Execute an ABAP Report From WebDynpro and Display the List Output In a WebDynpro View.

Applies to:
Webdynpro ABAP

Summary
This tutorial shows how to execute an ABAP report, which produces a list output, from a webdynpro screen and then displays the result in the same webdynpro screen. The tutorial demonstrates the use of dynamic programming in Webdynpro ABAP.

Author: Priyank Kumar Jain
Company: Bristlecone (www.bcone.com)
Created on: 08 August 2008

Author Bio
Priyank Jain is a senior development consultant working for Bristlecone India. He has been involved in various projects as a technical consultant in ABAP and Webdynpro.
Table of Contents
Create a report program .....................................................................................................................................3
  Report Selection screen ...............................................................................................................................3
  Report output ..............................................................................................................................................4
Creating webdynpro assistance class ................................................................................................................4
  Class Attributes .......................................................................................................................................4
  Class Methods .........................................................................................................................................5
Creating Webdynpro component and view .........................................................................................................6
  Creating the webdynpro component ...........................................................................................................6
  Creating the View .....................................................................................................................................6
    Creating the view context ............................................................................................................................7
    Creating the view layout .............................................................................................................................8
    Create actions for button and Inputfield ....................................................................................................9
    Implementing the event handlers of the view ............................................................................................11
    Coding the wddoModifyview method .........................................................................................................15
    Implementing Assistance class methods ....................................................................................................23
  Create Webdynpro Application ....................................................................................................................25
  Output ....................................................................................................................................................25
Limitations ......................................................................................................................................................28
  Related Content ......................................................................................................................................29
Disclaimer and Liability Notice ......................................................................................................................30
Create a report program

To start off, create a report program in ABAP that is going to be executed from the webdynpro application. In this example, we will create a report with a selection screen having parameters, checkboxes and radio-buttons. Name the report as ZTEST_REPORT. The report will display the data based on the selections made by the user on the selection screen. The selection screen will later be replicated in the webdynpro view for taking user inputs.

Report Selection screen

A sample selection screen looks like the one below

![Sample Selection Screen](image)

**Note:** Please note that this example does not deal with select-options and the usage of selection-screen blocks. It can of course be used and replicated in webdynpro but is not in the scope of this basic tutorial.
Report output

The report produces the following sample output. Note that this is a list output based on the inputs in the selection screen above.

Creating webdynpro assistance class

Before we create our webdynpro component and view, let’s first create a class that is going to serve as the assistance class for the webdynpro component. This class will define the methods that will execute the report as well as pass the selection-screen information to the report from webdynpro to the report.

Open transaction SE24 and create a new class. Name it as ZTESTASSISTANCE.

Create the class as a usual ABAP class.

Class Attributes

Create the attributes of the class as shown below
Here, I created a Z-table type ZTT_RADIO_LEADSEL for the attribute MT_RADIO_LEADSEL. It has a line type whose structure is as below. The other attributes have standard SAP types.

```
NODE_NAME  TYPE   CHAR255
NODE_REF   TYPE REF TO  IF_WD_CONTEXT_NODE
LEAD_SEL   TYPE   INT1
```

**Class Methods**

Declare the following methods in the class. The implementation of these methods will be done at a later stage in the tutorial.

**Method Name**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Level</th>
<th>Visibility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXEC_REPORT</strong></td>
<td>Instance</td>
<td>Public</td>
<td>Report name</td>
</tr>
<tr>
<td>REPNAME</td>
<td>Importing</td>
<td>STRING</td>
<td>Report name</td>
</tr>
<tr>
<td>TAB_OUTPUT</td>
<td>Exporting</td>
<td>STRING_TT</td>
<td>Table of strings</td>
</tr>
<tr>
<td>CT_PARAMS</td>
<td>Changing</td>
<td>RSPARMS_TT</td>
<td>RSPARMS table</td>
</tr>
<tr>
<td><strong>GET_REP_SELSCREEN_INFO</strong></td>
<td>Instance</td>
<td>Public</td>
<td>Report Name</td>
</tr>
<tr>
<td>REPNAME</td>
<td>Importing</td>
<td>STRING</td>
<td>Report Name</td>
</tr>
<tr>
<td>CT_SSCR</td>
<td>Exporting</td>
<td>RSBK_T_RSSCR</td>
<td>Selection-Screen info.</td>
</tr>
<tr>
<td>CT_TEXTTAB</td>
<td>Exporting</td>
<td>TEXTPOOL_TABLE</td>
<td>Text elements</td>
</tr>
<tr>
<td><strong>SET_ATTR_INFO</strong></td>
<td>Instance</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>CT_ATTR_INFO</td>
<td>Importing</td>
<td>ZTT_WDR_CONTEXT_ATTRIBUTE_INFO*</td>
<td></td>
</tr>
<tr>
<td>CT_SELTEXT</td>
<td>Importing</td>
<td>TABLE_OF_TEXTPOOL</td>
<td></td>
</tr>
<tr>
<td>CT_RADIO_LEADSEL</td>
<td>Importing</td>
<td>ZTT_RADIO_LEADSEL</td>
<td></td>
</tr>
<tr>
<td>CT_SSCR</td>
<td>Importing</td>
<td>RSBK_T_RSSCR</td>
<td></td>
</tr>
<tr>
<td><strong>GET_ATTR_INFO</strong></td>
<td>Exporting</td>
<td>ZTT_WDR_CONTEXT_ATTRIBUTE_INFO*</td>
<td></td>
</tr>
<tr>
<td>CT_ATTR_INFO</td>
<td>Exporting</td>
<td>ZTT_WDR_CONTEXT_ATTRIBUTE_INFO*</td>
<td></td>
</tr>
<tr>
<td>CT_SELTEXT</td>
<td>Exporting</td>
<td>TABLE_OF_TEXTPOOL</td>
<td></td>
</tr>
<tr>
<td>CT_RADIO_LEADSEL</td>
<td>Exporting</td>
<td>ZTT_RADIO_LEADSEL</td>
<td></td>
</tr>
<tr>
<td>CT_SSCR</td>
<td>Exporting</td>
<td>RSBK_T_RSSCR</td>
<td></td>
</tr>
</tbody>
</table>

