ADF Code Corner

104. How to show a confirmation dialog on panel tab selection

twitter.com/adfcodecorner

Abstract:

A requirement posted on the Oracle JDeveloper forum on OTN was to intercept the user selection of a tab panel to display a confirmation dialog if the current transaction is dirty or when business logic requires it. Because panel tab selection can be suppressed on the client side only, the solution explained in this article uses both, JavaScript and server side Java.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
04-DEC-2012

Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

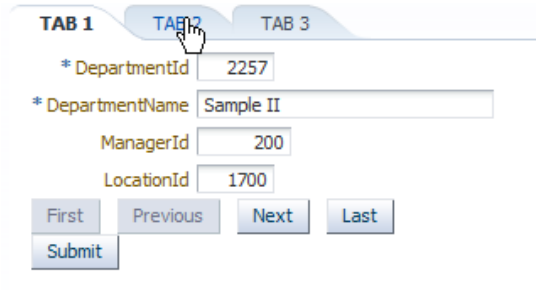
Introduction

The ADF Faces panel tab component, `af:panelTabbed`, uses the `af:showDetailItem` component for rendering the individual tabs. Each of this tabs, when selected by a user, raises a disclosure event that you, as the developer, can listen to on the client using JavaScript and the server using Java.

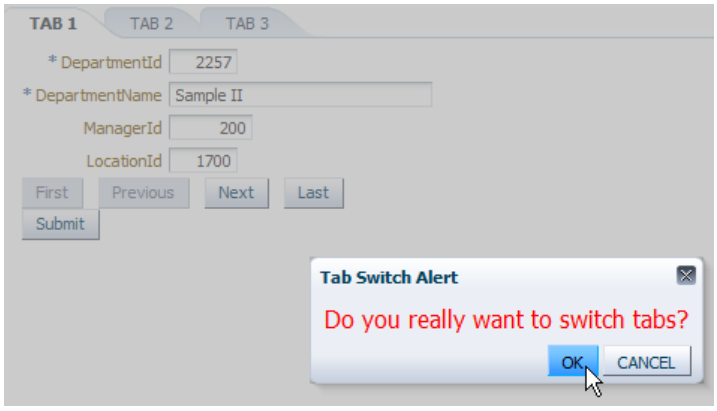
To implement a solution that intercepts the user tab selection to conditionally display a dialog to confirm the selection or stay on the current selected tab, JavaScript needs to be used to cancel the select event. The confirmation dialog is then launched using JavaScript, for when the launch condition is accessible on the client, or Java for server side conditions, like transaction state.

This article explains a solution that could be used both ways, client side and server side, simply by changing code comments in the sample. The default implementation uses server side Java to launch the confirmation dialog and, if confirmed, to switch tabs.

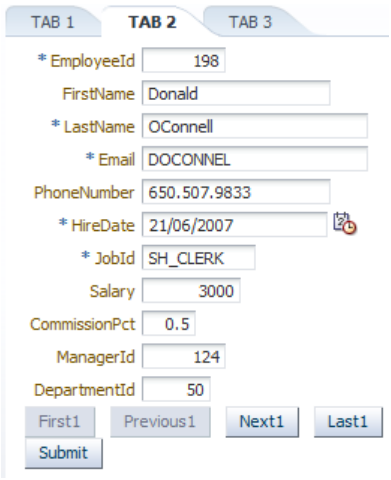
The images below show how the sample works. The sample has three tabs. The first and the second tab show ADF bound data. The third tab is empty. When you click on a non-current tab, the tab disclosure event is intercepted and cancelled. A server event is queued to evaluate the dialog launcg condition.



If the dialog needs to be launched, an af:popup is invoked from the server using JavaScript for the user to confirm or cancel navigation.



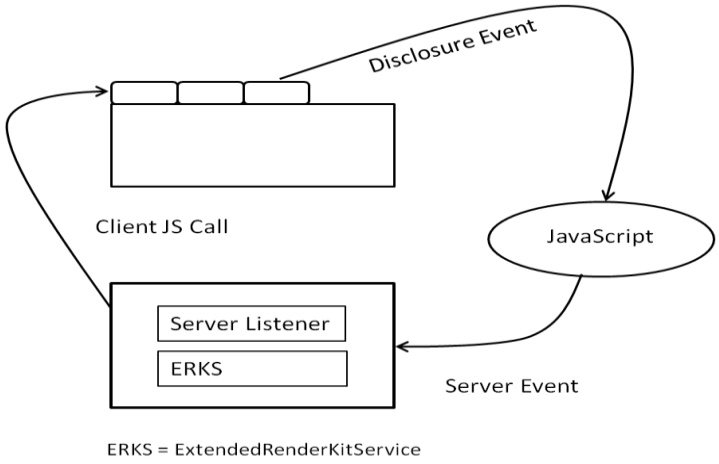
If the user confirms navigation, the non-current tab the user clicked on is selected.



