL" into your list data table.

```
*--- Internal table holding list data
DATA BEGIN OF gt_list OCCURS 0 .
INCLUDE STRUCTURE SFLIGHT  .
DATA   rowcolor(4) TYPE c .
DATA   cellcolors TYPE lvc_t_scol .
DATA   carrid_handle TYPE int4 .
DATA   connid_handle TYPE int4 .
DATA   cellstyles TYPE lvc_t_styl .
DATA END OF gt_list .
```

Code Part 25 – *Inserting an inner table to store cell display styles*

Fill this inner table for each field to be displayed as pushbutton.

```
DATA ls_style TYPE lvc_s_styl .
...
READ TABLE gt_list INDEX 7 .
ls_style-fieldname = 'SEATSMAX' .
ls_style-style = cl_gui_alv_grid=>mc_style_button .
APPEND ls_style TO gt_list-cellstyles .
MODIFY gt_list INDEX 7 .
```

Code Part 26 - *A sample code to make the cell at row 7 and column 'SEATSMAX' displayed as pushbutton*

As usual, we state our list data table field related with styles in the layout structure at field 'STYLEFNAME'.

e.g. `ps_layout-stylefname = 'CELLSTYLES' .`

Button click event is handled like hotspot click via the event "**`button_click`**" through its parameters "`es_col_id`" and "`es_row_no`" which contain the address of the clicked pushbutton cell.

Adding Your Own Functions

ALV Grid control has an open door letting you to add your own functions triggered by a button press on the ALV toolbar. For this, we mainly utilize two of ALV Grid events. We use the event "toolbar" to add the button and the event "**user_command**" to implement the new function.

In the method handling the "toolbar" event, we define a new button by filling a structure and appending it to the table attribute "`mt_toolbar`" of the object to whose reference we can reach via the parameter "`e_object`" of the event.

```
FORM handle_toolbar USING i_object TYPE REF TO cl_alv_event_toolbar_set .

    DATA: ls_toolbar  TYPE stb_button.

    CLEAR ls_toolbar.
    MOVE 3 TO ls_toolbar-butn_type.
    APPEND ls_toolbar TO i_object->mt_toolbar.

    CLEAR ls_toolbar.
    MOVE 'PER' TO ls_toolbar-function.                    "#EC NOTEXT
    MOVE icon_display_text TO ls_toolbar-icon.
    MOVE 'Passenger Info'(201) TO ls_toolbar-quickinfo.
    MOVE 'Passenger Info'(201) TO ls_toolbar-text.
    MOVE ' ' TO ls_toolbar-disabled.                      "#EC NOTEXT
    APPEND ls_toolbar TO i_object->mt_toolbar.

    CLEAR ls_toolbar.
    MOVE 'EXCH' TO ls_toolbar-function.                   "#EC NOTEXT
    MOVE 2 TO ls_toolbar-butn_type.
    MOVE icon_calculation TO ls_toolbar-icon.
    MOVE 'Payment in Other Currencies'(202) TO ls_toolbar-quickinfo.
    MOVE ' ' TO ls_toolbar-text.
    MOVE ' ' TO ls_toolbar-disabled.                      "#EC NOTEXT
    APPEND ls_toolbar TO i_object->mt_toolbar.

  ENDFORM .
```

Code Part 27 – ***Filling the structure for two new buttons***

The fields of the structure we fill are as follows:

Field	Description
FUNCTION	The function code for the function
BUTN_TYPE	Button type that will be added to the toolbar. Available button types are: 0 Button (normal) 1 Menu and default button 2 Menu 3 Separator 4 Radio button 5 Checkbox 6 Menu entry
ICON	Icon for the button (optional)
TEXT	Text for the button (optional)
QUICKINFO	Quick info for the button (optional)
DISABLED	Adds the button as disabled

Table 14 – *Fields of structure to be filled to add a new function*

In the *Code Part 22*, we are adding a separator line and two buttons one of which is a normal button whereas the other is a menu button. To handle a menu button which as added by choosing '1' or '2' as the button type, we must also implement some coding at the method handling the event "**menu_button**" to define functions as to be subentries. The functions of these subentries are also handled under the event "**user_command**".

```
FORM handle_menu_button USING    i_object TYPE REF TO cl_ctmenu
                                 i_ucomm TYPE syucomm .

  CASE i_ucomm .
    WHEN 'EXCH' .
      CALL METHOD i_object->add_function
        EXPORTING
          fcode       = 'EU'
          text        = 'Euro' .
      CALL METHOD i_object->add_function
        EXPORTING
          fcode       = 'TRL'
          text        = 'Turkish Lira' .
      .. ..
  ENDCASE .
ENDFORM.                    " handle_menu_button
```

