SDN Contribution

Web Dynpro ABAP Context Tool – Debug Jockey

Applies to:

SAP NetWeaver 2004s – SP7 or higher

Summary

A Web Dynpro for ABAP component that can easily be used in custom development to assist with analyzing context values at runtime. Debugging the context is difficult and involves complicated drill-downs on classes in the debugger. This tool can display every node, every element, and every attribute at once. It even allows you to change the values!

Author(s): Phil Soady

Company: SAP Australia

Created on: 9 Aug 2006

Author Bio

Phil has been with SAP for more than 16 years; 5 years in Walldorf as a developer, and 10 years plus as a technical consultant. Currently he is a solution architect, specializing in technical tools such as Web Dynpro and XI.

Table of Contents

Debugging Web Dynpro Contexts Using the Debug Jockey Tool

When you first start using Web Dynpro one of the obvious deficiencies is an easy way to view and change the value of the context as you debug. The context seems out of sight when debugging.

You can view the WD_CONTEXT as an ABAP Class OBJECT. This is very limited and timing consuming in terms of accessing attribute values.

As of NW04s SP8 you will notice a new debug option in the ABAP debugger called the DATA explorer.

Whilst the data explorer is better at viewing the context than straight ABAP object viewer, it is still very clumsy at handling large contexts and multiple clicks are still required to show 1 element at a time.

With Debug jockey all you need to do is embed and pass a simple reference to the context node WD_CONTEXT. The rest is taken care for you. You can view and edit the entire context in one easy place.

How Does Debug Jockey work?

Debug Jockey can be used in two different ways:

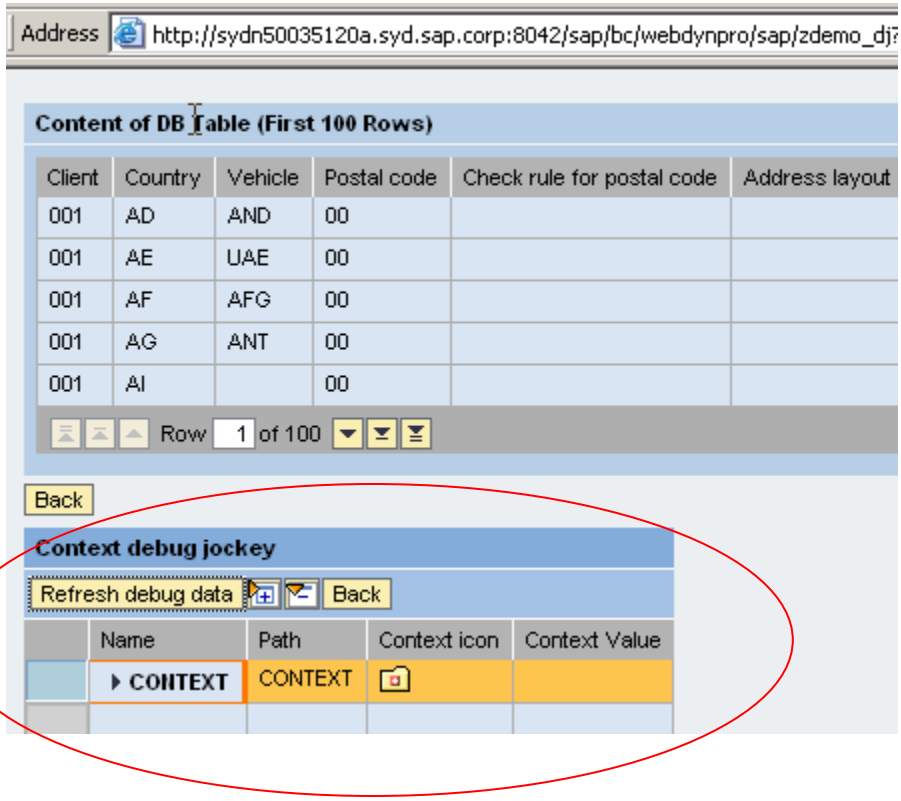
- You can embed the view in a view container next to you main view (recommended),
- You can navigate to and back from the component if you don't wish to define a view container to host debug jockey.

Demo Hosting Application is a Copy from NET310's Dynamic SE16

A simple piece of code that some may remember from the SE16 exercise (thanks to Stefan Ehret) was copied from the notes to make a simple shell application as a host to demonstrate DEBUG JOCKEY. I like the pseudo SE16 tool as a hosting application as is can easily have different context values due to its dynamic nature with very little code.

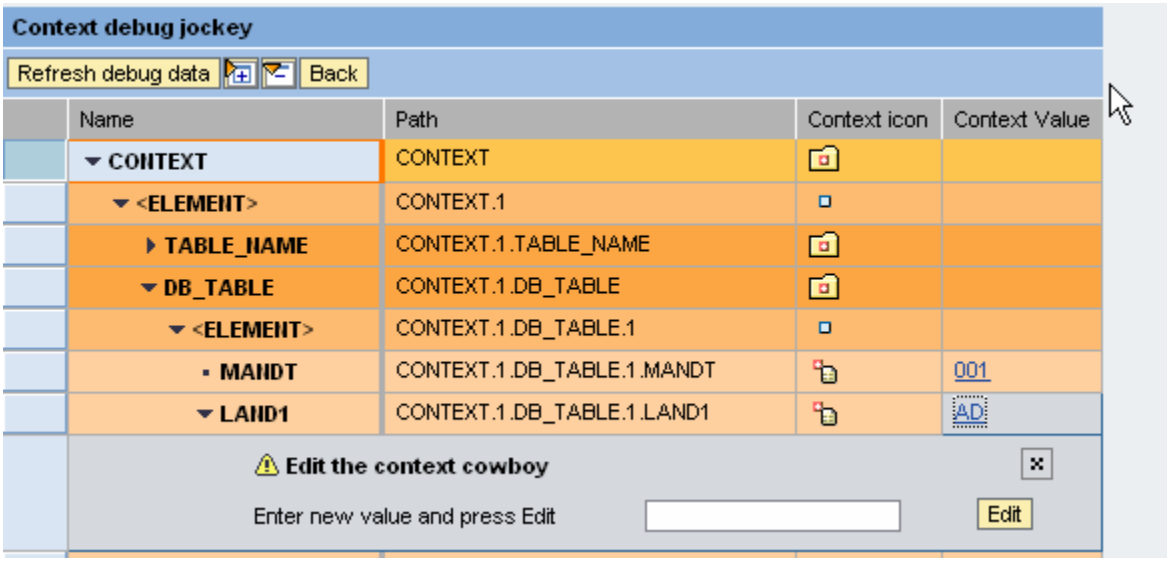
The following screen shot shows a result view with debug jockey when first loaded.

Debug Jockey is also a helpful for beginners to understand the Context.



Open the tree to browse the context.

You can click on attribute values to make a change.



The results are written back in source context when you press EDIT button.

I have supplied a transport with the development. If you can't load the transport, see the appendix which has screen dumps of code and views, etc., so you can attempt to create it by hand. Recreating it by hand is more for those with intermediate skills and not for beginners.

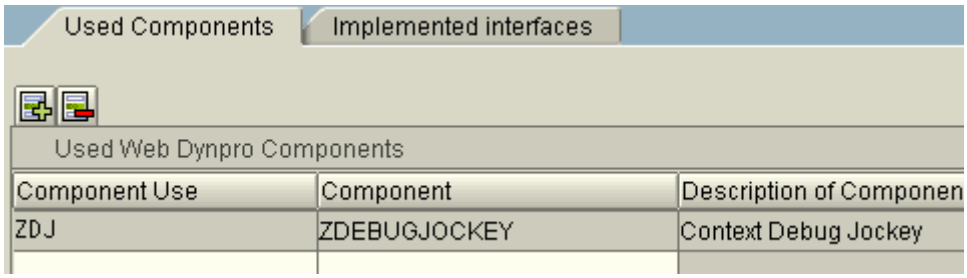
1. Load transport.

