WebDynpro ABAP in CRM 2007: Dynamic Context Nodes and Component Usages

Applies to:

SAP CRM ABAP 6.0

For more information, visit the User Interface Technology homepage.

Summary

A quick walkthrough using a simple example on how to dynamically create component usages, context nodes and embed them in a view.

Author: Arun Prakash Karuppanan

Company: Accenture

Created on: 19 January 2010

Author Bio

Arun Prakash Karuppanan is an application developer in SAP-CRM. He is currently employed with Accenture Services Private Ltd.

Table of Contents

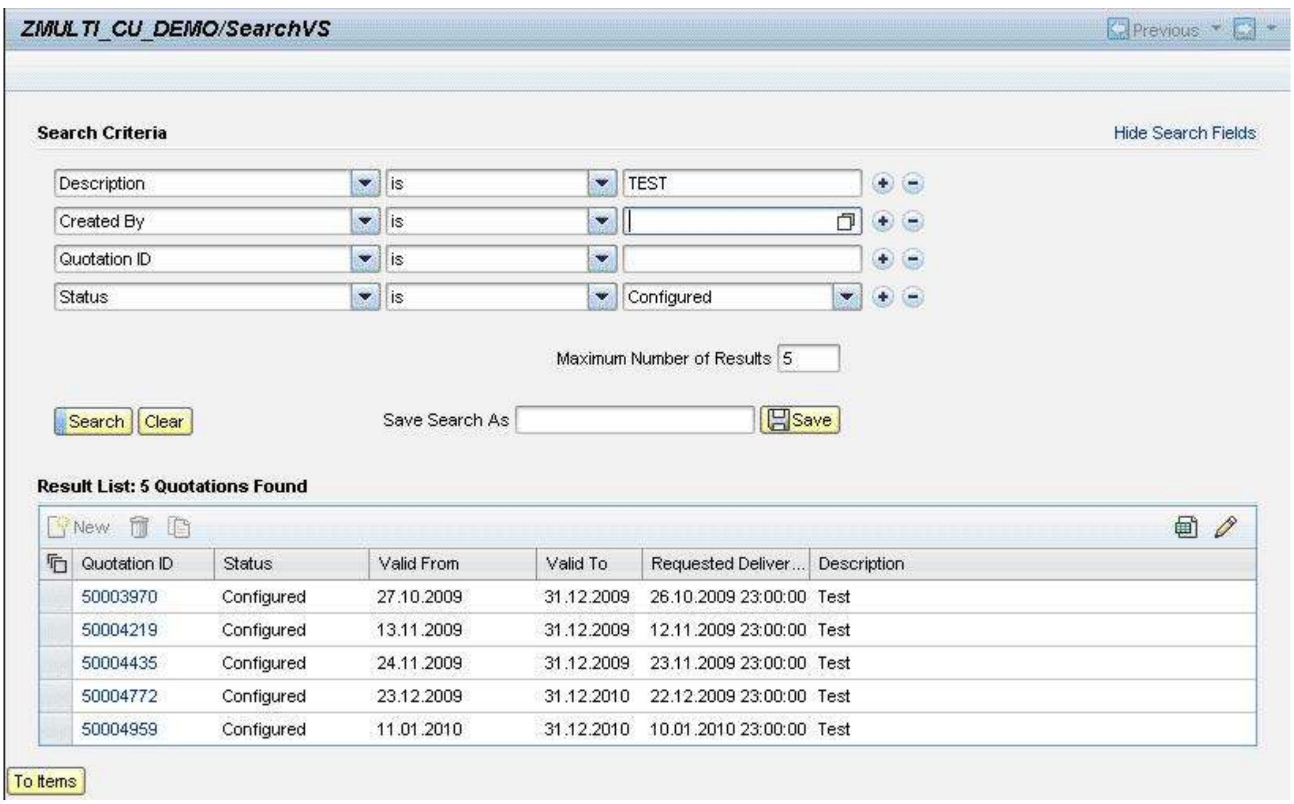
Introduction

The reusability of BSP components in the ABAP WebDynpro programming environment is a very useful feature. Most of our programming needs are satisfied by statically declaring the reused component as per number of usages required. But sometimes, the case may be that we will know the component and the number of usages required only during runtime. If the number of usages is too large or if there are too many components in question, it will be very cumbersome to statically declare all those. What we can do instead is, dynamically create the component usages, context nodes and embed them in the view container, all at runtime. Though a little bit complex, it is not as hard as you might think and the payoff is very good compared to the messy programming using static declarations. This example is based on CRM 6.0(CRM 2007). Examples for earlier versions are available online in SAP Help.

