Quick tips and tricks on Web Dynpro for ABAP – How to use ALV and Pop ups

Applies to:

SAP Web Dynpro for ABAP

Summary

This is a collection of quick tips and comprehensive ready-to-use ABAP code samples on different aspects of SAP Web Dynpro. Mainly the use of the ALV and pop-ups. It can be useful to anyone developing Web Dynpro for ABAP. For those who are fairly new to ABAP the code samples are documented and can mostly be copied straight away. For more seasoned developers this document can be useful to kick start their work on the discussed items.

Author(s): Joris Bots

Company: SAP Netherlands

Created on: 23 August 2006

Author Bio

Joris has been working for SAP The Netherlands since April 2006 as a certified ABAP Netweaver Development Consultant. Prior to that he has developed business applications in variety of 3- and 4- GL languages

Table of Contents

ALV in Web Dynpro

Use ALV in Web Dynpro

Incorporate ALV

Check one of these documents for understanding how to incorporate an ALV in your Web Dynpro

…

…

Get reference to ALV component and configuration model

To obtain a reference to the ALV component and its configuration model you can use the following code.

The configuration model can be used to set all sorts of properties of the ALV

```
  DATA: lr_salv_wd_table TYPE REF TO iwci_salv_wd_table,
        r_table TYPE REF TO CL_SALV_WD_CONFIG_TABLE.

* get reference to ALV component interface
  lr_salv_wd_table = wd_this->wd_cpifc_alvmain( ).


* get ConfigurationModel from ALV Component
  wd_this->r_table = lr_salv_wd_table->get_model( ).
```

Hide / add function buttons in your ALV

You can hide all function buttons of the ALV with one of these methods.

```
    * Set toolbar visibility to false.
      data:
        lr_function_settings type ref to if_salv_wd_function_settings.

        lr_function_settings ?= wd_this->r_table.
        lr_function_settings->set_visible( CL_WD_UIELEMENT=>E_VISIBLE-NONE ).

    * set default ALV Functions off
      data:
        lr_standard_functions type ref to if_salv_wd_std_functions.

        lr_standard_functions ?= wd_this->r_table.
        lr_standard_functions->set_sort_headerclick_allowed( ABAP_false ).
        lr_standard_functions->set_filter_filterline_allowed( ABAP_false ).
        lr_standard_functions->set_filter_complex_allowed( ABAP_false ).
        lr_standard_functions->set_sort_complex_allowed( ABAP_false ).
```

You can set individual functions with these methods:


```
CALL METHOD cl_salv_wd_config_table=>if_salv_wd_standard_functions~set_<x>_allowed
   EXPORTING
      Value = ABAP_true.
```

Where <x> is to be replaced with the property of your choice.

Enable the Settings dialog in your WD

Besides making the settings screen available, you also can set which functionality should be anabled inside the setting screen

```
* show SETTINGS button
  CALL METHOD wd_this->r_table->if_salv_wd_std_functions~
                       set_dialog_settings_allowed
    EXPORTING
      value = ABAP_true.

* show tab COLUMN SELECTION in Settings page
  CALL METHOD wd_this->r_table->if_salv_wd_std_functions~
                        set_column_selection_allowed
    EXPORTING
      value = ABAP_true.
```

Change the position of a column

```
DATA: ls_column TYPE salv_wd_s_column_ref.
ls_column-r_column->set_position( 3 ).
```

NOTE: By default all columns have position set to 0. Columns are added ‘left justified’ to the table. This means that you can change the position of just one column without the need to explicitly set the position of all columns…isn’t life beautiful with the ALV ?!

Automatically set number of visible rows to number of items

If you want your ALV to be as long as the list of rows in your dataset, you can use this trick.

Set the visible row count to -1 and disable footer to hide page buttons.