Transport was provided.

https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/com.sap.km.cm.docs/business_packages/a1-8-4/Context%20Debugger%20Tool%20for%20Web%20Dynpro%20ABAP.zip

2. Add Debug Jockey to your WD component.

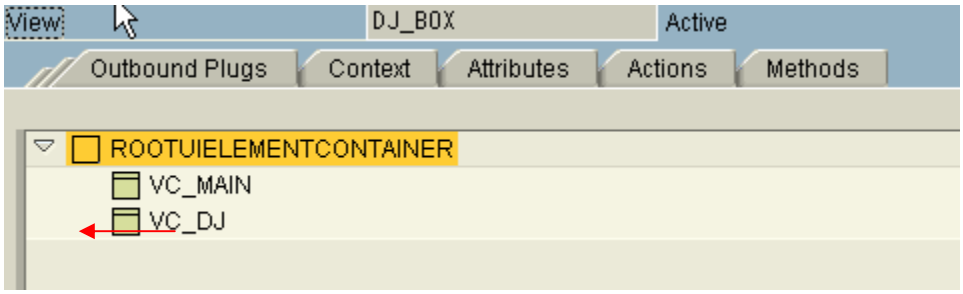
Declare used component on you WD component screen.



3. Declare a view container that will host Debug Jockey.

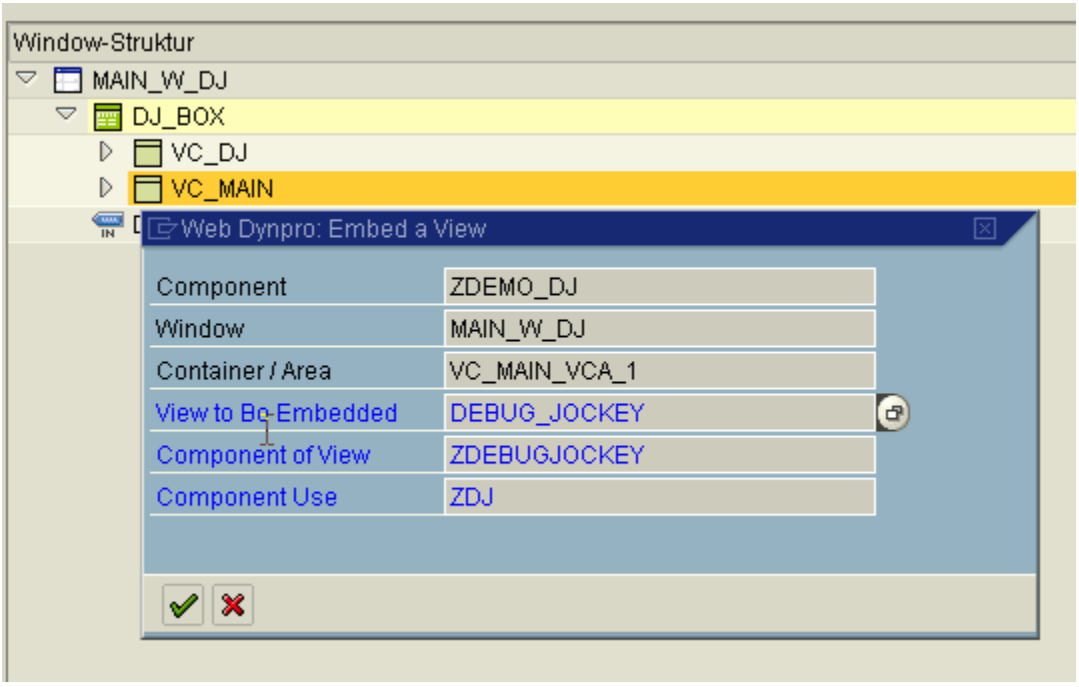
You can just add a view container to the end of an existing view

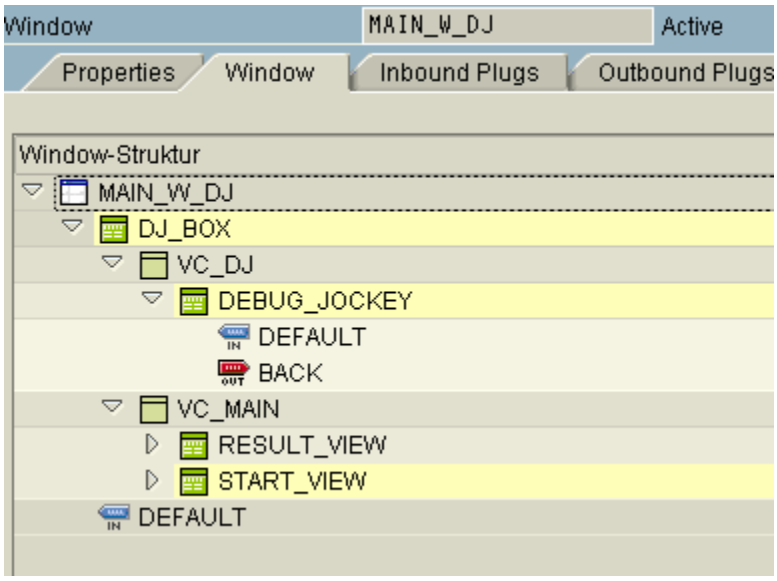
or declare new view that has 2 view containers. 1 to hold you normal view/s and the second to hold debug jockey.



4. Embed Debug jockey inside the View container for the relevant window/s

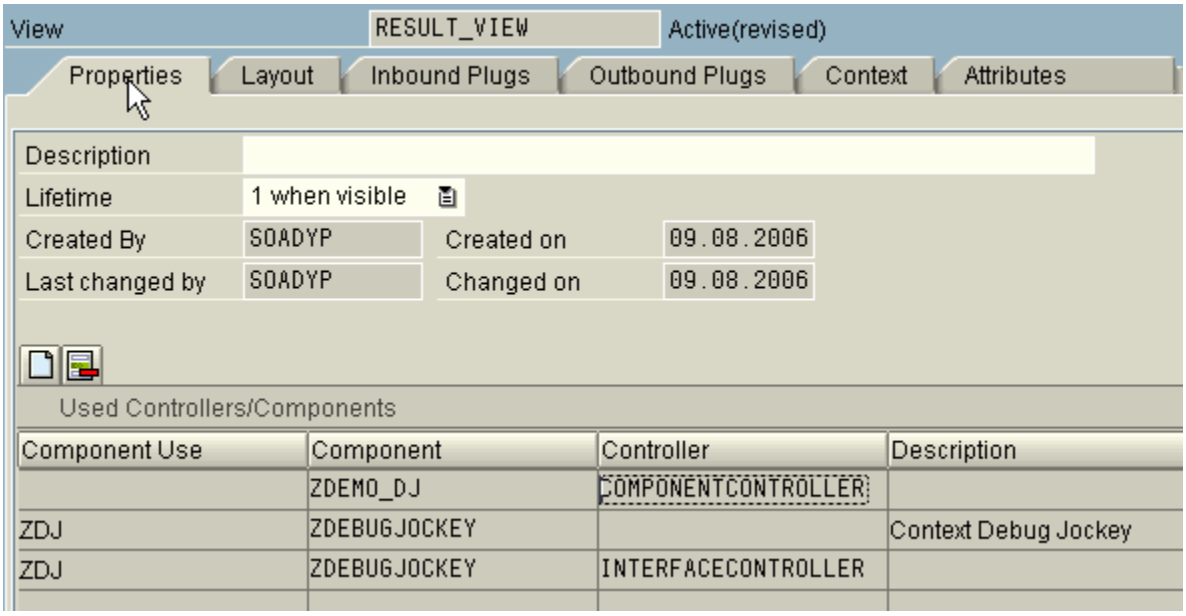
Here you see normal views embedded in VC_MAIN and the external component embedded in the second view container. Use right-click to embed external views.