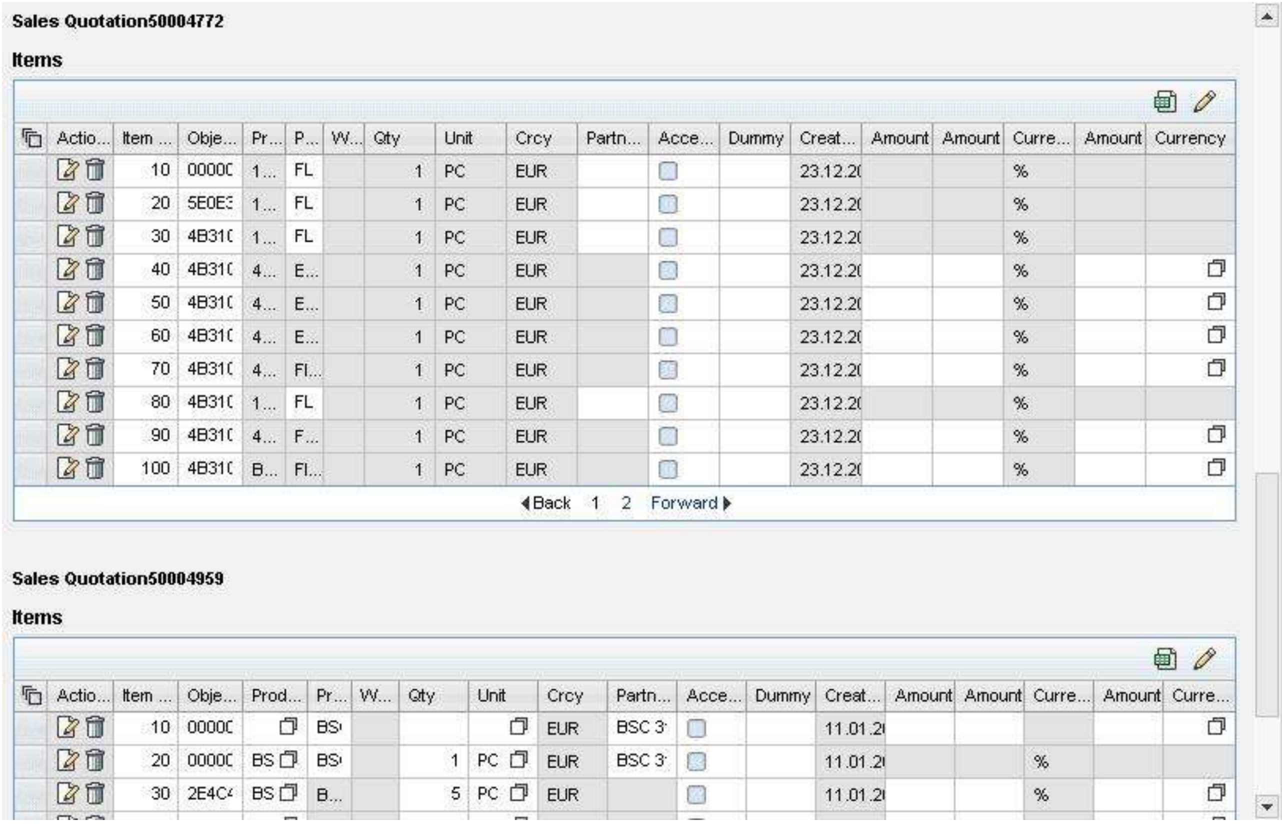
Demo Scenario

The concepts can be better explained by devising a requirement and programming for it. The reader is expected to be fairly familiar with programming in the ABAP WebDynpro environment. Concepts like component interfaces, component usage, context node binding, etc., should be known.

Suppose that we are required to develop a BSP component with two pages. The first page will contain a search view for sales quotations. This page will also contain a button, on clicking which, the user is taken to the next page. The second page will show items of all the sales quotations in the search result of the previous page. Please look at the pictures below to get an idea. The scenario might not be very imaginative, but will suffice for demonstration purposes.



Note: The first page uses the standard component 'BT115QS_SLSQ' for providing the search interface.