*ZTT_WDR_CONTEXT_ATTRIBUTE_INFO is a table type having WDR_CONTEXT_ATTRIBUTE_INFO as its line type.
Creating Webdynpro component and view

Creating the webdynpro component

Open transaction SE80 and create a new webdynpro component. Name it as ZWDTEST_REPORT. Assign the class created above as the assistance class for this webdynpro component.

<table>
<thead>
<tr>
<th>Web Dynpro Component</th>
<th>ZWDTEST_REPORT</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assistance Class</td>
<td>ZTESTASSISTANCE</td>
<td></td>
</tr>
<tr>
<td>Created By</td>
<td></td>
<td>Created On 00.09.2008</td>
</tr>
<tr>
<td>Last Changed By</td>
<td></td>
<td>Changed On 00.09.2008</td>
</tr>
<tr>
<td>Original Lang.</td>
<td>EN</td>
<td>Package</td>
</tr>
<tr>
<td>Accessibility Checks Active</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Creating the View

A view named MAIN will get created by default when the webdynpro component is created and saved. If not, then create one and name it as MAIN.
Creating the view context

Create nodes and attributes in the context tab of the view MAIN as shown in the following screenshot.

<table>
<thead>
<tr>
<th>Node</th>
<th>Attribute name</th>
<th>Attribute type</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT</td>
<td>REPNAME</td>
<td>STRING</td>
</tr>
<tr>
<td>SUBMIT</td>
<td>VISIBLE</td>
<td>WDUI_VISIBILITY</td>
</tr>
<tr>
<td>SELSCREEN_CONTAINER</td>
<td>VISIBLE</td>
<td>WDUI_VISIBILITY</td>
</tr>
<tr>
<td>RBG</td>
<td>TEXT</td>
<td>STRING</td>
</tr>
<tr>
<td>RPT_OUTPUT</td>
<td>OUTPUT_TEXT</td>
<td>string</td>
</tr>
</tbody>
</table>

Ensure that all the nodes created above have the default cardinality of 1:1 and selection 0:1. The initialization lead selection checkbox is checked for all the nodes.
Creating the view layout

Open the layout tab of the view and insert UI elements required to be displayed on the view. The layout of the view should look like the one shown in the following screenshot.

The UI elements list is shown below

As you can see above, the view contains a label, an input field for the name of the report to be executed, a selection screen container which will hold the dynamically replicated report selection screen and a text view element which will be used to show the list output of the report.

Bind the properties of the UI elements to the respective context attribute as mentioned in the following table.

<table>
<thead>
<tr>
<th>UI Element</th>
<th>Property</th>
<th>Node</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPNAME</td>
<td>VALUE</td>
<td>REPORT</td>
<td>REPNAME</td>
</tr>
<tr>
<td>SELSCREEN_CONTAINER</td>
<td>VISIBLE</td>
<td>SELSCREEN_CONTAINER</td>
<td>VISIBLE</td>
</tr>
<tr>
<td>SUBMIT</td>
<td>VISIBLE</td>
<td>SUBMIT</td>
<td>VISIBLE</td>
</tr>
<tr>
<td>TEXT_VIEW</td>
<td>TEXT</td>
<td>RPT_OUTPUT</td>
<td>OUTPUT_TEXT</td>
</tr>
</tbody>
</table>
Create actions for button and Inputfield