4. Declare external component and external interface controller usage.

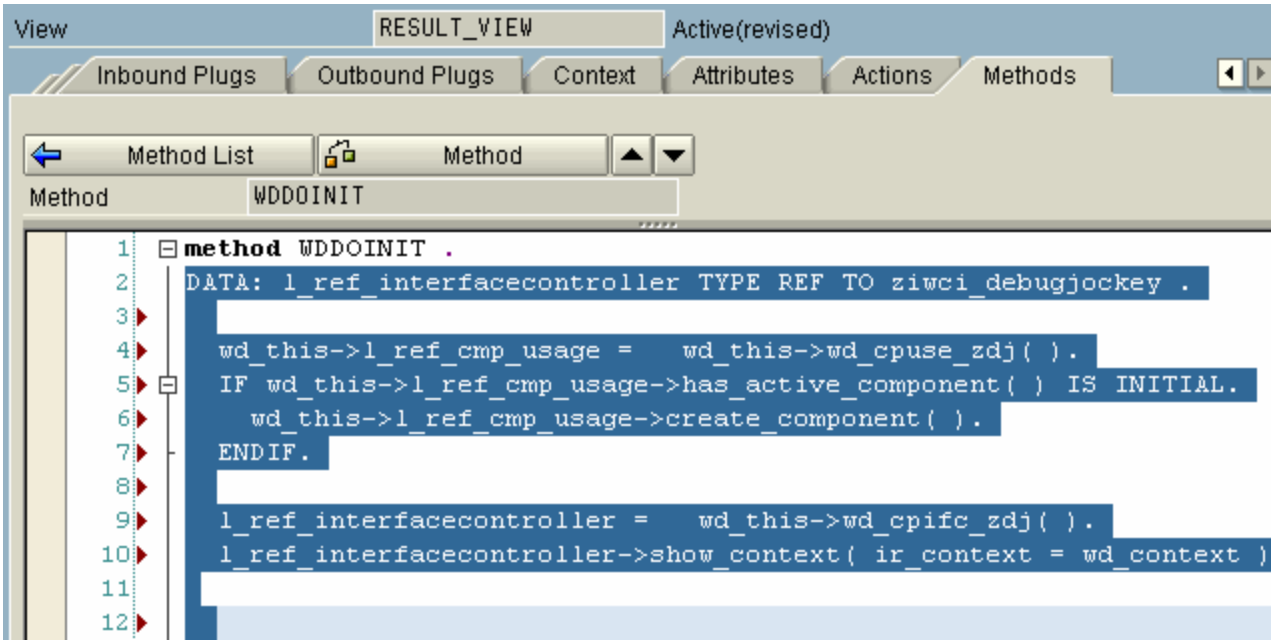
You declare debug jockey and its interface controller in any view controller you wish to debug.



5. Instantiate DJ component and interface controller in suitable method.

You can use WDDOINIT if you are debugging just one view. Otherwise and inbound plug handler may be a good place to put the following code if debugging multiple views.

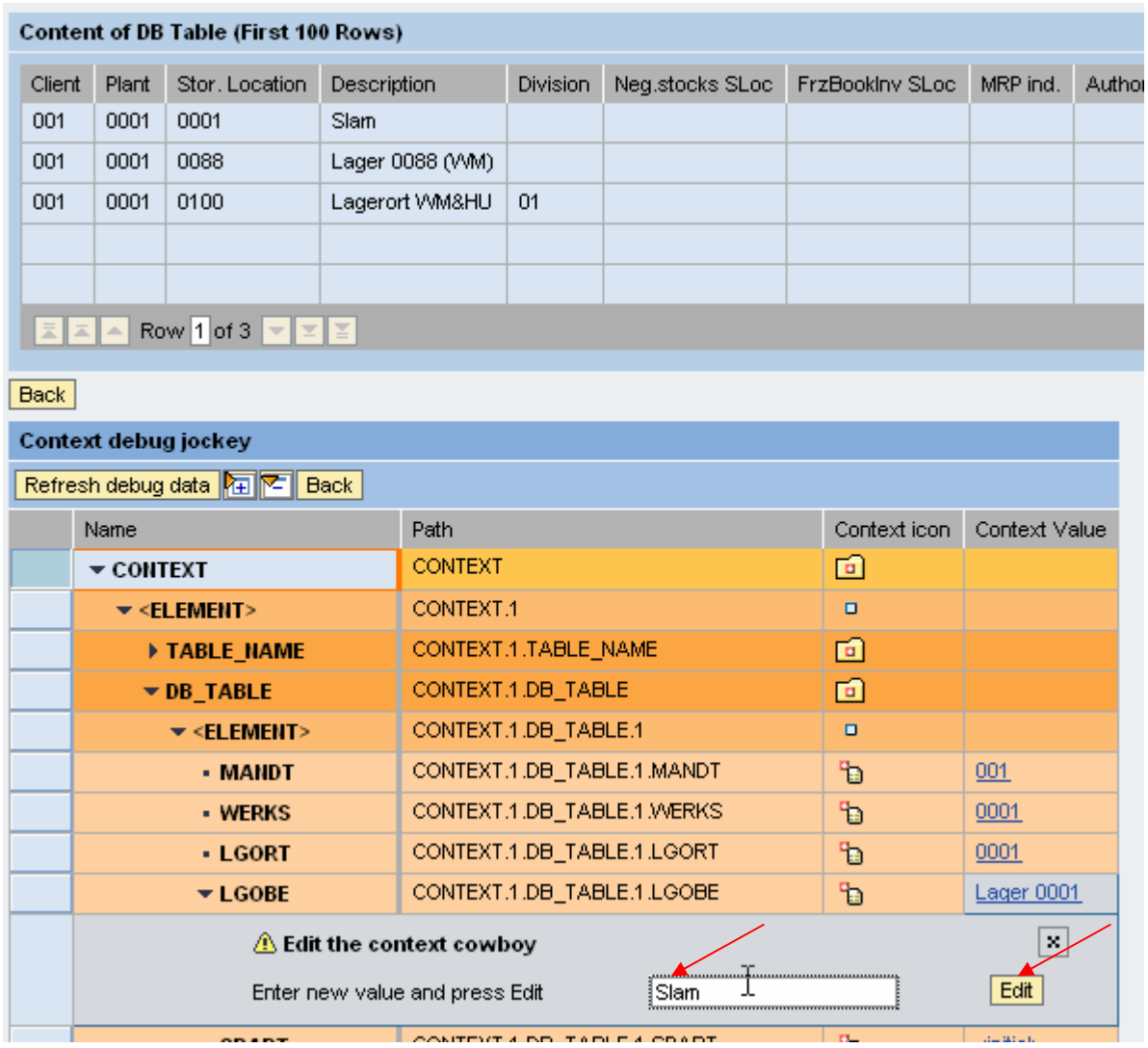
The code wizard can be used to generate code. You must have completed the usage declaration step described above prior to this step.



6. You are ready to go debugging.

When you wish to debug something, press the [Refresh debug data].

Here's an example.