Code Part 28 – *Adding two functions to be subentries for the menu button with function code 'EXCH'*

Now, to implement what to be executed when our button is pressed or our subentry is selected we need to program our functions in the method handling the event "**user_command**".


```
FORM handle_user_command USING i_ucomm TYPE syucomm .

  DATA lt_selected_rows TYPE lvc_t_roid .
  DATA ls_selected_row TYPE lvc_s_roid .

  CALL METHOD gr_alvgrid->get_selected_rows
       IMPORTING
          et_row_no     = lt_selected_rows .
  READ TABLE lt_selected_rows INTO ls_selected_row INDEX 1 .
  IF sy-subrc ne 0 .
    MESSAGE s000(su) WITH 'Select a row!'(203) .
  ENDIF .
  CASE i_ucomm .
    WHEN 'CAR' .
        READ TABLE gt_list INDEX ls_selected_row-row_id .
        IF sy-subrc = 0 .
          CALL FUNCTION 'ZDISPLAY_CARRIER_INFO'
               EXPORTING  carrid = gt_list-carrid
               EXCEPTIONS carrier_not_found = 1
                          OTHERS            = 2.
          IF sy-subrc NE 0 .
*--Exception handling
          ENDIF .
     ENDIF .
    WHEN 'EU' .
        READ TABLE gt_list INDEX ls_selected_row-row_id .
        IF sy-subrc = 0 .
          CALL FUNCTION 'ZPOPUP_CONV_CURR_AND_DISPLAY'
               EXPORTING monun = 'EU'
                         quant = gt_list-paymentsum.
        ENDIF .
       .. ..
  ENDCASE .
ENDFORM .
```

Code Part 29 – *Implementing functioning codes for new functions*

As you can see, we are using the method "**get_selected_rows**" to get which row is selected. Since the button with function 'EXCH' branches into subfunctions, we do not implement anything for it. Instead, we implement functional codes for subfunctions.

After all, to make ALV show our additional buttons, we must call the method "**set_toolbar_interactive**" for the ALV Grid instance after the instance is created.

e.g. `CALL METHOD gr_alvgrid->set_toolbar_interactive .`

Overriding Standard Functions

The ALV Grid control also gives the opportunity to override its standard functions. For this purpose, we utilize the event "**before_user_command**" and the method which sets ALV user command, called "**set_user_command**". At "**before_user_command**", you control the user command to process your own function and then use the method "**set_user_command**" to set the ALV Grid user command to space to avoid ALV Grid execute further with the standard function.

```
FORM handle_before_user_command USING i_ucomm TYPE syucomm .
    CASE e_ucomm .
      WHEN '&INFO' .
        CALL FUNCTION 'ZSFLIGHT_PROG_INFO' .
        CALL METHOD gr_alvgrid->set_user_command
                    EXPORTING i_ucomm = space.
    ENDCASE .
ENDFORM .
```

Code Part 30 – *Overriding the standard ALV Grid function '&INFO'*

Context Menus

The event related with context menus is "**context_menu_request**". When you right click on the ALV Grid area the event-handling method is triggered. Under this method you can add entries in the context menu. You can utilize "**GET_SELECTED…**" methods to retrieve which cell, row or column the user has right-clicked. Adding functions to the context menu is accomplished as in adding subfunctions to non-standard menu buttons as described in *section "Adding Your Own Functions"*.


```
FORM handle_context_menu_request USING i_object TYPE REF TO cl_ctmenu .
*--Here you can add some logic to retrive what was clicked and
*--decide what to add to the context menu
  CALL METHOD i_object->add_function
    EXPORTING
      fcode       = 'CREA'
      text        = 'Create Booking'(204) .

ENDFORM .
```

Code Part 31 – *Adding an entry for the context menu*

You can add a separator by "**add_separator**", a menu by "**add_menu**", and a submenu by "**add_submenu**" methods.

