Using Dynamic ALV with Web Dynpro ABAP with Editable Fields

Applies to:

Web Dynpro ABAP. For more information, visit the User Interface Technology homepage.

Summary

This tutorial explains the basics of creating an internal table whose structure is known only at runtime and editable ALV list display for dynamic internal tables.

Author: Mazin Mahmood

Company: Keane India Private Limited

Created on: 28 January 2010

Author Bio

Mazin Mahmood is a Senior Software Engineer at Keane India Private Limited and is working on Enterprise Portals, Web Dynpro for ABAP and Product and Portfolio Management.

Table of Contents

The Scenario

Let us suppose that you have an internal table whose structure is known only at runtime. This article shows us how to create the internal table and to display it as an ALV list with editable fields.

Creation of Internal Table and Context Node

We will first see how to create an internal table having 2 fields FLD1 and FLD2. Comments will be mentioned in text boxes embedded within the code for better understanding.

Code for Creating Dynamic Internal Table

```
data : lt_dyn_table type ref to data,
         lw_dyn_table type ref to data,
         lt_fieldcat type lvc_t_fcat,
         lw_fieldcat like line of lt_fieldcat.

field-symbols : <fs_table> type standard table,
                  <fs_wa> type any,
                  <fs_variable> type any.

  clear lt_fieldcat.
  lw_fieldcat-fieldname = 'FLD1'.
  lw_fieldcat-datatype = 'string'.
  lw_fieldcat-outputlen = '24'.
  lw_fieldcat-coltext = 'FLD1'.
  lw_fieldcat-seltext = lw_fieldcat-coltext.
  append lw_fieldcat to lt_fieldcat.


  lw_fieldcat-fieldname = 'FLD2'.
  lw_fieldcat-datatype = 'string'.
  lw_fieldcat-outputlen = '24'.
  lw_fieldcat-coltext = 'FLD2'.
  lw_fieldcat-seltext = lw_fieldcat-coltext.
  append lw_fieldcat to lt_fieldcat.


call method cl_alv_table_create=>create_dynamic_table
    exporting
      it_fieldcatalog           = lt_fieldcat
    importing
      ep_table                  = lt_dyn_table
    exceptions
      generate_subpool_dir_full = 1
      others                    = 2.
  if sy-subrc <> 0.
    message id sy-msgid type sy-msgty number sy-msgno
    with sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
  endif.
```

Note: The above code is used to create an internal table having 2 string fields with character length of 24. We use field symbols for this purpose.

Code for Populating Values within the Internal Table

```
assign lt_dyn_table->* to <fs_table>.
  create data lw_dyn_table like line of <fs_table>.
  assign lw_dyn_table->* to <fs_wa>.

assign component 'FLD1' of structure <fs_wa> to <fs_variable>.
      <fs_variable> = 'Text for FLD1'.

      assign component 'FLD2' of structure <fs_wa> to <fs_variable>.
      <fs_variable> =Text for FLD2.
append <fs_wa> to <fs_table>.
```

Code for Creation of Context Node

```
DATA:
rootnode_info TYPE REF TO if_wd_context_node_info,
dyn_node TYPE REF TO if_wd_context_node,
tabname_node TYPE REF TO if_wd_context_node,
struct_type  TYPE REF TO cl_abap_structdescr,

comp_tab TYPE cl_abap_structdescr=>component_table,
comp LIKE LINE OF comp_tab.

comp-name = 'FLD1'.
comp-type ?= cl_abap_datadescr=>describe_by_name( 'STRING' ).
      APPEND comp TO comp_tab.
comp-name = 'FLD2'.
comp-type ?= cl_abap_datadescr=>describe_by_name( 'STRING' ).
      APPEND comp TO comp_tab.

*Code for getting node info of root node
 rootnode_info = wd_context->get_node_info( ).

*Code for creating a new node with structure defined in comp_tab
struct_type = cl_abap_structdescr=>create( comp_tab ).

rootnode_info = rootnode_info->add_new_child_node(
    name                        = 'NEW_NODE'
    is_mandatory                = abap_true
    is_multiple                 = abap_true
    static_element_rtti         = struct_type
    is_static                   = abap_false
       ).
```

```
dyn_node = wd_context->get_child_node( name = 'NEW_NODE' ).
* Bind internal table to context node.
dyn_node->bind_table( <fs_table> ).
```