Content of DB Table (First 100 Rows)
Client
Plant
Stor. Location
Description
Division
Neg.stocks SLoc
FrzBookInv SLoc
MRP ind.
001
0001
0001
Slam
001
0001
0088
Lager 0088 (WM)
001
0001
0100
Lagerort WM&HU
01
Row 1 of 3
Back
Context debug jockey
Refresh debug data
Back
Name
Path
Context icon
Context Value
CONTEXT
CONTEXT
<ELEMENT>
CONTEXT.1
TABLE_NAME
CONTEXT.1.TABLE_NAME
DB_TABLE
CONTEXT.1.DB_TABLE
<ELEMENT>
CONTEXT.1.DB_TABLE.1
MANDT
CONTEXT.1.DB_TABLE.1.MANDT
001
WERKS
CONTEXT.1.DB_TABLE.1.WERKS
0001
LGORT
CONTEXT.1.DB_TABLE.1.LGORT
0001
LGOBE
CONTEXT.1.DB_TABLE.1.LGOBE
Lager 0001
Edit the context cowboy
Enter new value and press Edit
Slam
Edit

Appendix

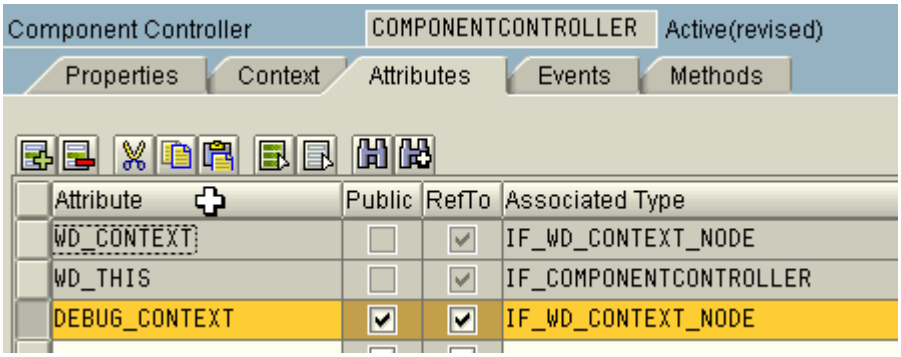
Manually create Debug Jockey without using the transport.

For whatever reason you don't wish to or can't import the transport into you system here is the code and screen shots.

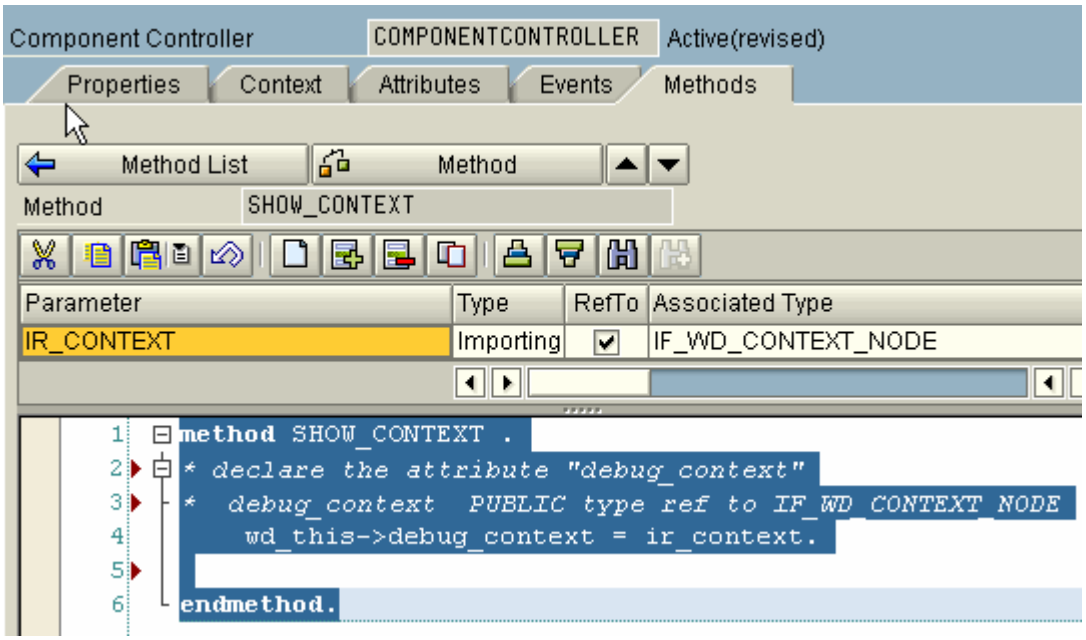
1. Create your ZDEBUGJOCKEY Web Dynpro component.

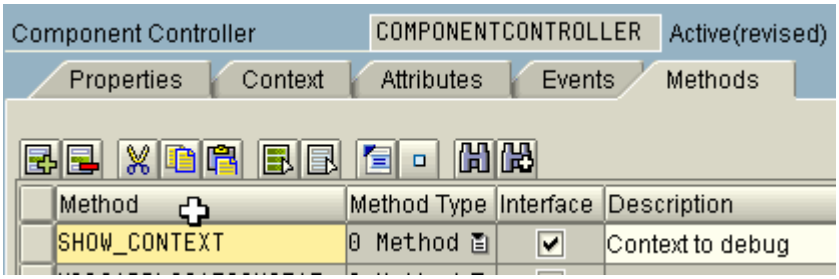
2. In the component controller:

2.1 Now define the attribute to hold the passed in context reference.



2.2 Define method **SHOW_CONTEXT** as shown below. Then mark the method as INTERFACE in the method List.

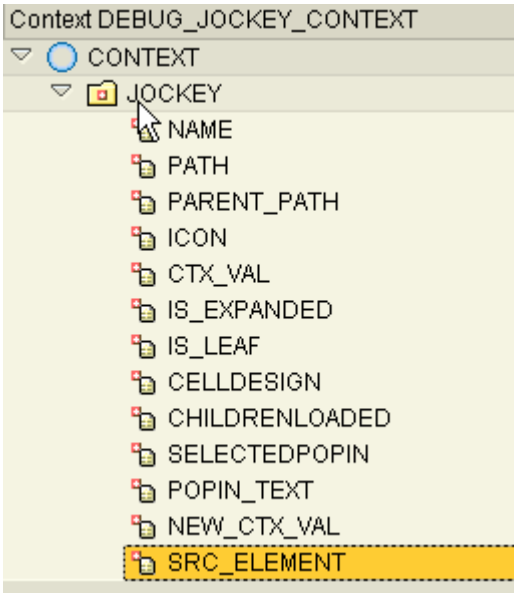




3. Now maintain the view created when you created you Web Dynpro component.

(DEBUG_JOCKEY_CONTEXT).

Declare the follow context:

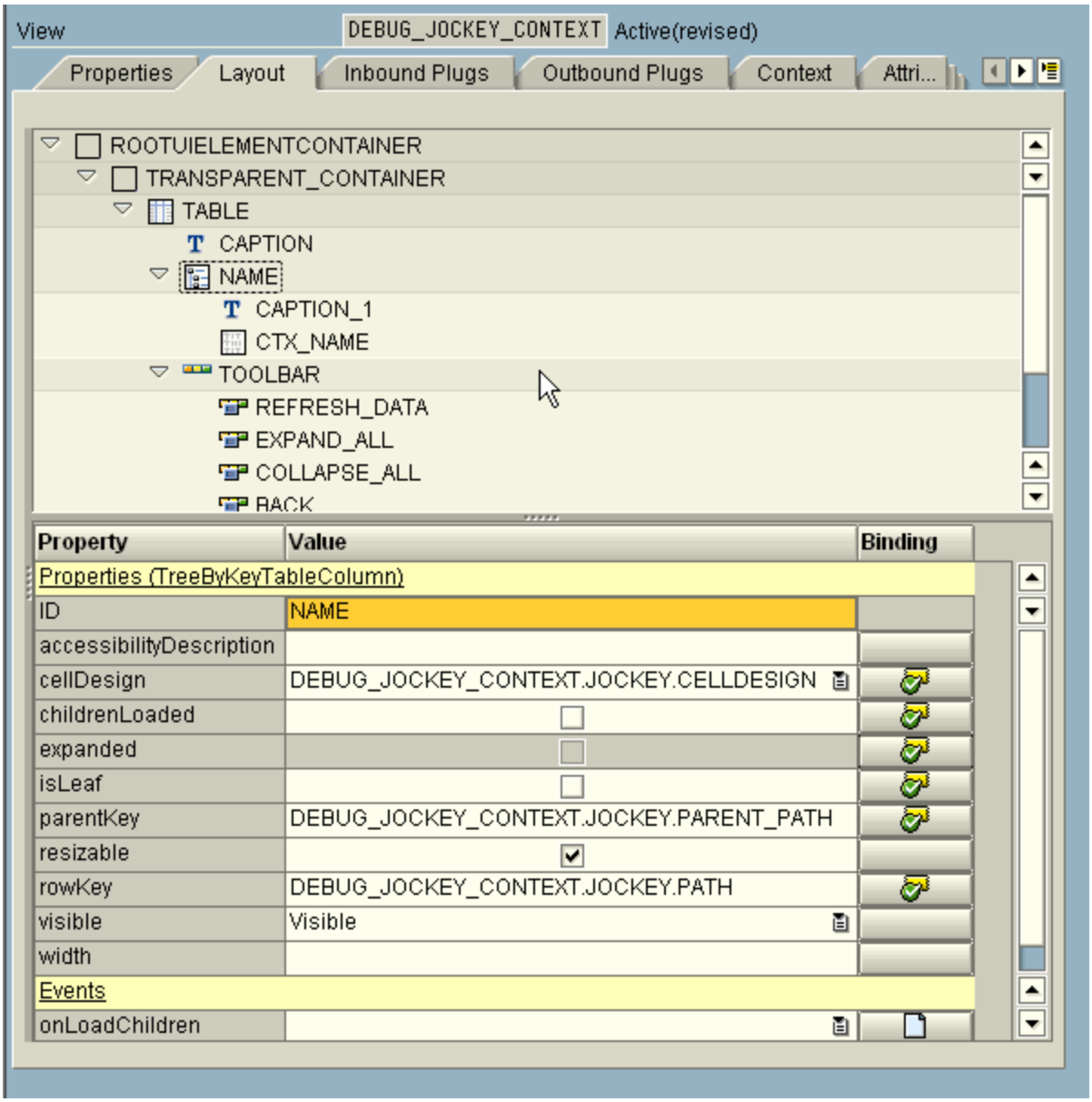
	JOCKEY is 0..n cardinality NAME, PATH, PARENT_PATH, ICON, CTX_VAL, CHILDRENLOADED, SELECTEDPOPIN NEW_CTX_VAL type string IS_EXPANED, IS_LEAF type WDY_BOOLEAN CELLDESIGN type WDUI_TABLE_CELL_DESIGN SRC_ELEMENT type IF_WD_CONTEXT_ELEMENT

4. Declare the UI Elements Part 1.

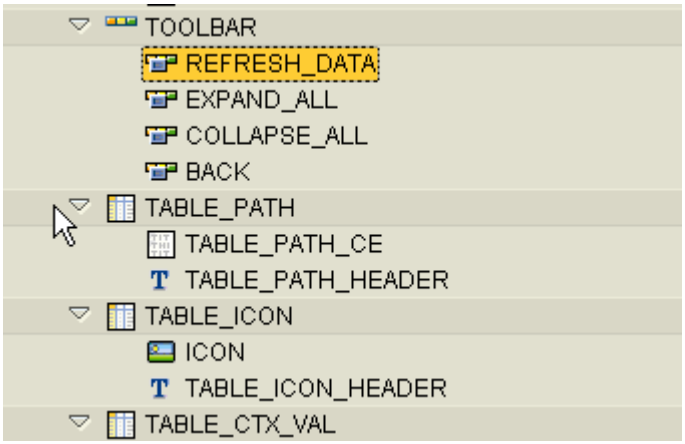
The important part here is this screen is getting the TreeByKeyTableColumn element right.

When adding the first column make sure you chose the right UI type then get ALL of the bindings set as below. There are 6 bindings on the column NAME.

The table has a toolbar with four ToolBarButtons. Go ahead and create the actions and leave the methods empty for now.



