ADF Code Corner

029. How-to build Oracle Forms style List-of-Values in ADF Faces

twitter.com/adfcodecorner

Abstract:

Oracle Forms allowed developers to associate a smart list of values with an inputtext field. The intelligence of the Forms LOV appears almost mind reading in that it doesn't pop up the LOV if the partial search condition entered by the user retrieves a single match only. If more than a single match is returned, a LOV dialog is opened with the pre-fetched list of data. Of course, Oracle Forms LOV can also be used the dumb way, in which the user hits the LOV button to then manually enter the query condition before pressing the execute query button.

In JDeveloper 11, the ADF Faces and ADF development teams have closed the gap that existed in 10.1.3 for LOV on the web. A new component, the af:inputListOfValues component, is introduced that, in combination with ADF Business Component's model driven list of values, can be configured to behave and act like LOVs in Oracle Forms.

This how-to document shows and explains how to build such a list of values and also unveils the little code to write to make the LOV pre-fetching a list of values based on the partial user input in the input text field or other custom code that should be reflected in the query.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
28-JUL-2008

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

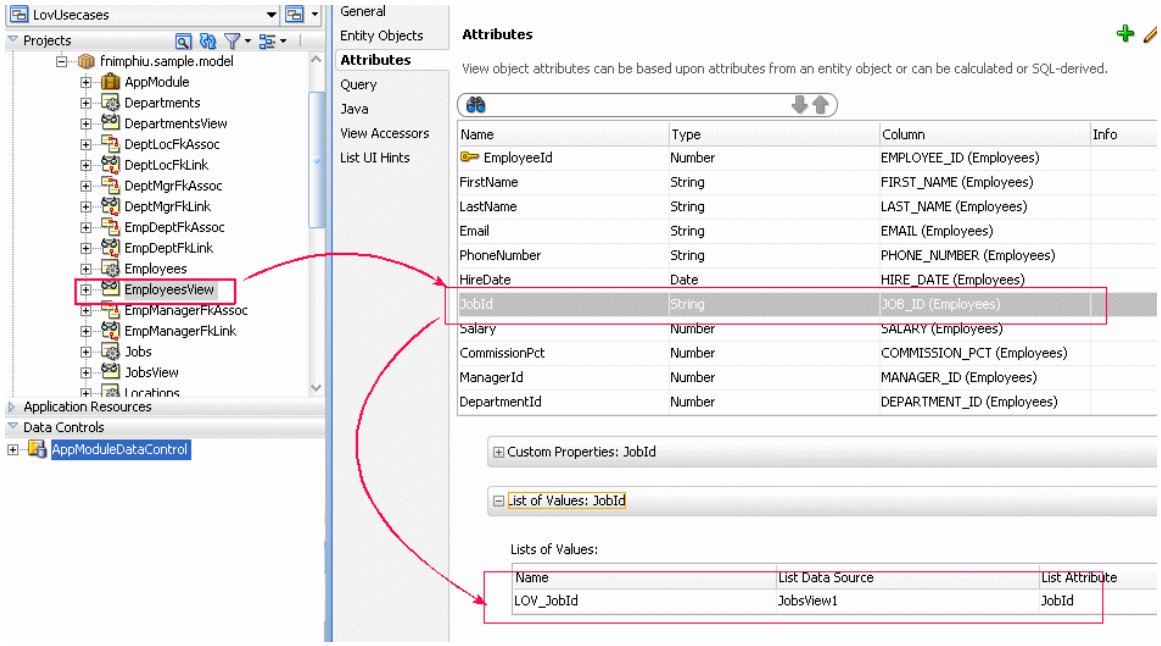
The af:inputListOfValues in ADF Faces Rich Client is an input text field with a launch icon to open a popup. The popup dialog shows the list of allowed values retrieved based on the application user provided search conditions. Setting the "autosubmit" property of the af:inputListOfValues component to true, also launches the popup automatically when the user tabs out of the input field with the field only partially edited. The idea of the af:inputListOfvalues is to help developers to allow users to lookup values from large data sets without paying the penalty of long running queries.

Note: The content of this how-to is not needed if you set the "Query List Automatically" on the model friven LOV. The code in this document is needed when the "Query List Automatically" checkbox is not set or if you want the search field to show the value of the initial search in the search field.

