ADF Code Corner

039. How-to declaratively launch a bounded task flow in a lightweight popup

Abstract:

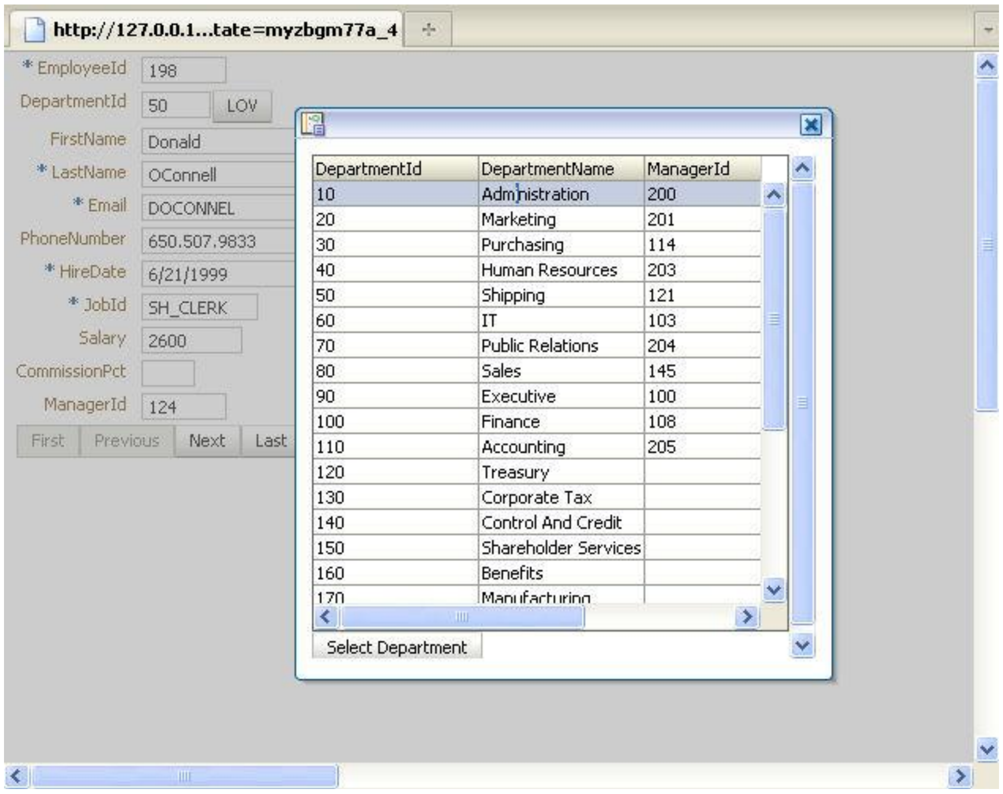
UI Shell is a layout pattern in which ADF bounded task flows are used within ADF dynamic regions to create desktop like user layouts that operate in a single document window. Bounded task flows that are displayed in ADF regions use ADF Faces page fragments (jsff) to display the UI content. In a blog article, David Giammona from the Oracle controller development team, described a pattern to use when a bounded task flow needs to be displayed in a popup dialog. The same pattern is what Lynn Munsinger and I describe in the upcoming "Oracle Fusion Developer Guide" book, published in January 2010 by McGraw Hill. The ADF region popup pattern described by David Giammona and in the book is a valid pattern to use still, but as a code centric approach appears to be less productive than a declarative approach. In Oracle JDeveloper 11g R1 Patch Set 1, the ADF controller team implemented a declarative approach for creating lightweight dialogs that can run bounded task flows in a lightweight dialog. This blog article provides you with the implementation details to implement this new declarative pattern.

twitter.com/adfcodecorner

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
17-NOV-2009

Introduction

The use case described in this blog article is simple as shown in the image below: An input form has "LOV" a launch button that displays a list of value in a popup dialog. The list of values is a custom implementation that runs a bounded task flow, simulating a more complex value selection process. For simpler use cases, for example if a table is all you need to look up to return a lookup value, you can use the same approach with a normal ADF Faces jspx page that you navigate to using a navigation case in ADFc.



