ADF Code Corner 77. Handling the af:dialog Ok and CANCEL buttons

Abstract:



The af:dialog component provides configurable button options for Ok, Yes, No and Cancel buttons to close the popup and return to the parent page. To handle the return event, developers can setup a dialog listener in a managed bean. This dialog listener however does not handle the cancel event, as this is a client side event only that you can handle in JavaScript only. But what is there is server side code that you need to execute in response to the cancel event fired on the client? This article explains how to call the and execute server side logic in response to the cancel dialog client-side event.

twitter.com/adfcodecorner

Author:

Frank Nimphius, Oracle Corporation twitter.com/fnimphiu 29-MAR-2011

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: <u>http://forums.oracle.com/forums/forum.jspa?forumID=83</u>

Introduction

The use case in this sample is a create-new row situation that is handled by a dialog. Users can either create an commit a newly created row or cancel it, in which case the new row changes need to be undone and removed. Another popular use case is a cofirmation dialog in which users press the *Ok* button to commit a previous change or use the *Cancel* button to roll it back.

Running the Oracle JDeveloper 11.1.1.4 workspace that you can download as sample #77 from the ADF Code Corner website, an ADF Faces page shows that allows you to create a new row for the HR Departments table. In the opened af:dialog instance, you can edit the new row and commit changes by pressing *Ok* or closing the dialog and undoing the row creation by pressing the *Cancel* button.

DepartmentId		DepartmentName
10		Administration
20		Marketing
30		Purchasing
40		Legal
50		Human Resources
60		Π
70		Public Relations
80		Sales
90		Executive
100		Finance
110		Accounting
120		Treasury
130		Corporate Tax
140		Control And Credit
160	Complete ou Cancel Department Cuast	
170	Complete or Cancel Department Creat	
100	* DepartmentId	
Create new Row	Departmentud	
	* DepartmentName	
	ManagerId	
	LocationId	
	Locatorita	
	0	K Cancel
		N

Set-up

The sample is based on the Oracle HR database schema and uses Oracle ADF Business Components as the business service. Note that – except for how to perform the commit action – there is no difference between using ADF Business Components, Enterprise JavaBeans, Web Services or JavaBeans as business services. ADF Business Components was chosen as a business service only because it is "quick and easy" to use.

The **DepartmentsView** view object, renamed to **allDepartments**, is added as a table to the page. A command button is added to create a new table row and launch the dialog.

Building the sample

To edit new department rows, drag and drop the af:popup component from the Oracle JDeveloper Component Palette and add the af:dialog tag as a child component. Ensure the dialog is configured to show the Ok/Cancel button pair, which you do using the Property Inspector (ctrl+shift+I). Drag the DepartmentsView view object instance from the Data Controls panel to the dialog and choose **ADF Form** from the context menu. Confirm the form creation dialog **without** creating navigation buttons or a submit button. The functionality of a submit button is handled by the af:dialog OK button.



To create the Commit functionality, which is called from the af:dialog listener in response to users pressing the OK button, you click the **Bindings** tab at the bottom of the visual page editor in Oracle JDeveloper. This opens the binding dialog shown in the image below.

In the binding editor, **press** the green plus icon and choose the **action** option in the opened **Insert Item** dialog to create a new action binding.



In the **Create Action Binding** dialog, select the Data Control node and choose **Commit** from the list of actions.

Note: If your Data Control is not ADF Business Components, then, in this step, you create a method binding instead of an action binding and choose the method that performs the persist operation. For example, in the EJB case this would be the *merge* method.

CreateNewDepartment.jspx	
Page Data Binding Definition	
This shows the Oracle ADF data bindings defined for your page. Select a binding to s Page Definition File: <u>adf/sample/view/pageDefs/CreateNewDepartmentPageDef.xr</u>	ee its relationship to the underlying Data
Bindings and Executables Contextual Events Parameters	🔆 Create Action Binding
Model Bindings Bindings Bind	Select a data collection and the action on the data objects of the selected of Data Collection:
	Select an Iterator:
	Apply to all i

Switch back to the page **Design** view. From the Data Controls panel, drag the **CreateInsert** operation to the page and drop it onto the existing command button. This configures the command button's **Action Listener** property to reference an action in the ADF binding layer when the button is pressed.



