Dynamic Context Menus in Web Dynpro ABAP.

Applies to:

Web Dynpro for ABAP, NW 7.0. For more information, visit the Web Dynpro ABAP homepage.

Summary

In this tutorial I want to explain how to define context menus dynamically in Web Dynpro ABAP. The application consists of two screens. On the first screen, the user can choose the flight by selecting an airline (CARRID) and a connection id (CONNID). On the second screen, we display two tables: The first one shows a flight list and the second one displays the booking details for a flight. Therefore you should have some basic knowledge in context mapping, as well as in firing navigation plugs.

Author: Patrick Johann

Company: SAP

Created on: 24.February 2009

Author Bio

Patrick Johann is in the first year of Vocational Training as an IT Specialist – Application Development at SAP. At the moment he is working in a SAP Net Weaver Product Management User Interaction Team an is doing Web Dynpro for ABAP.

Table of Content

Exercise Objectives:

- using and designing context menus dynamically

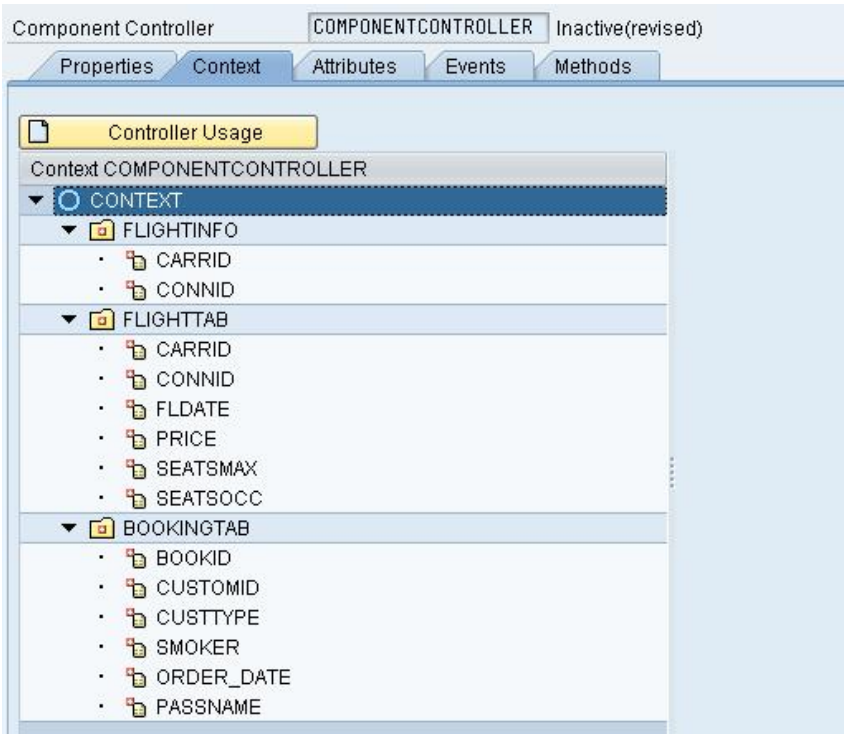
Process at runtime:

The user starts on the first view and chooses a flight connection. He or she can select an airline (CARRID) and a connection id from the selected airline (CONNID). By clicking the 'details' button the second view opens. On the second view, the selection is displayed again. Additionally there is a table of flights, filled with data that correspond with the user's selection. Moreover the table contains details like flight date, airfare, maximum capacity or occupied capacity. To get a table of flights for another connection, there is a button to navigate back. This navigation will be fired too when the user clicks the corresponding action item in the dynamically defined context menu. It appears by right-clicking the mouse in the details view. Furthermore it is possible to get another table which contains booking details (e.g. customer data, booking date...) for each single flight in the flights table also with the context menu. However it could be that the flight table is empty (because of a wrong carrier for example) which means that in this situation the booking table couldn't get any data. In this case the menu item is unable to click. In addition the content of the booking table gets deleted on navigation back, besides the old data still exists, although the user has entered a new selection. Otherwise he or she must update the booking table with new data to overwrite the old one. The booking table is invisible and only appears when the context menu item is clicked. The booking table shall only be an addition to the flight table, that's the reason why both tables are displayed in trays. Trays have the advantage to maximize and minimize manually by the user, so he only sees the details he needs.