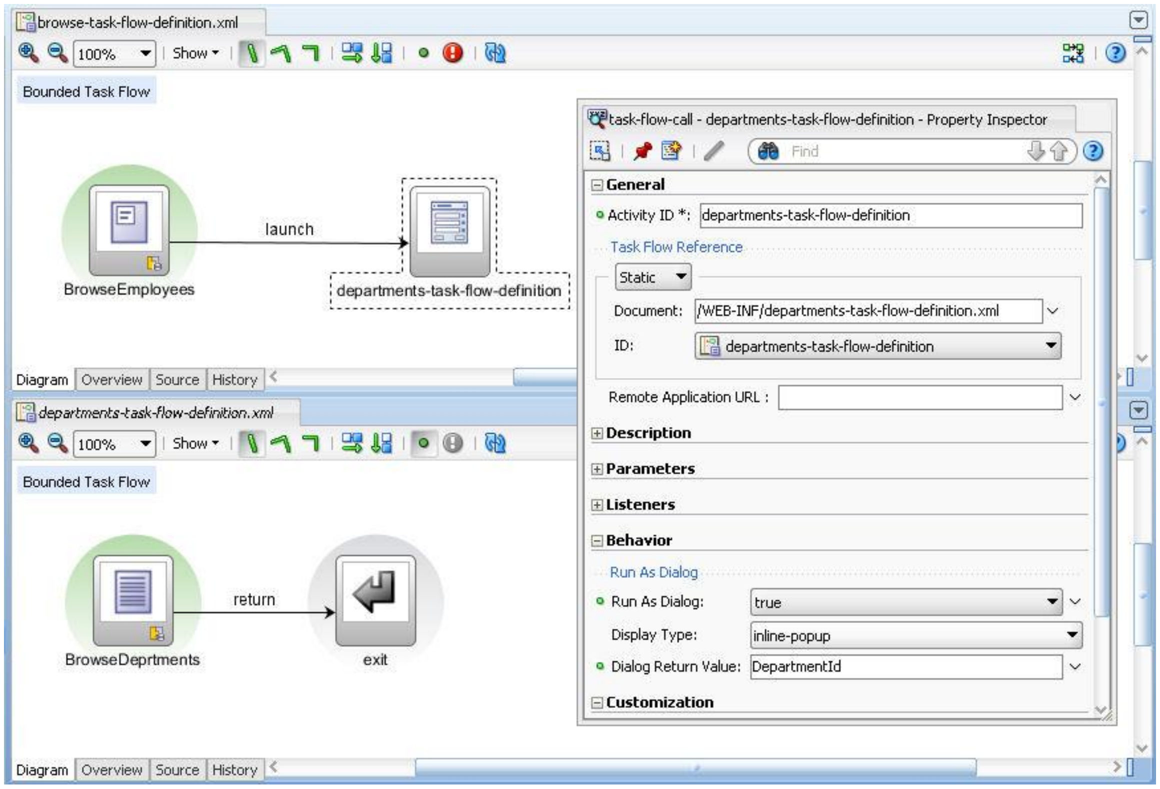
* EmployeeId	198
DepartmentId	50
FirstName	Donald
* LastName	OConnell
* Email	DOCONNEL
PhoneNumber	650.507.9833
* HireDate	6/21/1999
* JobId	SH_CLERK
Salary	2600
CommissionPct	
ManagerId	124

DepartmentId	DepartmentName	ManagerId
10	Administration	200
20	Marketing	201
30	Purchasing	114
40	Human Resources	203
50	Shipping	121
60	IT	103
70	Public Relations	204
80	Sales	145
90	Executive	100
100	Finance	108
110	Accounting	205
120	Treasury	
130	Corporate Tax	
140	Control And Credit	
150	Shareholder Services	
160	Benefits	
170	Manufacturing	

Calling a bounded task flows using a task flow call activity

A task flow call activity is a mean of accessing a bounded task flow from a parent flow. In the image below, the "departments-task-flow-definition" bounded task flow is called from the browse-task-flow-definition. As task flow call activity can be used by flows executed in an ADF region, in which case the called bounded task flow must also use page fragments to render the page content, or from a bounded task flow that executes using stand alone JSF pages (using the jspx document format).