You can also use "**disable_functions**" to disable some of the functions on the menu, "**enable_functions**" to enable, "**hide_functions**" to hide them and "**show_functions**" to make them displayed again. You pass the list of the function codes to those methods via the parameter "fcodes". These are all about the class "CL_CTMENU" since the context menu instantiates that class. For further functionalities refer to that class.

```
  DATA lt_fcodes TYPE ui_functions .

  CLEAR lt_fcodes.
  APPEND cl_gui_alv_grid=>mc_fc_col_optimize TO lt_fcodes.
  APPEND 'CREA' TO lt_fcodes .
  CALL METHOD e_object->disable_functions
        EXPORTING fcodes = lt_fcodes .
```

Code Part 32 – *An example filling the table to be passed to above functions*

About Printing

There are some events that you can utilize about printing. You simply write what is to be output in methods handling these events. Here is the list of events in this context.

print_end_of_list	Define output text to be printed at the end of the entire list
print_top_of_list	Define output text to be printed at the beginning of the entire list
print_end_of_page	Define output text to be printed at the end of each page
print_top_of_page	Define output text to be printed at the beginning of each page

Table 15 – *Events for printing*

To utilize one of the events above, let's proceed with the standard procedure since we did not add this event in our general scheme at section *D.1*.

First add a handler method in your handler class definition as:

e.g.
```
METHOD handle_print_top_of_list
            FOR EVENT print_top_of_list OF cl_gui_alv_grid .
```

Then implement the method in the implementation part of your local class.

e.g.
```
METHOD handle_print_top_of_list .
       WRITE:/ 'Flights Made on ', sy-datum .
ENDMETHOD .
```

And register this method as the handler.

e.g.
```
SET HANDLER gr_event_handler->handle_print_top_of_list FOR gr_alvgrid .
```

The output of these events can be examined at the print preview or in the printing.

Making ALV Grid Editable

This may be one of the mostly-used functionalities of the ALV Grid since as a developer we prefer to use an ALV Grid instead of a table control for some reasons that are known by all of you (at least for the sake of appearance). In fact, making the ALV Grid editable has nothing to do with events. However, since controlling data input which is explained in the next section is related, it is better that we deal with this topic here.

To make a column editable, it will be sufficient to set the field "EDIT" in the field catalog. The ALV Grid perceives if there are some editable fields and adds buttons for editing purposes. If you do not need these new buttons, you know how to exclude them.

To make individual cells editable, we will utilize the table we used for making a cell a pushbutton. As you remember, it was of type "LVC_T_STYL". If you have not added this inner table, add it now. For this procedure; add the name of the field to the field "FIELDNAME", and pass "cl_gui_alv_grid=>mc_style_enabled" to make a field editable and "cl_gui_alv_grid=>mc_style_disabled" to make a field non-editable, to the field "STYLE". You can use the one with "disable" when you make an entire column editable and want just a few of cells along it non-editable. As you remember from the pushbutton section we must tell the layout about this styling field.

e.g. `ps_layout-stylefname = 'CELLSTYLES' .`

Now, let's solidify the procedure by code parts below. We want our column "SEATSMAX" entirely editable except the case "CARRID" is 'XY' which is a rare case and we want our cells along the column 'PLANETYPE' editable if their respective 'CONNID' fields contain the value '02'.

Assume we have added our style table ("CELLSTYLES") to our list data table and tell the layout structure about it and we adjust the field catalog so that the column "SEATSMAX" has the property "EDIT" set to 'X'.


```
FORM adjust_editables USING pt_list LIKE gt_list[] .

  DATA ls_listrow LIKE LINE OF pt_list .
  DATA ls_stylerow TYPE lvc_s_styl .
  DATA lt_styletab TYPE lvc_t_styl .

  LOOP AT pt_list INTO ls_listrow .

    IF ls_listrow-carrid = 'XY' .
      ls_stylerow-fieldname = 'SEATSMAX' .
      ls_stylerow-style = cl_gui_alv_grid=>mc_style_disabled .
      APPEND ls_stylerow TO lt_styletab .
    ENDIF .
    IF ls_listrow-connid = '02' .
      ls_stylerow-fieldname = 'PLANETYPE' .
      ls_stylerow-style = cl_gui_alv_grid=>mc_style_enabled .
      APPEND ls_stylerow TO lt_styletab .
    ENDIF .

    INSERT LINES OF lt_styletab INTO ls_listrow-cellstyles .
    MODIFY pt_list FROM ls_listrow .
  ENDLOOP .
ENDFORM
```

Code Part 33 – *Conditionally setting fields to be editable or non-editable*

As usual, cell based settings override entire column settings. You can dynamically switch between cases in any proper part of your execution. Just fill your inner table as required and refresh table display; for entire column settings, set or unset the property "`EDIT`" of the field catalog for the column and reset the field catalog using "set_frontend_fieldcatalog".

As the last condition to be met for editability, you must call the method "**`set_ready_for_input`**" passing '1' to the parameter "`i_ready_for_input`".

Using this method you can switch between editable and non-editable mode. As you guess, passing '0' to the parameter while calling the method, switches to non-editable mode.