Procedure – Part A: **Template**

1. Start the Object Navigator (transaction code SE80). Create a new Web Dynpro component (suggested name: *'Z##_CONTEXT_MENU'*), one window and two views (suggested names: *'INPUT_VIEW'* and *'OUTPUT_VIEW'*).

2. Navigate to COMPONENTCONTROLLER. Create a context node named FLIGHTINFO (cardinality *1:1*) type of the dictionary structure *SFLIGHT* with the attributes CARRID and CONNID. At runtime this node enables the selection of a flight connection to display the flights according to the user's input. The flight data is going to be available in a table that is named FLIGHTTAB (cardinality *0:n SFLIGHT*). Add this as another node. Later our objective will be to fill a second table with booking information via a context menu item, so we need another node called BOOKINGTAB (cardinality *0:n SBOOK*). Add some attributes to each node (look at screenshot).



3. Now you have to map the nodes to the context of the view controllers. For INPUT_VIEW map FLIGHTINFO, because the user only have to do a selection on this view. For OUTPUT_VIEW map all three nodes.

4. Click on the *Layout* tab of the INPUT_VIEW. Use the Code Wizard to implement a form with two input fields for the attributes *Carrid* and *Connid*. Bind the FLIGHTINFO node to the form. After this, change the layout of the ROOTUIELEMENTCONTAINER. Choose *Matrix Layout* for the container and for each single element *MatrixHeadData* as *Layout Data*. Now every element appears in a new row. Furthermore create a button (suggested name: *'BUT_DETAILS'*) and give it a meaningful label.

5. Design the OUTPUT_VIEW. On this view all details get displayed. First of all repeat the last step, because we want to show what the input of the user was. However the input should be unable to change. Set the property *readOnly* for both input fields. Then create two trays in the ROOTUIELEMENTCONTAINER (suggested names: *'TRAY_FLIGHTS'* and *'TRAY_BOOKINGS'*). The advantage of trays is that the user can open and close them manually and only has the look on the details he or she needs.

6. Modify the layout and choose *Matrix Layout* and *MatrixHeadData* for the single elements again. Use the Code Wizard to implement the two tables in the trays. Don't forget to match the right tables to the right trays. Set the property *visibleRowCount* to 15 for FLIGHTTAB. Add a button (suggested name: *'BUT_BACK'*) to your layout.

7. The layout now should be completed. However before we start to code the necessary methods and events we'll have to apply navigation links from INPUT_VIEW to OUTPUT_VIEW. Do this as before. Create the events *'GO'* and *'BACK'* for the buttons to fire the Outbound Plugs.

8. Now it is possible to navigate between the two views in your Web Dynpro. If the application is executed, the tables will be empty. For this, it is necessary to create some methods to fill the tables. This has to be done in the component controller, because we need references to the context nodes. Create a method called 'FLIGHTTAB_FILL' and another one called 'BOOKINGTAB_FILL'.

    a. First we are coding the FLIGHTTAB_FILL method. Read the context node FLIGHTINFO (use the Code Wizard). Then do an array fetch to read the data from the dictionary table SFLIGHT by the keys Carrid and Connid from your FLIGHTINFO node. Export the data to a local declared table type of the static method ***elements_flighttab( )*** of your interface controller. Navigate to the FLIGHTTAB node with the method ***get_child_node( )*** and write

the data from the local declared table into the node. This happens by calling the Method ***bind_table( )***.

b. For BOOKINGTAB_FILL you nearly have to do the same things as in the method before. Read the context node FLIGHTTAB. Then do an array fetch on the dictionary table SBOOK by using the keys Carrid, Connid and Fldate. Store the data in a local declared table type of ***elements_bookingtab( )***. Navigate with ***get_child_node( )*** to the BOOKINGTAB node and write the data into it with ***bind_table( )***.