Sales Quotation50004772

Items

Actio...	Item ...	Obje...	Pr...	P...	W...	Qty	Unit	Crcy	Partn...	Acce...	Dummy	Creat...	Amount	Amount	Curre...	Amount	Currency
	10	0000C	1...	FL		1	PC	EUR				23.12.2(			%		
	20	5E0E3	1...	FL		1	PC	EUR				23.12.2(			%		
	30	4B31(	1...	FL		1	PC	EUR				23.12.2(			%		
	40	4B31(	4...	E...		1	PC	EUR				23.12.2(			%		
	50	4B31(	4...	E...		1	PC	EUR				23.12.2(			%		
	60	4B31(	4...	E...		1	PC	EUR				23.12.2(			%		
	70	4B31(	4...	Fl...		1	PC	EUR				23.12.2(			%		
	80	4B31(	1...	FL		1	PC	EUR				23.12.2(			%		
	90	4B31(	4...	F...		1	PC	EUR				23.12.2(			%		
	100	4B31(	B...	Fl...		1	PC	EUR				23.12.2(			%		

Back 1 2 Forward

Sales Quotation50004959

Items

Actio...	Item ...	Obje...	Prod...	Pr...	W...	Qty	Unit	Crcy	Partn...	Acce...	Dummy	Creat...	Amount	Amount	Curre...	Amount	Curre...
	10	0000C		BS(				EUR	BSC 3			11.01.2(					
	20	0000C	BS	BS(		1	PC	EUR	BSC 3			11.01.2(			%		
	30	2E4C4	BS	B...		5	PC	EUR				11.01.2(			%		

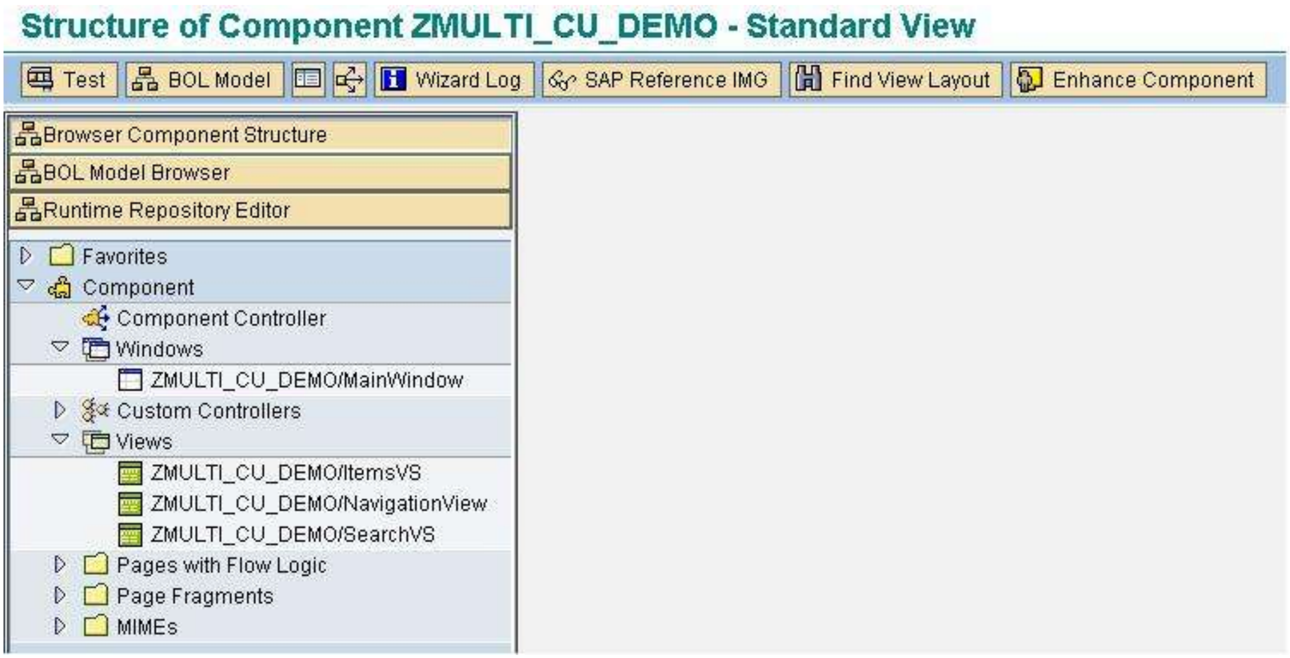
Note: The second page uses the standard component ‘BT115QIT_SLSQ’ for showing the items of all quotations. There are as many tables as the number of search results. The screenshot shows only a limited view.