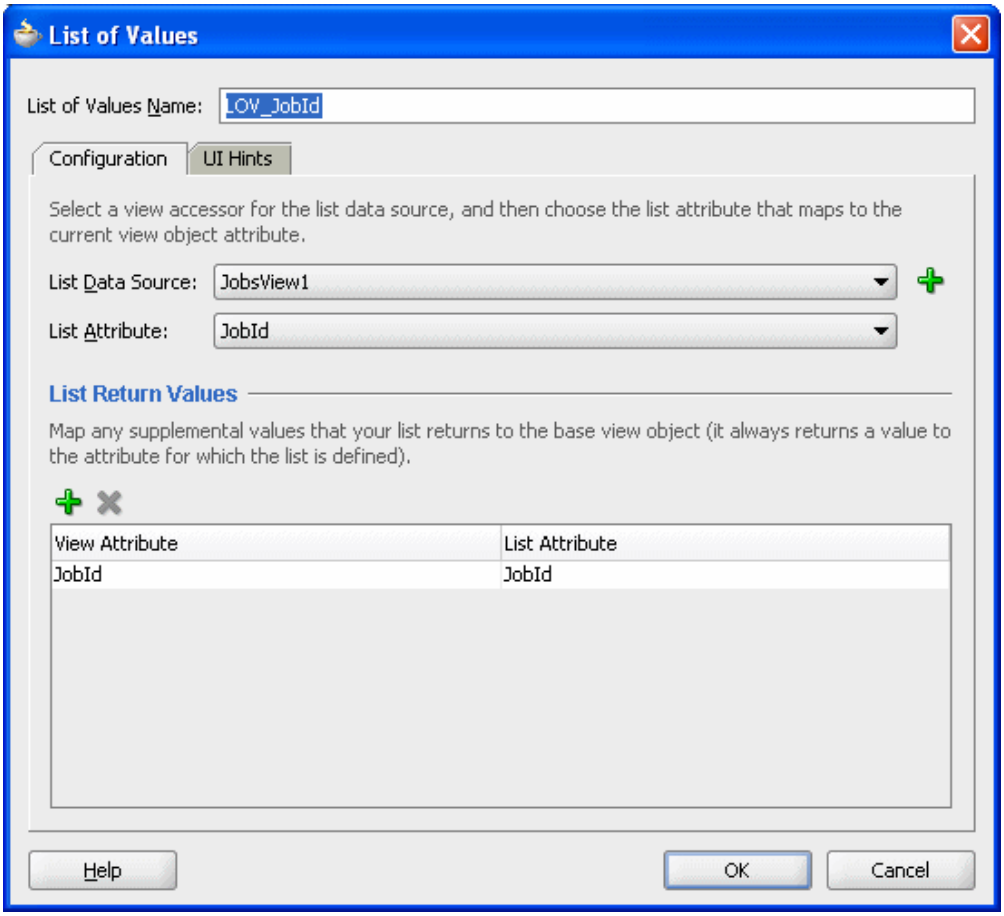
Building the LOV model part

A new feature in ADF Business Components of JDeveloper 11 is model driven list of values. Model driven list of values allow developers to define a dynamic list of lookup data for an attribute of a ViewObject. The benefit of model driven values is the same as for UI hints, the LOV is defined in a single place so that modifications to the list propagate to all usages of the list.

