ADF Code Corner

108. How-to launch a popup upon rendering of a page fragment in a region using JSF 2

twitter.com/adfcodecorner

Abstract:

A common requirement in Oracle ADF is to launch a popup dialog when a page fragment is rendered in a region.

In JDeveloper 11g R1, the best option was to use a hidden text field with its value referencing a managed bean method that then launches the popup from Java.

In JDeveloper 11g R2 and JDeveloper 12c, another option becomes available, which is the use of JSF 2 system events.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
21-AUG-2013

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

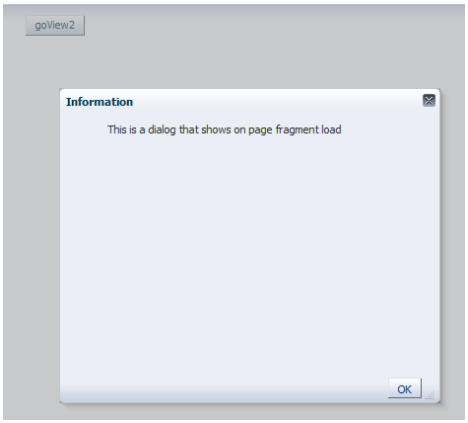
Launching a popup upon rendering a page fragment in a region is a frequent requirement among ADF application developers.

In JDeveloper 11g R1, and JavaServer Faces 1.2, the only reliable option is to use a hidden output text field that has its value property referencing a managed bean method, which then launches the popup.

With the new JavaServer Faces system events in JSF 2, there now is a better approach available for Oracle JDeveloper 11g R2 and JDeveloper 12c:

https://javaserverfaces.java.net/nonav/docs/2.1/vdldocs/facelets/f/event.html

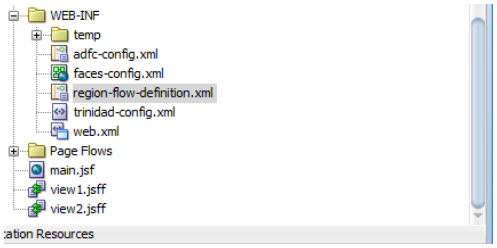
The following example shows how to invoke a dialog upon initial page fragment rendering and view navigation within an ADF region using Oracle JDeveloper 11g R2 (11.1.2.4)

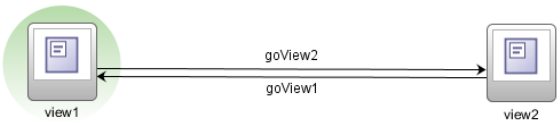


Setting up the use case

The use case for this example is simple and shows a region embedded in a JSF page exposing a task flow that has two page fragments view1.jsff and view2.jsff. The default page fragment, and also the fragment that launches the popup is view1.jsff.

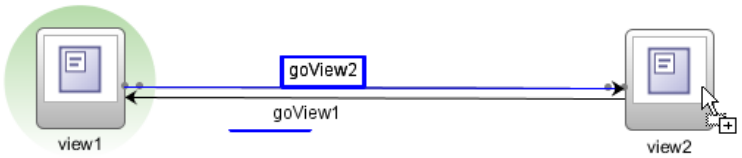
The wanted outcome is that a popup is launched whenever view1.jsff initially renders as the default view of the region and when the view is navigated to in context of task flow navigation.

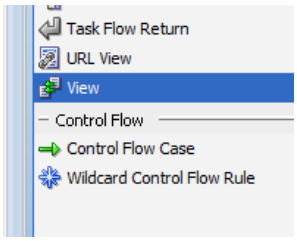




Creating view1.jsff

View1.jsff is created declaratively by dragging the view component from the task flow component palette to the bounded task flow definition.