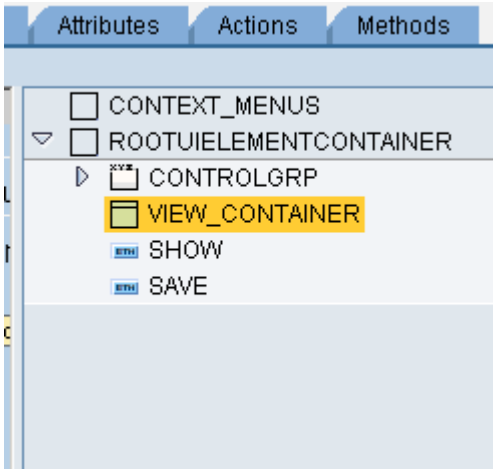
Note: cl_abap_structdescr=>component_table is used to describe the structure of the node. Assuming that we want to create a new node in Web Dynpro context under the root with the name of 'NEW_NODE' corresponding to this internal table, we can do it with the above method. Here <FS_TABLE> is the field symbol containing the data as a table.

Ok, so now we have the internal table and the node in the context at runtime. Now we need to display these in front end in the form of an ALV list display.

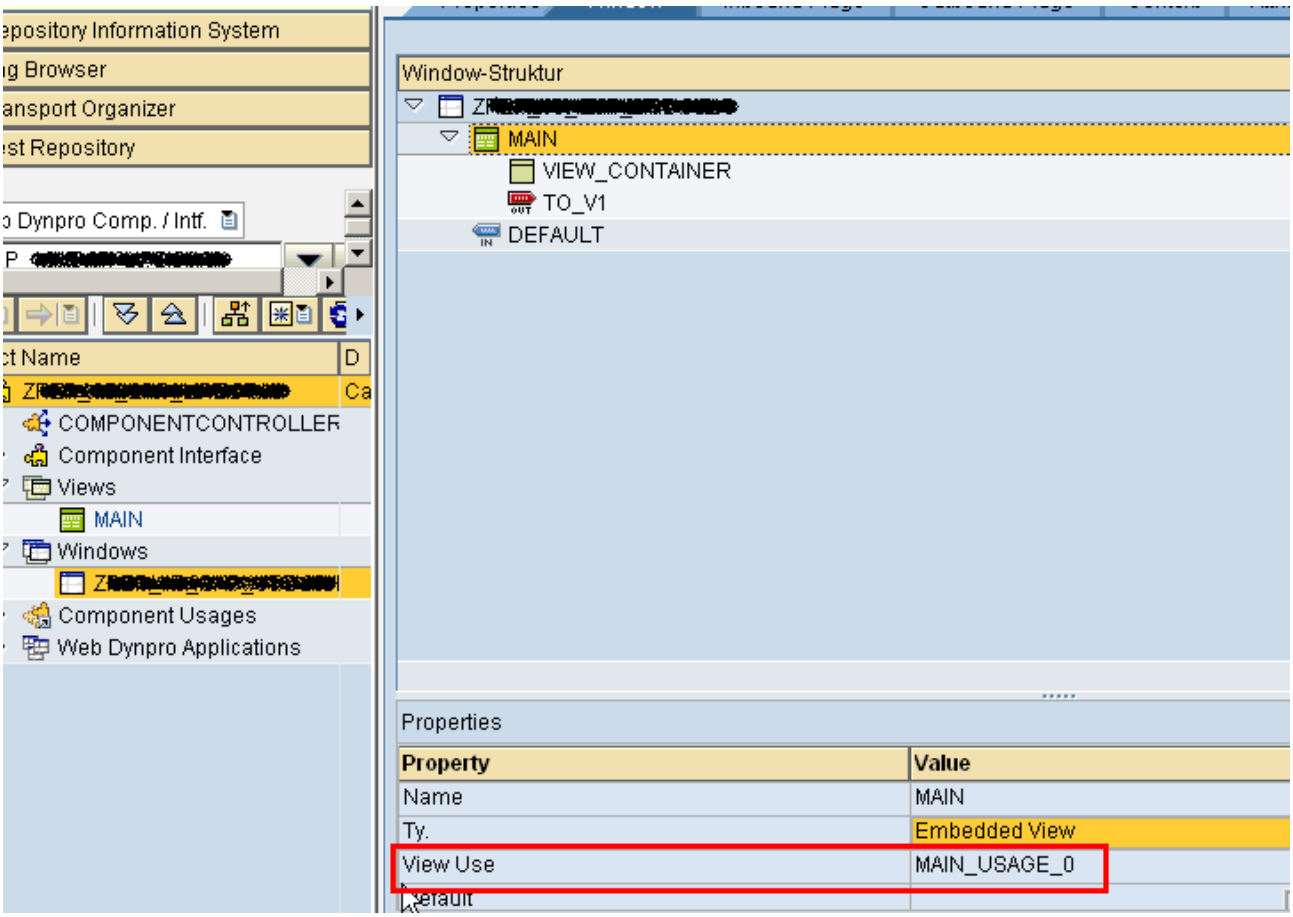
Prerequisites for ALV List Display

Creation of View Container

Before we code for ALV, we need to make sure of a couple of things.



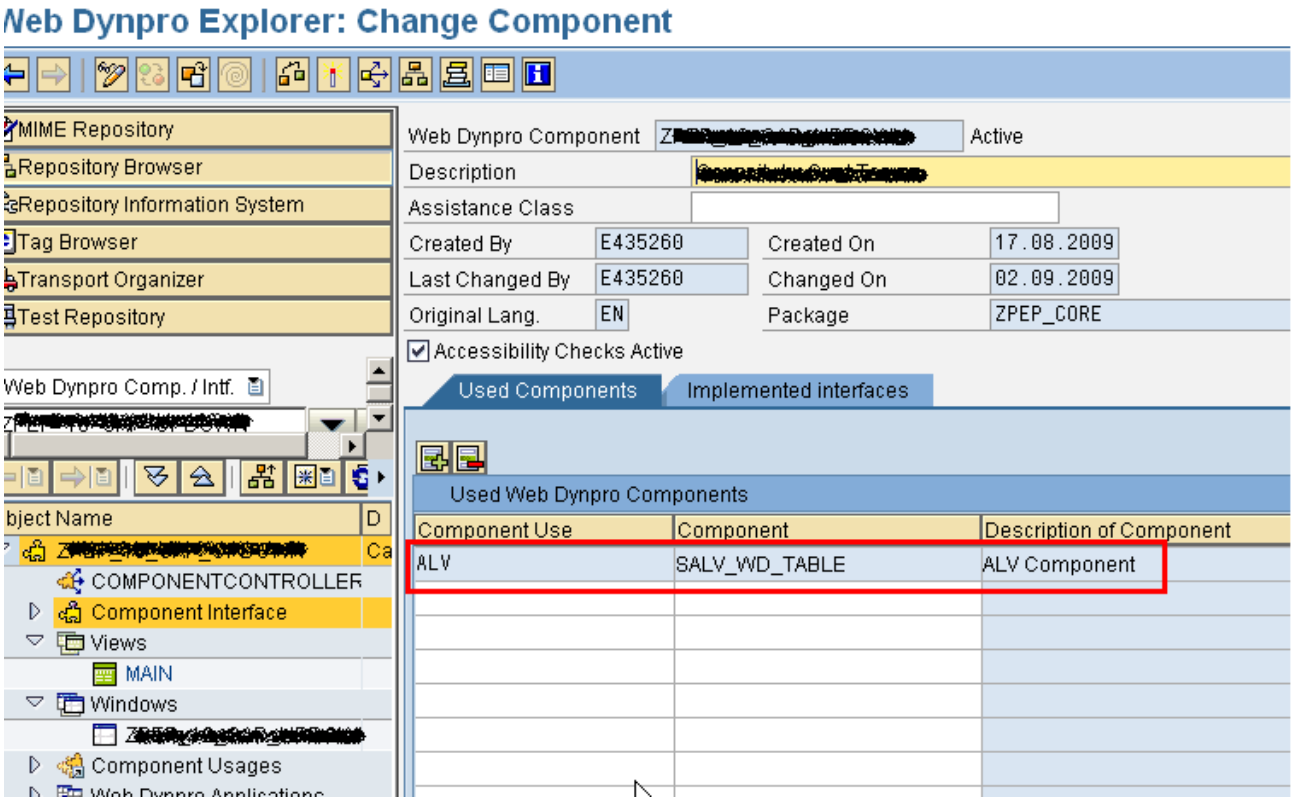
We first need to create a View Container UI element in the view as shown above. The view container is used to hold the view of the ALV list display.

