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.

Utilizing the New ALV Object Model

SAP DEVELOPER NETWORK

Applies To:

The code sample in this paper can be used on SAP NetWeaver 04 (Web AS 6.40) or later.

Summary

This code sample presents a report template that utilizes the new ALV Object Model.

An additional updated code sample is available that demonstrates a different class hierarchy using interface methods. Although you can download the updated code (refactoring.zip), we would encourage you to use the Refactoring Assistant to make the changes to the original code sample. If you need help with the Refactoring Assistant, see Tomas Ritter's Refactoring ABAP classes weblog describing this tool.

By: Thomas Jung

Company: Kimball International, Kimball Electronics Group

Date: 21 October 2005

Introduction

When developing custom reports, it is important to have a way of creating consistent yet feature-rich applications. The SAP ALV functionality goes a long way towards facilitating this need. For years, our company has had a simple program template for creating basic reports. This template already consisted of all the logic necessary to create and interact with the old REUSE_ALV function modules. The use of this template allows developers to focus on the specific business logic for their new report without having to spend any time on creating the UI. This also has the effect that nearly all of our custom reports have the same look and feel.

We are currently going through an upgrade from 4.6C to ECC 5.0 (Web AS 6.40). New in Web AS 6.40 is the ALV Object Model. The ALV OM is a more object-oriented (OO) approach to the entire suite of ALV output formats. Our goal was to recreate our template program in order to take advantage of this new ALV OM. At the same time, we wanted to get away from the approach of copying from template. We wanted to use OO to create most of our logic in an ABAP objects class. This way, instead of copying the logic, individual programs could just reuse this existing class. If specific logic needed to be included (perhaps to handle double-click navigation), this could now be done via inheritance from this base template class. In this way, we can add new features or make fixes to all applications based on this template class with relative ease.

Template Program

We still have a template program that you copy from. This program now is really just a shell to call the ALV OM template class. This is the place where you can supply your business logic. This logic might be coded locally within this copied program (or better – placed in a separate class and called from here). The main reason for still having a classic program as the starting point is only to easily support select-options/parameters. If there was an easy way to do this without a dialog program, we could go completely OO!

This template program has some very simple logic to fill an internal table from SFLIGHT for demonstration purposes.

```
    ************************************************************************
    * Program Name: KEG Program Template for        Creation: 10/07/2005
    *               ALV Object Model
    *                                                                      *
    * SAP Name    : ZESU_REPORT_TEMPLATE_ALV_OM      Application: U        *
    *                                                                      *
    * Author      : Thomas Jung                      Type: 1               *
    *_____   *
    * Description :   This program is a template for the ALV Object Model  *
    *_____   *
```

```
* Inputs:                                                               *
*                                                                       *
* Outputs:                                                              *
*_____*
* External Routines                                                     *
*                                                                       *
*_____*
* Return Codes:                                                         *
*_____*
* Ammendments:                                                          *
*    Programmer         Date      Req. #            Action              *
* ================  ==========  ======  ==============================*
*                                                                       *
*************************************************************************


report   zesu_report_template_alv_om.


*-----------------------------------------------------------------------*
* TABLES                                                                *
*-----------------------------------------------------------------------*
tables: sflight.


*-----------------------------------------------------------------------*
* INTERNAL TABLES                                                       *
*-----------------------------------------------------------------------*
data: itab type table of sflight.


*-----------------------------------------------------------------------*
* CLASSES                                                               *
*-----------------------------------------------------------------------*
data: keg_alv   type ref to zcl_es_alv_om.


*-----------------------------------------------------------------------*
* SELECTION SCREEN                                                      *
```

```
*----------------------------------------------------------------------*
selection-screen begin of block five with frame title text-017.
parameter: variant like disvariant-variant. "ALV GRID VARIANT
parameter: nodata1 as checkbox. "RUN ALV WITHOUT DATA
selection-screen end of block five.


*----------------------------------------------------------------------*
* AT SELECTION SCREEN                                                  *
*----------------------------------------------------------------------*
**Respond to the F4 Request by the User for Help on the ALV Grid
**Variant Selection
at selection-screen
  on value-request for variant.
  keg_alv->f4_layouts( changing  c_variant = variant ).


*----------------------------------------------------------------------*
* INITIALIZATION                                                       *
*----------------------------------------------------------------------*
**Initialize for Selection Screen Output
initialization.
  create object keg_alv exporting i_repid = sy-repid.
  keg_alv->get_default_layout(
          changing  c_variant = variant ).


*----------------------------------------------------------------------*
* START-OF-SELECTION                                                   *
*----------------------------------------------------------------------*
start-of-selection.

  keg_alv->auth_check( ).

  if nodata1 = 'X'.
  else.
    select * from sflight into table itab.
```

```
*       Perform to read data and do processing
    endif.


    keg_alv->set_report_title( 'Dialog Template'(t01) ).
    keg_alv->publish_alv( exporting i_variant = variant
                          changing  itab      = itab ).
```