To keep the sample simple, the views only contain a single command button to simulate navigation in the region. A simplified page source code of view1.jsff is shown below

```
<?xml version='1.0' encoding='UTF-8'?>
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
                xmlns:f="http://java.sun.com/jsf/core">
  <af:panelGroupLayout id="pgl3">
    ...
    <af:commandButton text="goView2" id="cb1" action="goView2"/>
    ...
```

```
    <af:popup childCreation="immediate" autoCancel="disabled" id="p1"
              contentDelivery="lazyUncached">
      <af:dialog id="d1" resize="on" stretchChildren="first" type="ok"
                 title="Information">
        ...
          <af:outputText value="This is a dialog that shows on page
                                fragment load" id="ot1"/>
        ...
      </af:dialog>
    </af:popup>
  </af:panelGroupLayout>
</ui:composition>
```

There are two things to mention about the source code shown above:

1. The af:popup component is set to immediately create its child objects by setting the popup **childCreation** property to **immediate**.
2. The root component of the page fragment is a group layout that contains to child components: the command button and the popup. With this we know that the popup component is within the same JSF naming container, which means we can perform relative searches as will be shown later.

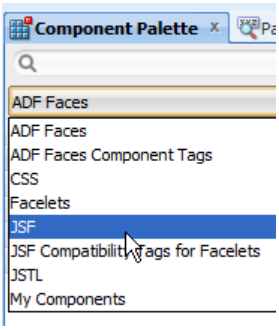
Defining the system event

System events in JSF 2 allow developers to add a component phase listener – f:event – to UI components, which then fire to defined times in the page lifecycle.

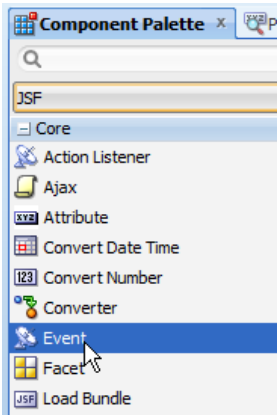
Select the view1,jsff page fragment in the JDeveloper Application Navigator and open the Structure Window (ctrl+shift+S).

Select the **af:panelGorupLayout** component (the oage fragment root component) as this components is unlikely to be partially refreshed when the application user works within the view.

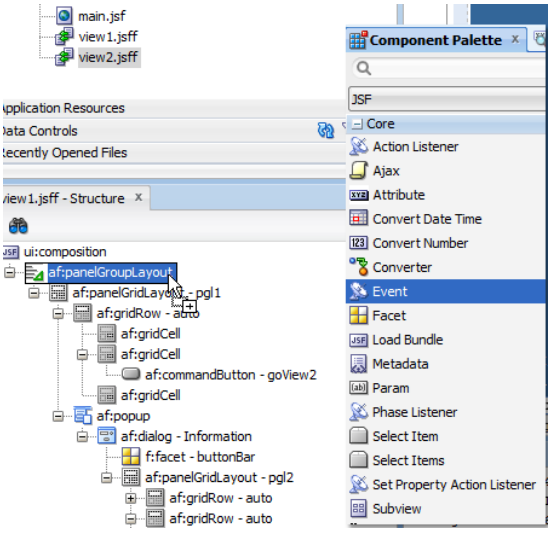
Open the Component panel (ctrl+shift+P) and switch the tag library selection to **JSF** as shown in the image below.



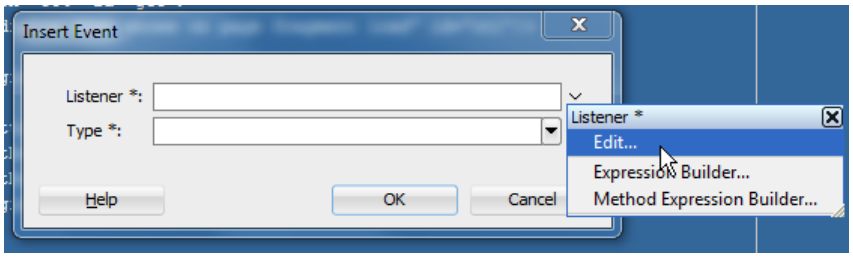
Then drag the **Event** component tag from the component palette …