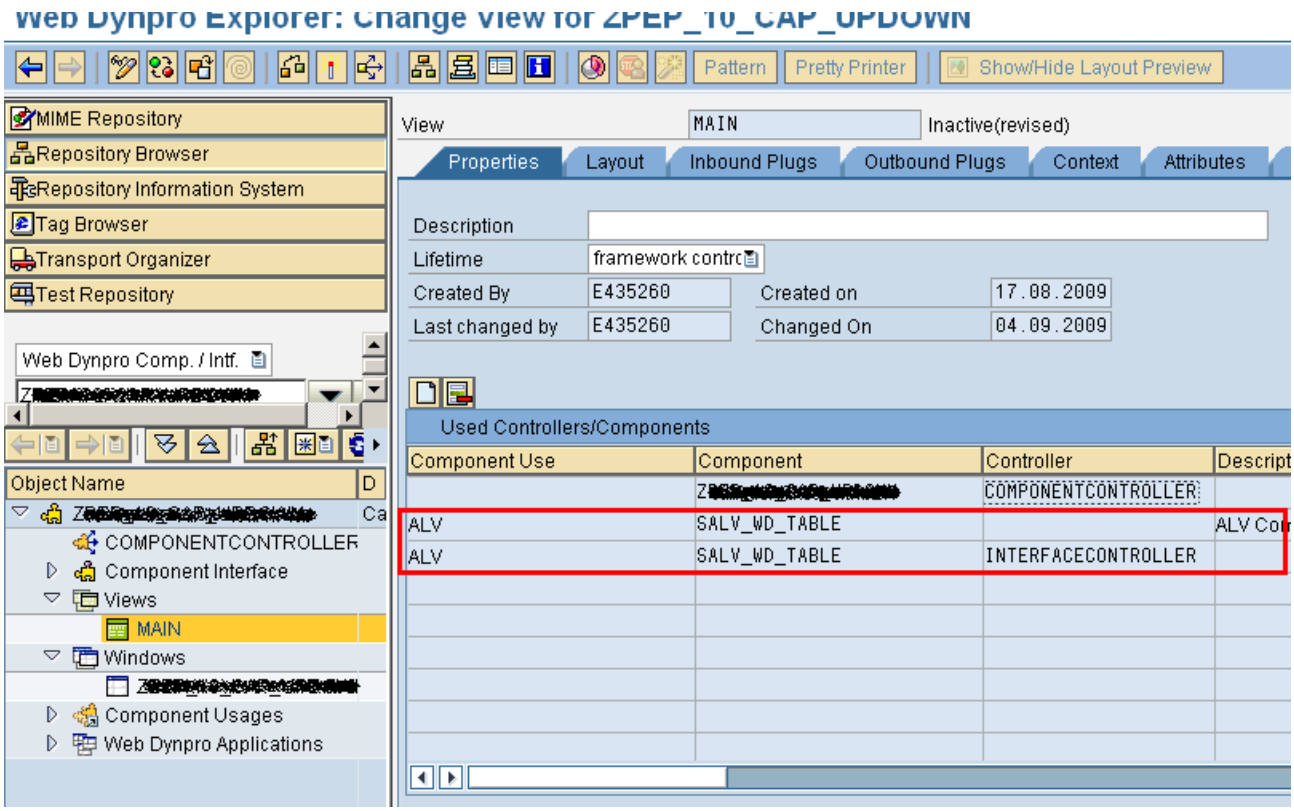


Once we create a view container, it will come under the window within the view. Find the view use of the main view and not this down. In our case, it is MAIN_USAGE_0.

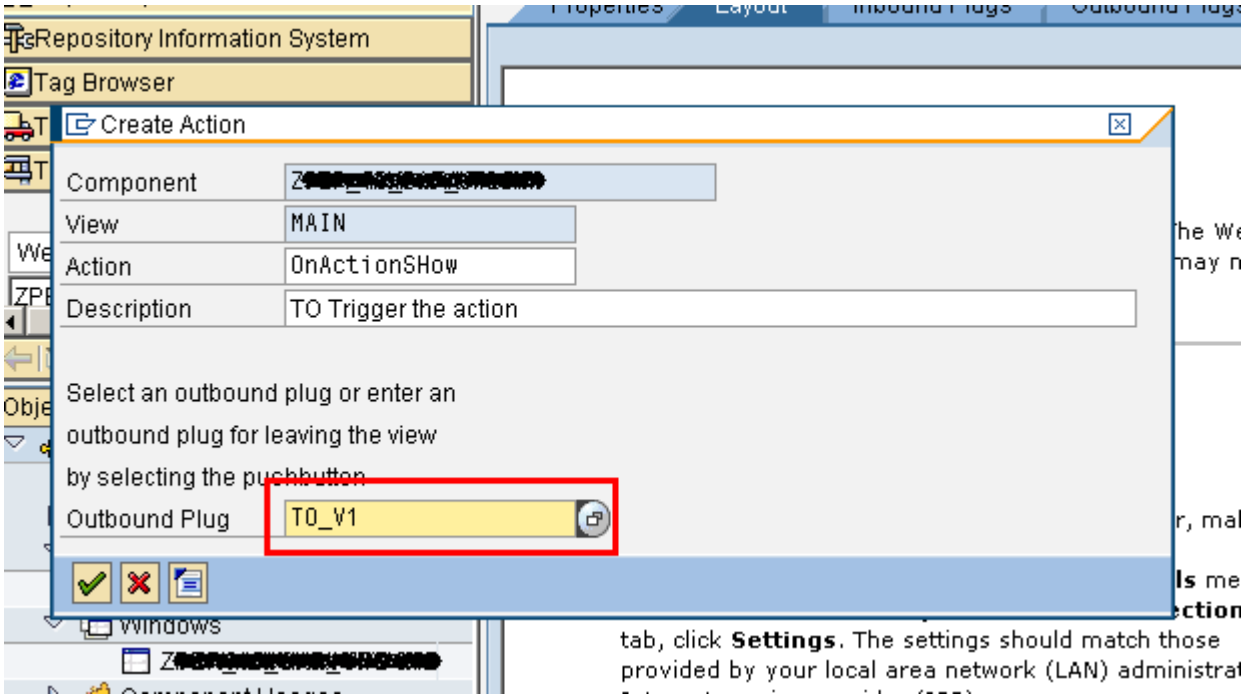
Declaration of the Standard ALV Web Dynpro Component

SAP delivers the standard ALV Web Dynpro component SALV_WD_TABLE which is used for ALV display. We can reuse this within our Web Dynpro report. Before we do that, we have to declare it within our component controller and view as shown below respectively.