In the opened dialog, keep the current values for the button text and the disabled state.

The following component property values will be overwritten if you change Value checkbox for any properties you don't want to change, then select Property Values:						
Keep Current Value	Property	Current Value				
~	text	Create new Row				
	id	cb1				
	actionListener					
✓ disabled						

To make the af:dialog instance in the managed bean that handles the dialog action, you create a JSF component binding. In the visual editor, **select** the dialog component and open the Property Inspector. In the Property Inspector, choose the **Binding** property and use the **down arrow** icon to open the **Binding** context menu. Press the **Edit** option to create or choose a managed bean to hold the dialog instance reference.

Complete or Cance Departmentid DepartmentName	I Department Creation #{DepartmentId.inputValue} #{DepartmentName.inputVa	National International Interna					
Managerid Locationid	#{Managerld.inputValue} #{LocationId.inputValue}	Cancel	3				
L		R	🖳 Popup - Property Inspect	or	🎁 bin	d	₽ ₽
			ContentDelivery: PartialTriggers:	<default> (lazy)</default>			•
			Animate: LauncherVar:	<default> (default)</default>		Binding Edit	2
			EventContext: PopupFetchListener:	<default> (self)</default>		Property Help	•
			PopupCanceledListener: Advanced Binding:			the component instance on a bean. This can be used to give programmatic access to a component from a backing bea or to move creation of the	e an, 🗸

When you create a new managed bean, ensure the **Create Class If It Does Not Exist** check box is selected.

Create Mana	Create Managed Bean					
Bean Name:	Bean Name: CreateDepartmentPageHelper					
Class Name:	CreateDepartmentPageHelper	Browse				
Package:	adf.sample.view Browse					
Extends:	java.lang.Object	Browse				
Scope:	request 🔹					
Generate						
Help	ОК	Cancel				

In the second dialog **Edit Property: Binding**, define a name for the af:dialog reference. In this example, the property name is chosen as "popup".

Complete or Cance	I Department Creation	×	ľ	1			
DepartmentId	#{DepartmentId.inputValue}						
DepartmentName	#{DepartmentName.inputValue}						
Managerid	#{Managerld.inputValue}	<					
LocationId	#{LocationId.inputValue}						
	OK Can	cel			b Edit Property:	Binding	x
					Managed Bean:	CreateDepartmentP 🕶	<u>N</u> ew
					Property:	popup 💌	N <u>e</u> w
					Help	ок	Cancel

Note: To partially refresh the table in response to the dialog outcome, you should also create a JSF component binding for the table. In this example, the JSF component binding for the table is created in the same managed bean that handles the dialog event.

Next, select the **Create New Row** button on the page and double click it to create a method action binding in the managed bean you created before. Because the command button has an ADF action binding reference, it is required to select the **Generate ADF Binding Code**, which ensures that the current functionality of the button doesn't change. The nice side-effect of this step is that it autogenerates helper methods that we use to access the binding layer and perform actions on it.

Bind Action Property							
Managed Bean:	CreateDepartmentPageHelper						
Method:	cb1_action						
✓ Generate ADF	Binding Code						
Help	OK Cancel						

Select the af:dialog and use the **down arrow** icon on the **DialogListener** property to add the logic that is executed when the user presses the Ok button in the dialog.