We need to define an action for the Submit button on the view. Create an action named as SUBMIT for the event OnAction of the button. The properties of the SUBMIT button will thus look like below.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Bind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties (Button)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>BUTTON</td>
<td></td>
</tr>
<tr>
<td>Layout Data</td>
<td>MatrixHeadData</td>
<td></td>
</tr>
<tr>
<td>contextMenuBehavior</td>
<td>inherit</td>
<td></td>
</tr>
<tr>
<td>contextMenuId</td>
<td></td>
<td></td>
</tr>
<tr>
<td>design</td>
<td>standard</td>
<td></td>
</tr>
<tr>
<td>enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>explanation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>imageFirst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>imageSource</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>Submit</td>
<td></td>
</tr>
<tr>
<td>textDirection</td>
<td>inherit</td>
<td></td>
</tr>
<tr>
<td>tooltip</td>
<td></td>
<td></td>
</tr>
<tr>
<td>visible</td>
<td>MAIN SUBMIT:VISIBLE</td>
<td></td>
</tr>
<tr>
<td>width</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>OnAction</th>
<th>SUBMIT</th>
</tr>
</thead>
</table>

Also, create another action for the OnEnter event of the input field REPNAME. Name this action as ENTER. The properties of the input field should look like the screenshot below.
Creating these actions will create the event handler methods named `ONACTION_SUBMIT` and `ONACTION_ENTER` in the view controller. Navigate to the ‘Methods’ tab to find these methods there.
Implementing the event handlers of the view

We now need to write the implementation code in these event handler methods.

- **ONACTION_ENTER**

  After the user enters the report name in the input field and presses enter, we will internally change the name of the report entered to upper case. To achieve this, write the following code in the ONACTION_ENTER method of the view.

```
METHOD onactionenter .

  DATA lo_nd_report TYPE REF TO if_wd_context_node.
  DATA lo_el_report TYPE REF TO if_wd_context_element.
  DATA ls_report TYPE wd_this->element_report.
  DATA lv_repname LIKE ls_report-repname.
  * navigate from <CONTEXT> to <REPORT> via lead selection
    lo_nd_report = wd_context->get_child_node( name = wd_this->wdctx_report ).
  * get element via lead selection
    lo_el_report = lo_nd_report->get_element(  ).
  * get single attribute
    lo_el_report->get_attribute(
      EXPORTING
        name = 'REPNAME`
      IMPORTING
        value = lv_repname ).

    IF lv_repname IS NOT INITIAL.
      TRANSLATE lv_repname TO UPPER CASE.
    ENDIF.

    lo_el_report->set_attribute(
      EXPORTING
        name = 'REPNAME`
      value = lv_repname ).

ENDMETHOD.
```
Execute an ABAP Report From WebDynpro and Display the List Output In a WebDynpro View.

• **ONACTION_SUBMIT**

Once the user presses the submit button, the inputs need to be passed to the report selection screen and the list output will be displayed on the view. To achieve this, write the following code in the ONACTION_SUBMIT method of the view.

(Note that replicating the selection screen of the report to the view is not covered in the tutorial yet. It is covered in the next section. This method will be called after the report selection screen is displayed so that the user can put the input values in it.)

```
METHOD onactionsubmit.

DATA: lo_nd_report TYPE REF TO if_wd_context_node,
     lo_el_report TYPE REF TO if_wd_context_element,
     ls_report TYPE wd_this->element_report,
     lv_repname LIKE ls_report-repname,
     itab_output TYPE string_tt,
     wa_output TYPE string.

DATA: lt_attr_info TYPE TABLE OF wdr_context_attribute_info,
     ls_attr_info TYPE wdr_context_attribute_info,
     lt_texttab TYPE TABLE OF textpool,
     ls_texttab TYPE textpool.

DATA: lt_rad_lead_sel TYPE TABLE OF zwd_radio_leadsel,
     ls_rad_lead_sel TYPE zwd_radio_leadsel,
     lt_sscr TYPE TABLE OF rsscr,
     ls_sscr TYPE rsscr.

DATA: lo_nd_rpt_output TYPE REF TO if_wd_context_node,
     lo_el_rpt_output TYPE REF TO if_wd_context_element,
     ls_rpt_output TYPE wd_this->element_rpt_output,
     lv_output_text LIKE ls_rpt_output-output_text.

TYPES: BEGIN OF typ_nam_val,
        name TYPE string,
        value TYPE string,
    END OF typ_nam_val.

DATA: lt_nam_val TYPE TABLE OF typ_nam_val,
     ls_nam_val TYPE typ_nam_val.

DATA: lr_node TYPE REF TO if_wd_context_node,
     lr_element TYPE REF TO if_wd_context_element,
     lr_node_rad TYPE REF TO if_wd_context_element,
     lv_leadsel_index TYPE i,
     lv_tabix TYPE sy-tabix.

* navigate from <CONTEXT> to <REPORT> via lead selection
  lo_nd_report = wd_context->get_child_node( name = wd_this->wdctx_report ).

* get element via lead selection
  lo_el_report = lo_nd_report->get_element(  ).

* get single attribute
  lo_el_report->get_attribute( EXPORTING
                               name = 'REPNAME'
                               IMPORTING
Execute an ABAP Report From WebDynpro and Display the List Output In a WebDynpro View.

```abap
value = lv_repname ).