Create a new action and attach it to an outbound plug. Here we have used the action for a button and attached the event trigger to the outbound plug 'TO_V1'.



Code to Call the ALV Display

Now that everything is set, what we would want is to call the ALV list display when we call the event after populating the internal table.

For this we can make use of the following code within the event created above.

```
data lo_cmp_usage type ref to if_wd_component_usage.
data : L_VIEW_CONTROLLER_API type ref to IF_WD_VIEW_CONTROLLER,
      lr_column      type ref to cl_salv_wd_column,            " Table fo rcolumn ref
      ls_columns     type salv_wd_s_column_ref,                " Structure for column ref
      lt_columns     type salv_wd_t_column_ref,                " Structure for column ref
      lr_input       type ref to cl_salv_wd_uie_input_field,
      l_value type ref to cl_salv_wd_config_table,
      lr_column_settings type ref to if_salv_wd_column_settings,
      lr_table_settings type ref to if_salv_wd_table_settings.


lo_cmp_usage =   wd_this->wd_cpuse_alv( ).
if lo_cmp_usage->has_active_component( ) is initial.
  lo_cmp_usage->create_component( ).
endif.


data lo_interfacecontroller type ref to iwci_salv_wd_table .
lo_interfacecontroller =   wd_this->wd_cpifc_alv( ).


  lo_interfacecontroller->set_data(
*   only_if_new_descr =                " wdy_boolean
    "r_node_data =
    dyn_node                    " ref to if_wd_context_node
  ).


l_value = lo_interfacecontroller->get_model( ).
  lr_std ?= l_value.
  lr_std->set_export_allowed( abap_false ).
  lr_std->set_pdf_allowed( abap_false ).
  lr_std->set_edit_append_row_allowed( abap_false ).
  lr_std->set_edit_check_available( abap_false ).
  lr_std->set_edit_delete_row_allowed( abap_false ).
  lr_std->set_edit_insert_row_allowed( abap_false ).

  call method l_value->if_salv_wd_column_settings~get_columns
    receiving
      value = lt_columns.
  loop at lt_columns into ls_columns.
    if sy-tabix eq 1.
  ls_columns-r_column->set_visible( value = '99' ). " This sets First column as invisible
    endif.
    else sy-tabix gt 6.
      lr_column = ls_columns-r_column.
      create object lr_input
        exporting
          value_fieldname = ls_columns-id.
      call method lr_column->set_cell_editor
```

```
        exporting
          value = lr_input.
      lr_table_settings ?= l_value.
      lr_table_settings->set_read_only( abap_false ). " Rest of the columns are editable
    end.

  endloop.
L_VIEW_CONTROLLER_API = WD_THIS->WD_GET_API( ).
L_VIEW_CONTROLLER_API->PREPARE_DYNAMIC_NAVIGATION(
source_window_name = 'Z******************'     "The name of the Window
source_vusage_name = 'MAIN_USAGE_0'            "The parameter shown
source_plug_name = 'TO_V1'                     "Define a New Outbound Plug
target_component_name = 'SALV_WD_TABLE'        "All Standard
target_component_usage = 'ALV'                 "All Standard
target_view_name = 'TABLE'                     "All Standard
target_plug_name = 'DEFAULT'                   "All Standard
target_embedding_position = 'MAIN/VIEW_CONTAINER'    "Main View name / *Container name
 ).

WD_THIS->FIRE_TO_V1_PLG( ).
```