9. Open the OUTPUT_VIEW and create an event called 'DETAILS'. Change the method behind the event and implement a call of the BOOKINGTAB_FILL method. The code wizard is able to generate the necessary coding. Therefore you don't have to write any coding for this step.

10. There should be run two things in the background if the user clicks on the details. First, the navigation plug should be fired and then the FLIGHTTAB_FILL method fills the table in the OUTPUT_VIEW. Use the HANDLEFROM_INPUT_VIEW method in the OUTPUT_VIEW to implement the method as the step before.

11. Solve the following problem: When the booking table is filled and you navigate back to the first view to do a new selection and you navigate forward again, the booking table is still visible with the old data. You have to clean up the table by navigate back. Look at method ONACTIONBACK. Get a local reference to the component controller and its node BOOKINGTAB. Overwrite it with a local declared table, which is `initial.` Use the ***bind_table( )*** method.

12. Finally you have to modify the hook method WDDOMODIFYVIEW of the OUTPUT_VIEW. We want to set the tray of the booking table only visible when it has data. Reference to the BOOKINGTAB and check if it is `initial`. Use the method ***set_visible( )*** and the values *'01'* for `none` and *'02'* for `visible`.

onactiondetails:

```
METHOD onactiondetails .

  DATA lo_componentcontroller TYPE REF TO ig_componentcontroller .

  lo_componentcontroller = wd_this->get_componentcontroller_ctr( ).
  lo_componentcontroller->bookingtab_fill( ).

ENDMETHOD.
```

onactionback:

```
METHOD onactionback .

  DATA lo_nd_init   TYPE REF TO if_wd_context_node.
  DATA it_init      TYPE        ig_componentcontroller=>elements_bookingtab.

  lo_nd_init = wd_context->get_child_node( name = wd_this->wdctx_bookingtab ).
  lo_nd_init->bind_table( it_init ).

  wd_this->fire_to_input_view_plg( ).


ENDMETHOD.
```

handlefrom_input_view:

```
METHOD handlefrom_input_view .
  wd_comp_controller->flighttab_fill(  ).

ENDMETHOD.
```

WDDOMODIFYVIEW:

```
METHOD wddomodifyview .

  DATA lo_ui_root   TYPE REF TO if_wd_view_element.
  DATA lo_container TYPE REF TO cl_wd_uielement_container.
  DATA lo_node      TYPE REF TO if_wd_context_node.
  DATA lo_element   TYPE REF TO if_wd_context_element.

  lo_ui_root = view->get_element( id = 'TRAY_BOOKING' ).
  lo_container ?= lo_ui_root.
  lo_node = wd_context->get_child_node( name = wd_this->wdctx_bookingtab ).

  IF lo_node IS INITIAL.
  ENDIF.

    lo_element = lo_node->get_element(  ).

  IF lo_element IS INITIAL.
    lo_container->set_visible( '01' ).
  ELSE.
     lo_container->set_visible( '02' ).
  ENDIF.
ENDMETHOD.
```

flighttab_fill:

```
METHOD flighttab_fill .

  DATA lo_nd_flightinfo TYPE REF TO if_wd_context_node.
  DATA lo_el_flightinfo TYPE REF TO if_wd_context_element.
  DATA ls_flightinfo TYPE wd_this->element_flightinfo.
  DATA lt_flighttab TYPE if_componentcontroller=>elements_flighttab.

  lo_nd_flightinfo = wd_context->get_child_node( name = 'FLIGHTINFO' ).

  lo_el_flightinfo = lo_nd_flightinfo->get_element(  ).

* get all declared attributes
  lo_el_flightinfo->get_static_attributes(
    IMPORTING
      static_attributes = ls_flightinfo ).

  SELECT * FROM sflight INTO TABLE lt_flighttab WHERE
    carrid = ls_flightinfo-carrid AND
    connid = ls_flightinfo-connid.

  lo_nd_flightinfo = wd_context->get_child_node( name = 'FLIGHTTAB' ).
  lo_nd_flightinfo->bind_table( lt_flighttab ).
```

```
ENDMETHOD.
```