Output

In the following screen shot you can see the format of the output. It looks very much like the old Reuse ALV Grid. We have created a common header (the processing logic for this is in the template class) with all the information that we feel is import.

Template Class

The vast majority of the coding and logic resides in the template class, ZCL_ES_ALV_OM. There are several global attributes:

Attribute	Level	Visi...	Rea...	Typing	Associated Type		Description
REPID	Instance	Private	☐	Type	SYREPID	⇨	ABAP Program: Current Ma
ALV	Instance	Private	☐	Type Ref	CL_SALV_TABLE	⇨	Basis Class for Simple Table
ALV_MSG	Instance	Private	☐	Type Ref	CX_SALV_MSG	⇨	ALV: General Error Class w
TITLE	Instance	Private	☐	Type	STRING	⇨	Report Title

The following are the methods in this class. Several of them are delivered empty (such as the event handlers). These can be redefined if this class is inherited to provide more specific functionality.

Analyze	Level	Visi...	M...	Description
CONSTRUCTOR	Instance	Publi	🔒	CONSTRUCTOR
F4_LAYOUTS	Instance	Publi		Respond to the F4 Request by the User for
GET_DEFAULT_LAYOUT	Instance	Publi		Get Default Layout
ON_USER_COMMAND	Instance	Publi	⨠▶	On User Command Event Handler
ON_BEFORE_USER_COMMAND	Instance	Publi	⨠▶	On Before User Command Event Handler
ON_AFTER_USER_COMMAND	Instance	Publi	⨠▶	On After User Command Event Handler
ON_DOUBLE_CLICK	Instance	Publi	⨠▶	On Double Click Event Handler
ON_LINK_CLICK	Instance	Publi	⨠▶	On Link Click Event Handler
ON_TOP_OF_PAGE	Instance	Publi	⨠▶	On Top Of Page Event Handler
ON_END_OF_PAGE	Instance	Publi	⨠▶	On End of Page Event Handler
AUTH_CHECK	Instance	Publi		Authorization Check
PUBLISH_ALV	Instance	Publi		Prepare and Publish the ALV Grid
SET_REPORT_TITLE	Instance	Publi		Set the Report Title
PROCESS_REPORT_HEADERS	Instance	Prote		Process the Report Headers
PROCESS_TOP_OF_LIST	Instance	Prote		Process The Top of List for On-Line Display
PROCESS_TOP_OF_LIST_PRINT	Instance	Prote		Process the Top of List for Print Output
PROCESS_LAYOUT	Instance	Prote		Process the Layout Options
PROCESS_FUNCTIONS	Instance	Prote		Process the ALV Grid status Functions
SET_COLUMNS	Instance	Prote		Set Column Options
REGISTER_EVENTS	Instance	Prote		Register (Please only register those events yo

CONSTRUCTOR

This is the entrance point to the program. In this case, all it does is record the program name from the hosting program. This program name is then used during the custom authorization check and for the processing of the ALV variants.

```
method CONSTRUCTOR.

*Importing    I_REPID    TYPE SYREPID

  me->repid = i_repid.
```

```
endmethod.
```

F4_LAYOUTS

This is the method that is called in the at selection-screen on value-request for variant event. It will hook into the ALV OM to supply the F4 Value Help.

```
METHOD f4_layouts.
*Changing      C_VARIANT    TYPE SLIS_VARI     Layout


  DATA: ls_layout TYPE salv_s_layout_info,
        ls_key    TYPE salv_s_layout_key.


  ls_key-report = me->repid.


  ls_layout = cl_salv_layout_service=>f4_layouts(
    s_key    = ls_key
    restrict = if_salv_c_layout=>restrict_none ).


  c_variant = ls_layout-layout.
ENDMETHOD.
```