Complete or Cancel Department Creation		
DepartmentId #{DepartmentId.inputValue}		
DepartmentName #{DepartmentName.inputValue}		
Managerld #{Managerld.inputValue}		
LocationId #{LocationId.inputValue}		
OK Can	Dialog - Complete or Cancel Department Creation - Property Inspe	ctor
	🔣 📌 🖹 🥒	
	Width:	
	Height:	• •
	Margin:	- ~ II
	Padding:	• ~
	Border Color:	
	Border Style:	
	Border Width:	• •
	Outline Color:	
	Outline Style:	• ~
	Outline Width:	• •
Source Bindings Preview History		
ing: IntegratedWebLogicServer - Log	Modai: (default> (true)	°
	Resize: commons.com commons.com commons.com commons.com commons.com"/>commons.com commons.com"/>commons.com commons.com"/>commons.com commons.com	`` \
	DialogListener:	DialogListener
	PartialTriggers:	Edit
	⊒ Advanced	Method Expression Builder
	Binding:	Reset to Default

Provide a meaningful name for the **Method** and make sure it is created in the same managed bean that you created to hold the af:dialog component binding reference.

С	omplete or Cance	I Department Creation	×								
	DepartmentId	#{DepartmentId.inputValue}									
	DepartmentName	#{DepartmentName.inputValue}									
	Managerid	#{Managerld.inputValue}		×							
	LocationId	#{LocationId.inputValue}	1								
		OK Can	Q	Dialog - C	omplete	or Cancel D	epartment Crea	ation - Pr	ор	erty Inspector	
		UK Call	5	3 📌 🛛	9 🥖	•	(iii	Find			Ś
				une data a				1		[
				wiath:					_	%	
				Height:					•	%	
				Margin:					•	%	
				Padding					-	%	
				Border	🍝 Edi	t Property:	DialogListene	er			x
				Border							
				Border	<u>M</u> ana	ged Bean:	CreateDepart	tmentPag	jeł	Helper 🔻 🛛 <u>N</u> ew	
				Dorder	Meth	od:	onDialogActio	n	-	New	
				Outline	<u> </u>		UnbialogActio		-	100	
				Outline		Help		C	ж	Canc	el
				Outline	_			_		12	

As shown below, the dialog handler method has access to the DialogEvent object, from which you determine the Outcome. The Outcome that can be handled on the server includes **ok**, **no** and **yes**.

<pre>public void onDialogAction(DialogEvent dialogEvent) { Outcome outcome = dialogEvent.getOutcome();</pre>						
if	(outcome == Outcome.){					
1	123 📲 cancel	Outcome				
1	class	Class				
esign History	123 🏪 no	Outcome				
u IntegratedW	123 🚼 ok	Outcome				
, integrateum	🔚 🏪 valueOf(String)	Outconse				
	🖃 🌯 valueOf(Class <t>,S</t>	tring) T				
	🖻 🏪 values()	Outcome[]				
	123) 🏪 yes	Outcome				
	Regular Items: Press "Otrl-Space" for	Smart Items QuickDoc 🖃				

To quote the ADF Faces tag documentation for the af:dialog component::

When using the dialog type button configurations, action outcomes of type "ok", "yes", "no" and "cancel" can be intercepted on the client with a "dialog" type listener.

Only "ok", "yes" and "no" events will be propagated to the server. The ESC key, "cancel" button and close icon queues a client dialog event with a "cancel" outcome. Dialog events with a "cancel" outcome will not be sent to the server. Propagation of dialog events to the server can be blocked, as with any RCF event, by calling cancel() on the JS event object. Use the af:clientListener with a type of dialog to listen for a dialog client event.

This means that though the dialog Ok action can be handled on the server, the **Cancel** action cannot. However, for the use case demoed in the sample, we need to get the cancel event notification in a managed bean on the server.

The solution to this requirement is to listen for the cancel event using the af:clientListener component and JavaScript on the client.