**read context attributes**
CALL METHOD wd_assist->get_attr_info
  IMPORTING
    ct_attr_info     = lt_attr_info
    ct_seltext       = lt_texttab
    ct_radio_leadsel = lt_rad_lead_sel
    ct_sscr          = lt_sscr.

IF lt_attr_info IS NOT INITIAL.
  LOOP AT lt_attr_info INTO ls_attr_info.
    lr_node = wd_context->get_child_node( name = ls_attr_info-name ).
    IF lr_node IS NOT INITIAL.
      lr_element = lr_node->get_element( ).
      IF lr_element IS NOT INITIAL.
        lr_element->get_attribute(
          EXPORTING
            name = ls_attr_info-name
          IMPORTING
            value = ls_nam_val-value ).
      ENDIF.
    ENDIF.
    ls_nam_val-name = ls_attr_info-name.
    APPEND ls_nam_val TO lt_nam_val.
  ENDLOOP.
ENDIF.

******radio buttons
LOOP AT lt_rad_lead_sel INTO ls_rad_lead_sel.
  lr_node_rad = ls_rad_lead_sel-node_ref.
  IF lr_node_rad IS NOT INITIAL.
    lv_leadsel_index = lr_node_rad->get_lead_selection_index( ).
  ENDIF.
  IF lv_leadsel_index GT 0.
    CLEAR lv_tabix.
    LOOP AT lt_sscr INTO ls_sscr WHERE matchcode = ls_rad_lead_sel-node_name.
      ADD 1 TO lv_tabix.
      IF lv_tabix = lv_leadsel_index.
        READ TABLE lt_nam_val INTO ls_nam_val WITH KEY name = ls_sscr-name.
        IF sy-subrc EQ 0.
          ls_nam_val-value = 'X'.
          MODIFY lt_nam_val FROM ls_nam_val INDEX sy-tabix TRANSPORTING value.
        ENDIF.
      ENDIF.
    ENDLOOP.
  ENDIF.
ENDLOOP.

**construct selection screen parameters table**
DATA : lt_params TYPE TABLE OF rsparams,
       ls_params TYPE rsparams.

LOOP AT lt_nam_val INTO ls_nam_val.
  ls_params-selname = ls_nam_val-name.
  ls_params-kind = 'P'.
  ls_params-low = ls_nam_val-value.
  APPEND ls_params TO lt_params.
ENDLOOP.
```
IF lv_repname IS NOT INITIAL.
    REFRESH itab_output.
    TRANSLATE lv_repname TO UPPER CASE.
    CALL METHOD wd_assist->exec_report
        EXPORTING
            repname    = lv_repname
        IMPORTING
            tab_output = itab_output
        CHANGING
            ct_params  = lt_params.
ENDIF.

**create a string to show in text view
* navigate from <CONTEXT> to <RPT_OUTPUT> via lead selection
  lo_nd_rpt_output = wd_context->get_child_node( name = wd_this->wdctx_rpt_output ).
* get element via lead selection
  lo_el_rpt_output = lo_nd_rpt_output->get_element(  ).

DATA crlf(2) TYPE c.
crlf = cl_abap_char_utilities=>cr_lf.
LOOP AT itab_output INTO wa_output.
    CONCATENATE lv_output_text wa_output INTO lv_output_text SEPARATED BY crlf.
ENDLOOP.
* get single attribute
  lo_el_rpt_output->set_attribute( EXPORTING
        name = `OUTPUT_TEXT`
        value = lv_output_text ).
ENDMETHOD.
Coding the wddoModifyview method

For this example tutorial, the wddoModifyView method is the most important method of the view. In this method, we will try to recreate the selection of the report name entered by the user on the webdynpro view. This involves dynamically creating and adding UI elements on the webdynpro view. In our example report, the selection screen consists of one parameter, two sets of radio buttons and one checkbox. In this method, we will write the code to display these on the webdynpro view.

Write the following code in the wddoModifyView method.

```plaintext
METHOD wddomodifyview .
  DATA : lo_nd_report TYPE REF TO if_wd_context_node,
    lo_el_report TYPE REF TO if_wd_context_element,
    ls_report TYPE wd_this->element_report,
    lv_repname LIKE ls_report-repname.

  **fields for selection screen
  DATA : lt_sscr TYPE TABLE OF rsscr,
    lt_radio TYPE TABLE OF rsscr,
    ls_radio TYPE rsscr,
    ls_sscr TYPE rsscr,
    lt_texttab TYPE TABLE OF textpool,
    ls_texttab TYPE textpool.

  **nodes
  DATA : lr_container TYPE REF TO cl_wd_uielement_container,
    lr_node_info TYPE REF TO if_wd_context_node_info,
    lr_child_node_info TYPE REF TO if_wd_context_node_info,
    lt_child_nodes TYPE TABLE OF wdr_context_child,
    ls_child_nodes TYPE wdr_context_child,
    lr_element TYPE REF TO cl_wd_uielement.