GET_DEFAULT_LAYOUT

This method is called from the INITIALIZATION event of the dialog program to preload the default ALV variant.

```
METHOD get_default_layout.
*Changing      C_VARIANT    TYPE SLIS_VARI     Layout
  DATA: ls_layout TYPE salv_s_layout_info,
        ls_key    TYPE salv_s_layout_key.


  ls_key-report = me->repid.


  ls_layout = cl_salv_layout_service=>get_default_layout(
    s_key    = ls_key
    restrict = if_salv_c_layout=>restrict_none ).


  c_variant = ls_layout-layout.
```

```
      ENDMETHOD.
```

EVENT Handlers

The event handler methods (ON_USER_COMMAND, ON_BEFORE_USER_COMMAND, ON_AFTER_USER_COMMAND, ON_DOUBLE_CLICK, ON_LINK_CLICK, ON_TOP_OF_PAGE, and ON_END_OF_PAGE) are all defined, but not implemented. This is where an inheriting class can provide a specific function such as forward navigation. For these methods to be called they must be registered in the method REGISTER_EVENTS.

AUTH_CHECK

This method is called at the very beginning of processing in the dialog program to perform our company's custom authorization check.

```
      METHOD auth_check.

        AUTHORITY-CHECK OBJECT 'Z_ABAP_CHK'
                   ID 'BUKRS' DUMMY
                   ID 'ACTVT' DUMMY
                   ID 'WERKS' DUMMY
                   ID 'REPID' FIELD me->repid.


        IF sy-subrc NE 0.
          MESSAGE e024(zes_job).
        ENDIF.
      ENDMETHOD.
```

SET_REPORT_TITLE

This method can be called from the dialog program to set the title for the ALV display.

```
      METHOD set_report_title.
      *Importing    I_TITLE      TYPE CSEQUENCE
        me->title = i_title.
      ENDMETHOD.
```

PUBLISH_ALV

This is the main method of the ALV template class. This method is called after all business logic is complete. The internal table with the final report results are passed into this method. From this point, all the UI processing and interaction with the ALV OO takes place.

```
      METHOD publish_alv.
      *Importing    I_VARIANT  TYPE SLIS_VARI     Layout
      *Changing     ITAB       TYPE TABLE
```

```
TRY.
    cl_salv_table=>factory(
      EXPORTING
        list_display = abap_false
      IMPORTING
        r_salv_table = alv
      CHANGING
        t_table      = itab ).
  CATCH cx_salv_msg INTO alv_msg.
    MESSAGE alv_msg TYPE 'I'.
    EXIT.
ENDTRY.


me->process_functions( ).

me->set_columns( ).

me->process_layout( i_variant ).

me->register_events( ).

me->process_report_headers( ).


alv->display( ).


ENDMETHOD.
```

PROCESS_FUNCTIONS

This method is called during the PUBLISH_ALV method processing. It controls which GUI functions are available in the ALV output screen. You can redefine this method to create custom buttons/menu options or remove standard ones. By default this method will activate all standard functions plus the XML export function. It also disables the Lotus function (since we don't use Lotus at our company).

```
METHOD process_functions.
*... Functions
*... activate ALV generic Functions
*... include own functions by setting own status
*  alv->set_screen_status(
*    pfstatus      =  'SAPLSLVC_FULLSCREEN'
*    report        =  'SAPLSLVC_FULLSCREEN' "me->repid
*    set_functions = alv->c_functions_all ).
```

```
      DATA: lr_functions TYPE REF TO cl_salv_functions_list.

      lr_functions = alv->get_functions( ).

      lr_functions->set_all( abap_true ).

      lr_functions->set_export_xml( abap_true ).

      lr_functions->set_view_lotus( abap_false ).


    ENDMETHOD.
```