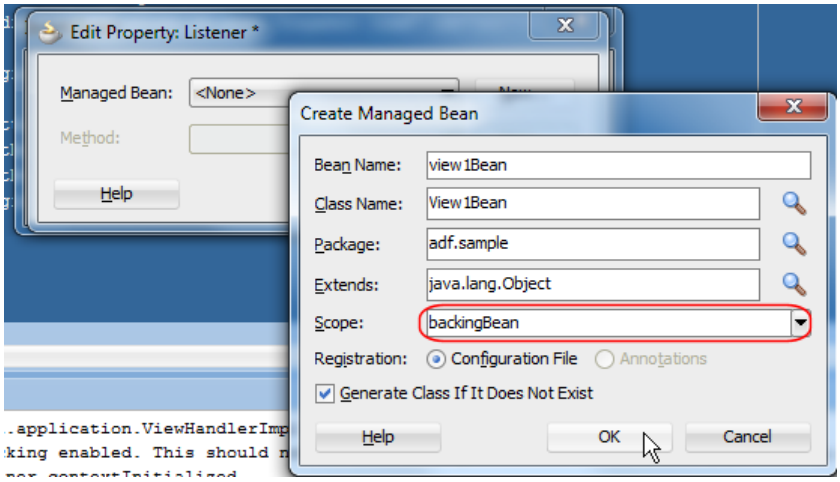
… and drop it on top of the **af:panelGroupLayout** component.



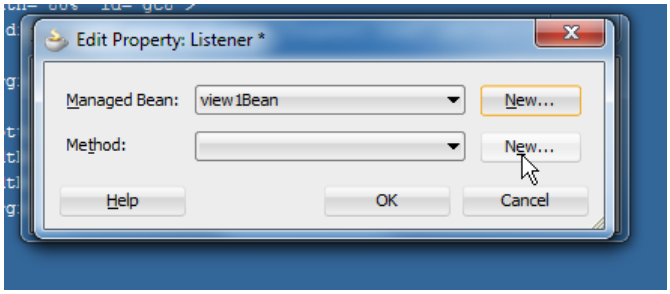
Use the **Insert Event** dialog to create and/or map the tag to a managed bean method.



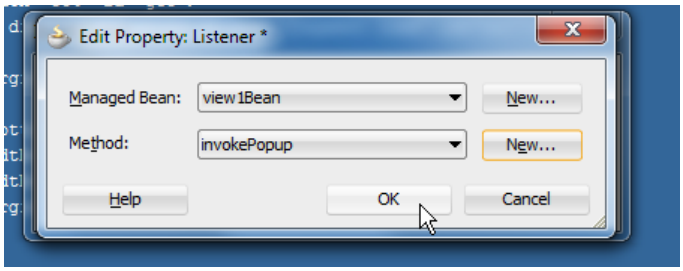
Create a new managed bean or select an existing managed bean to create or reference the event listener method in. Ensure that the managed bean is in backing bean scope as the view is exposed in a bounded task flow **(!)**.



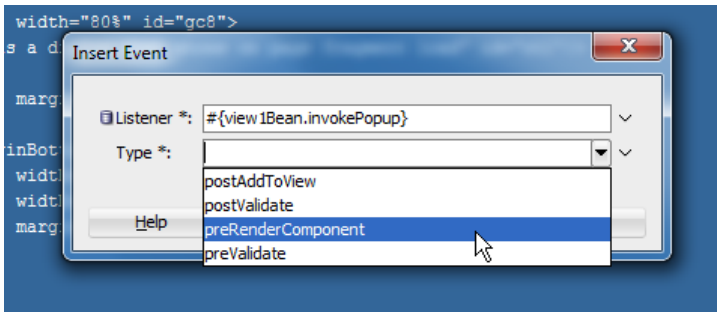
Press **New** next to the **Method** field to create the event listener method skeleton and signature in the managed bean.