  **attributes table
  DATA :
    lt_attr_info TYPE TABLE OF wdr_context_attribute_info,
    ls_attr_info TYPE wdr_context_attribute_info,
    comp_tab TYPE cl_abap_structdescr=>component_table,
    comp LIKE LINE OF comp_tab,
    lr_struct_type TYPE REF TO cl_abap_structdescr.

  **ui element class references
  DATA : lr_label TYPE REF TO cl_wd_label,
    lr_inputfield TYPE REF TO cl_wd_input_field,
    lr_checkbox TYPE REF TO cl_wd_checkbox,
    lr_radiobutton_idx
    TYPE REF TO cl_wd_radiobutton_group_by_idx.

  TYPES : BEGIN OF typ_rbnode,
    name TYPE string,
  END OF typ_rbnode.

  DATA : lt_rbnode TYPE TABLE OF typ_rbnode,
    ls_rbnode TYPE typ_rbnode,
    lt_children TYPE cl_wd_uielement=>tt_uielement,
    ls_children LIKE LINE OF lt_children.

  DATA : lv_string TYPE string,
    lv_path TYPE string,
    lv_text type string.
```
Execute an ABAP Report From WebDynpro and Display the List Output In a WebDynpro View.

```abap
TYPES : BEGIN OF elem_ids,
  id TYPE string,
END OF elem_ids.
DATA : lt_ids TYPE TABLE OF elem_ids,
  ls_ids TYPE elem_ids.

* navigate from <CONTEXT> to <REPORT> via lead selection
lo_nd_report = wd_context->get_child_node
  ( name = wd_this->wdctx_report ).
* get element via lead selection
lo_el_report = lo_nd_report->get_element( ).

* get single attribute
lo_el_report->get_attribute(
  EXPORTING
  name = 'REPNAME',
  IMPORTING
  value = lv_repname ).

IF lv_repname IS NOT INITIAL.
  TRANSLATE lv_repname TO UPPER CASE.
**get the selection screen information for the report
CALL METHOD wd_assist->get_rep_selscreen_info
  EXPORTING
    repname = lv_repname
  IMPORTING
    ct_sscr = lt_sscr
    ct_texttab = lt_texttab.

**create a separate table for radio buttons
LOOP AT lt_sscr INTO ls_sscr WHERE flag1 = '03'.
**03 means radiobutton
  APPEND ls_sscr TO lt_radio.
ENDLOOP.
SORT lt_radio BY matchcode.
DELETE ADJACENT DUPLICATES FROM lt_radio COMPARING matchcode.