Note that the message displayed in the popup can be changed to dynamically contain context information using Expression Language. So don't take this sample as final and look for what your application requirements are for this.

Implementation

As shown in the image below, ADF Faces raises the tab disclosure event on the browser client. Custom JavaScript is used to dispatch the new tab selection to either client-side (outlined in sample) or server side logic (implemented in sample). Note that two disclosure events are raised by the framework when a user selects a tab. The first disclosure event is by the current tab that becomes un-selected. The disclosed state of this component is "true". The second event is by the tab that the user clicked on. The disclosed state is "false". The sample uses the same function to handle the disclosure events of all tabs. The function looks for "false" as the disclosed state to get a handle to the tab (`af:showDetailItem`) the user wants to navigate to.



The disclosure event is dispatched by an `af:serverListener` tag. A managed bean method is invoked on the server for the application to evaluate whether or not a confirmation dialog needs to be displayed.

If a dialog needs to be displayed then this is handled through the `ExtendedRenderKitService` class invoking JavaScript on the client. Though you could use the `RichPopup` Java API to launch the dialog, a client attribute is needed to be set on the dialog, for which you require JavaScript, thus, the use of the `ExtendedRenderKitService` class.

Sample Implementation

To be able to respond to the tab disclosure event, each `af:showDetailItem` must be configured with an `af:clientListener` tag as shown in the image below. The client listener calls a custom JavaScript function that saves the client Id of the tab the user clicked on to switch the focus to.

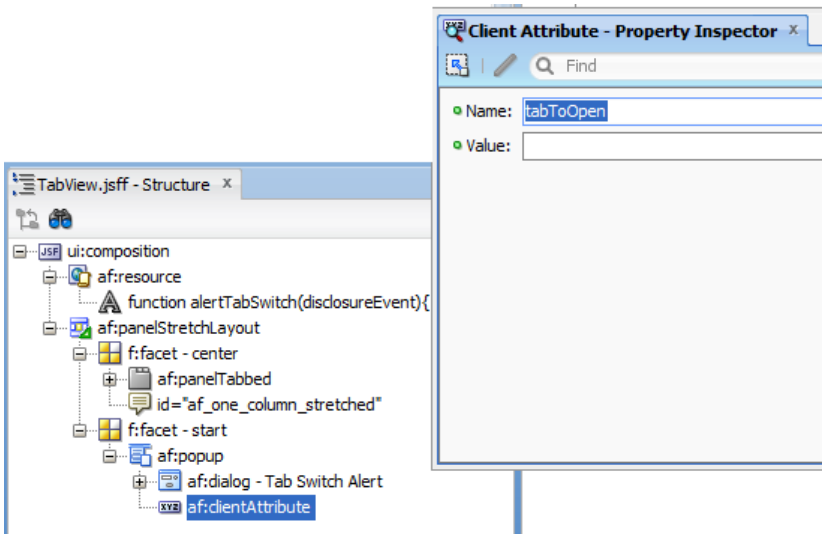
```
function alertTabSwitch(disclosureEvent){
  var tab = disclosureEvent.getSource();
  if(tab.getDisclosed()==false){
  /*
   * Uncomment the code below if the decision of whether or not
   * the popup should display is made on the client
   */

   // var popup = tab.findComponent('p1');
   // popup.show();
   // popup.setProperty("tabToOpen",tab.getClientId());

   /*
    * comment the code lines below if the decision of whether
    * or not the popup should display is made on the client
    */
```

```
        //create a custom event
        AdfCustomEvent.queue(tab,"checkDialogOpenCondition");

        //!!! don't comment the code line below. This is always required
        //no matter if the check for launching the dialog is made on the
        //client or the server

        disclosureEvent.cancel();
    }
}
```

The function contains code for launching a confirmation dialog based on a client decision (commented out) and server side decision (active in the sample). In either way, the tab selection event is cancelled and the tab is selected programmatically if the user confirms the tab selection.

The `af:ServerListener` configuration added to each `af:showDetailItem` component is shown in the image below.

Also shown in the image above, each `af:showDetailItem` contains an `af:serverListener` tag, for the client to call server-side code to evaluate whether or not to launch the confirmation dialog.



The af:popup component that displays the confirmation uses a custom client attribute to hold the client Id of the tab to open in case the user confirms the tab-switching. This information is passed in by JavaScript (see image above).

