Working with the Roadmap UI Element in Web Dynpro ABAP

Applies to:

Web Dynpro ABAP

Summary

This tutorial shows the use of the Roadmap UI element in Web Dynpro ABAP applications. The tutorial shows navigation between views using roadmap steps.

Author: Priyank Kumar Jain

Company: Bristlecone

Created on: 10th September 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 Web Dynpro.

Table of Contents

Objective Exercise

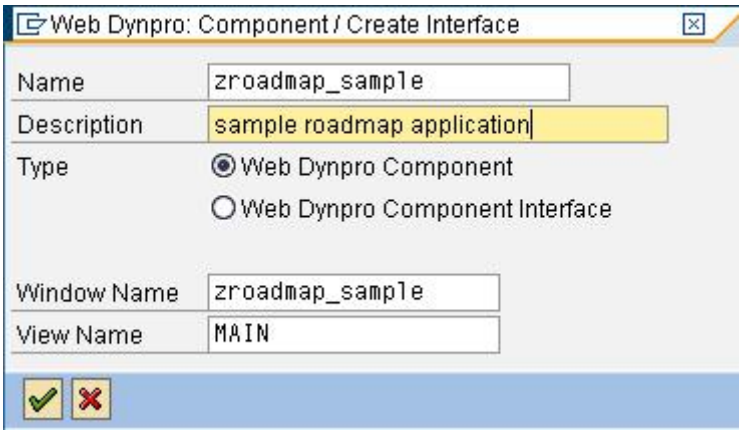
Create an application that uses a roadmap to navigate across four views. The first view will be a starting point for the application, the second one will accept the user input, the third will display the output as per the user's input and the fourth will be the last view to finish the process. Create an application with proper navigation using the Roadmap UI element in Web Dynpro ABAP.

Prerequisites

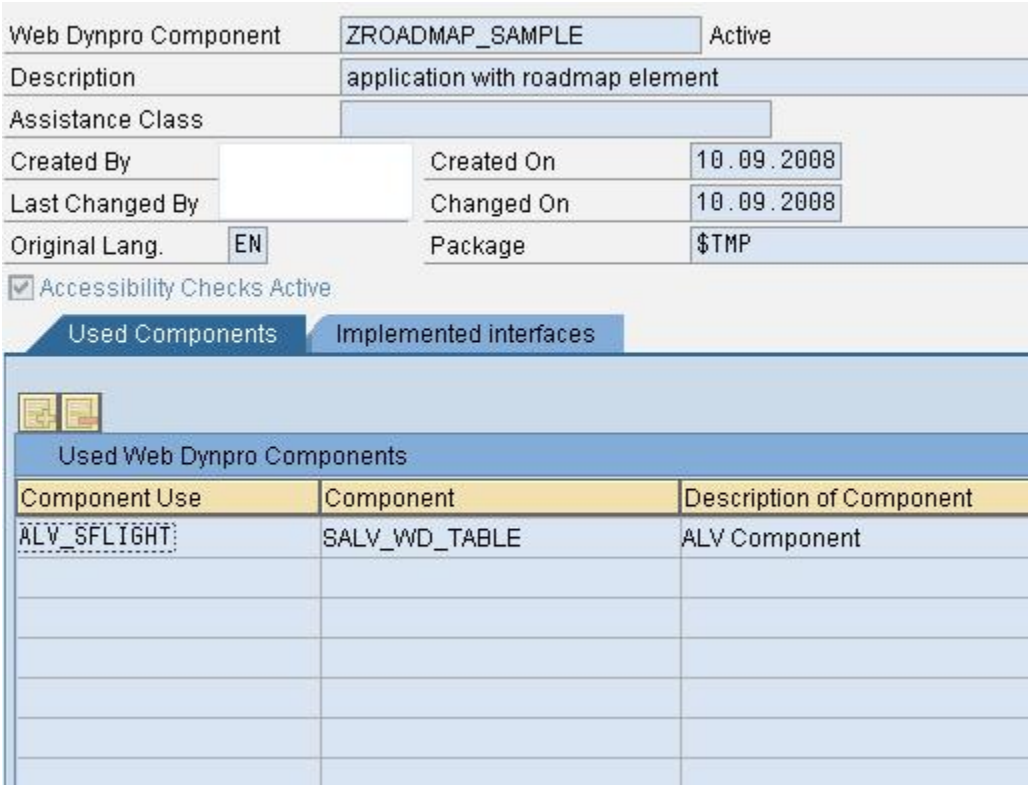
Basic knowledge of programming in ABAP and Web Dynpro for ABAP is required.

Create Web Dynpro component

To accomplish the objective of the exercise mentioned above, we will first create a new Web Dynpro component in SE80. Name it as ZROADMAP_SAMPLE.



Create a usage for the standard Web Dynpro ALV component SALV_WD_TABLE in the component just created. Name this usage as ALV_SFLIGHT.

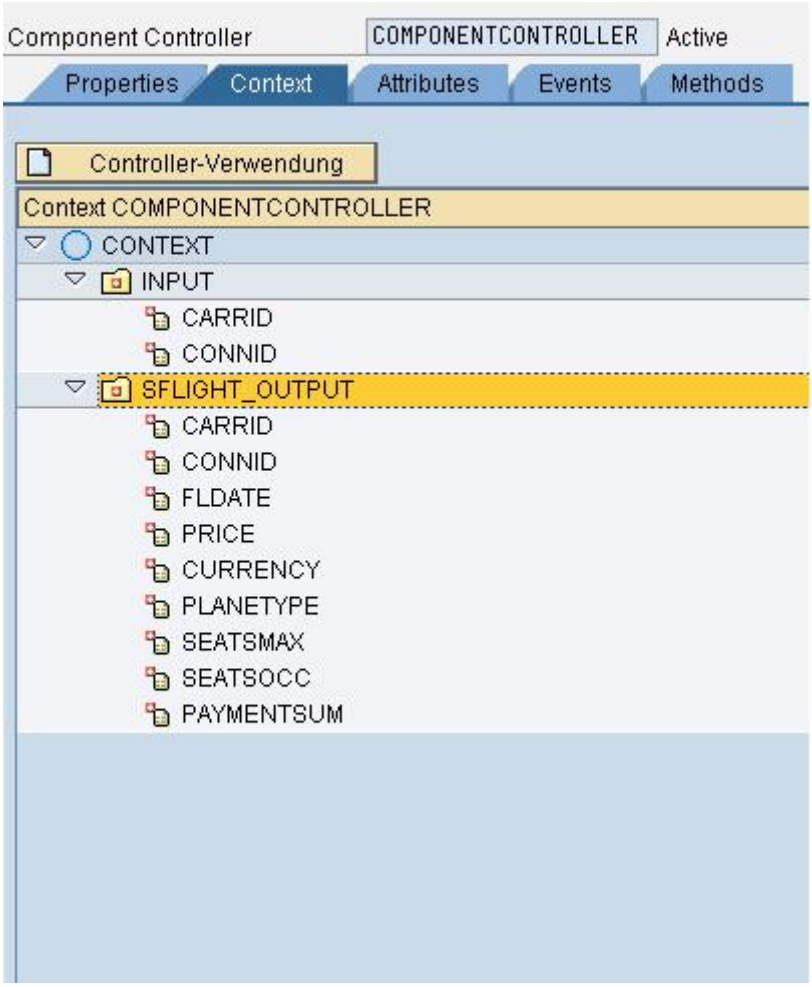


Create context attributes in component controller

Create a node in the context of the component controller. Name it as INPUT. Create two attributes CARRID and CONNID under this node. These are of the same type as the fields CARRID and CONNID of the SFLIGHT table. We will use these fields as the input to be taken from the user.