You can use the iview scrollbars to scroll. This can be a performance gain, since the ALV has to be loaded once, not with every scroll.

```
DATA: lr_table_settings TYPE REF TO if_salv_wd_table_settings.
"get reference of table settings by cast
lr_table_settings ?= wd_this->r_table .


"set rowcount to -1 and disable footer
```



```
      lr_table_settings->set_visible_row_count( -1 ).
      lr_table_settings->set_footer_visible( 0 ).
```

Set properties of individual columns

An easy way to set individual properties of columns is this:

Get a table of references to all columns of an ALV and loop over it. This way you can set each column as needed. See chapter *Get reference to ALV component and configuration model* if you don't know how to get a reference to the configuration model.

```
* init ColumnSettings
  DATA: lr_column_settings TYPE REF TO if_salv_wd_column_settings,
        lr_col_header TYPE REF TO cl_salv_wd_column_header.
  lr_column_settings ?= wd_this->r_table.

* get table of column settings - each line one column
  DATA: lt_columns TYPE salv_wd_t_column_ref.
  lt_columns = lr_column_settings->get_columns( ).

* loop over table - in each loop another column can be modified
  DATA: ls_column TYPE salv_wd_s_column_ref.

* define visible columns (fields) by naming them,
* exclude others by setting visibility to none
  DATA: ls_tooltip TYPE string.
  LOOP AT lt_columns INTO ls_column.

    " get header of column
    lr_col_header = ls_column-r_column->get_header( ).

    " do settings here
 ENDLOOP.
```

Put a button (or an other UI element) inside the cells of a column

It is possible to set all sorts of UI elements inside the cells of a column. Those elements are called cell editors. In this example we set a button as cell editor. Create a button element and set its tooltip (optional) . Then set the button as the cell editor of the button. See chapter *Set properties of individual columns* if you don't know how to get a table of references to all columns (lt_columns)

```
    LOOP AT lt_columns INTO ls_column.
    CASE ls_column-id.
        WHEN 'DBC_ICO'.
          DATA: lr_button TYPE REF TO cl_salv_wd_uie_button.
          CREATE OBJECT lr_button.
          " set width to 1 to make column as small as possible
          " the width is always adapted to minimum width of celleditor
          ls_column-r_column->set_width( '1' ).
```



```
" image source is in the value of the cell
lr_button->set_image_source_fieldname( ls_column-id ).

" retrieve translatable text for tooltip
ls_tooltip = cl_wd_utilities=>get_otr_text_by_alias( 'NNL1/DBC' ).

  CALL METHOD lr_button->set_tooltip
    EXPORTING
      value = ls_tooltip.

  CALL METHOD ls_column-r_column->set_cell_editor
    EXPORTING
      value = lr_button.
```

Use S/M/L text from domain for column header

By default the column text shown in the ALV are the S (short) texts from the domain. You can set the M or L text like this: See chapter *Set properties of individual columns* if you don't know how to get a table of references to all columns (lt_columns)

```
LOOP AT lt_columns INTO ls_column.
    " get header of column
    lr_col_header = ls_column-r_column->get_header( ).
    " change columnheader text to medium text of domain
      CALL METHOD lr_col_header->set_ddic_binding_field
        EXPORTING
          value = if_salv_wd_c_column_settings=>ddic_bind_medium.
ENLOOP.
```

Messages

Show a message

To invoke a message that is shown in the Web Dynpro try the following. Use transaction SE91 to manage the message texts

```
" report message gpart_not_found
CALL METHOD wd_comp_controller->message_manager->report_t100_message
  EXPORTING
    msgid = 'my_messageclass'
    msgno = '503'
    msgty = 'E'.
```