SET_COLUMNS

Before the introduction of the ALV OM, you would create a Field Catalog to manipulate the number of columns and/or the settings for these columns Now there is an OO approach to this where you ask the ALV OM for a Columns Object (cl_salv_columns) and manipulate through it. The standard implementation of this method exposes all columns and sets the optimize width. This method can be redefined to create custom column processing.

```
    METHOD set_columns.
*... SET the columns
      DATA: lr_columns TYPE REF TO cl_salv_columns.

      lr_columns = alv->get_columns( ).

      lr_columns->set_optimize( abap_true ).
    ENDMETHOD.
```

PROCESS_LAYOUT

This method contains all the logic to process the ALV Grid Variants. It sets the current layout from a parameter on the dialog screen. It also sets the types of variants that can be saved (local, global, or both). The standard processing uses the dialog program name as the Variant key and doesn't restrict the type of variant that can be saved. Once again you can redefine this method to change the default processing of the ALV variants.

```
    METHOD process_layout.
*Importing     I_VARIANT    TYPE SLIS_VARI     Layout

*... set layout
      DATA: lr_layout TYPE REF TO cl_salv_layout,
            ls_key    TYPE salv_s_layout_key.

      lr_layout = alv->get_layout( ).
*... set the Layout Key
      ls_key-report = me->repid.
```

```
    lr_layout->set_key( ls_key ).
*... set usage of default Layouts
    lr_layout->set_default( abap_true ).
*... set Layout save restriction
    lr_layout->set_save_restriction( if_salv_c_layout=>restrict_none ).
*... set initial Layout
    IF i_variant IS NOT INITIAL.
      lr_layout->set_initial_layout( i_variant ).
    ENDIF.
  ENDMETHOD.
```

REGISTER_EVENTS

This method is used to register any of the event handlers. By default no events are registered. However, all the coding is in place, but commented out. You can just redefine this method and uncomment any events that you want to code for.

```
  METHOD register_events.
*... register to the events of cl_salv_table
    DATA: lr_events TYPE REF TO cl_salv_events_table.
    lr_events = alv->get_event( ).


*... register to the events (Please only register those events you are
using).
*   SET HANDLER me->on_user_command        FOR lr_events.
*   SET HANDLER me->on_before_user_command FOR lr_events.
*   SET HANDLER me->on_after_user_command  FOR lr_events.
*   SET HANDLER me->on_double_click        FOR lr_events.
*   SET HANDLER me->on_top_of_page         FOR lr_events.
*   SET HANDLER me->on_end_of_page         FOR lr_events.
  ENDMETHOD.
```

PROCESS_REPORT_HEADERS

This method is used to process the logic of the report header. You can have different output based upon whether the ALV is displayed or printed. For our processing we will separate these two approaches into two separate methods. That way they can be inherited and redefined individually if necessary.

```
  METHOD process_report_headers.
    me->process_top_of_list( ).
```

```
    me->process_top_of_list_print( ).
  ENDMETHOD.
```