The server notification then is handled by the af:serverListener component, which invokes a managed bean method on the server.

entName.inputValue}	Component Palette	
ld.inputValue}		
ld.inputValue}	abi races	0
CK Cancel	ADF Faces	
	Common Components	
	Layout	
	✓ Operations	
	Attribute Drag Source	<u>^</u>
	🛃 Attribute Drop Target	1
	Auto Suggest Behavior	
	👌 Calendar Drop Target	
	I Client Attribute	
	🔉 Client Listener	
	🕒 Collection Drop Target	~

Drag the **ClientListener** component from the Oracle JDeveloper Component palette into the dialog. In the opened **Insert Client Listener** dialog, set the **Method** attribute to the name of the JavaScript method to call when the dialog cancel action is invoked. The **Type** attribute must be set to **dialog**.

ADF CODE CORNER Handling the af:dialog OK and CANCEL buttons

Insert Client Listener	×
• Method *: onDialogCancel • Type *: dialog	
Help	OK Cancel

The JavaScript function determines the type of dialog action. If the dialog event is **Cancel**, then a server listener is used to queue a custom event that gets propagated to the server. To add the server listener, drag and drop the **Server Listener** component entry from the Component Palette to the af:dialog.

Complete or Cancel	I Department Creation		
DepartmentId	#{DepartmentId.inputValue}		
DepartmentName	#{DepartmentName.inputValue}		
Managerid	#{Managerld.inputValue}		
LocationId	#{LocationId.inputValue}	Component Polatta	
	Сан	ADF Faces	•
		<i>6</i> 6	>
		ADF Faces	
		Common Components	
		Layout	
		✓ Operations	
		Deturn Action Listener	î
		The Disk Taut Edites Jacob Relation	
		Rich Text Editor Insert Benavior	
		Scroll Component Into View Behavior	
		No Server Listener	•
		🕺 Set Action Listener	
		🔊 Set Property Listener	
			~

Select the af:serverListener entry in the Oracle JDeveloper Structure window and open the Property Inspector. Set the **Type** property to a name that later is used to identify this server listener instance. In the example, we called it *DialogCancelHandlerEvent*. The **Method** property is bound to a managed bean that accepts a single argument of type ClientEvent.

af:form	Server Listener - Property Inspector	
🛄 af:table - t1		
🔅 🗐 af:column - #{binding	🔝 🖌 🖉 🖉 🥒 🔞 Find 🛛 👋 😭	0
🕀 🗏 af:column - #{binding	a Tuna 8. DialooCancebiandlarEvent	
🕀 📄 af:column - #{binding	- Type T biologeancemanalerevent	
🕀 🗐 af:column - #{binding	Method:	×
Table facets		Method X
af:commandButton - Crea		Edit
街 af:popup		Method Expression Builder
😑 📰 af:dialog - Complete d		Reset to Default
		□ Property Help
 — KarlientListener 		a method reference to an event of
af:serverListener		the signature void(ClientEvent)
Dialog facets		
Document facets		

You can create the managed bean method using the **Edit** option in the context menu that opens when pressing the down arrow icon next to the **Method** property field.

CreateNewDepartment.js	ox - Structure	
📌 🗟		
tive.page		
ocument		innunge Cutinus -Cutinum aut Midt -CuCum MCt -Cummun
af:messages		
al:10mi af:table - t1	Server Listener - Prope	erty Inspector
ian:table - t1 ian:table - t1 ian:table - t1 ian:table - t1	🖳 l 🏓 🖹 l 🥒	(# Find
af:column - #{binding af:column - #{binding	e Edit Property: M	1ethod
af:commandButton - Crea	Managed Bean:	CreateDepartmentPageHelper▼ New
af:popup ⊡… 😨 af:dialog - Complete (Method:	Create Method
af:panelFormLay af:clientListener Joialog facets	Help	Method Name: onDialogCancel
Document facets		Help OK Cancel

To add the JavaScript function to the page, drag the **Resource** component from the Component palette onto the page and ensure it becomes a direct child of the af:document component.