Pop ups

Show a Web Dynpro as a (modal) popup

To show a WebDynpro as a popup we need quite a bit of code. The example below shows a 'Used' Web Dynpro window. If you want to show a 'local' window as a popup, please check the other methods of interface if_wd_window_manager. Please note that it depends on Portal settings whether or not the pop up is shown as a truly modal pop up.

```
DATA: l_ref_cmp_usage TYPE REF TO if_wd_component_usage.

" get handle to usage comp with name 'popup_text'
```



```
    " and make sure it is instantiated

    " Note the name of the usage component has 'wd_cpuse' in front of its name
    l_ref_cmp_usage =   wd_this->wd_cpuse_popup_text( ).
    IF l_ref_cmp_usage->has_active_component( ) IS INITIAL.
      l_ref_cmp_usage->create_component( ).
    ENDIF.


    " optionally call a method of the used component to fill data, etc
    DATA: l_ref_interfacecontroller TYPE REF TO iwci_ish_nl_wd_dbc_textpopup .
    l_ref_interfacecontroller =   wd_this->wd_cpifc_popup_text( ).
    l_ref_interfacecontroller->prepare_popup(
      i_some_data = 'this can be used in called window'
    ).


  DATA: l_cmp_api TYPE REF TO if_wd_component,
    l_window_manager TYPE REF TO if_wd_window_manager,
    lv_title TYPE string.

    " get window_manager
    l_cmp_api = wd_comp_controller->wd_get_api( ).
    l_window_manager = l_cmp_api->get_window_manager( ).

    " and finally call the sucker! (...the pop up window)
    wd_this->m_pop_text = l_window_manager->create_window_for_cmp_usage(
    interface_view_name = 'WINDOW_OF_TEXTPOPUP'
    component_usage_name = 'POPUP_TEXT'
    close_in_any_case = ABAP_false
    title = 'Title in title bar'
    ).


   " and show the popup
   wd_this->m_pop_text->open( ).
```

Hide the windows close button in upper right of popup

You might want more control over how the pop up is closed. To do this you can add a close button on the pop up. In that case you want to disable the 'windows style' close button in the upper right to prevent users from closing it that way. The code is simple. (See previous item if you are not sure ho to get the reference to M_POPUP_TEXT)

```
    " hide windows close-button
    wd_this->m_pop_text->set_close_button( ABAP_false ).
```

Have a pop up close itself

If you want a pop up to close itself you need to get a reference to the embedding window.

It's not hard, just see below.

```
  " close popup, no questions asked!
```



```
DATA: l_api TYPE REF TO if_wd_view_controller,
      l_window_ctlr TYPE REF TO if_wd_window_controller,
      l_popup TYPE REF TO if_wd_window.


" get embedding window

l_api = wd_this->wd_get_api( ).
l_window_ctlr = l_api->get_embedding_window_ctlr( ).


" and close the pop up

IF l_window_ctlr IS BOUND.
  l_popup = l_window_ctlr->get_window( ).
  l_popup->close( ).
ENDIF.
```

Show a dialog pop up and handle the button-clicks

To show a Dialog pop-up and handle its button clicks use the following method. Note that the pop up is always modal.

```
DATA: l_cmp_api TYPE REF TO if_wd_component,
      l_window_manager TYPE REF TO if_wd_window_manager,
      l_popup TYPE REF TO if_wd_window,
      l_api TYPE REF TO if_wd_view_controller,

      l_text TYPE string_table.


" Get Window manager

l_cmp_api = wd_comp_controller->wd_get_api( ).
l_window_manager = l_cmp_api->get_window_manager( ).


" fill table with text

INSERT 'first line to appear in dialog' INTO TABLE l_text.
INSERT 'Second line to appear in dialog' INTO TABLE l_text.


" Create the pop up
" See interface if_wd_window for buttonkinds and message types
l_popup = l_window_manager->create_popup_to_confirm(
          text            = l_text
          button_kind     = if_wd_window=>co_buttons_yesnocancel
          message_type    = if_wd_window=>co_msg_type_question
          window_title    = 'My cool title' )
          window_position = if_wd_window=>co_center
          close_button    = ABAP_false
          ).


" subscribe YES button to action ACT_YES

l_popup->subscribe_to_button_event(
  button = if_wd_window=>co_button_yes
  action_name = 'ACT_YES'
  action_view = l_api
```



```
    is_default_button = ABAP_true
    ).
```

Events

Catch parameters from portal events

To collect parameters of an event triggered by the portal eventing (like relative Navigation) do the the following in the DEFAULT HANDLER method of your window (handler method of the startup-plug).

The result is a table with all parsed parameters.

```
  DATA: all_parameters TYPE tihttpnvp.

* Get all params from inbound portal event
  CALL METHOD wdevent->get_data
    EXPORTING
      name  = '_ALL_URL_PARAMETERS'
    IMPORTING
      value = all_parameters
```


Disclaimer and 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.