bookingtab_fill:

```
METHOD bookingtab_fill .
  DATA lo_nd_flighttab TYPE REF TO if_wd_context_node.
  DATA lo_el_flighttab TYPE REF TO if_wd_context_element.
  DATA ls_flighttab    TYPE        wd_this->element_flighttab.
  DATA it_bookingtab   TYPE        if_componentcontroller->elements_bookingtab.

  lo_nd_flighttab = wd_context->get_child_node( name = wd_this->wdctx_flighttab ).

*   get element via lead selection
  lo_el_flighttab = lo_nd_flighttab->get_element(  ).

*   get all declared attributes
  lo_el_flighttab->get_static_attributes(
    IMPORTING
      static_attributes = ls_flighttab ).

  SELECT * FROM SBOOK INTO TABLE lt_bookingtab WHERE
  carrid = ls_flighttab-carrid AND
  connid = ls_flighttab-connid AND
  fldate = ls_flighttab-fldate.

  lo_nd_flighttab = wd_context->get_child_node( name = 'BOOKINGTAB' ).
  lo_nd_flighttab->bind_table( it_bookingtab ).

ENDMETHOD.
```

Procedure – Part B: Context Menu

(For source code look at the abap-coding below)

1. Edit the hook-method WDDOONCONTEXTMENU in the OUTPUT_VIEW.

2. Start with the declaration of local attributes. Create a local reference attribute to the class ***cl_wd_menu*** and also two reference attributes to the class ***cl_wd_menu_action_item***. With these attributes we design our context menu dynamically and display it later at runtime by handing it over to the returning parameter *menu*.

3. Now you have to design the action-items for your context menu. Call the static method ***new_menu_action_item( )*** from the class ***cl_wd_menu_action_item***. Supply the parameters *ID*, *On_Action* (*'DETAILS'* and *'BACK'*) and *Text*. Assign the method call to a declared action-item attribute. Repeat the whole step for the second item.

4. The declaration is now complete. In addition create the context menu (***cl_wd_menu=>new_menu( )***) and add the items to the created context menu with the method ***add_item( )***.

5. Display the menu by handing over the local menu to the returning parameter *menu*.

6. If the user tries to display the *Bookings* table when the *Flights* table does not contain any entries, a runtime error will occur. Therefore you must check that the *Flights* table is not empty. If it is, the BOOKINGTAB_FILL event of the context menu must be grayed out.

    a. Reference the context node (***if_wd_context_node***) as well as the context element FLIGHTTAB(***if_wd_context_element***), by navigating from context

to the FLIGHTTAB node via lead selection. Call the method ***get_element( ),*** which reads the content of the table.

b. Use an IF/ELSE construction to check if the table is `initial`. If it is, the optional parameter `enabled` of the action item element, which triggers the *'DETAILS'* event, has to be set to `abap_false`. It is an optional parameter because `abap_true` is the default value and it only has to be set if you want to disable an element. If the table has data, use the ELSE statement to set the action item to `enabled`.