PROCESS_TOP_OF_LIST

This method has the logic to work with the ALV OM to create the report header. The ALV OM has an output format that is metadata based. That means that you don't use write statements or HTML, but instead a neutral formatting method. The ALV OM itself will then interpret this data and produced the best output type for the current situation. In the processing you will see that we use GRIDs and FLOWs to control the layout, which is very similar to BSP or WebDynpro.

```abap
  METHOD process_top_of_list.


    DATA: lr_grid   TYPE REF TO cl_salv_form_layout_grid,
          lr_grid_1 TYPE REF TO cl_salv_form_layout_grid,
          lr_flow   TYPE REF TO cl_salv_form_layout_flow,
          lr_label  TYPE REF TO cl_salv_form_label,
          lr_text   TYPE REF TO cl_salv_form_text,
          l_text TYPE string.
    CREATE OBJECT lr_grid.


    IF me->title IS NOT INITIAL.
      lr_grid->create_header_information(
        row    = 1
        column = 1
        text    = me->title
        tooltip = me->title ).
    ENDIF.

*... in the cell [2,1] create a grid
    lr_grid_1 = lr_grid->create_grid(
                  row    = 2
                  column = 1 ).

*... in the cell [1,1] of the second grid create a label
    lr_text = lr_grid_1->create_text(
      row      = 1
```

```
     column  = 1
     colspan = 2
     text    = 'Kimball International (Kimball Electronics Group)'(ki1)
     tooltip = 'Kimball International (Kimball Electronics Group)'(ki1) ).


  lr_flow  = lr_grid_1->create_flow(
    row     = 2
    column  = 1 ).


  lr_label = lr_flow->create_label(
    text    = 'Program:'(t02)
    tooltip = 'Program: '(t02) ).


  lr_text = lr_flow->create_text(
    text    = sy-cprog
    tooltip = sy-cprog ).


  lr_flow  = lr_grid_1->create_flow(
    row     = 3
    column  = 1 ).


  lr_label = lr_flow->create_label(
    text    = 'System:'(t03)
    tooltip = 'System:'(t03) ).


  lr_text = lr_flow->create_text(
    text    = sy-sysid
    tooltip = sy-sysid ).


  lr_flow  = lr_grid_1->create_flow(
    row     = 3
    column  = 2 ).


  lr_label = lr_flow->create_label(
```

```
      text    = 'Client:'(t04)
      tooltip = 'Client:'(t04) ).


  lr_text = lr_flow->create_text(
      text    = sy-mandt
      tooltip = sy-mandt ).


  lr_flow  = lr_grid_1->create_flow(
        row     = 4
        column  = 1 ).


  DATA: date1(12) TYPE c.
  DATA: time1(8) TYPE c.
  WRITE sy-datum TO date1.
  WRITE sy-uzeit TO time1.
  DATA: tzonesys TYPE tznzonesys.
  SELECT SINGLE tzonesys FROM ttzcu INTO tzonesys.
  lr_label = lr_flow->create_label(
      text    = 'Date:'(t05)
      tooltip = 'Date:'(t05) ).


  lr_text = lr_flow->create_text(
      text    = date1
      tooltip = date1 ).



  lr_flow  = lr_grid_1->create_flow(
        row     = 4
        column  = 2 ).
  lr_label = lr_flow->create_label(
      text    = 'Time:'(t06)
      tooltip = 'Time:'(t06) ).


  lr_text = lr_flow->create_text(
```

```
    text    = time1
    tooltip = time1 ).


  lr_text = lr_flow->create_text(
    text    = tzonesys
    tooltip = tzonesys ).


  IF sy-timlo NE sy-uzeit.
    WRITE sy-datlo TO date1.
    WRITE sy-timlo TO time1.
    lr_flow  = lr_grid_1->create_flow(
       row     = 5
       column  = 1 ).
    lr_label = lr_flow->create_label(
      text    = 'Local Date:'(t07)
      tooltip = 'Local Date:'(t07) ).


    lr_text = lr_flow->create_text(
      text    = date1
      tooltip = date1 ).



    lr_flow  = lr_grid_1->create_flow(
       row     = 5
       column  = 2 ).
    lr_label = lr_flow->create_label(
      text    = 'Local Time:'(t08)
      tooltip = 'Local Time:'(t08) ).


    lr_text = lr_flow->create_text(
      text    = time1
      tooltip = time1 ).


    lr_text = lr_flow->create_text(
```

```
       text    = sy-zonlo

       tooltip = sy-zonlo ).

    ENDIF.


    alv->set_top_of_list( lr_grid ).


  ENDMETHOD.
```

PROCESS_TOP_OF_LIST_PRINT

For the purpose of our template, we want to produce the same output when printed as when displayed on-line. Therefore the standard implementation of this method will just call over to its online counterpart. However thought redefinition, an individual application can create specific processing for the print version of the header that differs from the on-line version.

```
METHOD process_top_of_list_print.

  me->process_top_of_list( ).

ENDMETHOD.
```

Author Bio

Thomas Jung is an applications developer for the Kimball Electronics Group. He has been involved in SAP implementations at Kimball as an ABAP developer for over 9 years. He has done work in the Microsoft world with VB and .NET Development, but his first love remains as always: ABAP. For several years, Tom has been involved in the use of BSP development at Kimball and more recently in the introduction of ABAP web services for critical Interfaces and WebDynpro ABAP. He holds the Special Interest Group Chair position for Web Technologies within ASUG (America's SAP User's Group). He is also the co-author of the SAP PRESS Book, Advanced BSP Programming.