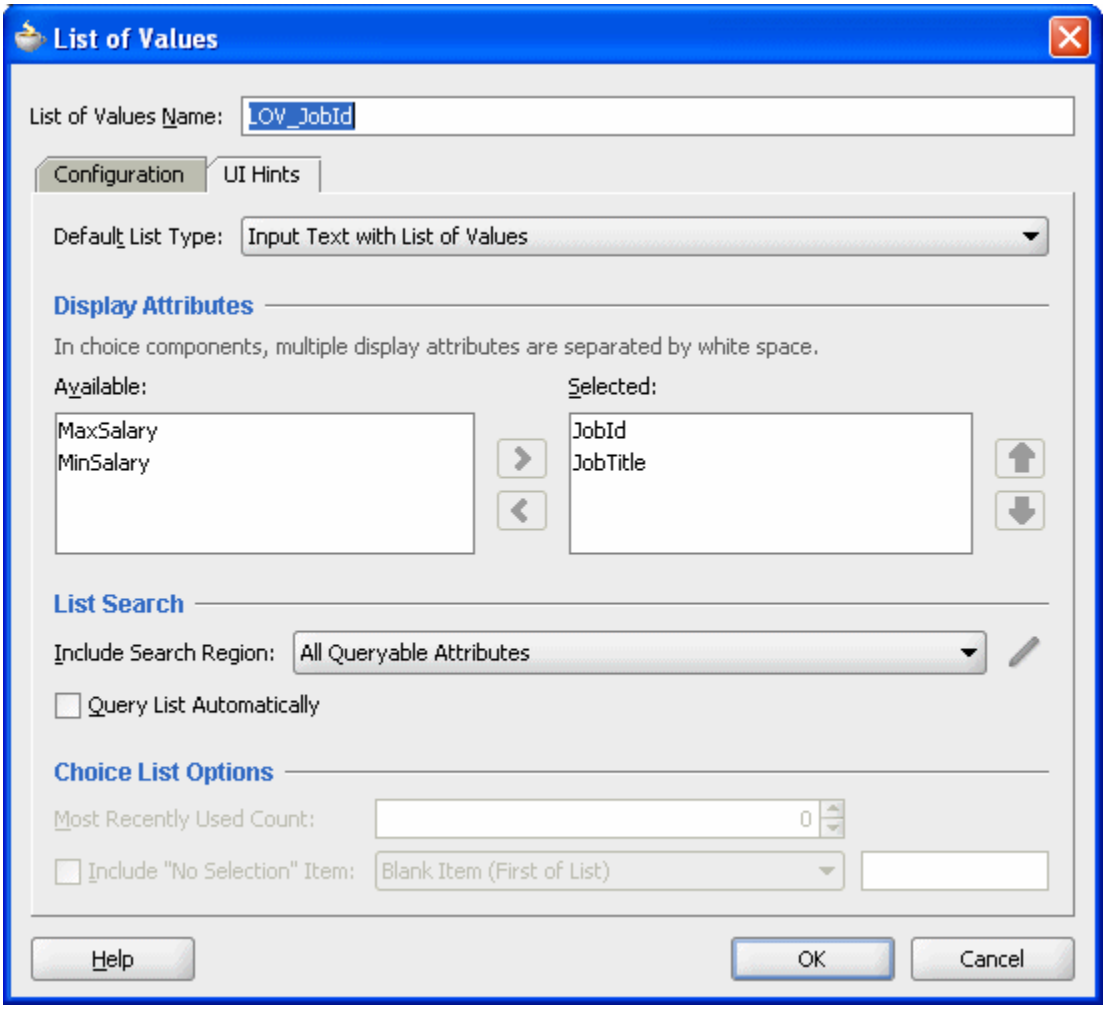
In the image below, a list of values is defined for the "JobId" attribute of the EmployeesView. To define a list of values, select an attribute and click the green cross icon on the right of the list of values section header (not shown in the image)



In the opened dialog, select the view object that serves as the list data source. In the example below the list is read from the JobsView1 ViewObject. The list attribute that matches the target attribute is "JobId".

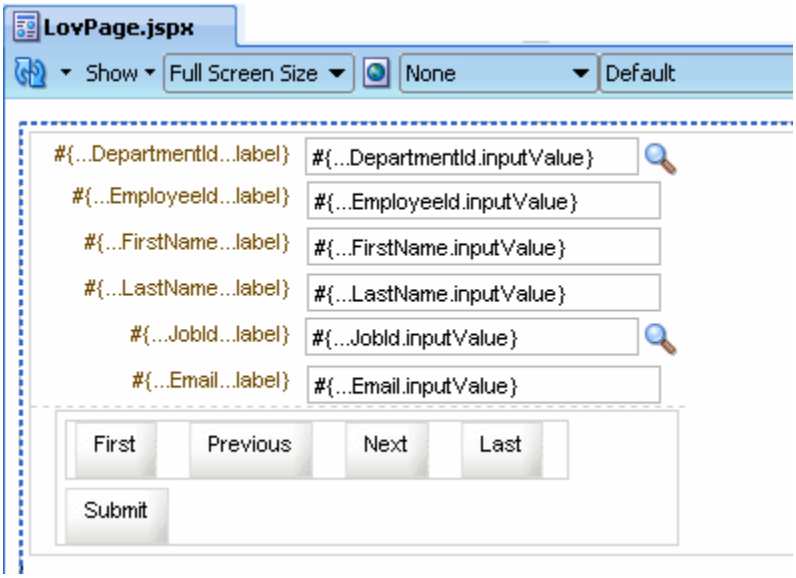


Clicking onto the "UI Hints" tab allows the developer to specify the display attributes, the attributes that are shown in the list of values dialog. This example defines the JobId and the JobTitle as the display values. In ADF Faces, all attributes will show as query fields in the list of value dialog. This is independent from the attributes selected for display.