Create another node named SFLIGHT_OUTPUT. This node is of the type dictionary structure SFLIGHT. Select the attributes to be added under this node from the table SFLIGHT. This node will serve as the output node and will be bound to the ALV to display the result.

The component controller context should look like below.



We will also create a method in the component controller. This method will fetch the data based on the inputs given by the user. Create a method named GET_DATA in the component controller and put the following code in it.

```
METHOD get_data .
DATA lo_nd_input TYPE REF TO if_wd_context_node.

DATA :  lo_el_input TYPE REF TO if_wd_context_element,
        ls_input TYPE wd_this->element_input,
        lv_carrid TYPE wd_this->element_input-carrid,
        lv_connid TYPE wd_this->element_input-connid,
        itab TYPE TABLE OF sflight,
        ls_where(72) TYPE c,
        lt_where TYPE TABLE OF string.
```

```
    DATA : lo_nd_sflight_output TYPE REF TO if_wd_context_node,
           lo_el_sflight_output TYPE REF TO if_wd_context_element,
           ls_sflight_output TYPE wd_this->element_sflight_output.

  lo_nd_input = wd_context->get_child_node( name = wd_this->wdctx_input ).

*   get element via lead selection
  lo_el_input = lo_nd_input->get_element( ).

*   get single attribute
  lo_el_input->get_attribute(
    EXPORTING
      name =  `CARRID`
    IMPORTING
      value = lv_carrid ).

  lo_el_input->get_attribute(
    EXPORTING
      name =  `CONNID`
    IMPORTING
      value = lv_connid ).

**construct dynamic where clause
  IF lv_carrid IS NOT INITIAL.
    CONCATENATE 'CARRID = ' '''' lv_carrid '''' INTO ls_where.
    APPEND ls_where TO lt_where.
  ENDIF.

  IF lv_connid IS NOT INITIAL.
    CONCATENATE 'CONNID = ' '''' lv_connid '''' INTO ls_where.
    IF lt_where IS NOT INITIAL.
      CONCATENATE 'AND' ls_where INTO ls_where.
    ENDIF.
    APPEND ls_where TO lt_where.
  ENDIF.

  SELECT * FROM sflight INTO TABLE itab WHERE (lt_where).
  IF itab IS NOT INITIAL.
*         navigate from <CONTEXT> to <SFLIGHT_OUTPUT> via lead selection
    lo_nd_sflight_output = wd_context->get_child_node(
                        name = wd_this->wdctx_sflight_output ).

    lo_nd_sflight_output->bind_table( itab ).
  ENDIF.

ENDMETHOD.
```

Create Web Dynpro Views

We need to create views for our application. One view will contain the roadmap element with steps while others will be used for different steps in the application.

Create MAIN view

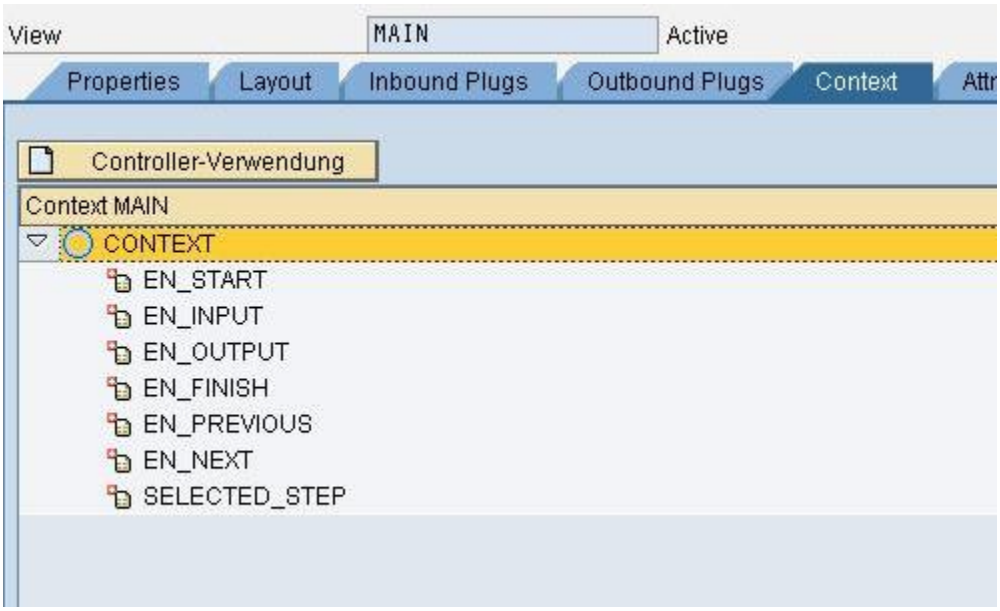
Create a view named MAIN. This view will contain a Roadmap element, Buttons for navigation and a container for the other views.

Create context attributes in MAIN view

Create the following attributes under the CONTEXT node in the context tab.

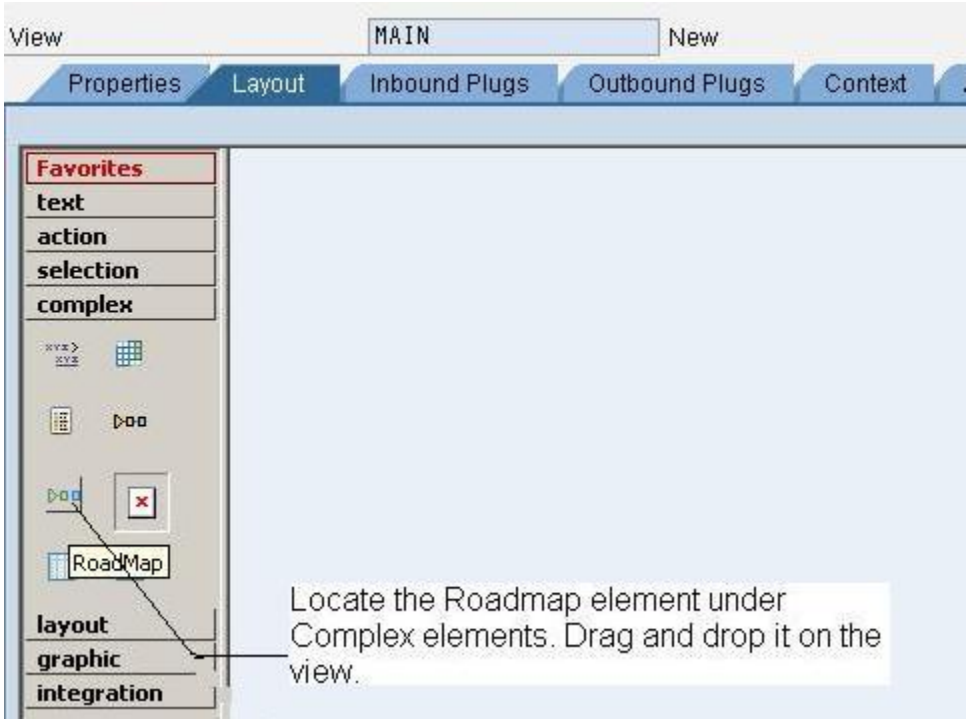
Attribute name	Type	Purpose
EN_START	WDY_BOOLEAN	Enable the START step
EN_INPUT	WDY_BOOLEAN	Enable INPUT step
EN_OUTPUT	WDY_BOOLEAN	Enable OUTPUT step
EN_FINISH	WDY_BOOLEAN	Enable FINISH step
EN_PREVIOUS	WDY_BOOLEAN	Enable PREVIOUS button
EN_NEXT	WDY_BOOLEAN	Enable NEXT button
SELECTED_STEP	STRING	Selected step of roadmap

The context should look like below.