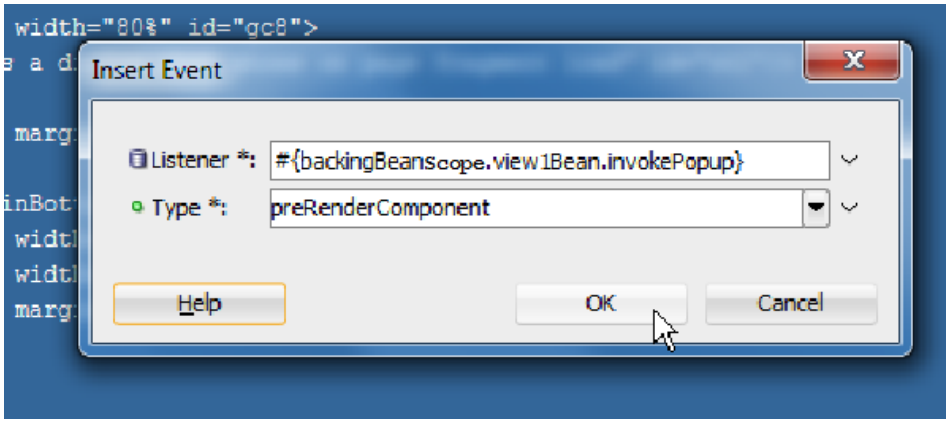
Press **OK** to close the managed bean creation dialog.



In the **Insert Event** dialog, choose the **preRenderComponent** phase as the event type. This way the referenced managed bean is called whenever the component is rendered.



Press **Ok** to add the event tag to the parent component (the panel group in this example)



The view1Bean managed bean referenced by the listener is shown below:

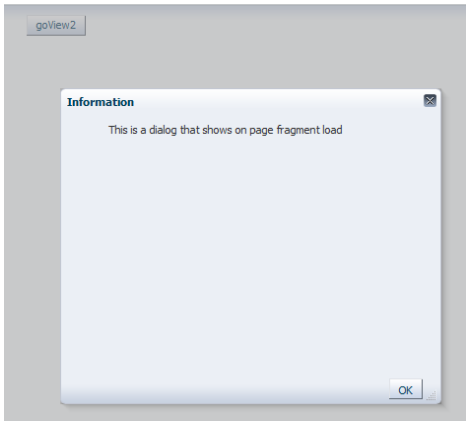
```
public class View1Bean {
 private String sayHello;
 private boolean popupLaunched = false;

 public View1Bean() {}


 public void invokePopup(ComponentSystemEvent componentSystemEvent) {
        //find popup relative to event component and
        //and avoid double invocation
        if (!popupLaunched) {
            UIComponent pageFragmentRoot =
                       componentSystemEvent.getComponent();

            //show dialog
           RichPopup p1 =
                 (RichPopup)pageFragmentRoot.findComponent("p1");

           RichPopup.PopupHints hints = new RichPopup.PopupHints();
           p1.show(hints);

        }
```

```
        popupLaunched = true;
      }
}
```

Conclusion

Using component events (aka. system events) in JSF 2 provides an optimized solution for launching ADF Faces popups upon view rendering in a region. This article explained the declarative steps to implement this solution.

At runtime, for this example, the popup below shows when the page containing the region renders and when view1.jsff is navigated to from view2.jsff



The completed page source is shown below:

```
<af:panelGroupLayout id="pgl3">
   <af:panelGridLayout id="pgl1">
        ...
        <af:commandButton text="goView2" id="cb1" action="goView2"/>
        ...
         <f:event listener="#{backingBeanScope.view1Bean.invokePopup}"
                  type="preRenderComponent"/>
      <af:popup childCreation="immediate" autoCancel="disabled" id="p1"
                contentDelivery="lazyUncached">
          <af:dialog id="d1" resize="on" stretchChildren="first"
                     type="ok" title="Information">
                ...
          <af:outputText value="This is a dialog that shows on page
          ...                        fragment load" id="ot1"/>
      </af:dialog>
    </af:popup>
```

```
  </af:panelGroupLayout>
</ui:composition>
```

RELATED DOCOMENTATION

⊠	System event tag: https://javaserverfaces.java.net/nonav/docs/2.1/vdldocs/facelets/f/event.html
⊠	
⊠	