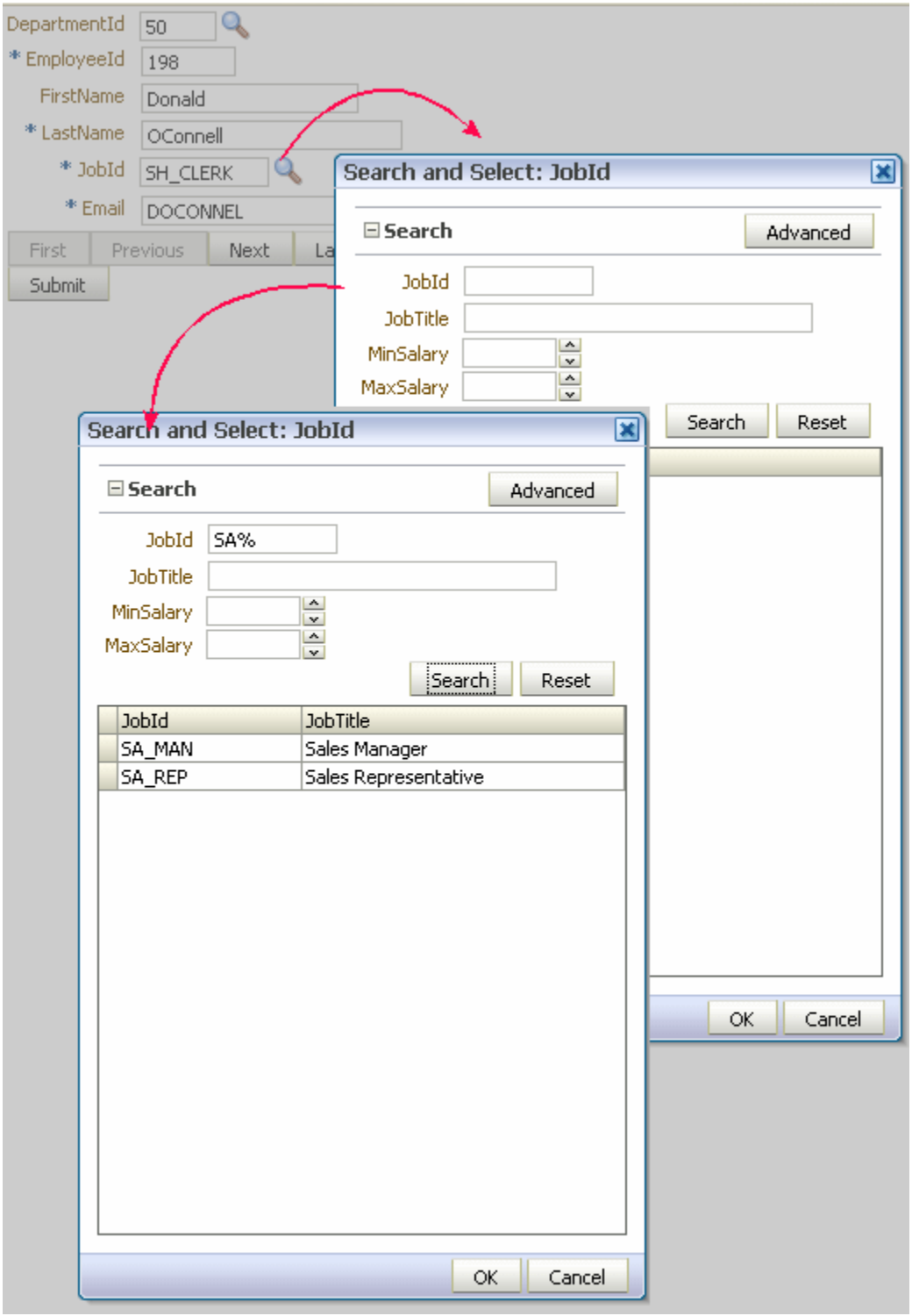


Creating the ADF Faces form

A ViewObject that is dragged from the DataControl palette and dropped as a form to the JSF page is automatically added to the page with the LOV attributes defined as af:inputListOfValues. At runtime and at designtime, the LOV component shows a search icon on the right side.