To create a task flow call activity, certainly the easiest way to do this is to drag and drop the bounded task flow onto the parent flow diagram and draw a navigation case line to it. Then, from a command component, you reference the navigation case name in the action property. Note that unlike when opening pages in an external dialog, the navigation case does not need to have the "dialog:" prefix in the navigation case name, if the task flow call activity is used. This is because the configuration to launch the bounded task flow as a dialog is defined in the task flow call activity properties, as shown in the image below. To launch the popup as an inline popup, in addition to set the "Run As Dialog" option to "true", you also need to set the "Display Type" property to "inline-popup". Now, when the user navigates within the application and accesses the task flow call activity, the bounded task flow it references is called in a dialog.



task-flow-call - departments-task-flow-definition - Property Inspector

General
Activity ID *: departments-task-flow-definition
Task Flow Reference
Static
Document: /WEB-INF/departments-task-flow-definition.xml
ID: departments-task-flow-definition
Remote Application URL :
Description
Parameters
Listeners
Behavior
Run As Dialog
Run As Dialog: true
Display Type: inline-popup
Dialog Return Value: DepartmentId
Customization

As shown in the image above, the "departments-task-flow-definition" bounded task flow is exited via the "exit" return activity, which automatically closes the dialog for you as well. Not shown in the image above, within the "Parameters" section of the bounded task flow, a return parameter - using the "Return Values Definitions" option - is defined that passes a value from the bounded task flow back to the calling task flow.

In the example above, the return value is defined as shown below

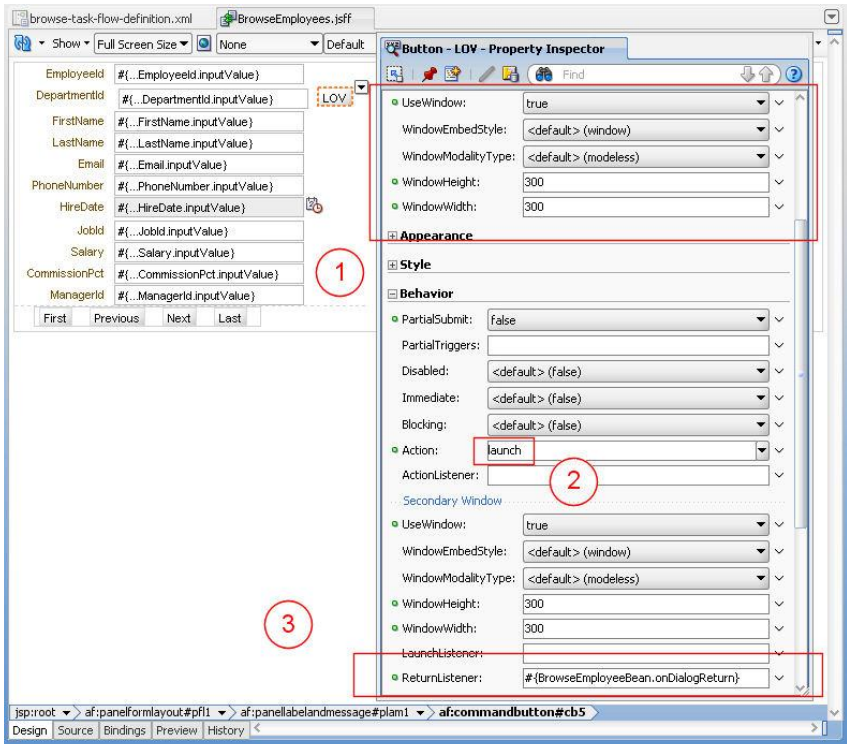
```
<task-flow-definition id="departments-task-flow-definition">
 <default-activity>BrowseDeprtments</default-activity>
    <return-value-definition>
 <name>DepartmentId</name>
 <value>#{pageFlowScope.returnValue}</value>
 <class>oracle.jbo.domain.Number</class>
    </return-value-definition>
 . . .
```

A return listener, that you configure on the command component that navigates to the task flow call activity that launches the dialog, can be implemented using a managed bean to read the return value and do something with it. An example code is shown later in this article.

Displaying a bounded task flow in a popup launched from an ADF region

In the example, I use a commandButton to navigate to the task flow call activity, which then launches the dialog. The following configuration needs to be done for the command button

- Specify the UseWindow property and define the size of the window that is opened by the navigation
- Reference the navigation case name - "launch" in this example - from the command button action property
- Define a return listener that reads the return value from the return event



EmployeeId	#{...EmployeeId.inputValue}
DepartmentId	#{...DepartmentId.inputValue}
FirstName	#{...FirstName.inputValue}
LastName	#{...LastName.inputValue}
Email	#{...Email.inputValue}
PhoneNumber	#{...PhoneNumber.inputValue}
HireDate	#{...HireDate.inputValue}
JobId	#{...JobId.inputValue}
Salary	#{...Salary.inputValue}
CommissionPct	#{...CommissionPct.inputValue}
ManagerId	#{...ManagerId.inputValue}