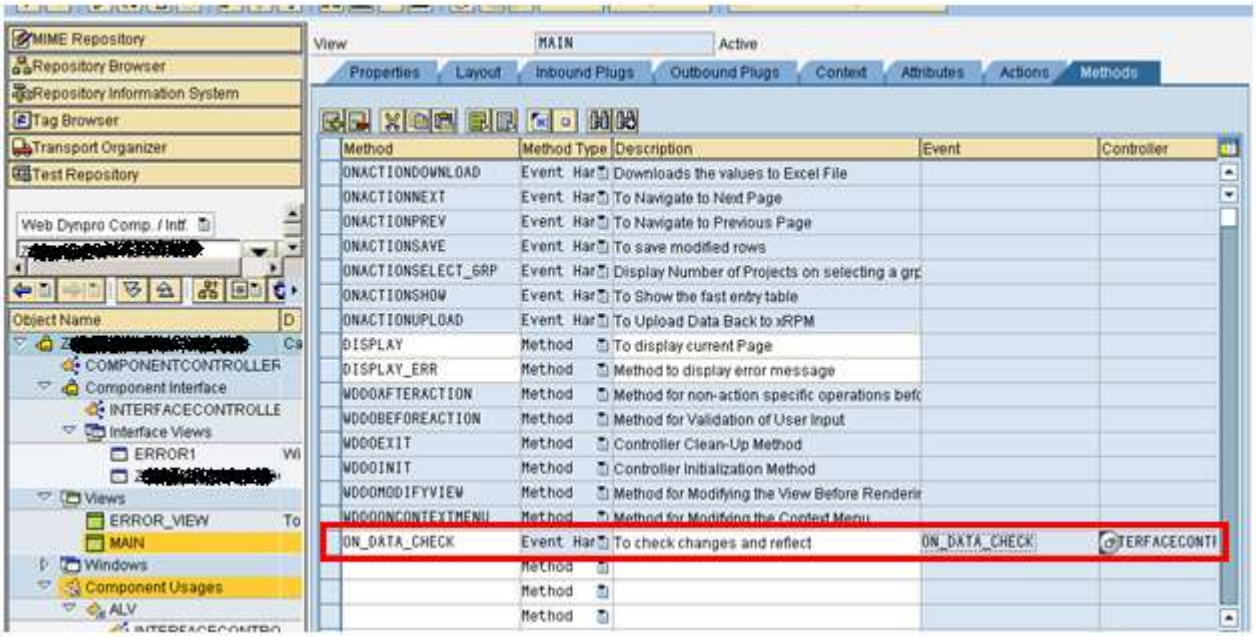
The above code when called is responsible for displaying the ALV list.

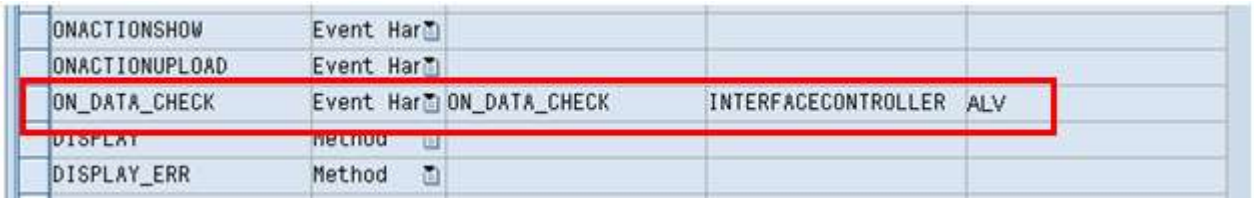
Checking the Data after User Input

Now that everything has been created, the ALV is called successfully once the action is executed. The only thing remaining now is to check if the user has changed the data and find out where. It would be difficult to compare the old values of context nodes and the new values. So a more efficient method has been developed.

Prerequisites

The first thing that we have to do is to create a new method ON_DATA_CHECK in the Web Dynpro (Be sure that this name is not changed). Give method type as event handler and Event as ON_DATA_CHECK. Give the controller as INTERFACECONTROLLER and Component Usage as ALV as shown.