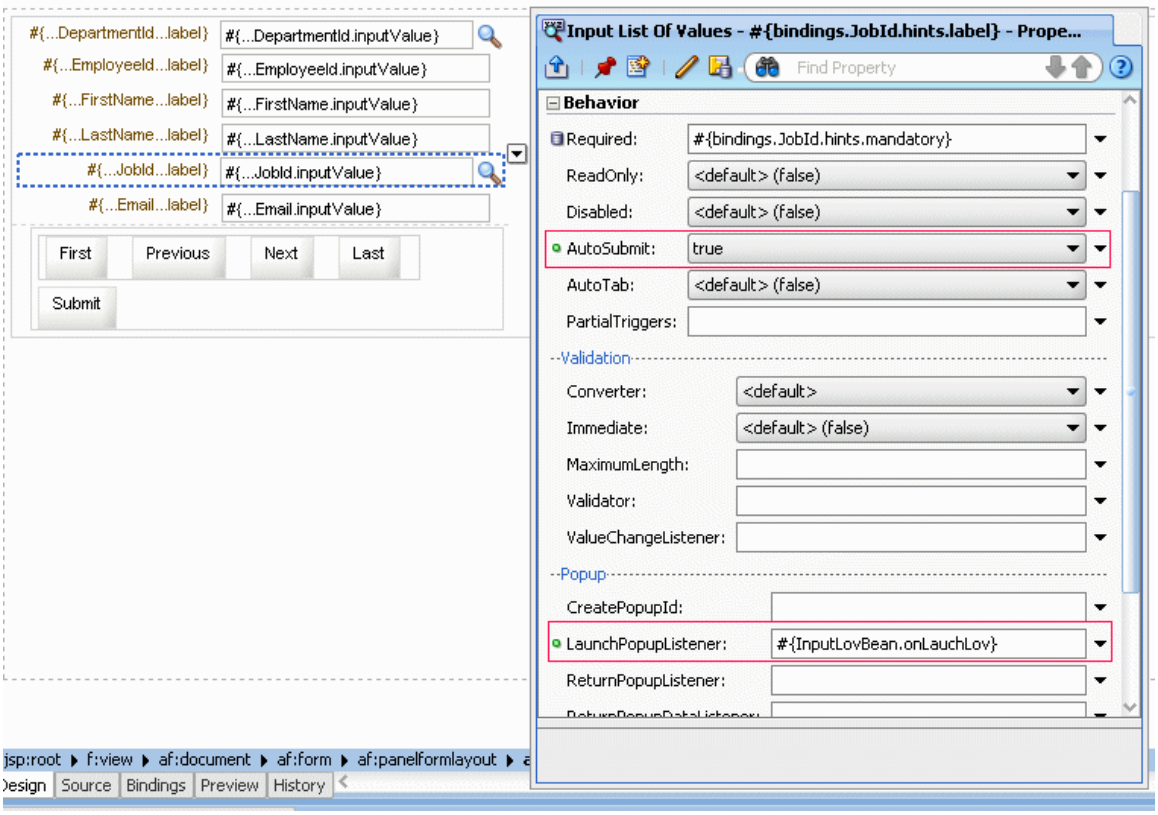


If the user presses the search icon, the list of values dialog is opened for the user to further specify the lookup query. Typing "SA%", as in the example below, retrieves all data that start with "SA". Selecting a value from the list and pressing the "OK" button updates the LOV input text field.


DepartmentId
50
* EmployeeId
198
FirstName
Donald
* LastName
OConnell
* JobId
SH_CLERK
* Email
DOCONNEL
First
Previous
Next
Submit
Search and Select: JobId
Search
Advanced
JobId
JobTitle
MinSalary
MaxSalary
Search
Reset
OK
Cancel
Search and Select: JobId
Search
Advanced
JobId
SA%
JobTitle
MinSalary
MaxSalary
Search
Reset
JobId
JobTitle
SA_MAN
Sales Manager
SA_REP
Sales Representative
OK
Cancel