5. Now create 2 columns for the path and icon:

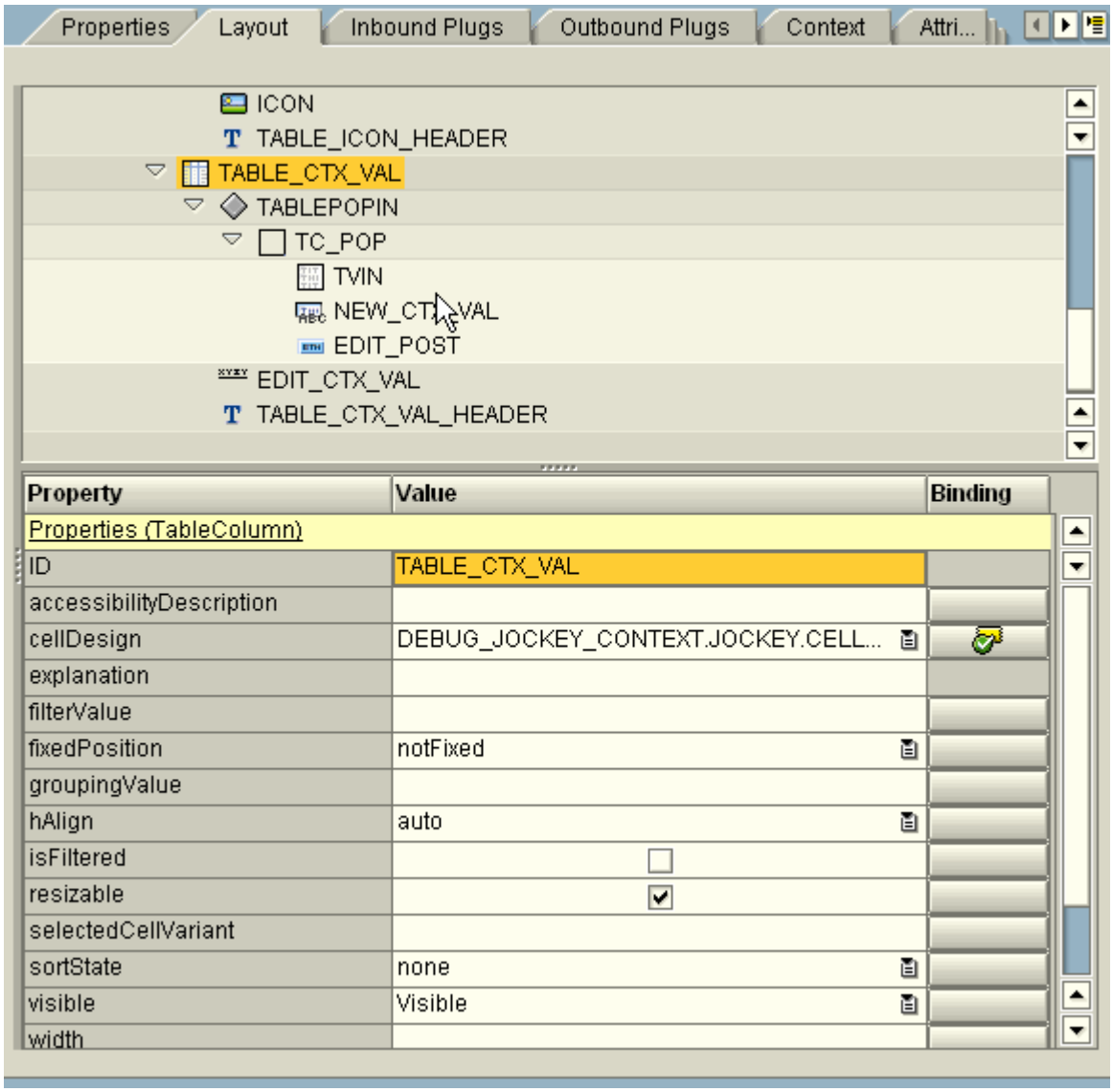


TABLE_PATCH_CE is bound to DEBUG_JOCKEY_CONTEXT.JOCKEY.PATH

ICON is bound to DEBUG_JOCKEY_CONTEXT.JOCKEY.ICON

6. The Context value column is now added.

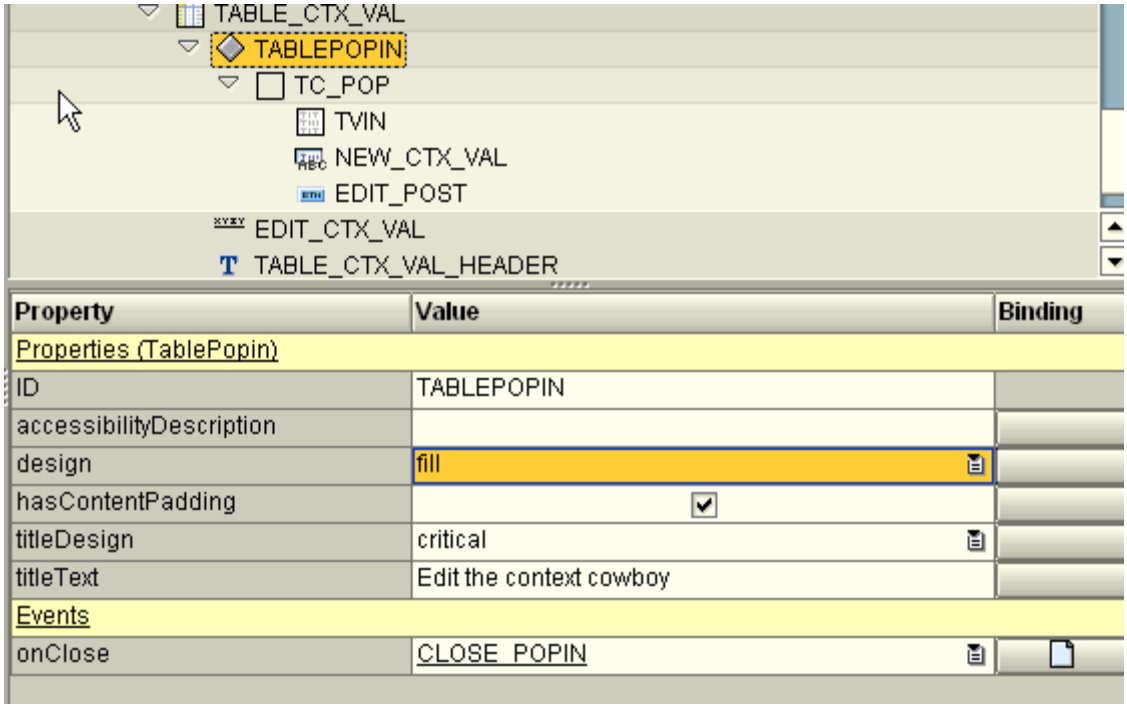
The are a few things to get right here.



TABLE_CTX_VAL celldesign is bound DEBUG_JOCKEY_CONTEXT.JOCKEY.CELLDESIGN.

7. The POPIN TablePopin property sheet is shown below.

Import is the close action CLOSE_POPIN.



8. Add a transparent container inside the popin.

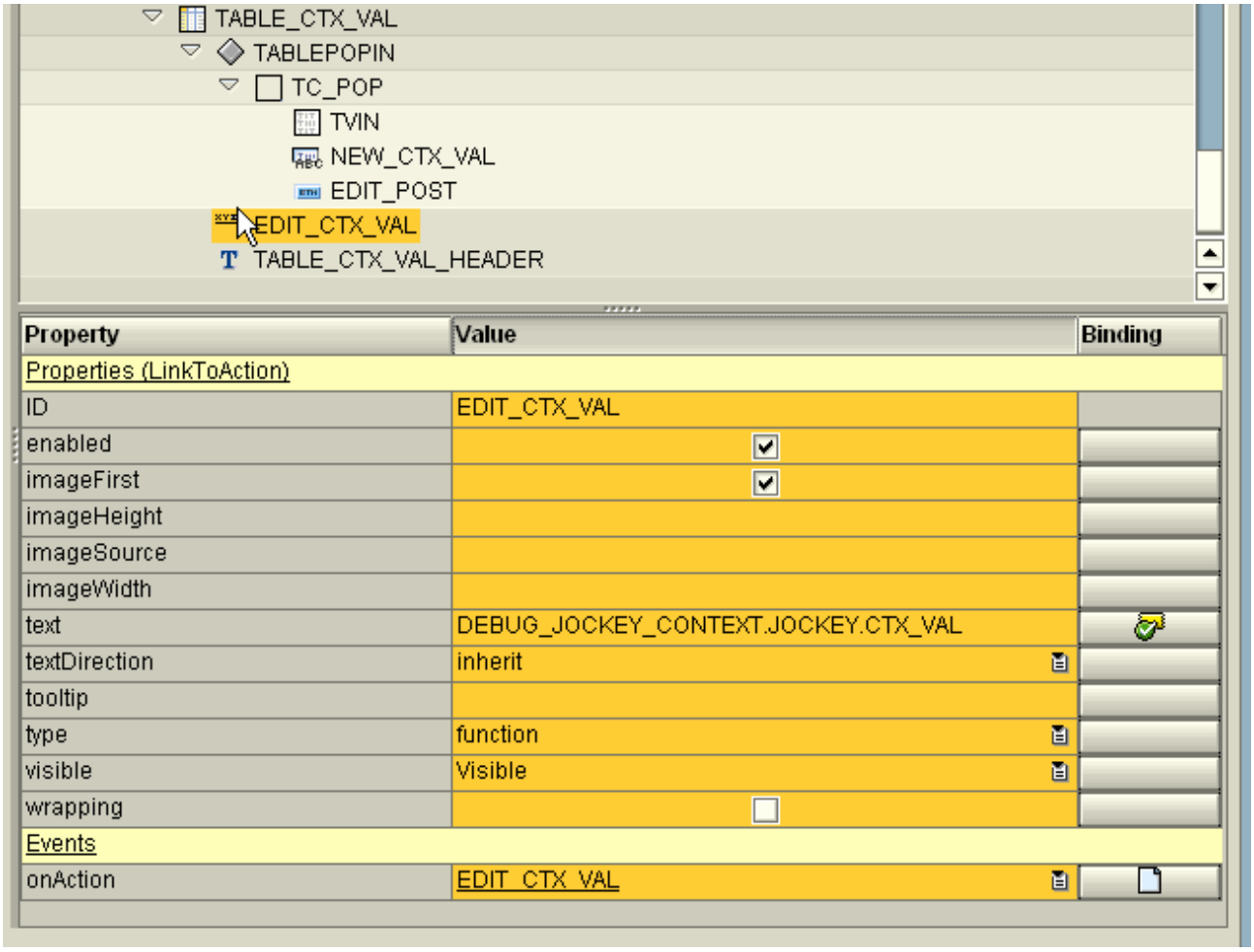
There are no screen shots for the next three UI elements. Just add as follows.

Add a textview and bind it to DEBUG_JOCKEY_CONTEXT.JOCKEY.POPIN_TEXT.

Add an input field NEW_CTX_VAL and Bind it to DEBUG_JOCKEY_CONTEXT.JOCKEY.NEW_CTX_VAL.