Controlling Data Changes

As we can now make our ALV Grid editable we may require controlling input data. The ALV Grid has events "**data_changed**" and "**data_changed_finished**". The former method is triggered just after the change at an editable field is perceived. Here you can make checks for the input. And the second event is triggered after the change is committed.

You can select the way how the control perceives data changes by using the method "**register_edit_event**". You have two choices:

i. After return key is pressed: To select this way, to the parameter "`i_event_id`" pass "`cl_gui_alv_grid=>mc_evt_enter`".
ii. After the field is modified and the cursor is moved to another field: For this, pass "`cl_gui_alv_grid=>mc_evt_modifies`" to the same parameter.

To make events controlling data changes be triggered, you must select either way by calling this method. Otherwise, these events will not be triggered.

To control field data changes, ALV Grid uses an instance of the class "`CL_ALV_CHANGED_DATA_PROTOCOL`" and passes this via the event "**data_changed**". Using methods of this class, you can get and modify cell values and produce error messages. Here are some of those methods:

get_cell_value	Gets the cell value. You pass the address of the cell to the interface.
modify_cell	Modifies the cell value addressed via parameters.
add_protocol_entry	Add a log entry. You make use of standard message interface with message type, message id, etc…
protocol_is_visible	Make the error table visible or not.
refresh_protocol	Refreshing log entries.

Table 16 – *Methods to use for controlling data changes*

With the reference of the instance, you can reach information about modifications. These useful attribute tables are:

MT_MOD_CELLS	Contains addresses of modified cells with "`row_id`"s and "`fieldname`"s.
MP_MOD_ROWS	Contains modified rows. Its type is generic.
MT_GOOD_CELLS	Contains cells having proper values
MT_DELETED_ROWS	Contains rows deleted from the list
MT_INSERTED_ROWS	Contains rows inserted to the list

Table 17 – *Attribute tables to be used for controlling data changes*

Utilizing these methods and attributes you can check and give proper message and also modify the cell content.


```
FORM handle_data_changed USING ir_data_changed
                         TYPE REF TO cl_alv_changed_data_protocol.

  DATA : ls_mod_cell  TYPE lvc_s_modi  ,
         lv_value     TYPE lvc_value  .

  SORT ir_data_changed->mt_mod_cells BY row_id .
  LOOP AT ir_data_changed->mt_mod_cells
                         INTO ls_mod_cell
                         WHERE fieldname = 'SEATSMAX' .
    CALL METHOD ir_data_changed->get_cell_value
                EXPORTING i_row_id     =  ls_mod_cell-row_id
                          i_fieldname  =  'CARRID'
                IMPORTING e_value      =  lv_value .

    IF lv_value = 'THY' AND ls_mod_cell-value > '500' .
      CALL METHOD ir_data_changed->add_protocol_entry
           EXPORTING
             i_msgid = 'SU'
             i_msgno = '000'
             i_msgty = 'E'
             i_msgv1 = 'This number can not exceed 500 for '
             i_msgv2 = lv_value
             i_msgv3 = 'The value is et to ''500'''
             i_fieldname = ls_mod_cell-fieldname
             i_row_id = ls_mod_cell-row_id .

      CALL METHOD ir_data_changed->modify_cell
             EXPORTING i_row_id    = ls_mod_cell-row_id
                       i_fieldname = ls_mod_cell-fieldname
                       i_value     = '500' .
    ENDIF .
  ENDLOOP .
ENDFORM                    " handle_data_changed
```

Code Part 34 – *Checking the input together with anon-input value, adding a log and modifying the cell content*

The event "**data_changed**" makes you aware about F4 functions. It sets the appropriate parameter from the group with respect to where it was triggered. These parameters are { E_ONF4, E_ONF4_BEFORE, E_ONF4_AFTER }.

Linking F1 Help to Fields

To link your own F1 help to a field, you should simply utilize the event "**onf1**". At this point, it is assumed that you know how to define, implement and register an event. The event has following parameters at its interface:

- E_FIELDNAME of type LVC_FNAME
- ES_ROW_NO of type LVC_S_ROID
- ER_EVENT_DATA reference to type CL_ALV_EVENT_DATA

And, here is a simple sample code for the method.

```
METHOD handle_on_f1 .

  PERFORM f1_help USING e_fieldname es_row_no .
  er_event_data->m_event_handled = 'X' .

ENDMETHOD .
```

Code Part 35 - *A sample code for the event"onf1"*

Here we set the attribute "er_event_data->m_event_handled" to prevent further processing of standard F1 help.