**dynamically create context nodes & attributes for the
**selection screen elements
**dynamically create view elements based on
**field information retrieved
IF lt_sscr IS NOT INITIAL.
  lr_container ?= view->get_element( 'SELSCREEN_CONTAINER' ).
  lr_node_info = wd_context->get_node_info( ).
  LOOP AT lt_sscr INTO ls_sscr.
    CLEAR ls_attr_info.
    ls_attr_info-name = ls_sscr-name.
    **for type , based on the type received, set the
    **webdynpro context attribute type
    **parameters on selection screen
    IF ls_sscr-kind EQ 'P'.
      TRANSLATE ls_sscr-type TO UPPER CASE.
    **assign webdynpro types for selection screen element types
    CASE ls_sscr-type.
      WHEN 'D' OR 'I',
        ls_attr_info-type_name = ls_sscr-type.
```

SAP DEVELOPER NETWORK | sdn.sap.com
BUSINESS PROCESS EXPERT COMMUNITY | bpx.sap.com
© 2008 SAP AG
WHEN 'F' OR 'P'.
    ls_attr_info-type_name = 'F'.
WHEN 'C' OR 'N' OR 'G'.'
    ls_attr_info-type_name = 'STRING'.
ENDCASE.
**select options on selection screen
ELSEIF ls_sscr-kind EQ 'S'.
******
**here we need to create dynamic component usage for
**select options component
**Not covered as a part of this tutorial. Refer tutorial on
**using select-options in webdynpro
******
ENDIF.
APPEND ls_attr_info TO lt_attr_info.
ENDLOOP.
CLEAR ls_attr_info.

**get all existing nodes in a table -
this is to check for duplication of nodes being created
CALL METHOD wd_context->get_child_nodes
RECEIVING
    child_nodes = lt_child_nodes.

*build context nodes and attributes -
one each for each selection screen parameter
LOOP AT lt_attr_info INTO ls_attr_info.
**build attribute information
    CLEAR comp.
    comp-name = ls_attr_info-name.
    comp-type ?=
        cl_abap_datadescr=>describe_by_name( ls_attr_info-type_name ).
        APPEND comp TO comp_tab.
    lr_struct_type = cl_abap_structdescr=>get( comp_tab ).
**build node and pass attribute information
**node should not already exist
    READ TABLE lt_child_nodes INTO ls_child_nodes
    WITH KEY name = ls_attr_info-name.
    IF sy-subrc NE 0.
        CALL METHOD lr_node_info->add_new_child_node
            EXPORTING
                name = ls_attr_info-name
                is_mandatory = abap_true
                is_multiple = abap_true
                static_element_rtti = lr_struct_type
                is_static = abap_false
            RECEIVING
                child_node_info = lr_child_node_info.
        ENDIF.
    ENDLOOP.
**create ui elements
    IF lr_container IS NOT INITIAL.
    **get all ui elements already in container to avoid duplication
    CALL METHOD lr_container->get_children
      RECEIVING
        the_children = lt_children.
    LOOP AT lt_children INTO ls_children.
lr_element := ls_children.
ls_ids-id = ls_children->id.
APPEND ls_ids TO lt_ids.
ENDLOOP.

**flow layout for the transparent container
cl_wd_flow_layout=>new_flow_layout
  ( container = lr_container ).
LOOP AT lt_attr_info INTO ls_attr_info.
  READ TABLE lt_sscr INTO ls_sscr
    WITH KEY name = ls_attr_info-name.
  CASE ls_sscr-flag1.
    WHEN '01'. "PARAMETERS
      **read the selection screen texts from lt_texttab table
      READ TABLE lt_texttab INTO ls_texttab
        WITH KEY id = 'S' key = ls_attr_info-name.
      IF sy-subrc EQ 0.
        lv_text = ls_texttab-entry.
      ELSE.
        lv_text = ls_attr_info-name.
      ENDIF.
      CONDENSE lv_text.
      lr_label = cl_wd_label=>new_label
        ( id = ls_attr_info-name text = lv_text ).
      cl_wd_flow_data=>new_flow_data( element = lr_label ).
    ENDIF.
    **input field to be added
    CLEAR lv_string.
    CONCATENATE 'I_' ls_attr_info-name INTO lv_string.
    READ TABLE lt_sscr INTO ls_sscr
      WITH KEY name = ls_attr_info-name.
    IF sy-subrc EQ 0.
      lr_inputfield = cl_wd_input_field=>new_input_field
        ( id = lv_string length = ls_sscr-olength ).
    ENDIF.
    cl_wd_flow_data=>new_flow_data( element = lr_inputfield ).
  END_CASE.
**node path for binding "value" and "label for" properties
  CLEAR lv_path.
  CONCATENATE ls_attr_info-name '.' ls_attr_info-name
    INTO lv_path.
CALL METHOD lr_inputfield->bind_value( lv_path ).
CALL METHOD lr_label->set_label_for( lv_string ).
ENDIF.
WHEN '09'. **CHECKBOX
**read the selection screen text from lt_texttab table
READ TABLE lt_texttab INTO ls_texttab
  WITH KEY id = 'S' key = ls_attr_info-name.
IF sy-subrc EQ 0.
  lv_text = ls_texttab-entry.
ELSE.
  lv_text = ls_attr_info-name.
ENDIF.
CONDENSE lv_text.
lr_label = cl_wd_label=>new_label
  ( id = ls_attr_info-name text = lv_text ).
cl_wd_flow_data=>new_flow_data( element = lr_label ).
**label should not already exist
CLEAR ls_ids.
READ TABLE lt_ids INTO ls_ids
  WITH KEY id = ls_attr_info-name.
IF sy-subrc NE 0.
  CALL METHOD lr_container->add_child
    EXPORTING
      the_child = lr_label.
ENDIF.
CLEAR lv_string.
CONCATENATE 'R_' ls_attr_info-name INTO lv_string.
READ TABLE lt_sscr INTO ls_sscr
  WITH KEY name = ls_attr_info-name.
IF sy-subrc EQ 0.
  CLEAR lv_path.
  CONCATENATE ls_attr_info-name '.' ls_attr_info-name
    INTO lv_path.
  lr_checkbox = cl_wd_checkbox=>new_checkbox
    ( id = lv_string bind_checked = lv_path ).
ENDIF.
cl_wd_flow_data=>new_flow_data( element = lr_checkbox ).
** checkbox should not already exist
CLEAR ls_ids.
READ TABLE lt_ids INTO ls_ids