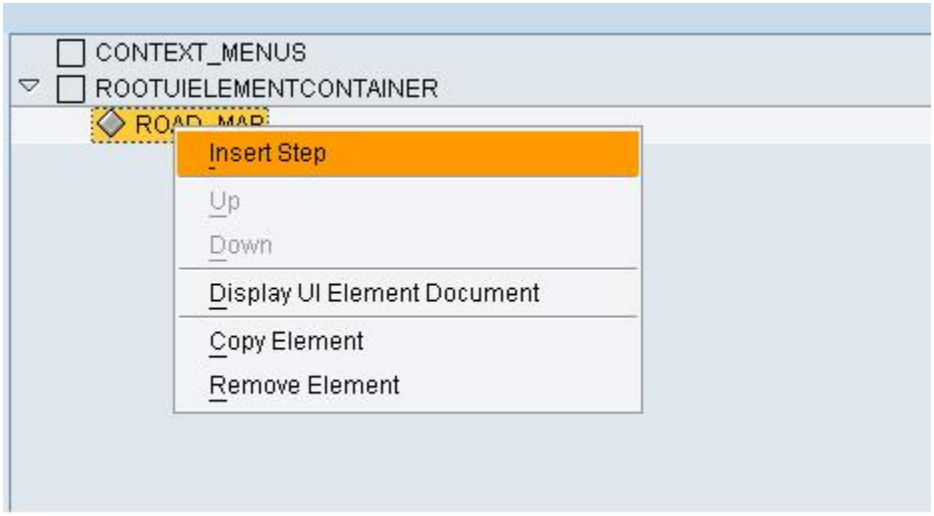
Define view layout

Place a Roadmap element on the MAIN view as shown below.

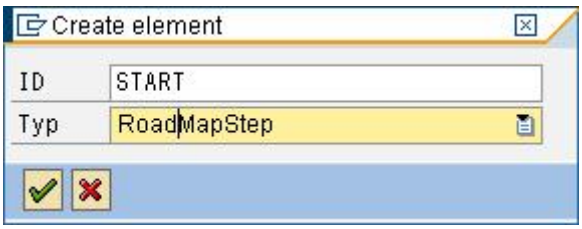


Now add four steps to the Roadmap element. Addition of one step is shown as an example below.

Add a roadmap step to the roadmap element.



The following window appears. Supply the ID as START and the type as RoadMapStep.

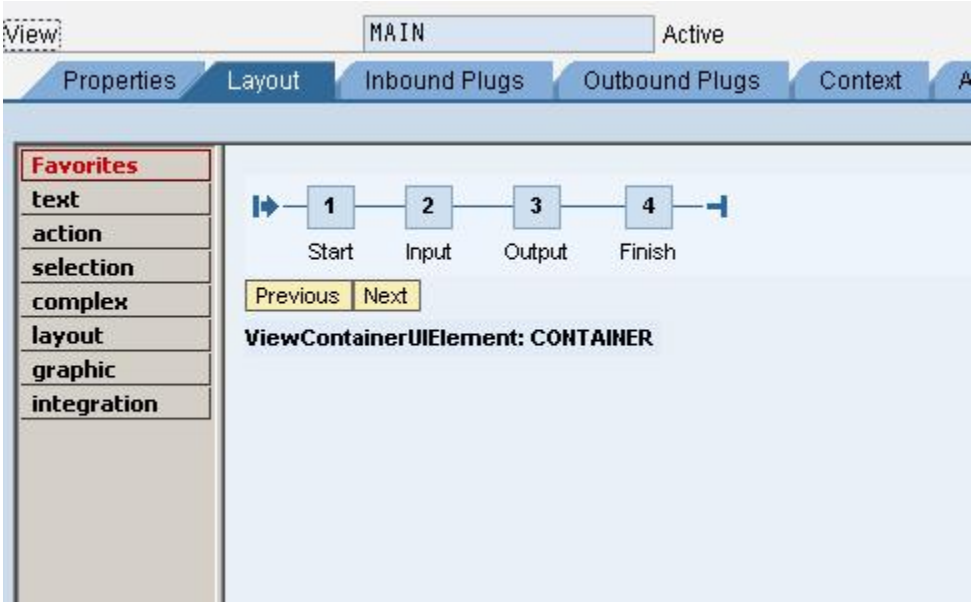


In the properties window, set the name of this step as '1', description as "Start". Bind the "Enabled" property to the context attribute EN_START that was created above.

Similarly, create three more steps for the roadmap. Name them as INPUT, OUTPUT and FINISH. Bind their 'enabled' property to the respective context attributes.

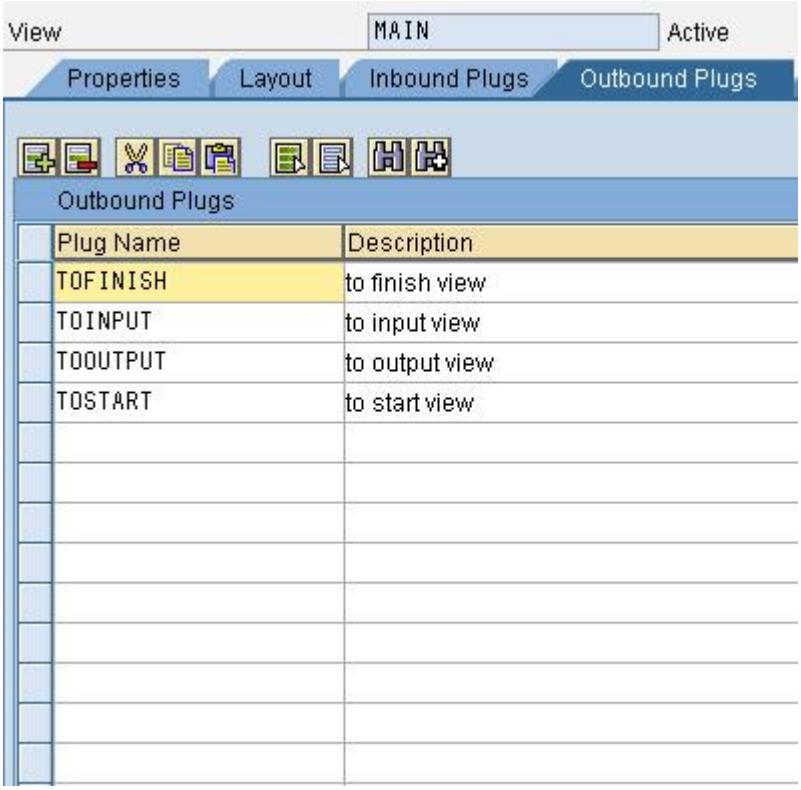
Place two buttons "Next" and "Previous" on the view for enabling navigation. Bind the 'enabled' property of these buttons to the respective context attribute created above.
Also place a viewcontainerUIelement named CONTAINER. This container will embed the other views.