Add a Button (EDIT) with a simple action. Create the action and leave the method empty for now.

9. Finally the actually Context value is added to the table. It is added as a link.



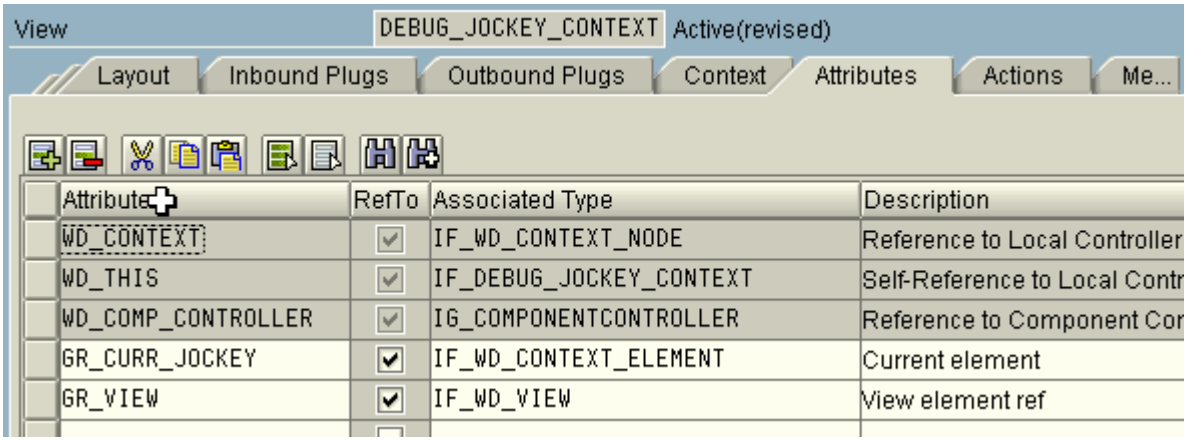
Add the Link to action EDIT_CTX_VAL. Bind the value to DEBUG_JOCKEY_CONTEXT.JOCKEY.CTX_VAL.

The Action EDIT_CTX_VAL showed be created and left empty.

10. New Attributes required on the view controller

GR_CURR_JOCKEY IF_WD_CONTEXT_ELEMENT

GR_VIEW IF_WD_VIEW



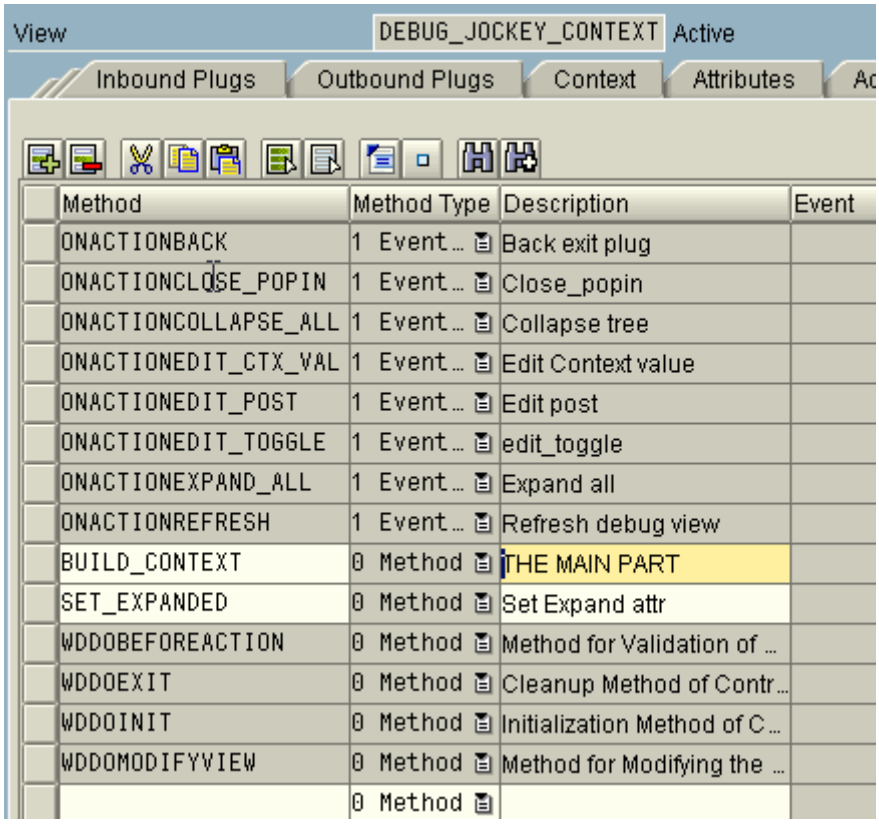
11. Actions TAB

You action tab should contain the following 8 actions.

BACK	Back exit plug
CLOSE_POPIN	Close_popin
COLLAPSE_ALL	Collapse tree
EDIT_CTX_VAL	Edit Context value
EDIT_POST	Edit post
EDIT_TOGGLE	edit_toggle
EXPAND_ALL	Expand all
REFRESH	Refresh debug view

12. Methods

Finally, the code part. The whole action starts when the button refresh data is pressed. This method Builds the local View CONTEXT Jockey using a recursive call to the main build method. The other methods are for a handling the POPIN to process the change value.