In the demo system, the component ‘BT115QIT_SLSQ’ has suitably been enhanced such that it can be independently reused with any component and will show all the items put in the collection wrapper of the context node, ‘BTADMINMULTSEL’. If you try out this example, you might encounter some problems with this component. In that case, please understand the concepts used and use some other scenario.

Building the Component

Layout of the Component

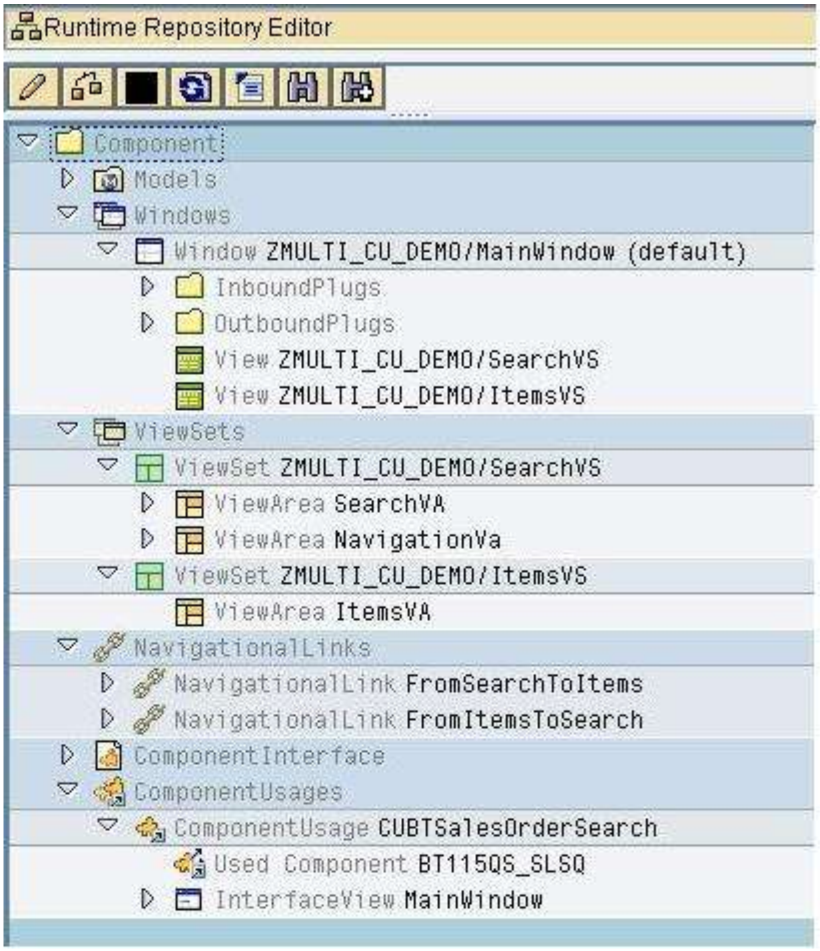
Since the reader is familiar with using the BSP workbench, I will skip the steps for the initial setup. I have created a component 'ZMULTI_CU_DEMO'. The component has only one Window. Take a look at the screenshots below and the attached notes. The screen caps show the structure of the component and the runtime repository.



Note: The first page is 'SearchVS'. This is a viewset that has two view areas. The first view area will load the component 'BT115QS_SLSQ', for the search interface. The second view area will contain the NavigationView that you see in the screenshot below. This page NavigationView will contain only a button for navigating to the next view.

The other view that you see is the ItemsVS. You may create this page as a viewset or a normal page. All the view areas in this page will be generated dynamically. Each view area will host all the dynamically created component usages of component 'BT115QIT_SLSQ'.

Runtime Repository View



Note: Notice that only one component has been declared under component usage, the search component and nothing has been assigned to the view area of the ItemsVS page.

Initial Context Node Bindings

The component controller of our demo component will have two context nodes. One is ‘Result’, which is bound to ‘BTQRSLSQUOT’ node of the search component ‘BT115QS_SLSQ’. Search result is accessible to our component via this node. The other node is ‘BTADMINI’ which will only act as a blueprint node for generating the dynamic context nodes.