The MAIN view should look like below.



Create outbound plugs

Create four outbound plugs, one for each roadmap step in the MAIN view. Name them as in the screenshot below.



Create Event handlers in MAIN view

We now need to create event handler methods for events of different elements of the MAIN view.

For the “Previous” button, create an action for the onAction event. Name this action as ‘PREVIOUS’. Similarly, create an action NEXT for the onAction event of the NEXT button.

For the Roadmap element, create an action named SELECT_STEP for the onSelect Event. Add an importing parameter called STEP to its event handler method.

We will come to the implementation of these event-handlers at a later stage in the tutorial.

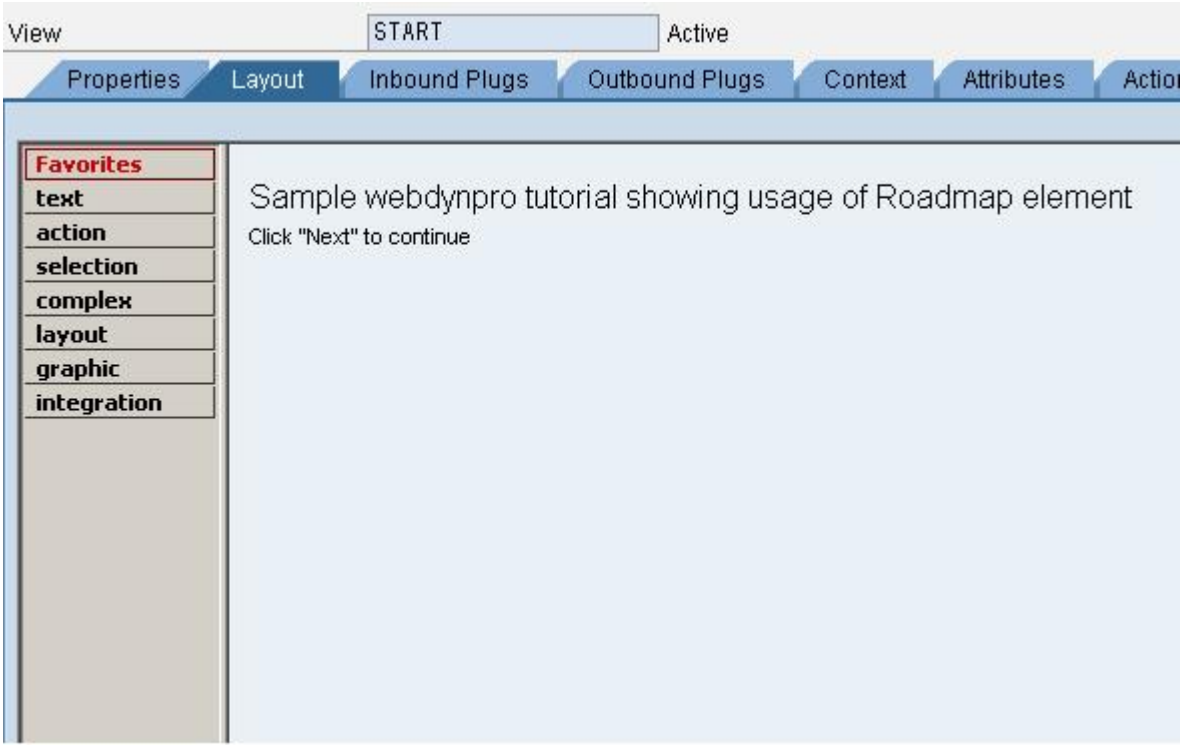
Create the remaining views

Create four more views in the Web Dynpro component, one corresponding to each of the four roadmap steps. For our exercise, we will use the data in the table SFLIGHT. There will be a START view which will serve as the starting point for our application. The INPUT view will accept a couple of fields as input from the user. The OUTPUT view will display the data retrieved from SFLIGHT table in an ALV grid. The FINISH view will be the last view in the sequence to inform that the application has ended.

The four views should look like below.

1) The START view.

Create a view named START. This is just a starting view for our example and contains some static text. Create an inbound plug named FROMMAIN in this view.

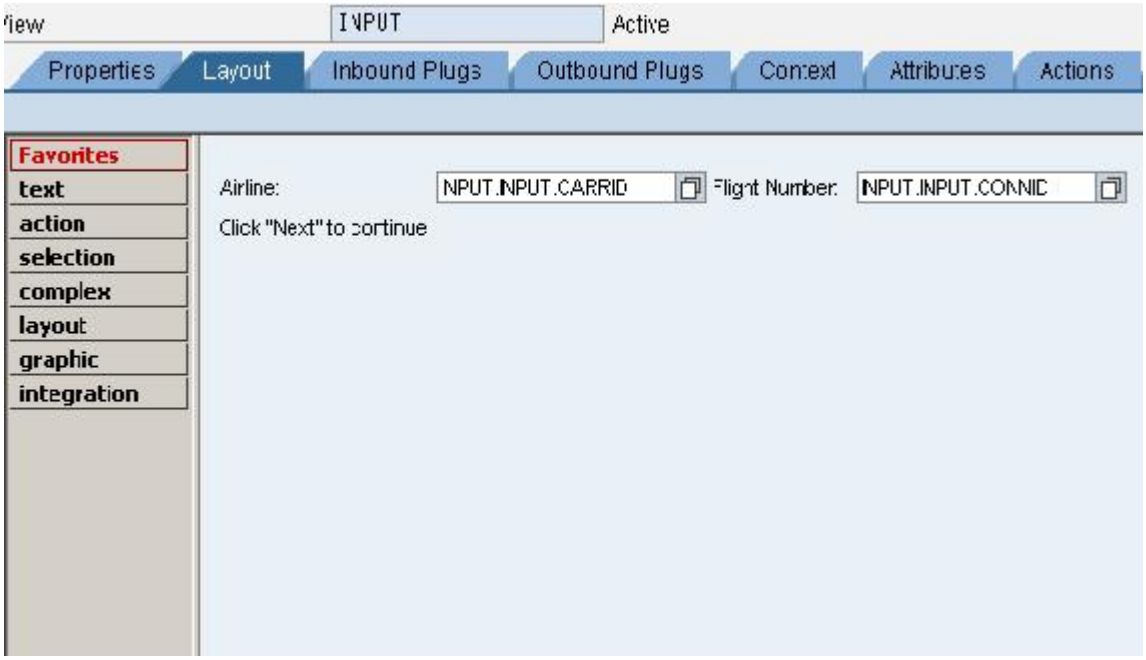


2) The INPUT view

Create a view named INPUT. This is the second view in the sequence and corresponds to the second step of the roadmap. In the context of this view, drag and drop the component controller node INPUT to the view controller context. This creates a node in the view controller context with mapping to the component controller node.