The method ON_DATA_CHECK is a standard method in the ALV controller SALV_WD_TABLE. It can be called within any method within our Web Dynpro with the code below.

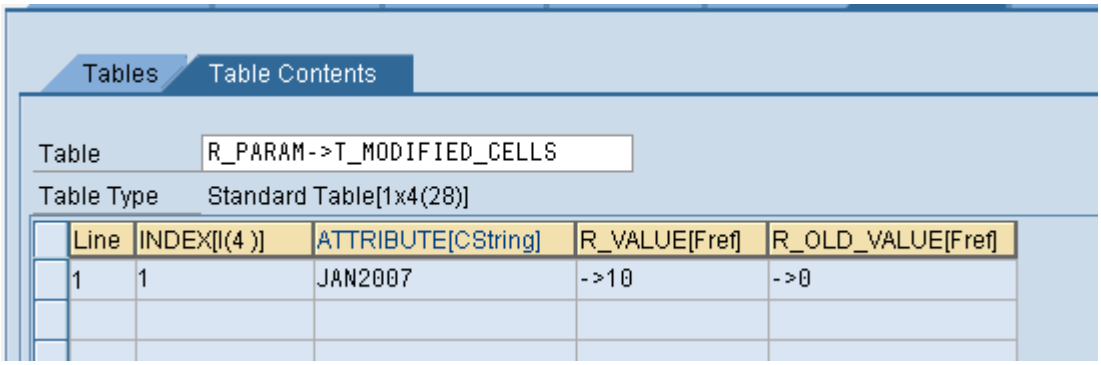
Code

```
DATA: l_ref_interfacecontroller TYPE REF TO iwci_salv_wd_table .
l_ref_interfacecontroller = wd_this->wd_cpifc_alv( ).
l_ref_interfacecontroller->data_check( ).
```

Validation

On executing the above code, the system checks if any data has been changed in the ALV and then calls the ON_DATA_CHECK method if there are any changes. Otherwise, nothing happens.

The method ON_DATA_CHECK populates the table r_param during runtime. The Table R_PARAM->T_MODIFIED_CELLS has a structure as shown below. The INDEX indicates the row of the context and ATTRIBUTE indicates the column name of the context (FLD1). R_VALUE has modified value and R_OLD_VALUE refers to previous value.



The below code within the ON_DATA_CHECK method is then used to move the new value to an internal table where it can be used and/or updated back to the context. <fs_context> represents an internal table containing the context data of the Web Dynpro.

```
data : wa_param like line of r_param->t_modified_cells.
data : lo_nd_projects type ref to if_wd_context_node,
       lo_el_projects type ref to if_wd_context_element.

field-symbols : <fs_context> type standard table,
                <fs_context2> type any,
                <fs_variable2> type any,
                <fs_variable> type any.

lo_nd_projects = wd_context->get_child_node( name =
  'NEW_NODE' ).
  lo_el_projects = lo_nd_projects->get_element(  ).
  call method lo_nd_projects->get_static_attributes_table
    importing
      table = <fs_context>.

loop at r_param->t_modified_cells into wa_param.
*Loop at changed entries and modify them in the dynamic internal table
    read table <fs_context> index wa_param-index into <fs_context2>.
    assign component wa_param-attribute of structure <fs_context2> to <fs_variable>.
    assign wa_param-r_value->* to <fs_variable2>.
    move <fs_variable2> to <fs_variable>.
```

Endloop.

```
lo_nd_projects = wd_context->get_child_node( name = 'NEW_NODE' ).
lo_el_projects = lo_nd_projects->get_element(  ).
lo_nd_projects->bind_table( <fs_context> ).
```

If all of the above has been done correctly, you should be able to get a dynamic editable ALV output.

Hope that this was useful ☺

Related Content

Blog for ALV by Krishnakumar

Blog for ALV by Marilyn Pratt

Blog for ALV by Stefan Weber

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.