The method ‘WD_USAGE_INITIALIZE’ has only the coding for binding with the search component.

```
CASE iv_usage->usage_name.

  WHEN 'CUBTSalesOrderSearch'.
    CALL METHOD iv_usage->bind_context_node
      EXPORTING
        iv_controller_type  = cl_bsp_wd_controller=>co_type_component
        iv_node_2_bind      = 'BTQRSLSQUOT'
        iv_target_node_name = 'RESULT'.

ENDCASE.
```

Generating Component Usages & Context Nodes Dynamically

The first page has a button for navigating to the second page. When the user clicks this button, we will generate as many component usages and context nodes as required. For each search result, one component usage and one context node will be generated. The component usage will be for the component '**BT115QIT_SLSQ**'. The reference to the dynamically created context node will be stored in a table type attribute in the component controller and will be linked to a specific component usage using the same table. It will contain the items of the order in search result. This dynamic context node will be bound to the '**BTADMINIMULTSEL**' node of the generated component usage**(BT115QIT_SLSQ)**. Thus, when this usage is inserted into a view container(a view area in a view set), we will get the desired result.

- Create a table type attribute in your component controller to store the dynamic instances. This way, we can get hold of them if we wish to modify content or destroy them.

```
private section.
  types:
    BEGIN OF ltype_cu_instance,
         cu_name TYPE string,              “name of component usage
         cnode_name TYPE string,           “name of context node
         model TYPE REF TO cl_bsp_model,   “instance of context node
         bo type ref to cl_crm_bol_entity, “instance of order entity(BTAdminH)
         controller TYPE REF TO cl_bsp_wd_controller, “instance of view controller
         END OF ltype_cu_instance .
  types:
    l_cu_list TYPE STANDARD TABLE OF ltype_cu_instance .

  data GT_CU_INSTANCES type L_CU_LIST .
```

- Create a method create_component_usage in the component controller class. We will be calling this method to create the dynamic component usages.

```
METHOD create_component_usage.
  CALL METHOD me->repository->create_cmp_usage_def
    EXPORTING
      iv_usage_name           = iv_usage_name
      iv_reference_usage_name = iv_reference_usage_name
      iv_component_name       = iv_component_name
    EXCEPTIONS
      duplicate_entry         = 1
      OTHERS                  = 2.
  IF sy-subrc <> 0.
  ENDIF.
ENDMETHOD.
```

- Create a method *register_cu_instance* in the component controller class. We will be calling this method to store the references to the dynamically created usages. It has only import parameters.

```
IV_CU_NAME      Type   STRING
IV_CNODE_NAME   Type   STRING
IR_MODEL        Type Ref To   CL_BSP_MODEL
IR_BO           Type Ref To   CL_CRM_BOL_ENTITY
IR_CONTROLLER   Type Ref To   CL_BSP_WD_CONTROLLER
```

```
METHOD register_cu_instance.

  FIELD-SYMBOLS <fs_cu_inst> TYPE ltype_cu_instance.
  IF iv_cu_name IS NOT INITIAL.
    READ TABLE gt_cu_instances WITH KEY cu_name = iv_cu_name ASSIGNING <fs_cu_inst>.
    IF sy-subrc EQ 0.
*Context node name
      IF iv_cnode_name IS NOT INITIAL.
        <fs_cu_inst>-cnode_name = iv_cnode_name.
      ENDIF.
*Context node instance
      IF ir_model IS BOUND.
        <fs_cu_inst>-model = ir_model.
      ENDIF.
*Order header entity
      IF ir_bo IS BOUND.
        <fs_cu_inst>-bo = ir_bo.
      ENDIF.
*View controller hosting this component usage
      IF ir_controller IS BOUND.
        <fs_cu_inst>-controller = ir_controller.
      ENDIF.
    ELSE.
      APPEND INITIAL LINE TO gt_cu_instances ASSIGNING <fs_cu_inst>.
      IF iv_cu_name IS NOT INITIAL.
        <fs_cu_inst>-cu_name = iv_cu_name.
      ENDIF.
      IF iv_cnode_name IS NOT INITIAL.
        <fs_cu_inst>-cnode_name = iv_cnode_name.
      ENDIF.
      IF ir_model IS BOUND.
        <fs_cu_inst>-model = ir_model.
      ENDIF.
      IF ir_bo IS BOUND.
        <fs_cu_inst>-bo = ir_bo.
      ENDIF.
      IF ir_controller IS BOUND.
        <fs_cu_inst>-controller = ir_controller.
      ENDIF.
    ENDIF.
  ENDIF.
ENDMETHOD.
```