In the layout of this view, create two input fields for CARRID and CONNID attributes. Bind their “value” property to the respective attributes of the INPUT node.

The view should look like below.



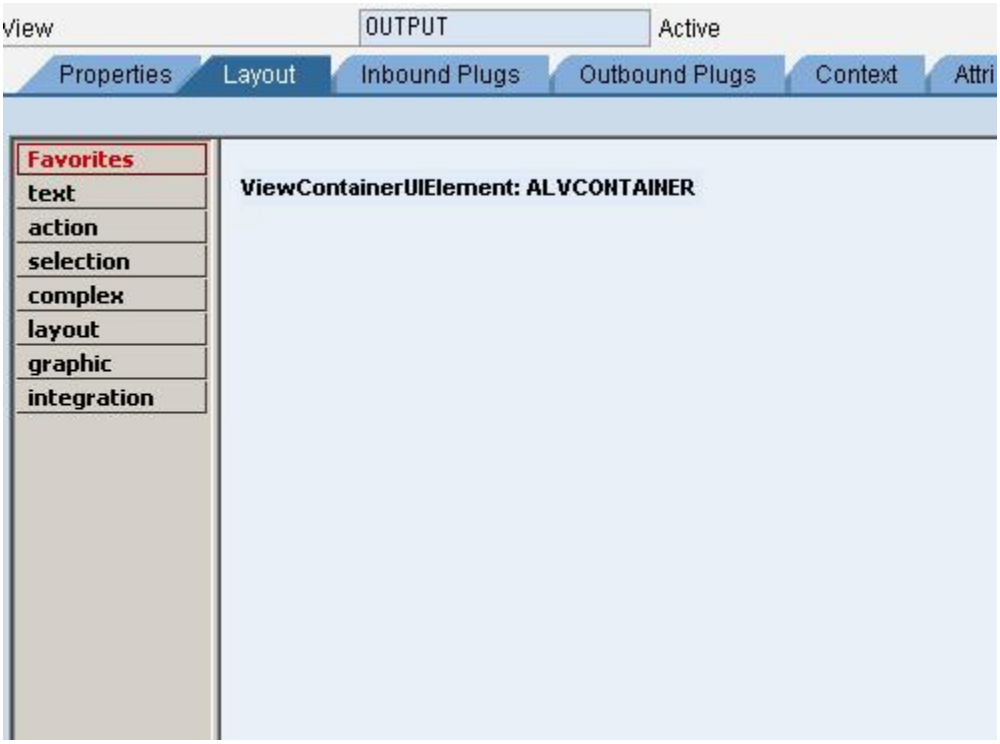
Also create an inbound plug named FROMMAIN in this view.

3) The OUTPUT view

This view will correspond to the third step of the roadmap. This will only display an ALV grid for displaying the data. Add a viewcontainerUIelement to the layout of this view.

In the context, drag and drop the SFLIGHT_OUTPUT node from the component controller to the view controller. This will create the same node in the view controller with mapping to the component controller node.

The view should look like below.



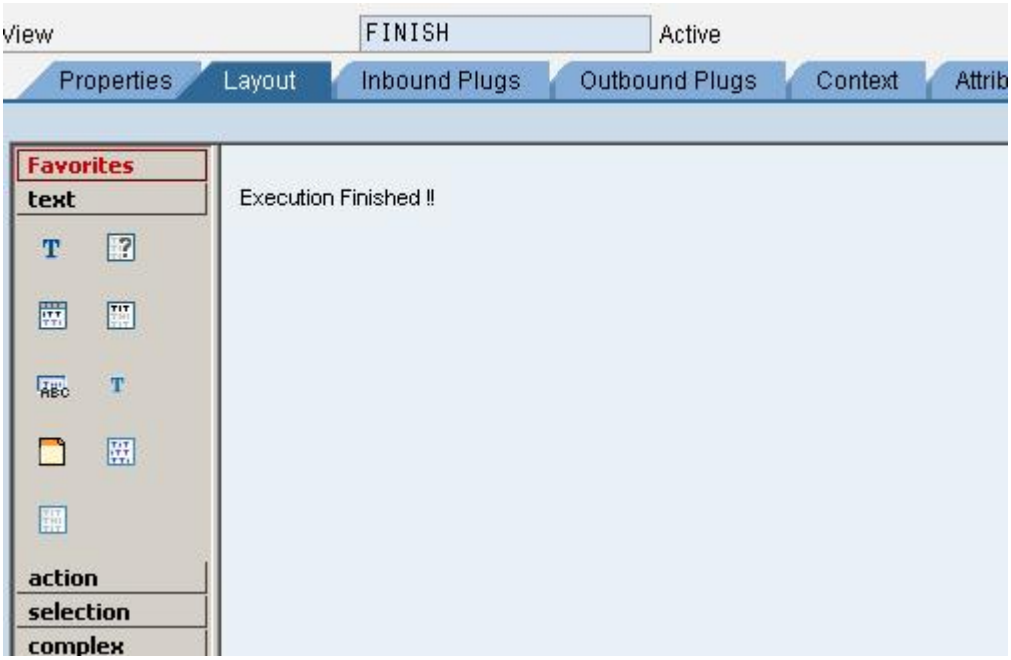
As for the preceding views, create an inbound plug named FROMMAIN in this view as well. This will create an event handler method with the name HANDLEFROMMAIN. Place the following code in this method. This method will retrieve the data using the component controller method created earlier.

```
METHOD handlefrommain .
  wd_comp_controller->get_data( ).
ENDMETHOD.
```

4) The FINISH view

This is the last view in the sequence. We will just put a text message in this view to inform the user that the application has finished executing.

The view will just have some static text and will look like below.



Create the inbound plug named FROMMAIN in this view too.

Implement Event handlers and methods of the MAIN view

Now we will write the code to implement the event handlers that we created in the MAIN view.

We created the following event handlers earlier.

ONACTIONNEXT

ONACTIONPREVIOUS

ONACTIONSELECT_STEP

In addition to these, create the following methods in the MAIN view.

ENABLE_FINISH_STEP – has one importing parameter ENABLE of type WDY_BOOLEAN

ENABLE_INPUT_STEP – has one importing parameter ENABLE of type WDY_BOOLEAN

ENABLE_NEXT_BUTTON – has one importing parameter ENABLE of type WDY_BOOLEAN

ENABLE_OUTPUT_STEP – has one importing parameter ENABLE of type WDY_BOOLEAN

ENABLE_PREV_BUTTON – has one importing parameter ENABLE of type WDY_BOOLEAN

ENABLE_START_STEP – has one importing parameter ENABLE of type WDY_BOOLEAN

SET_STEP – has one importing parameter STEP of type STRING

Following are the implementations of each of these methods and the event handlers.