	Component Palette
▼ ▼ ■ ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●	ADF Faces
	66
	ADF Faces
	Common Components
	Layout
	Operations Market Lext Benavior
	E Iterator
	🚸 Panel Dashboard Behavior
	🔁 Poll
	Reset Action Listener
	💁 Resource
	🔉 Return Action Listener
• Type *: javascript	撞 Rich Text Editor Insert Behavior
	Scroll Component Into View Behavior
Help OK Cancel	🕺 Server Listener
	🕺 Set Action Listener
	🔉 Set Property Listener
	Show Popup Behavior
	Show Popup Behavior Show Printable Page Behavior
	Show Popup Behavior Show Printable Page Behavior Skip Link Target
	Show Popup Behavior Show Printable Page Behavior Skip Link Target Switcher

```
<af:document>
 <af:resource type="javascript">
  function onDialogCancel(evt) {
     var outcome = evt.getOutcome();
     if(outcome == AdfDialogEvent.OUTCOME CANCEL) {
       //call server
       var eventSource = evt.getSource();
       var immediate = true;
       AdfCustomEvent.queue(
             eventSource,
             "DialogCancelHandlerEvent",
             { },immediate);
             evt.cancel();
       }
    }
  </af:resource>
  . . .
```

Note that the custom event that is queues on the server listener instance is set to be immediate to bypass any form update for the cancel action.

The event object that is passed into the JavaScript function is AdfDialogEvent. The JavaScript documentation is accessible online from otn.oracle.com. The image below shows an example for the AdfDialogEvent event.

```
AddfDialogEvent

AddfDialogEvent

CRACLE: JavaScript API Reference for Oracle ADF Faces

Overview Package Class Tree Deprecated Index Help

PREV NEXT
FRAMES NOFRAMES All Classes

SUMMARY: FELD | CONSTR | METHOD DETAIL: FELD | CONSTR | METHOD

oracle.adf.view.js.event

Class AddfDialogEvent

org.ecmascript.object.Object

+--oracle.adf.view.js.event.AdfBaseEvent
+--oracle.adf.view.js.event.AdfPhasedEvent
+--oracle.adf.view.js.event.AdfPhasedEvent
+--oracle.adf.view.js.event.AdfPhasedEvent
+--oracle.adf.view.js.event.AdfPhasedEvent
```

Managed Bean code

All sample code used by this example is saved in the same managed bean. The commented code listing is shown below.

```
import javax.faces.context.FacesContext;
import oracle.adf.model.BindingContext;
import oracle.adf.model.binding.DCIteratorBinding;
import oracle.adf.share.ADFContext;
import oracle.adf.view.rich.component.rich.RichPopup;
import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.adf.view.rich.context.AdfFacesContext;
import oracle.adf.view.rich.event.DialogEvent;
import oracle.adf.view.rich.event.DialogEvent.Outcome;
import oracle.adf.view.rich.render.ClientEvent;
import oracle.binding.BindingContainer;
import oracle.binding.OperationBinding;
import oracle.jbo.Row;
import oracle.jbo.server.ViewRowImpl;
public class CreateDepartmentPageHelper {
 //old "current row" value is saved in view scope in case the row
 //creation is cancelled, in which case this row needs to become
 //current again
 final String OLD_CURR_KEY_VIEWSCOPE_ATTR = "__oldCurrentRowKey__";
 private RichPopup popup;
 private RichTable departmentsTable;
 public CreateDepartmentPageHelper() {}
 public void setPopup(RichPopup popup) {
   this.popup = popup;
 }
 public RichPopup getPopup() {
 return popup;
 }
 //system generated method when you create a managed bean method for
 //a component that has an ADF binding referenced in its action
 //listener
 public BindingContainer getBindings() {
   return BindingContext.getCurrent().getCurrentBindingsEntry();
 }
```