7. Save and activate all objects. Create a Web Dynpro application and test it.

WDDOONCONTEXTMENU (solution):

```
METHOD wddooncontextmenu.

  DATA:
        lo_menu                 TYPE REF TO    cl_wd_menu,
        lo_menu_item            TYPE REF TO    cl_wd_menu_action_item,
        lo_menu_item2           TYPE REF TO    cl_wd_menu_action_item,
        lo_nd_flighttab         TYPE REF TO    if_wd_context_node,
        lo_el_flighttab         TYPE REF TO    if_wd_context_element.

  lo_nd_flighttab = wd_context->get_child_node(
   name = wd_this->wdctx_flighttab ).

  lo_el_flighttab = lo_nd_flighttab->get_element(  ).

* create menu
IF lo_el_flighttab IS INITIAL.
    lo_menu_item = cl_wd_menu_action_item=>new_menu_action_item(
         id = 'ACTION'
         on_action = 'DETAILS'
         text = 'show/update bookings'
          enabled = abap_false
         ).
ELSE.
   lo_menu_item = cl_wd_menu_action_item=>new_menu_action_item(
         id = 'ACTION'
         on_action = 'DETAILS'
         text = 'show/update bookings'
         ).
ENDIF.
    lo_menu_item2 = cl_wd_menu_action_item=>new_menu_action_item(
         id = 'BACKACTION'
         on_action = 'BACK'
         text = 'Zurück zur Eingabe'
         ).

* add items to context menu
    lo_menu = cl_wd_menu=>new_menu( ).
    lo_menu->add_item( lo_menu_item ).
    lo_menu->add_item( lo_menu_item2 ).

* display the menu
    menu = lo_menu.
```

```
ENDMETHOD.
```

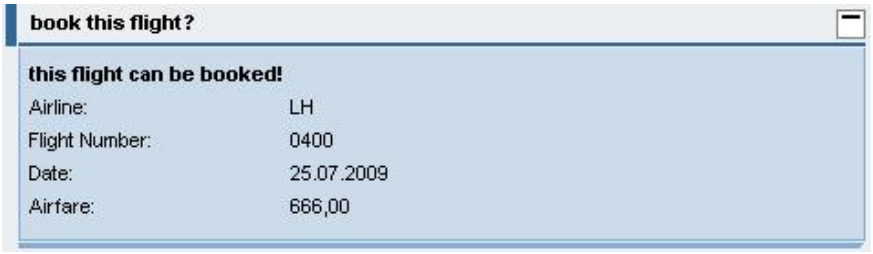
Procedure – Part C: Additional elements (optional)

(For source code look at the abap-coding below)

Process at runtime:

The user gets another context menu entry named 'book this flight'. If he or she books a flight, an arriving message tells you that the booking has been successful. On the OUTPUT_VIEW, a third tray is visible when the selected flight is bookable. This tray shows a short summary of the flight again (e.g. airline, connection, price…). The tray and the context menu item are bookable and only enabled when the flight date is in the future and there are still seats available in the economy class.

1. Change the layout of the OUTPUT_VIEW. Insert a new tray (suggested name: '*TRAY_BOOKFLIGHT*') with *MatrixHeadData*. Now add four text-views and label them. Bind the texts to the node attributes *Carrid*, *Connid*, *Fldate* and *Price*. The result should looks like this:



2. Navigate to the WDDOMODIFYVIEW method in your view controller. Here you have to change an UI-element dynamically, again like in part A12. Reference to the new tray an use the method **set_visible( )** to hide the tray. Change the visibility to the value '02' (visible) if the selected flight is in future and have blank seats. For this, you have to reference to the FLIGHTTAB node and check if the *fldate* is bigger than *sy-datum* (the actual date) and *seatsmax* is bigger than *seatsocc*. But you have to handle another problem, because this application will have a runtime error when the flight table is empty and the hook method tries to get the properties of this table. You have to verify that your reference to the context element `is not initial`. Only if it is the case, the method shall do the check which has been described previously.

3. The user gets a runtime error when he or she has entered an invalid flight connection, because the bindings of the texts views of the short summary can't get data from the FLIGHTTAB. You have to check if the FLIGHTTAB is filled with data. In case of an empty FLIGHTTAB you can fire the navigation link to the INPUT_VIEW, so the user has to enter a flight connection again. Change the handlefrom_input_view method in the OUPUT_VIEW and reference to the context node element FLIGHTTAB after the call of the COMPONENTCONTROLLER method FLIGHTTAB_FILL, which you already have done before. If the FLIGHTTAB element is `initial`, fire the outbound-plug.