Event Handlers – Implementation

ONACTIONNEXT – This handles the click of the “Next” button.

```
METHOD onactionnext .

  DATA lo_el_context TYPE REF TO if_wd_context_element.
  DATA ls_context TYPE wd_this->element_context.
  DATA lv_selected_step TYPE wd_this->element_context-selected_step.

*   get element via lead selection
  lo_el_context = wd_context->get_element( ).
*   get single attribute
  lo_el_context->get_attribute(
    EXPORTING
      name =  `SELECTED_STEP`
    IMPORTING
      value = lv_selected_step ).

  CASE lv_selected_step.

    WHEN 'START'.
      lo_el_context->set_attribute(
          EXPORTING
            name =  `SELECTED_STEP`
            value = 'INPUT' ).
      wd_this->set_step( step = 'INPUT' ).
      wd_this->enable_next_button( enable = 'X' ).
      wd_this->enable_prev_button( enable = 'X' ).
wd_this->fire_toinput_plg( ).
    WHEN 'INPUT'.
      lo_el_context->set_attribute(
          EXPORTING
            name =  `SELECTED_STEP`
            value = 'OUTPUT' ).

      wd_this->set_step( step = 'OUTPUT' ).
      wd_this->enable_next_button( enable = 'X' ).
      wd_this->enable_prev_button( enable = 'X' ).
      wd_this->fire_tooutput_plg( ).
    WHEN 'OUTPUT'.
      lo_el_context->set_attribute(
          EXPORTING
            name =  `SELECTED_STEP`
            value = 'FINISH' ).
      wd_this->set_step( step = 'FINISH' ).
      wd_this->enable_next_button( enable = '' ).
      wd_this->enable_prev_button( enable = 'X' ).
      wd_this->fire_tofinish_plg( ).
    WHEN 'FINISH'.
      wd_this->set_step( step = 'FINISH' ).
      wd_this->enable_next_button( enable = '' ).
      wd_this->enable_prev_button( enable = 'X' ).
  ENDCASE.
ENDMETHOD.
```

ONACTIONPREVIOUS – This handles the click of the “Previous” button.

```
METHOD onactionprevious .
  DATA lo_el_context TYPE REF TO if_wd_context_element.
  DATA ls_context TYPE wd_this->element_context.
  DATA lv_selected_step TYPE wd_this->element_context-selected_step.
*   get element via lead selection
  lo_el_context = wd_context->get_element( ).
*   get single attribute
  lo_el_context->get_attribute(
    EXPORTING
      name =  `SELECTED_STEP`
    IMPORTING
      value = lv_selected_step ).
  CASE lv_selected_step.

    WHEN 'FINISH'.
      lo_el_context->set_attribute(
          EXPORTING
            name =  `SELECTED_STEP`
            value = 'OUTPUT' ).
      wd_this->set_step( step = 'OUTPUT' ).
      wd_this->enable_next_button( enable = 'X' ).
      wd_this->enable_prev_button( enable = 'X' ).
      wd_this->fire_tooutput_plg( ).
    WHEN 'OUTPUT'.
      lo_el_context->set_attribute(
          EXPORTING
            name =  `SELECTED_STEP`
            value = 'INPUT' ).
      wd_this->set_step( step = 'INPUT' ).
      wd_this->enable_next_button( enable = 'X' ).
      wd_this->enable_prev_button( enable = 'X' ).
      wd_this->fire_toinput_plg( ).

    WHEN 'INPUT'.
      lo_el_context->set_attribute(
          EXPORTING
            name =  `SELECTED_STEP`
            value = 'START' ).
      wd_this->set_step( step = 'START' ).
      wd_this->enable_next_button( enable = 'X' ).
      wd_this->enable_prev_button( enable = '' ).
      wd_this->fire_tostart_plg( ).


    WHEN 'START'.
      lo_el_context->set_attribute(
          EXPORTING
            name =  `SELECTED_STEP`
            value = 'START' ).
      wd_this->set_step( step = 'START' ).
      wd_this->enable_next_button( enable = 'X' ).
      wd_this->enable_prev_button( enable = '' ).
      wd_this->fire_tostart_plg( ).
  ENDCASE.
ENDMETHOD.
```

ONACTIONSELECT_STEP – This handles the selection of a step on the Roadmap element.

```
METHOD onactionselect_step .
  DATA lo_el_context TYPE REF TO if_wd_context_element.
* get element via lead selection
  lo_el_context = wd_context->get_element( ).
* @TODO handle not set lead selection
  CASE step.
    WHEN 'START'.
    lo_el_context->Set_attribute(
      EXPORTING
        name =  `SELECTED_STEP`
        value = 'START' ).

      wd_this->enable_prev_button( enable = '' ).
      wd_this->enable_next_button( enable = 'X' ).
      wd_this->set_step( step = 'START' ).
      wd_this->fire_tostart_plg( ).
    WHEN 'INPUT'.
    lo_el_context->Set_attribute(
      EXPORTING
        name =  `SELECTED_STEP`
        value = 'INPUT' ).

      wd_this->enable_prev_button( enable = 'X' ).
      wd_this->enable_next_button( enable = 'X' ).
      wd_this->set_step( step = 'INPUT' ).
      wd_this->fire_toinput_plg( ).
    WHEN 'OUTPUT'.
          lo_el_context->Set_attribute(
      EXPORTING
        name =  `SELECTED_STEP`
        value = 'OUTPUT' ).
      wd_this->enable_prev_button( enable = 'X' ).
      wd_this->enable_next_button( enable = 'X' ).
      wd_this->set_step( step = 'OUTPUT' ).
      wd_this->fire_tooutput_plg( ).
    WHEN 'FINISH'.
    lo_el_context->Set_attribute(
      EXPORTING
        name =  `SELECTED_STEP`
        value = 'FINISH' ).

      wd_this->enable_prev_button( enable = 'X' ).
      wd_this->enable_next_button( enable = '' ).
      wd_this->set_step( step = 'FINISH' ).
      wd_this->fire_tofinish_plg( ).
  ENDCASE.
ENDMETHOD.
```

Methods – Implementation

ENABLE_FINISH_STEP – To enable/disable the “Finish” step

```
method ENABLE_FINISH_STEP .
      DATA lo_el_context TYPE REF TO if_wd_context_element.
*   get element via lead selection
    lo_el_context = wd_context->get_element( ).

    lo_el_context->set_attribute(
      EXPORTING
        name =  `EN_FINISH`
        value = enable ).
endmethod.
```

ENABLE_INPUT_STEP – To enable/disable the “Input” step

```
Method ENABLE_INPUT_STEP .
    DATA lo_el_context TYPE REF TO if_wd_context_element.
*   get element via lead selection
    lo_el_context = wd_context->get_element( ).

    lo_el_context->set_attribute(
      EXPORTING
        name =  `EN_INPUT`
        value = enable ).
endmethod.
```

ENABLE_OUTPUT_STEP – To enable/disable the “Output” step

```
method ENABLE_OUTPUT_STEP .
      DATA lo_el_context TYPE REF TO if_wd_context_element.
*   get element via lead selection
    lo_el_context = wd_context->get_element( ).

    lo_el_context->set_attribute(
      EXPORTING
        name =  `EN_OUTPUT`
        value = enable ).
endmethod.
```

ENABLE_START_STEP – To enable/disable the “Start” step

```
method ENABLE_START_STEP .

    DATA lo_el_context TYPE REF TO if_wd_context_element.
*   get element via lead selection
    lo_el_context = wd_context->get_element( ).

    lo_el_context->set_attribute(
      EXPORTING
        name =  `EN_START`
        value = enable ).
endmethod.
```