Linking F4 Help to Fields

For the last section, we will deal with linking F4 help to fields. It is easy. As usual, define, implement and register the event "**onf4**" at proper places in your code. For F4 help, you must register the fields whose F4 request will trigger the "**onf4**" event. For this you must prepare a table of type "LVC_T_F4" and register this table using the method "**register_f4_for_fields**". While preparing table you must include a line for each field which will trigger F4 event. For each field in the structure;

- Pass the fieldname to 'FIELDNAME'
- Set 'REGISTER' to make the field registered,
- Set 'GETBEFORE' to provide field content transport before F4 in editable mode
- Set 'CHNGEAFTER' to make the data changed after F4 in editable mode.


```
DATA: lt_f4 TYPE lvc_t_f4 WITH HEADER LINE .
.. ..
    lt_f4-fieldname = 'PRICE'.
    lt_f4-register = 'X' .
    lt_f4-getbefore = 'X' .
    APPEND lt_f4 .

    CALL METHOD gr_alvgrid->register_f4_for_fields
      EXPORTING
        it_f4  = lt_f4[] .
```

Code Part 36 – *Preparing table for the fields to be registered to trigger F4 event*

```
  METHOD handle_on_f4 .

    PERFORM f4_help USING e_fieldname es_row_no .
    er_event_data->m_event_handled = 'X' .

  ENDMETHOD .
```

Code Part 37 – *A sample "onf4" method implementation*

Again, we set the attribute "`er_event_data->m_event_handled`" to prevent further processing of standard F4 help.

A Piece of Troubleshooting

No	Question	Answer
1	I register my local class as the event handler for all ALV instances. How can it be known which instance evokes the handler?	As a property of ABAP OOP, you can use the predefined parameter "`sender`" in the event interface. This parameter refers to the caller instance.
2	When I wanted to filter with respect to a field, it couldn't handle it. There were matching records but it didn't retrieve them.	The most probable problem is that you are trying to filter a text field and since you didn't set the "`lowercase`" property at the field catalog, ALV just looks for the upper case. Set this property for that field while generating the field catalog.
3	I make some style settings to individual rows or cells before I call first display method. However, I can't get the result as I want.	Follow all the steps described in related sections. 1. You may have forgotten to call "**`set_ready_for_input`**" if you are talking about making cells editable. 2. There may be a filter which makes the row of styled cell invisible or a sorting may have changed the order of the row making it somewhere downwards so that you can't realize it.
4	The event "**`data_changed`**" is not triggered	You should have not called the method "**`register_edit_event`**" by which you also select the way ALV Grid perceives changes
5	My function buttons are not visible on the toolbar	You should have forgotten to call the method "**`set_toolbar_interactive`**".
6	Can I modify the grid title during execution?	Yes. Use the method "**`set_gridtitle`**".
7	I want to add a pushbutton with an icon for a column. Can I do that?	Yes. Set the property 'ICON" for the column to 'X' at the field catalog. Then do the style setting as to make it pushbutton. Pass the icon name to the field as its content.
8	Can I hide the header?	Yes. Set the property "`no_headers`" of the layout structure to 'X'.
9	Can I hide the ALV toolbar?	Yes. Set the property "`no_toolbar`" of the layout structure to 'X'.
10	It cannot find my local class on the line where I declare my event handler reference variable. When I take my class definition above data declarations, then variables are not recognized.	Before your global data definitions, make the interpreter be aware that your local class is defined somewhere in your program. For this, use "`CLASS <class> DEFINITION DEFERRED`" or simply make your definitions related to this class after it is included.
11	Can I display hierarchical lists using the ALV Grid Control?	No. ALV Grid Control does not support hierarchical lists.

12	I made an editable ALV Grid and I am utilizing "**data_changed**" event to control the data input. However, I got log entries some of which exist there although they are corrected. Is there a refresh problem? Where?	Exactly. Use the method "REFRESH_PROTOCOL" of the instance came via the parameter at appropriate place to overcome.
13	I utilize "**print_top_of_page**" event. However, at preview, it sets the line size to the maximum of 80 and the list size. How can I overcome this?	Its line-size is restricted so. Try to change your page header layout.
14	Does the ALV Grid Control support Drag & Drop?	At other than SAPGUI for HTML interfaces, the ALV Grid control supports Drag & Drop. However, this topic was not included in the context of this tutorial.
15	Displayed columns are not as I programmed	Check whether a display variant is activated. If this is not the reason, inspect your field catalog again.

Author Bio

Serdar Şimşekler is an SAP application developer working for *Havelsan Inc., Turkey*. He has experience with ABAP program development. He is also studying for an M.A. degree at the department of philosophy at METU. ssimsekler@yahoo.com