The code for the ALL methods is just listed below.

```
method ONACTIONREFRESH .
   DATA:
    node_jockey   TYPE REF TO if_wd_context_node.
*****************************************************************

  node_jockey =
wd_context->get_child_node( name = if_debug_jockey_context=>wdctx_jockey ).
  node_jockey->invalidate( ).


    wd_this->build_context(
      ir_node =     wd_comp_controller->debug_context
      iv_prev_path =  ''          ).


endmethod.
```

```
METHOD BUILD_CONTEXT Parameters
IR_NODE        Importing    REF   IF_WD_CONTEXT_NODE
IV_PREV_PATH   Importing          STRING
```

```
METHOD build_context .
  DATA:
     l_api_componentcontroller TYPE REF TO if_wd_component,
     lr_windows_manager TYPE REF TO if_wd_window_manager,
     lr_wdw type REF TO if_wd_window,
     lt_text TYPE string_table,
      node_jockey   TYPE REF TO if_wd_context_node,
      elem_jockey   TYPE REF TO if_wd_context_element,
      stru_jockey   TYPE if_debug_jockey_context=>element_jockey ,
      curr_elem_stru TYPE if_debug_jockey_context=>element_jockey ,
      lv_prev_path  TYPE string,
      lv_parent_node TYPE string,
      lv_parent_elem TYPE string,
      lv_path       TYPE string,
      lv_index      TYPE string,
      lv_lines      TYPE i,

      src_node_info TYPE REF TO if_wd_context_node_info,
      src_attr_set  TYPE wdr_context_attr_info_map,
      src_attr      TYPE wdr_context_attribute_info,
      src_elem_set  TYPE wdr_context_element_set,
      src_elem      TYPE REF TO if_wd_context_element ,
      src_child_node_set TYPE  wdr_context_child_map,
      src_child_node TYPE wdr_context_child.

*****************************************************************

  IF ir_node IS BOUND.
  ELSE.
    l_api_componentcontroller = wd_comp_controller->wd_get_api( ).
    lr_windows_manager =
    l_api_componentcontroller->get_window_manager( ).
```

```
    APPEND 'Context to debug is not bound.' TO lt_text.
    APPEND 'Use the SHOW_CONTEXT on interface controller' TO lt_text.
    APPEND 'to pass a reference to the Context ' TO lt_text.
    APPEND 'to debug to the debug component.' TO lt_text.
    lr_wdw =  lr_windows_manager->create_popup_to_confirm(
                    text           = lt_text
                    button_kind    = '1'      ).

     lr_wdw->open( ).

    exit.
  ENDIF.



  lv_prev_path = iv_prev_path.
* get the node we use to show data
  node_jockey = wd_context-
>get_child_node( name = if_debug_jockey_context=>wdctx_jockey ).


*Get the metadata for currently processed node
  src_node_info = ir_node->get_node_info( ).
  src_attr_set  = src_node_info->get_attributes( ).

* BANG the current node in the tree
  CLEAR stru_jockey.
  stru_jockey-name = src_node_info->get_name( ).

  IF lv_prev_path = space.
    stru_jockey-path =  stru_jockey-name.
  ELSE.
    CONCATENATE lv_prev_path '.' stru_jockey-name
       INTO stru_jockey-path.
  ENDIF.


  stru_jockey-parent_path = lv_prev_path.
  stru_jockey-icon = 'ICON_WD_VALUE_NODE'.

*  stru_jockey-ctx_val = '<NODE>'.
  stru_jockey-celldesign = '06'.

  src_elem_set = ir_node->get_elements(  ).
  DESCRIBE TABLE src_elem_set LINES lv_lines.
  IF lv_lines > 0.
    stru_jockey-is_leaf = abap_false.
  ELSE.
    stru_jockey-is_leaf = abap_true.
  ENDIF.

  node_jockey->bind_structure(  new_item            = stru_jockey
                                set_initial_elements = abap_false ).

* record the parent key of node
  lv_parent_node = stru_jockey-path.
```

```
  LOOP AT src_elem_set INTO src_elem.
    CLEAR stru_jockey.

    lv_index = sy-tabix.
    stru_jockey-name = '<ELEMENT>'.
    CONCATENATE lv_parent_node '.' lv_index INTO  lv_path.
    CONDENSE lv_path NO-GAPS.


    stru_jockey-path = lv_path.
    stru_jockey-parent_path = lv_parent_node.
    stru_jockey-icon = 'ICON_ELEMENT'.
    stru_jockey-celldesign = '07'.
*    stru_jockey-ctx_val = lv_index.
    DESCRIBE TABLE src_attr_set LINES lv_lines.
    IF lv_lines = 0.
      stru_jockey-is_leaf = abap_false.
    ELSE.
      stru_jockey-is_leaf = abap_false.
    ENDIF.
    node_jockey->bind_structure(  new_item            = stru_jockey
                                set_initial_elements = abap_false ).

    curr_elem_stru = stru_jockey.

    lv_parent_elem = lv_path.

* add the attributes
    LOOP AT  src_attr_set INTO src_attr.
      lv_index = sy-tabix.
      stru_jockey-parent_path = curr_elem_stru-path.
      CONCATENATE lv_parent_elem '.' src_attr-name
                  INTO stru_jockey-path .

      stru_jockey-name = src_attr-name.
      stru_jockey-is_leaf = abap_true.
      stru_jockey-icon = 'ICON_WD_VALUE_ATTR'.
      stru_jockey-celldesign = '08'.
      stru_jockey-src_element = src_elem.

      src_elem->get_attribute(
        EXPORTING  name   = src_attr-name
         IMPORTING value  =   stru_jockey-ctx_val          ).

     IF stru_jockey-ctx_val is INITIAL.
        stru_jockey-ctx_val = '<initial>'.
     ENDIF.


      node_jockey->bind_structure(  new_item            = stru_jockey
                                  set_initial_elements = abap_false ).


    ENDLOOP.

    src_child_node_set = src_elem->get_child_nodes( ).

    LOOP AT src_child_node_set INTO src_child_node.
      wd_this->build_context(
```

```
        ir_node      =   src_child_node-node
        iv_prev_path =   lv_parent_elem   ).

    ENDLOOP.


  ENDLOOP.


ENDMETHOD.


method ONACTIONBACK .

  DATA:
    lr_VIEW_api TYPE REF TO if_wd_view_controller,
    lr_wdw_ctrl TYPE REF TO if_wd_window_controller.



  lr_view_api = wd_this->wd_get_api( ).
  lr_wdw_ctrl = lr_view_api->get_embedding_window_ctlr( ).

  lr_wdw_ctrl->fire_plug( plug_name  = 'BACK' ).


endmethod.


Method ONACTIONCLOSE_POPIN .
  context_element->set_attribute( name = 'SELECTEDPOPIN' value = '' ).
endmethod.


method ONACTIONCOLLAPSE_ALL .
  wd_this->set_expanded( is_expanded = abap_false ).
endmethod.


method ONACTIONEDIT_CTX_VAL parameters

WDEVENT              CL_WD_CUSTOM_EVENT

CONTEXT_ELEMENT      ref   IF_WD_CONTEXT_ELEMENT

ID                   STRING.

method ONACTIONEDIT_CTX_VAL .
  data wd_table_cell_editor type ref to cl_Wd_view_element.
  data wd_table_column      type ref to cl_wd_table_column.
  data wd_popin             type ref to cl_wd_table_popin.
  data lv_str TYPE string.

  wd_table_cell_editor ?= wd_this->gr_view->get_element( id ).
  wd_table_column ?= wd_table_cell_editor->get__parent( ).
  wd_popin = wd_table_column->get_popin( ).

*Tell context which POPIN IS UP...
   context_element->set_attribute( name = 'SELECTEDPOPIN'
```

```
          value = wd_popin->id ).

    context_element->set_attribute( name = 'IS_EXPANDED'
          value = abap_true ).

    context_element->get_attribute( EXPORTING name  = 'ICON'
                                    IMPORTING value = lv_str ).

    if lv_str <> 'ICON_WD_VALUE_ATTR'.
        lv_str = 'Edit supported on Attributes only'.
    else.
        lv_str = 'Enter new value and press Edit'.
    endif.

    context_element->set_attribute( name = 'POPIN_TEXT'
          value = lv_str ).

    wd_this->gr_curr_jockey = context_element.

endmethod.
```

```
method ONACTIONEDIT_POST .
  data: stru_jockey TYPE If_Debug_Jockey_Context=>element_jockey.

  wd_this->gr_curr_jockey->get_static_attributes(
    IMPORTING
       STATIC_ATTRIBUTES = stru_jockey    ).



  stru_jockey-src_element->set_attribute(
       VALUE  = stru_jockey-new_ctx_val
       name   = stru_jockey-name
         ).

endmethod.

method ONACTIONEDIT_TOGGLE .
endmethod.

method ONACTIONEXPAND_ALL .
 wd_this->set_expanded( is_expanded = abap_true  ).
endmethod.
```

```
Parameters for method SET_EXPANDED

IS_EXPANDED    Importing   0     WDY_BOOLEAN

method SET_EXPANDED .

      DATA:

      node_jockey    TYPE REF TO if_wd_context_node,
      elem_jockey    TYPE REF TO if_wd_context_element,
      stru_jockey    TYPE if_debug_jockey_context=>element_jockey ,
      lt_jockey      TYPE if_debug_jockey_context=>elements_jockey .

Field-SYMBOLS : <j> type  if_debug_jockey_context=>element_jockey .
*   navigate from <CONTEXT> to <JOCKEY> via lead selection
    node_jockey = wd_context-
>get_child_node( name = if_debug_jockey_context=>wdctx_jockey ).

    node_jockey->get_static_attributes_table(
        IMPORTING TABLE  = lt_jockey   ).

    LOOP AT lt_jockey ASSIGNING <j> .
      <j>-is_expanded = is_expanded.
    ENDLOOP.


node_jockey->bind_elements( new_items  = lt_jockey ).


endmethod.



method WDDOMODIFYVIEW .

  if first_time = abap_true.
    wd_this->gr_view = view.
  endif.


endmethod.
```


Related Content

1. Context selection versus lead selection from Thomas Szuecs
2. Thomas Jung intro videos
3. HELP on CONTEXT APIs

Copyright