ENABLE_NEXT_BUTTON – To enable/disable the “Next” button

```
method ENABLE_NEXT_BUTTON .
    DATA lo_el_context TYPE REF TO if_wd_context_element.
```

```
    DATA ls_context TYPE wd_this->element_context.
    DATA lv_en_next TYPE wd_this->element_context-en_next.

*   get element via lead selection
    lo_el_context = wd_context->get_element( ).

    lo_el_context->set_attribute(
      EXPORTING
        name =  `EN_NEXT`
        value = enable ).
endmethod.
```

ENABLE_PREV_BUTTON – To enable/disable the “Previous” button

```
method ENABLE_PREV_BUTTON .

    DATA lo_el_context TYPE REF TO if_wd_context_element.
    DATA ls_context TYPE wd_this->element_context.
    DATA lv_en_previous TYPE wd_this->element_context-en_previous.
*   get element via lead selection
    lo_el_context = wd_context->get_element( ).

    lo_el_context->set_attribute(
      EXPORTING
        name =  `EN_PREVIOUS`
        value = enable ).
endmethod.
```

SET_STEP – Enable/disable other steps when a step is selected

```
METHOD set_step .
  CASE step.
    WHEN 'START'.
      wd_this->enable_start_step( enable = 'X' ).
      wd_this->enable_input_step( enable = '' ).
      wd_this->enable_output_step( enable = '' ).
      wd_this->enable_finish_step( enable = '' ).
    WHEN 'INPUT'.
      wd_this->enable_start_step( enable = 'X' ).
      wd_this->enable_input_step( enable = 'X' ).
      wd_this->enable_output_step( enable = '' ).
      wd_this->enable_finish_step( enable = '' ).

    WHEN 'OUTPUT'.
      wd_this->enable_start_step( enable = 'X' ).
      wd_this->enable_input_step( enable = 'X' ).
      wd_this->enable_output_step( enable = 'X' ).
      wd_this->enable_finish_step( enable = '' ).
    WHEN 'FINISH'.
      wd_this->enable_start_step( enable = 'X' ).
      wd_this->enable_input_step( enable = 'X' ).
      wd_this->enable_output_step( enable = 'X' ).
      wd_this->enable_finish_step( enable = 'X' ).
  ENDCASE.
ENDMETHOD.
```

Write the following code in the wddomodifyview method of the MAIN View.

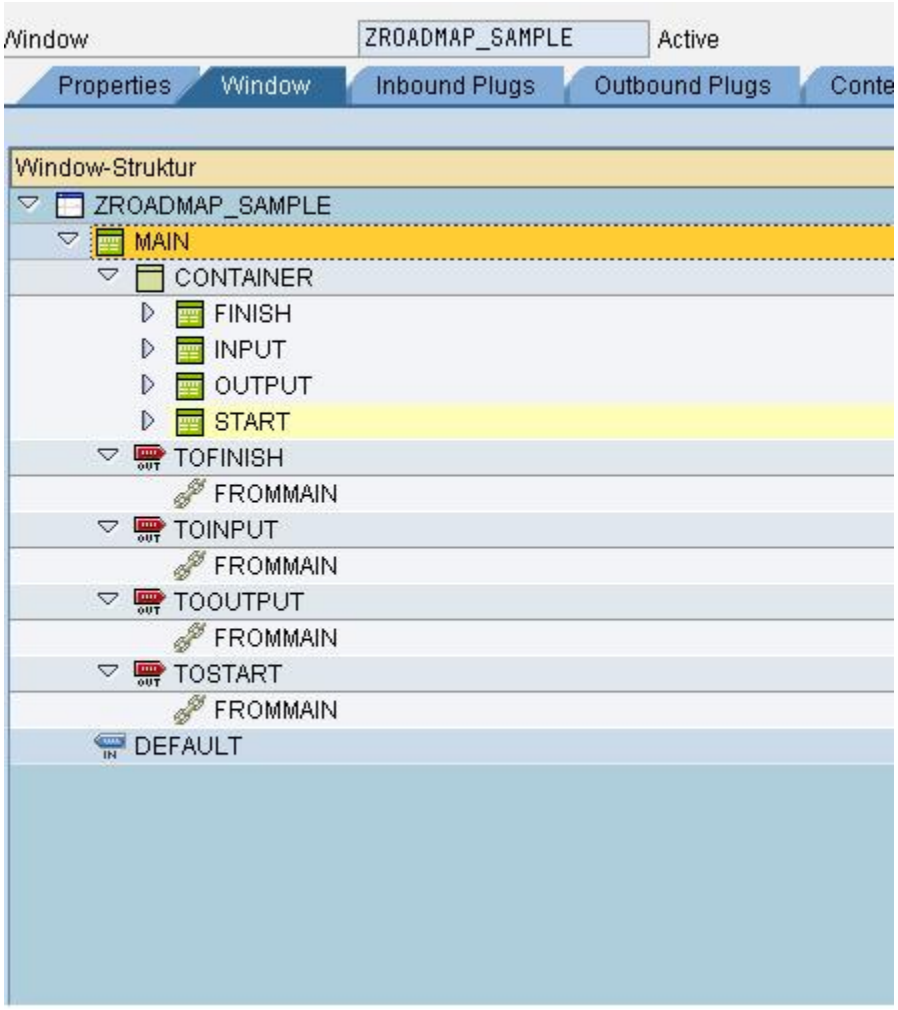
WDDOMODIFYVIEW

```
METHOD wddomodifyview .
    DATA lo_el_context TYPE REF TO if_wd_context_element.
    DATA ls_context TYPE wd_this->element_context.
    DATA lv_en_start TYPE wd_this->element_context-en_start.
    DATA lv_en_input TYPE wd_this->element_context-en_input.
    DATA lv_en_output TYPE wd_this->element_context-en_output.
    DATA lv_en_finish TYPE wd_this->element_context-en_finish.
    DATA lv_en_previous TYPE wd_this->element_context-en_previous.
    DATA lv_en_next TYPE wd_this->element_context-en_next.
    DATA lv_selected_step TYPE wd_this->element_context-selected_step.

  IF first_time = abap_true.
*     get element via lead selection
    lo_el_context = wd_context->get_element( ).

wd_this->enable_start_step( enable = 'X' ).
wd_this->enable_input_step( enable = '' ).
wd_this->enable_output_step( enable = '' ).
wd_this->enable_finish_step( enable = '' ).
wd_this->enable_prev_button( enable = '' ).
wd_this->enable_next_button( enable = 'X' ).

**set selected step as START
    lo_el_context->set_attribute(
  EXPORTING
    name =  `SELECTED_STEP`
    value = 'START' ).
  ENDIF.
ENDMETHOD.
```