 $// {\tt command}$ action that create a new row in the departments table and

```
//then opens an edit dialog for commit/cancel
 public String cb1 action() {
  BindingContainer bindings = getBindings();
  //get current row and save its rowKey in view scope for later use
  DCIteratorBinding dciter =
          (DCIteratorBinding) bindings.get("allDepartmentsIterator");
  Row oldCcurrentRow = dciter.getCurrentRow();
  //ADFContext is a convenient way to access all kinds of memory
  //scopes. If you like to be persistent in your ADF coding then this
  //is what you want to use
  ADFContext adfCtx = ADFContext.getCurrent();
  adfCtx.getViewScope().put(OLD CURR KEY VIEWSCOPE ATTR,
                        oldCcurrentRow.getKey().toStringFormat(true));
  //perform row create
  OperationBinding operationBinding =
                         bindings.getOperationBinding("CreateInsert");
  Object result = operationBinding.execute();
  if (!operationBinding.getErrors().isEmpty()) {
    return null;
  }
  //access the popup dialog and bring it up. The reference is
  //through a JSF component binding reference using the popup
  //"binding" property
  RichPopup popup = this.getPopup();
  RichPopup.PopupHints hints = new RichPopup.PopupHints();
  //empty hints renders dialog in center of screen
  popup.show(hints);
  return null;
public void onDialogAction(DialogEvent dialogEvent) {
  Outcome outcome = dialogEvent.getOutcome();
  //the dialog event only propagates yes, no, ok actions to the
  //server. The cancel outcome is only available on the browser
  //client. If there is a need to handle the cancel even then you
  //need to use a clientListener and JavaScript as we do on this
  //example
 if(outcome == Outcome.ok) {
    //commit
    BindingContainer bindings = getBindings();
    OperationBinding operationBinding =
                   bindings.getOperationBinding("Commit");
    Object result = operationBinding.execute();
       if (!operationBinding.getErrors().isEmpty()) {
          //handle errors if any
          //...
       return;
```

```
}
       AdfFacesContext.getCurrentInstance().addPartialTarget(
                                     this.getDepartmentsTable());
    }
   }
 public void setDepartmentsTable(RichTable departmentsTable) {
      this.departmentsTable = departmentsTable;
  }
 public RichTable getDepartmentsTable() {
    return departmentsTable;
  }
//method that is called from the serverListener on the client. The
//server listener is used to queue a custom event and pass information
//from the client to the server using JavaScript. It's actually doing
//this Ajax thing that everyone wants to do using the XmlHTTPRequest
//object
public void onDialogCancel(ClientEvent clientEvent) {
   BindingContainer bindings = getBindings();
   RichPopup popup = this.getPopup();
  popup.hide();
   //the cancel operation is executed with immediate=true to bypass the
   //model update. Therefore we manually delete the new row from the
   //iterator
   DCIteratorBinding dciter =
           (DCIteratorBinding) bindings.get("allDepartmentsIterator");
   Row currentRow = dciter.getCurrentRow();
   dciter.removeCurrentRow();
   //set current row back to original row
   ADFContext adfCtx = ADFContext.getCurrent();
   String oldCurrentRowKey =
       (String) adfCtx.getViewScope().get(OLD CURR KEY VIEWSCOPE ATTR);
   dciter.setCurrentRowWithKey(oldCurrentRowKey);
   AdfFacesContext.getCurrentInstance().addPartialTarget(
                                      this.getDepartmentsTable());
   FacesContext fctx = FacesContext.getCurrentInstance();
   //we don't want to continue with the remainder of the lifecycle and
  //thus skip the rest
   fctx.renderResponse();
  }
}
```

Download

You can download the sample workspace explained in this article as sample #77 from the ADF Code Corner website

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

To run it, ensure you have a database with the HR schema available. Change the database configuration to access your database and run the contained JSPX page. You can edit and Ok the dialog or directly press Cancel. Pressing **Ok** will commit the change to your database. Cancel will undo the changes and remove the row.

RELATED DOCOMENTATION

af:dialog tag documentation
http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_dialog.html