4. Before we can implement our new action item in the context menu we have to create a new event called 'BOOKING'. The purpose of the event is to report a success message if the flight has been booked. Use the code wizard, implement a success message and send a significant message text. Delete the following generated coding and complete your abap statement with a dot.

5. Finally we have to call the new event in the context menu. Navigate to the hook method WDDOONCONTEXTMENU and change it. Declare a new action item as you have done it in part B. Check again, if the flight date is after the actual date and there is at least one seat left in economy class. You have to check, if the local element `is not initial`, too. It would be the same problem as in step 2, so check this first. An `If/Else`-construction controls the creation of the action item and set it `enabled`, when the flight is bookable or unable if it is not. Add the new item with the method add_item( ) only, when the first check has been passed. That means you have to do this check for adding the new item again because the user gets a runtime error if the context menu wants to display an item which has never been created.

WDDOMODIFYVIEW (additional source code):

```
  lo_ui_root2 = view->get_element( id = 'TRAY_BOOKFLIGHT' ).
  lo_container2 ?= lo_ui_root2.
  lo_node2 = wd_context->get_child_node( name = wd_this->wdctx_flighttab ).
  lo_element2 = lo_node2->get_element(  ).

IF lo_element2 IS NOT INITIAL.

 lo_element2->get_static_attributes(
      IMPORTING
        static_attributes = ls_flighttab ).

  IF ls_flighttab-fldate > sy-datum AND
     ls_flighttab-seatsmax > ls_flighttab-seatsocc.
    lo_container2->set_visible( '02' ).
  ELSE.
     lo_container2->set_visible( '01' ).
  ENDIF.

ENDIF.
```

handlefrom_input_view (additional source code):

```
  DATA:       lo_nd_flighttab     TYPE REF TO if_wd_context_node,
              lo_el_flighttab     TYPE REF TO if_wd_context_element.

  lo_nd_flighttab = wd_context->get_child_node( name = 'FLIGHTTAB' ).
  lo_el_flighttab = lo_nd_flighttab->get_element( ).

  IF lo_el_flighttab IS INITIAL.
    wd_this->fire_to_output_view_plg(
    ).
  ENDIF.
```

ONACTIONBOOKING:

```
METHOD onactionbooking .

*   get message manager
  DATA lo_api_controller     TYPE REF TO if_wd_controller.
  DATA lo_message_manager    TYPE REF TO if_wd_message_manager.

  lo_api_controller ?= wd_this->wd_get_api( ).

  CALL METHOD lo_api_controller->get_message_manager
    RECEIVING
      message_manager = lo_message_manager.

*   report message
  CALL METHOD lo_message_manager->report_success
    EXPORTING
      message_text = 'your booking has been successful!'.

ENDMETHOD.
```

WDDOONCONTEXTMENU (additional source code):

```
If lo_flighttab IS NOT INITIAL.
  lo_flighttab->get_static_attributes(
      IMPORTING
        static_attributes = ls_flighttab ).
ENDIF.

IF lo_flighttab IS NOT INITIAL.
  IF ls_flighttab-fldate > sy-datum AND
    ls_flighttab-seatsmax > ls_flighttab-seatsocc.

    lo_menu_item2 = cl_wd_menu_action_item=>new_menu_action_item(
          id = 'BOOK'
          on_action ='BOOKING'
          text = 'book this flight'
    ).
    ELSE.
     lo_menu_item2 = cl_wd_menu_action_item=>new_menu_action_item(
         id = 'BOOK'
         on_action ='BOOKING'
         text = 'book this flight'
         enabled = abap_false
   ).
    ENDIF.
ENDIF.

IF lo_flighttab IS NOT INITIAL.
    lo_menu->add_item( lo_menu_item2 ).
  ENDIF.
```

Related Contents

For more information, visit the Web Dynpro ABAP homepage.

For more information, visit the User Interface Technology homepage.

Copyright