Embed views in window

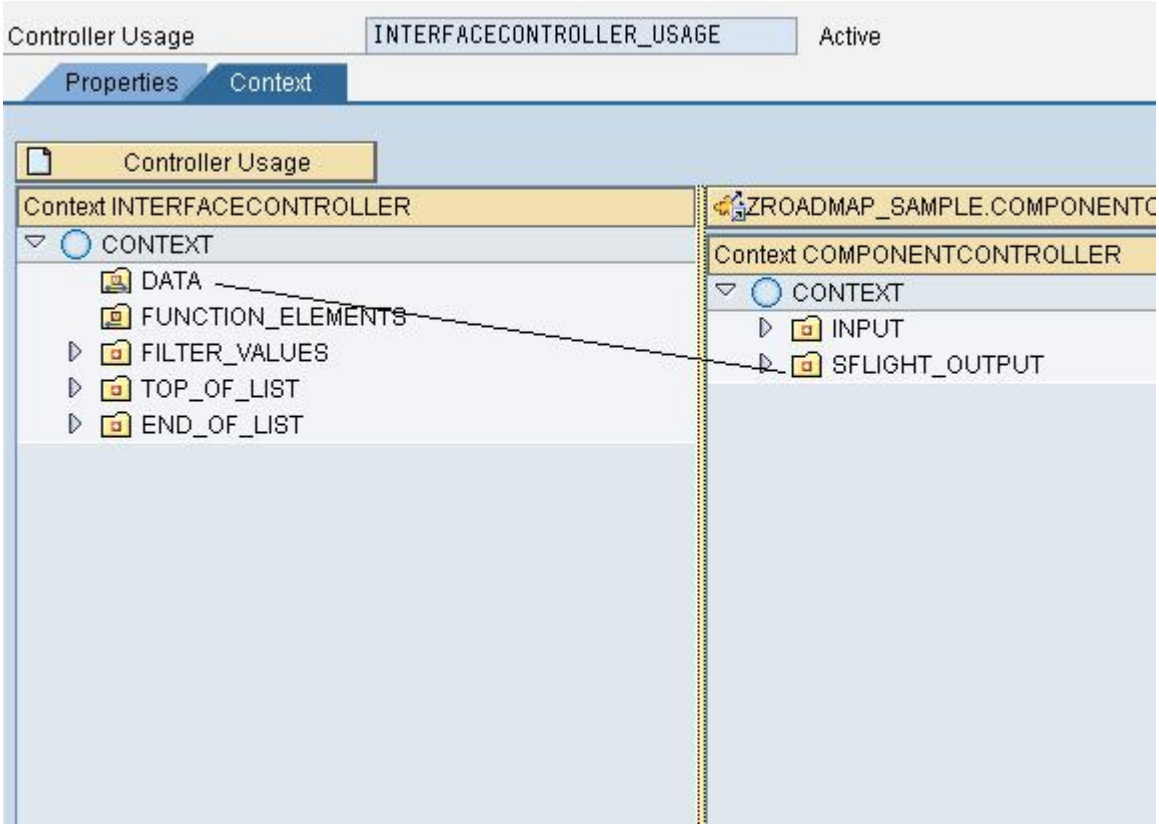
The next step is to embed the views in the default window of the component. Embed the four views – START, INPUT, OUTPUT and FINISH in the container of the MAIN view. Also, define the navigation link for the four outbound plugs of the MAIN view to the respective views. Remember, we created one inbound plug in each of the four views.

The window will look like below.



Map ALV interface controller node

Go to Component usages->ALV_SFLIGHT->INTERFACECONTROLLER_USAGE. Click on the “Create controller usage” button. Here, drag the SFLIGHT_OUTPUT node of the component controller and drop it on the DATA node of the interface controller. A double-sided arrow will appear on the DATA node.

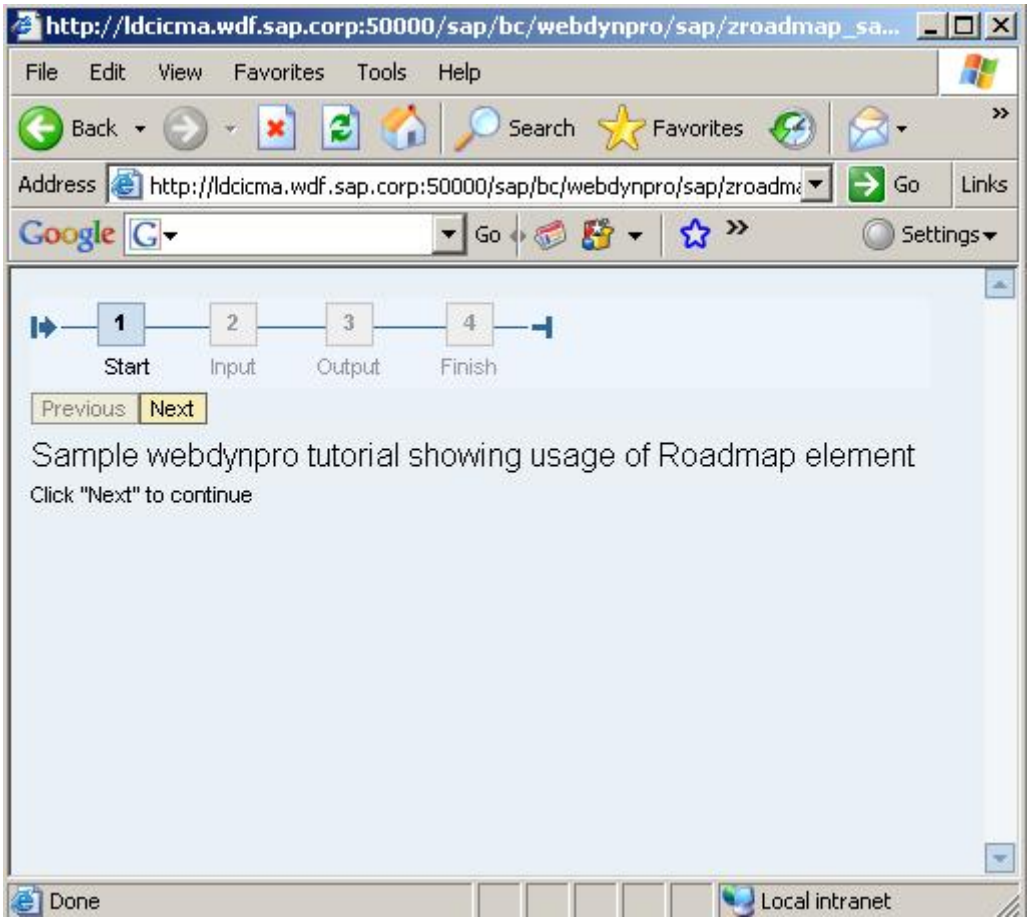


Create and test the Web Dynpro application

We are almost done. Create a Web Dynpro application and save it. Test the application and see the results of what you have done so far.

Result

The output of the application should look like the one below. You will observe that navigation to a preceding step is possible by either clicking on the steps of the roadmap directly or via the previous and next buttons. Navigation to a succeeding step is only possible via the NEXT button. While you are on the START or FINISH steps, the previous and next buttons remain disabled respectively.



Dynamic Programming

It is also possible to work with the Roadmap dynamically. This means that you can add or remove steps from the roadmap at the runtime. There are standard Web Dynpro classes that enable you to achieve this functionality. The reference of a roadmap itself can be got using the CL_WD_ROAD_MAP class. The ADD_STEP method can be used to add a new step while the REMOVE_STEP method can be used to remove an existing step.

The class for the roadmap step is CL_WD_ROAD_MAP_STEP. There are methods that allow you to change the name and description of the roadmap step at runtime.

You can explore these classes and methods if you want to achieve the functionality of the Roadmap element dynamically.

Related Content

Read more about the Roadmap element in the SAP library

For more information, visit the User Interface Technology homepage.

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.