  WITH KEY id = lv_string.
IF sy-subrc NE 0.
  CALL METHOD lr_container->add_child
    EXPORTING
      INDEX     =
      the_child = lr_checkbox.
    CALL METHOD lr_label->set_label_for( lv_string ).
ENDIF.
**RADIOBUTTONS
WHEN '03'.
**radio buttons are handled differently - done further below
ENDCASE.
ENDLOOP.
********************************************************
DATA : node_name TYPE string,
       node_path TYPE string.
DATA : itemlist TYPE TABLE OF if_main=>element_rbg,
ls_itemlist TYPE if_main=>element_rbg,
item_node TYPE REF TO if_wd_context_node.

**create nodes for radiobutton group -
as many as the number of entries in lt_radio table**

LOOP AT lt_radio INTO ls_radio.
    READ TABLE lt_child_nodes INTO ls_child_nodes
    WITH KEY name = ls_radio-matchcode.
    IF sy-subrc NE 0.
        CLEAR comp.
        REFRESH comp_tab.
        comp-name = 'TEXT'.
        comp-type => cl_abap_datadescr=>describe_by_name( 'STRING' ).
        APPEND comp TO comp_tab.
        lr_struct_type = cl_abap_structdescr=>get( comp_tab ).
        node_name = ls_radio-matchcode.
        CALL METHOD lr_node_info->add_new_child_node
            EXPORTING
                name = node_name
            RECEIVING
                child_node_info = lr_child_node_info.
        CONCATENATE node_name '.' 'TEXT' INTO node_path.
    **create radiobutton group by index ui element**
    REFRESH itemlist.
    lr_radiobutton_idx =
        cl_wd_radiobutton_group_by_idx=>new_radiobutton_group_by_idx
            ( id = node_name bind_texts = node_path ).
    LOOP AT lt_sscr INTO ls_sscr WHERE flag1 = '03'
        AND matchcode = ls_radio-matchcode.
        READ TABLE lt_texttab INTO ls_texttab
            WITH KEY id = 'S' key = ls_sscr-name.
        IF sy-subrc EQ 0.
            ls_itemlist-text = ls_texttab-entry.
        ELSE.
            ls_itemlist-text = ls_sscr-name.
        ENDIF.
        APPEND ls_itemlist TO itemlist.
    ENDLOOP.
    item_node = wd_context->get_child_node( node_name ).
    item_node->bind_table( itemlist ).

**set lead selection to first element while creating**
    item_node->set_lead_selection_index( '1' ).
    cl_wd_flow_data=>new_flow_data
        ( element = lr_radiobutton_idx ).

**should not already exist**
    CLEAR ls_ids.
    READ TABLE lt_ids INTO ls_ids WITH KEY id = node_name.
IF sy-subrc NE 0.
CALL METHOD lr_container->add_child
EXPORTING
   the_child = lr_radiobutton_idx.
ENDIF.
ENDIF.
ENDDO.

******end of creation of nodes for radiobutton groups*****
ENDIF.
ENDIF.
ENDIF.

**get lead selection of nodes for radiobutton and pass it to the attributes of assistance class**

DATA : lt_rad_lead_sel TYPE TABLE OF zwd_radio_leadsel,
   ls_rad_lead_sel TYPE zwd_radio_leadsel,
   lv_node_name TYPE string.

LOOP AT lt_radio INTO ls_radio.
   CLEAR ls_rad_lead_sel.
   lv_node_name = ls_rad_lead_sel-node_name = ls_radio-matchcode.

   ls_rad_lead_sel-node_ref =
      wd_context->get_child_node( name = lv_node_name ).
   IF ls_rad_lead_sel-node_ref IS NOT INITIAL.
      ls_rad_lead_sel-lead_sel =
         ls_rad_lead_sel-node_ref->get_lead_selection_index( ).
   ENDIF.
   APPEND ls_rad_lead_sel TO lt_rad_lead_sel.
ENDDO.

*********************************************************
IF lt_attr_info IS NOT INITIAL.
   CALL METHOD wd_assist->set_attr_info
      EXPORTING
         ct_attr_info     = lt_attr_info
         ct_seltext       = lt_texttab
         ct_radio_leadsel = lt_rad_lead_sel
         ct_sscr          = lt_sscr.
   ENDIF.

**set visibility of submit button and selection screen container**
DATA lo_nd_submit TYPE REF TO if_wd_context_node.
DATA lo_el_submit TYPE REF TO if_wd_context_element.
DATA ls_submit TYPE wd_this->element_submit.
DATA lo_nd_selscreen_container TYPE REF TO if_wd_context_node.
DATA lo_el_selscreen_container TYPE REF TO if_wd_context_element.
DATA ls_selscreen_container TYPE wd_this->element_selscreen_container.
DATA lv_visible LIKE ls_selscreen_container-visible.
* navigate from <CONTEXT> to <SUBMIT> via lead selection
  lo Nd_submit =
      wd_context->get_child_node( name = wd_this->wdctx_submit ).
* get element via lead selection
  lo_el_submit = lo_nd_submit->get_element(  ).

* navigate from <CONTEXT> to <SELSCREEN_CONTAINER> via lead selection
  lo Nd_selscreen_container =
      wd_context->get_child_node( name = wd_this->wdctx_selscreen_container ).
* get element via lead selection
  lo_el_selscreen_container = lo_nd_selscreen_container->get_element(  ).

  IF lv_repname IS NOT INITIAL.
    lo_el_submit->set_attribute( EXPORTING
        name = 'VISIBLE'
        value = '2' ).
    lo_el_selscreen_container->set_attribute( EXPORTING
        name = 'VISIBLE'
        value = '2' ).
  ELSE.
    lo_el_submit->set_attribute( EXPORTING
        name = 'VISIBLE'
        value = '1' ).
    lo_el_selscreen_container->set_attribute( EXPORTING
        name = 'VISIBLE'
        value = '1' ).
  ENDIF.
ENDMETHOD.
Implementing Assistance class methods

Now is the time to have a look at the implementation of the methods of the assistance class. Below is the source code for the methods of the assistance class.

**EXEC_REPORT method**

This method actually executes the report program and returns the output list. Write the following source code in the implementation of this method.