- Create a method *get_component_usages* in the component controller class. We will be calling this method to obtain the names of the dynamically created component usages as a stringtab. This has only returning parameter.

```
METHOD get_component_usages.
  DATA ls_cu_instance LIKE LINE OF gt_cu_instances.
  LOOP AT gt_cu_instances INTO ls_cu_instance.
    APPEND ls_cu_instance-cu_name TO et_component_usages.
  ENDLOOP.
ENDMETHOD.
```

- Create a method *get_cnode_to_bind* in the component controller class. We will be calling this method to obtain the name of the dynamically created context node that needs to be bound to the ***BTADMINMULTSEL*** node of the component usage instance. This information will be available in the table type attribute *gt_cu_instances*, that we created earlier. We pass the component usage name as import parameter and get the context node name as returning parameter. We will see later how this context node name is linked to this usage.

```
METHOD get_cnode_to_bind.

  DATA: ls_cu_instance LIKE LINE OF gt_cu_instances,
        ls_model LIKE LINE OF m_models,
        lr_cnode TYPE REF TO cl_bsp_wd_context_node,
        lr_ent TYPE REF TO cl_crm_bol_entity,
        lr_col TYPE REF TO if_bol_bo_col,
        lv_cnode_name TYPE string.

  READ TABLE gt_cu_instances WITH KEY cu_name = iv_cu_name INTO ls_cu_instance.
  IF sy-subrc EQ 0.
    ev_cnode_name = ls_cu_instance-cnode_name.
  ENDIF.
  lv_cnode_name = ev_cnode_name.
  TRANSLATE lv_cnode_name TO LOWER CASE.

*Put the relevant item collection in the collection wrapper of this context node
  READ TABLE m_models INTO ls_model WITH KEY model_id = lv_cnode_name.
  IF sy-subrc EQ 0.

    lr_ent = ls_cu_instance-bo.
    lr_ent = lr_ent->get_related_entity( iv_relation_name = 'BTOrderHeader' ).
    lr_ent = lr_ent->get_related_entity( iv_relation_name = 'BTHeaderItemsExt' ).
    lr_col = lr_ent->get_related_entities( iv_relation_name = 'BTOrderItemAll' ).
    lr_cnode ?= ls_model-instance.
    lr_cnode->collection_wrapper->set_collection( lr_col ).

  ENDIF.

ENDMETHOD.
```

- Create a method *bind_context_node2* in the component controller class. We will be calling this method to bind a dynamically created context node to the **BTADMINMULTSEL** node of the component usage instance of **BT115QIT_SLSQ**.

```
IV_CONTROLLER_TYPE     Importing     Type   CHAR1
IV_NAME                Importing     Type   STRING
IV_TARGET_NODE_NAME    Importing     Type   SEOCMPNAME
IV_NODE_2_BIND         Importing     Type   SEOCMPNAME
IR_USAGE               Importing     Type Ref To   CL_BSP_WD_COMPONENT_USAGE

METHOD bind_context_nodes2.

  DATA: lv_node_2_bind TYPE REF TO cl_bsp_wd_context_node,
        ls_model LIKE LINE OF m_models,
        lv_cnode_name TYPE string.

  lv_node_2_bind = ir_usage->if_bsp_wd_component_usage~get_context_node(
iv_node_2_bind ).
```

```
  DATA: lv_target_controller TYPE REF TO cl_bsp_wd_controller,
        lv_target_node TYPE REF TO cl_bsp_wd_context_node.

  CHECK me->do_binding = abap_true.

  TRY.
      lv_target_controller = me.

      lv_cnode_name = iv_target_node_name.
      TRANSLATE lv_cnode_name TO LOWER CASE.

      READ TABLE m_models WITH KEY model_id = lv_cnode_name INTO ls_model.
      IF sy-subrc EQ 0.
        lv_target_node ?= ls_model-instance.
      ELSE.
        RAISE EXCEPTION TYPE cx_bsp_wd_incorrect_implement
          EXPORTING
            controller = iv_name.
      ENDIF.


*       Share collection wrapper
    lv_node_2_bind->collection_wrapper->attach( lv_target_node->collection_wrapper ).

    CATCH cx_sy_move_cast_error cx_sy_ref_is_initial.
      RAISE EXCEPTION TYPE cx_bsp_wd_incorrect_implement.
  ENDTRY.

ENDMETHOD.                   "bind_context_nodes2
```