UseWindow:	true
WindowEmbedStyle:	<default> (window)
WindowModalityType:	<default> (modeless)
WindowHeight:	300
WindowWidth:	300
PartialSubmit:	false
PartialTriggers:	
Disabled:	<default> (false)
Immediate:	<default> (false)
Blocking:	<default> (false)
Action:	launch
ActionListener:	
UseWindow:	true
WindowEmbedStyle:	<default> (window)
WindowModalityType:	<default> (modeless)
WindowHeight:	300
WindowWidth:	300
LaunchListener:	
ReturnListener:	#{BrowseEmployeeBean.onDialogReturn}

The task flow call activity defined the return value as shown next:

```
<task-flow-call id="departments-task-flow-definition">
 <task-flow-reference>
 <document id="__4">/WEB-INF/departments-task-flow-
definition.xml</document>
 <id id="__6">departments-task-flow-definition</id>
 </task-flow-reference>
 <run-as-dialog id="__2">
 <display-type id="__5">
 <inline-popup/>
 </display-type>
 <dialog-return-value id="__3">
 DepartmentId
 </dialog-return-value>
 </run-as-dialog>
</task-flow-call>
```

Finally, the managed bean code that is referenced from the ReturnListener property on the command button sets the return value to theinputTextField

```
import java.util.HashMap;
import oracle.adf.controller.ControllerContext;
import oracle.adf.controller.TaskFlowId;
import oracle.adf.model.BindingContext;
import oracle.adf.view.rich.component.rich.input.RichInputText;
import oracle.binding.AttributeBinding;
import oracle.binding.BindingContainer;
import oracle.binding.ControlBinding;
import org.apache.myfaces.trinidad.event.ReturnEvent;

public class EmployeeBean {
 private RichInputText deptIdField;
 public EmployeeBean() {}

 public void onDialogReturn(ReturnEvent returnEvent) {

  BindingContext bindingContext =
                BindingContext.getCurrent();
  BindingContainer bindings =
                bindingContext.getCurrentBindingsEntry();
  ControlBinding control = bindings.getControlBinding("DepartmentId");
   AttributeBinding deptId = (AttributeBinding) control;
   deptId.setInputValue(returnEvent.getReturnValue());
 }

 public void setDeptIdField(RichInputText deptIdField) {
 this.deptIdField = deptIdField;
 }

 public RichInputText getDeptIdField() {
 return deptIdField;
```

```
  }
}
```

Important lesson to learn: The Key to Success

I mentioned that task flow call activities that are called from a bounded task flow in an ADF region call bounded task flows that also use page fragments to display the UI content. Note that to get this sample to work, the bounded task flow that is displayed within the opened inline popup must use JSPX documents. If you try referencing a bounded task flow that uses page fragments from a task flow call activity that has the dialog option set, an error message is shown in the visual design time. When you look at the sample workspace I provide with this blog article, you see that the dialog content uses stand alone JSF pages, whereas the calling bounded task flow uses page fragments.

This brings me back to my thesis that both popup patterns, the one first explained by David Giammona, which also is documented in the ADF Developer Guide book, and the new external dialog patten explained in this article, have their reason to exist. If you don't want to maintain two versions of a bounded task flow, or if you want to reuse bounded task flows in dialogs and in a non-dialog call, then the existing code centric pattern might be a better choice. However, given that ADF seeks for simplicity and not complexity, the external dialog framework example used in this article is the option I'd recommend you using for as long as it meets the needs of the use case you are going to implement. Also note that the code centric solution gives you a bit more control over where in the page the dialog pops up. The good about using the external dialog framework though is that the launch event listener and the return event listener make it easy to pass information to the opened dialog (bounded task flow) and receive return values. Also there is no need to manually ensure that the region within the dialog is refreshed when called again from the task flow call activity

Download

You can **download a small example from ADF Code Corner**:

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

and run it with Oracle JDeveloper 11g R1 PatchSet1. Make sure you change the database connection to access a local database with the HR schema installed. Run the Start.jspx page

RELATED DOCOMENTATION

☒	David Giammona, ADF region popup pattern - http://blogs.oracle.com/DavidGiammona/regions_dynamic_regions/
☒	Oracle Fusion Developer Guide, by Frank Nimphius and Lynn Munsinger - http://www.mhprofessional.com/product.php?cat=112&isbn=0071622543