```abap
METHOD exec_report.
  TYPES : BEGIN OF t_olist,
    filler1(255) TYPE c,
  END OF t_olist.
  DATA : report_name(20) TYPE c,
    ls_wa TYPE string,
    lobj TYPE TABLE OF abaplist,
    olist TYPE TABLE OF t_olist,
    wa_olist TYPE t_olist.

  report_name = repname.
  SUBMIT (report_name) WITH SELECTION-TABLE ct_params EXPORTING LIST TO MEMORY AND RETURN.

  CALL FUNCTION 'LIST_FROM_MEMORY'
    TABLES
      listobject = lobj
    EXCEPTIONS
      not_found = 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.

    **RAW format is returned from memory - convert it to readable format - type C**
    CALL FUNCTION 'LIST_TO_ASCI'
      EXPORTING
        list_index = -1
      TABLES
        listasci = olist
        listobject = lobj
      EXCEPTIONS
        empty_list = 1
        list_index_invalid = 2
        others = 3.
    IF sy-subrc <> 0.
      MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
        WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
    ENDIF.

    DATA lv_lin TYPE i.
    DESCRIBE TABLE olist LINES lv_lin.
    IF lv_lin GT 2.
      **first two lines in the report are the report text and then a line of hyphens**
      **these may not be required in the output**
    LOOP AT olist INTO wa_olist.
      IF sy-tabix GT 2.
        ls_wa = wa_olist-filler1.
        APPEND ls_wa TO tab_output.
      ENDIF.
    ENDLOOP.
```
**free the memory object**

```abap
REFRESH lobj.
CALL FUNCTION 'LIST_FREE_MEMORY'
  TABLES
    listobject = lobj.
ENDMETHOD.
```

### GET_REP_SELSCREEN_INFO method

This method retrieves the information about the selection screen of the report and is implemented as below

```abap
METHOD get_rep_selscreen_info.
  DATA : lv_repname TYPE rsvar-report,
         p_sscr TYPE TABLE OF rsscr,
         lt_texttab TYPE TABLE OF textpool.
  lv_repname = repname.
  LOAD REPORT lv_repname PART 'SSCR' INTO p_sscr.
  IF p_sscr IS NOT INITIAL.
    DELETE p_sscr WHERE numb EQ 0.
    READ TEXTPOOL lv_repname INTO lt_texttab LANGUAGE sy-langu.
    IF sy-subrc EQ 0.
      ct_texttab = lt_texttab.
    ENDIF.
  ENDIF.
  ct_sscr = p_sscr.
ENDMETHOD.
```

### SET_ATTR_INFO method

This method sets the class attributes as per the values retrieved from the webdynpro component. This method is called from wddoModifyView method of the view MAIN.

```abap
METHOD set_attr_info.
  mt_attr_info = ct_attr_info.
  mt_seltext = ct_seltext.
  mt_radio_leadsel = ct_radio_leadsel.
  mt_sscr = ct_sscr.
ENDMETHOD.
```

### GET_ATTR_INFO method

This method returns the class attributes to be used in webdynpro. The attributes are set during the execution of wddoModifyView method and are retrieved while submitting the inputs and executing the report program. This method is called in ONACTION_SUBMIT event handler.

```abap
METHOD get_attr_info.
  ct_attr_info = mt_attr_info.
  ct_seltext = mt_seltext.
  ct_radio_leadsel = mt_radio_leadsel.
  ct_sscr = mt_sscr.
ENDMETHOD.
```
Create Webdynpro Application
Create a webdynpro application for executing and testing what we have done so far.

Output
Right-click on the webdynpro application created in the step above and observe the output. Ensure that the Report program, the assistance class, the webdynpro component and view are activated without errors.

1) Right-click on the webdynpro application and then click on “Test”. The browser will open and the following screen appears.

2) Type the name of the report (ZTEST_REPORT) created earlier and press Enter. The selection screen elements of the report will be displayed on the screen.
Execute an ABAP Report From WebDynpro and Display the List Output In a WebDynpro View.
3) Provide inputs and click on the submit button

![WebDynpro output](image)

The output is visible as shown below.
Limitations

This tutorial gives a basic idea on how to execute a simple report from webdynpro, making available the list output in webdynpro itself. Developers can use this document as a reference to work on more complex and advanced requirements not covered as a part of this. For example, this tutorial does not cover the usage of select-options in the report selection screen. Also, selection screen elements like blocks or tab-strips are not covered. Please feel free to share more ideas or comments so that going further even things such as these can be covered.
Execute an ABAP Report From WebDynpro and Display the List Output In a WebDynpro View.

Related Content

Dynamic Programming in webdynpro
Dynamic Programming – Webdynpro ABAP
Using select-options in webdynpro ABAP
Working with Select-options in webdynpro ABAP
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.