- Now, we come to the event handler of the button in the first view, for moving to the next page. When the user clicks this button, we start creating the component usages and context nodes

```
METHOD eh_on_button.

  DATA: lr_col TYPE REF TO if_bol_bo_col,
        lr_iterator TYPE REF TO if_bol_bo_col_iterator,
        lr_coco TYPE REF TO zl_zmulti_c_bspwdcomponen_impl,
        lr_ent TYPE REF TO cl_crm_bol_entity,
        lr_header type ref to cl_crm_bol_entity,
        model TYPE REF TO cl_bsp_model,
        lv_cu_name TYPE string,
        lv_cnode_name TYPE string.


  lr_coco ?= me->comp_controller.
* destroy existing dynamic component usage instances
* lr_coco->free_dyn_cus( ).

* for each search result, one usage and one node
  lr_col ?= lr_coco->typed_context->result->collection_wrapper.
```

```
  lr_iterator ?= lr_col->get_iterator( ).
  lr_ent ?= lr_iterator->get_first( ).

  WHILE lr_ent IS BOUND.
    lr_ent ?= lr_ent->get_related_entity( iv_relation_name = 'BTADVSSlsQuot' ).
    lr_header ?= lr_ent->get_related_entity( iv_relation_name = 'BTOrderHeader' ).
    lv_cu_name = lr_header->get_property_as_string( iv_attr_name = 'OBJECT_ID' ).
    CONCATENATE 'qh' lv_cu_name INTO lv_cnode_name.
    CONCATENATE 'qhn' lv_cu_name INTO lv_cu_name.

*   create dynamic component usage
    lr_coco->create_component_usage( iv_usage_name = lv_cu_name
                                            iv_reference_usage_name = ''
                                            iv_component_name = 'BT115QIT_SLSQ'
).
*   create dynamic context node(model)
    model = lr_coco->create_model(
        class_name     = 'ZL_ZMULTI_C_BSPWDCOMPONEN_CN01'   "blueprint class
        model_id       = lv_cnode_name ).                   "#EC NOTEXT

*   Register instances with component controller
    lr_coco->register_cu_instance( iv_cu_name = lv_cu_name
                                   iv_cnode_name = lv_cnode_name
                                     ir_model = model
                                       ir_bo = lr_ent ).
    CLEAR model.
    lr_ent ?= lr_iterator->get_next( ).

  ENDWHILE.
*   Navigate to next page
  view_manager->navigate( source_rep_view = rep_view outbound_plug =
'FromSearchToItems' ).

ENDMETHOD.
```

Note: In this method, note that we pass nothing for the import parameter iv_reference_usage_name when creating a component usage. Thus, we will avoid sharing view controllers between generated component usages.