Configuring af:inputListOfValues for auto completion

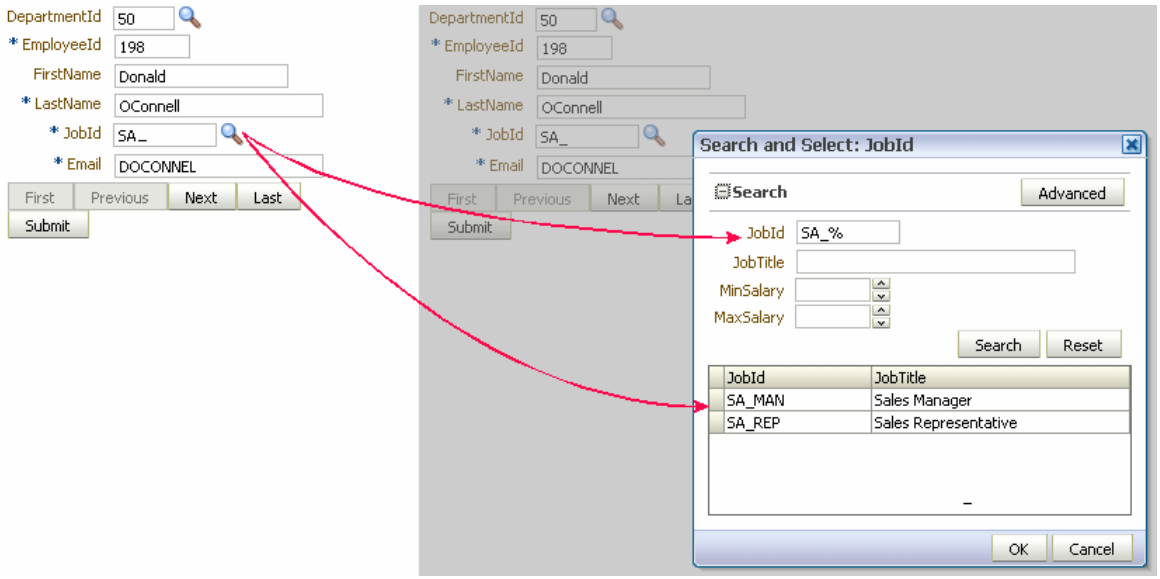
To enable intelligent LOVs like in Oracle Forms, two additional configuration steps are required: i) setting the autosubmit property to "true" ii) defining a popup launch listener to read the partial search string entered by the user to perform a query with it



With the autosubmit property set to "true", ADF Faces checks the user provided input string as soon as the user tabs out of the input text field. If the search string returns a single data match only then the LOV dialog isn't shown and instead the value is directly copied to the LOV input text field. If there are more values that meet the search criteria, then the list of value dialog is shown. However, to this point the dialog comes up empty for the user to enter the query criteria and to execute search.

Enabling the LOV to query for partial user entries

A LaunchPopupListener helps to make the list of value Forms like. The LaunchPopupListener allows developers to read the user entered search criteria and to add it to the LOV search form before executing the query. In this example, the user input string is added to the JobId search field, adding a "%" wildcard to the string. The next time the user enters partial data into the LOV input text field to then press the search icon, the input string is copied to the JobId field of the popup and the search is executed.



The LaunchPopupListener is defined as a method in a managed bean. The managed bean and the method signature can be created declaratively by clicking onto the triangle icon on the right side of the LaunchPopupListener property field. The icon opens a context menu with the "Edit" option to launch the managed bean selection / creation dialog.

```
package fnimphiu.sample;

import oracle.adf.view.rich.component.rich.input.RichInputListOfValues;
import oracle.adf.view.rich.event.LaunchPopupEvent;

import oracle.adfinternal.view.faces.model.binding.FacesCtrlLOVBinding;


public class InputLovBean {
  public InputLovBean() {
  }

 public void onLauchLov(LaunchPopupEvent launchPopupEvent) {
  String submittedValue =
     (String) launchPopupEvent.getSubmittedValue();
   //only perform query if value is submitted
   if (submittedValue!=null &&submittedValue.length()>0){
     RichInputListOfValues lovComp =
         (RichInputListOfValues)launchPopupEvent.getComponent();
      FacesCtrlLOVBinding.ListOfValuesModelImpl lovModel = null;
      lovModel =
         (FacesCtrlLOVBinding.ListOfValuesModelImpl)lovComp.getModel();
         lovModel.getCriteria().getCurrentRow()
                  .setAttribute("JobId",submittedValue+"%");
         lovModel.applyCriteria();
         lovModel.performQuery(lovModel.getQueryDescriptor());
    }
  }
}
```

The managed bean code access the launchPopupEvent, which is passed in by JavaServer Faces. From here, the list of value component - of type RichInputListOfValues - becomes accessible. The component's list of values model provides access to the underlying ViewCriteria. By default the ViewCriteria contains a single row with the attributes of the ViewObject that represents the model driven LOV. In the example above, the ViewCriteria has an attribute "JobId" that can be accessed and used to set the user provided search string. The rest then is to execute the search query.

RELATED DOCOMENTATION

⊠	
⊠	
⊠	