The handlePopupOkCancel JavaScript function is called by the confirmation dialog to close the popup with or without switching the tab focus.

```
function handlePopupOkCancel(actionEvent){
  var popupButton = actionEvent.getSource();
  var butPressed = popupButton.getProperty('popupAction');
  var dialog = popupButton.getParent();
  var popup  = dialog.getParent();
  if(butPressed == 'OK'){
    var tabToFindAndFocusOnString = popup.getProperty("tabToOpen");
    //look for the client Id (thus findComponent() can be used)
    if(tabToFindAndFocusOnString.length > 0){
     var tab =
     AdfPage.PAGE.findComponentByAbsoluteId(tabToFindAndFocusOnString);
      tab.setDisclosed(true);
      actionEvent.cancel();
      popup.hide();
    }
  }
  else{
    //close popup and stay on page
    actionEvent.cancel();
    popup.hide();
   }
}
```

The function is called from the OK, CANCEL command button on the confirmation dialog. The buttons are custom af:commandButtons that use an af:clientListener to call the JS function to close the dialog.

The af:popup component is defined as shown below:

```
<af:popup childCreation="deferred" autoCancel="disabled" id="p1"
          clientComponent="true" contentDelivery="immediate"
          binding="#{backingBeanScope.dialogHandlerBean.popupP1}">
  …
</af:popup>
```

The popup component JSF binding to the managed bean is used in the server listener code to access the popup client Id before launching the popup through the ExtendedRenderKitService. The code of this managed bean method is shown below

```
/**
 * Managed bean called from the client to check whether user panel tab
```

```
 * switch is acceptable or if there exist reason to ask the user if
 * he/she really wants to leave the current tab. The sample contains
 * client side code in its TabView.jsff page that would allow you to
 * perform the dialog open decision on the client side (e.g based on
 * the outcome of a client side verification
 *
*/
public class DialogHandlerBean {
 //JSF component reference to the popup to open. The managed bean is in
 //backing bean scope so that component references are okay
 private RichPopup popupP1;
 public DialogHandlerBean() {}

 //the af:serverListener defined on each af:showDetailItem component
 //in TabView.jsff calls the server when the user clicks on a disclosed
 //tab. The information contained in the call contains a reference to
 //the RichShowDetailComponent. This is automatically done by the

 af:serverListener referencing the
 //client side tab instance
 public void onCheckDialogOpen(ClientEvent clientEvent) {

 //do we need to challenge the user ?
 boolean tabSwitchOk = false;

 //get the tab that has been pressed to navigate to
 RichShowDetailItem tab =
          (RichShowDetailItem) clientEvent.getComponent();


/* ****YOUR TODO ************** */
  //Check if tab switch is ok or if the user needs to be asked. For
  //example, the ControllerContext class gives you access to the
  //ViewPortContext that exposes
  //an API to check if the transaction is dirty ....
  //tabSwitchOk = <your checking code here>
/* ****END YOUR TODO *********** */


 //the Trinidad ExtendedRenderKitService allows you to write JavaScript
 //command to the client with the response issued in return to the
 //af:serverListener call

  FacesContext fctx = FacesContext.getCurrentInstance();
  ExtendedRenderKitService erks =
    Service.getRenderKitService(fctx, ExtendedRenderKitService.class);
```

```
  //if tab switch is false, show a popup dialog for the user to decide
  //whether to navigate off the current tab or stay (e.g. to provide
  //additional information)

  if (tabSwitchOk != true) {
     String tabId = tab.getClientId();
     //compose the JS that accesses the client side popup instance and
     //shows it.
     //A client side property is set to the af:clientAttribute defined
     //on the af:popup component to pass the tab Id to navigate to the
     //command button in the dialog
     String showDialogScript =
            "var popup = AdfPage.PAGE.findComponentByAbsoluteId('" +
             popupP1.getClientId() + "'); popup.show();" +
            "popup.setProperty('tabToOpen','" + tabId + "')";
             erks.addScript(fctx, showDialogScript);
     }
     else{
      String noDialogScript = "var tab =
        AdfPage.PAGE.findComponentByAbsoluteId('"+tab.getClientId()+
        "');" tab.setDisclosed(true);";
      erks.addScript(fctx, noDialogScript);
    }
  }

  public void setPopupP1(RichPopup popupP1) {
    this.popupP1 = popupP1;
  }

  public RichPopup getPopupP1() {
    return popupP1;
  }
}
```

Download

The Oracle JDeveloper 11g R2 (11.1.2.3) sample for this article can be downloaded as sample #104 from the ADF Code Corner website

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

You need to configure the database connection to point to a local database with the HR schema enabled.

RELATED DOCOMENTATION

⊠	
⊠	
⊠	