Generating View Controllers, Context Node Binding, View Embedding

Now, we navigate to the second page. This page on loading, must present us with all the dynamic component usages embedded in it. The code present in “DO_INIT_CONTEXT” is presented below.

```
METHOD do_init_context.

  DATA: lr_coco TYPE REF TO zl_zmulti_c_bspwdcomponen_impl,
        lr_rep_view TYPE REF TO cl_bsp_wd_rep_view,
        lr_usage TYPE REF TO cl_bsp_wd_component_usage,
        lr_controller TYPE REF TO cl_bsp_wd_view_controller,
        lr_sub_ctrl TYPE REF TO cl_bsp_controller2,
        lr_bo TYPE REF TO cl_crm_bol_entity,
        ls_subctrlr LIKE LINE OF m_subcontrollers,
        lv_cu_name TYPE string,
        lv_view_name TYPE string,
        lv_view_area TYPE string,
        lv_cnode_name TYPE seocmpname.

  lr_coco ?= me->comp_controller.

  CLEAR gt_component_usages.
  gt_component_usages = lr_coco->get_component_usages( ).

*Delete all previous instances of dynamically created controllers.
  LOOP AT m_subcontrollers INTO ls_subctrlr.
    delete_controller( ls_subctrlr-component_id ).
  ENDLOOP.

*Refresh view area
  REFRESH me->viewarea_cont.

  LOOP AT gt_component_usages INTO lv_cu_name.

    lv_view_area = sy-tabix.
    CONCATENATE 'VA' lv_cu_name INTO lv_view_area.

*Create repository view
    CONCATENATE lv_cu_name 'BT115QIT_SLSQ/ItemsList' INTO lv_view_name SEPARATED BY
'.'.
    lr_rep_view = rep_view->create_instance_by_name( lv_view_name ).

*Get the correct context node to be bound
    CLEAR lv_cnode_name.
    lv_cnode_name = lr_coco->get_cnode_to_bind( iv_cu_name = lv_cu_name ).

*Embed the repository view in a new view area
*If the passed view area is not already present, it will be automatically created
    CALL METHOD me->bind_view
      EXPORTING
        rep_view                  = lr_rep_view
        viewarea                  = lv_view_area
*     suppress_area_init        = SPACE
*     suppress_area_init_single = SPACE
*     iv_suppress_context_init  = ABAP_FALSE
*     iv_force_binding          = ABAP_FALSE
```

```
      RECEIVING
        controller                 = lr_controller
        .

*Update information about view controller instance
    lr_coco->register_cu_instance( iv_cu_name = lv_cu_name ir_controller =
lr_controller ).

    lr_usage ?= lr_coco->get_component_usage( iv_usage_name = lv_cu_name ).

*Custom bind method -
    CALL METHOD lr_coco->bind_context_nodes2
      EXPORTING
        iv_controller_type  = cl_bsp_wd_controller=>co_type_component
*     iv_name             =
        iv_target_node_name = lv_cnode_name
        iv_node_2_bind      = 'BTADMINIMULTSEL'
        ir_usage            = lr_usage
        .

  ENDLOOP.

ENDMETHOD.
```

Note: When creating the repository view, note that I have used **'BT115QIT_SLSQ/ItemsList'** interface view of the BT115QIT_SLSQ component. The repository view name should be of the format **<some_name>.<interface_view_name>.**

- Our next task is to embed each component usage in a view area on the ItemsVS viewset. The htm code is as below. It is self explanatory.

```
<%@page language="abap" %>
<%@extension name="thtmlb" prefix="thtmlb" %>
<%@extension name="chtmlb" prefix="chtmlb" %>
<%@extension name="bsp" prefix="bsp" %>
<%
  Data: lv_rows type string,
  lv_lines type i,
  lv_va_name type string,
  lv_label type string.
  describe table controller->gt_component_usages lines lv_lines.
  lv_rows = lv_lines.
%>
<thtmlb:grid cellSpacing = "1"
             columnSize  = "1"
             height      = "100%"
             rowSize     = "<%= lv_rows %>"
             width       = "100%" >
  <%
  loop at controller->gt_component_usages into lv_va_name.
  concatenate 'VA' lv_va_name into lv_va_name.
  lv_rows = sy-tabix.
```

```
    lv_label = lv_va_name.
    replace first occurrence of 'VAqhn' in lv_label with 'Sales Quotation'.
    %>
    <thtmlb:gridCell colSpan     = "1"
                     columnIndex = "1"
                     rowIndex    = "<%= lv_rows %>"
                     rowSpan     = "1" >
     <br/><br/>

      <thtmlb:label id     = "<%= lv_va_name %>"
                    text   = "<%= lv_label %>"
                    design = "HEADER2" >
      </thtmlb:label>
      <bsp:call comp_id = "<%= controller->GET_VIEWAREA_CONTENT_ID( lv_va_name ) %>"
                url     = "<%= controller->GET_VIEWAREA_CONTENT_URL( lv_va_name ) %>"
  />
    </thtmlb:gridCell>
    <%
    endloop.
    %>
  </thtmlb:grid>
  <thtmlb:button id      = "s2button"
                 text    = "Back to Search"
                 onClick = "to_search_view" />
```

That's it, we are done. This wasn't as difficult as one would have thought. The dynamically created instances will stay alive as long as the parent controller is alive, unless you intentionally destroy/delete them. So, it is always advisable to clear those when you exit the application work area. I'm leaving it to the reader to take care of those details.

Related Content

[Dynamic layout manipulation in WD ABAP]

[Dynamic programming in WebDynpro ABAP]

[Dynamic Programming – SAP Online Help]

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.