ADF Code Corner

062. How-to use the af:autoSuggestBehavior component tag with ADF bound data sources

twitter.com/adfcodecorner

Abstract:

The ADF Faces auto suggest behavior tag implements dynamic value suggest for input text fields, as many users know it from Internet applications, like Google. The ADF Faces af:autoSuggestBehavior tag refrences a managed bean that returns the list of suggested completion items based on the current user input. This article explains how to build auto suggest behavior with ADF bound data sources by example of ADF Business Components.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
19-OCT-2010

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

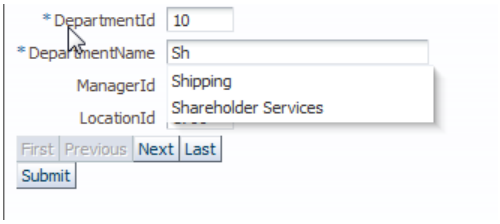
Introduction

The use case of auto suggest is sinple: A user types into an input text field and the system queries the underlying data source for possible completion strings.

The tag documentation for the ADF Faces af:autoSuggestBehavior components shows an example of a managed bean that exposes a public method with a single String argument to accept the suggest query event. In this article, we take it from there and show how developers can query auto suggest options through the ADF binding layer.

The sample used with this article provides text completion for the DepartmentName attribute of a View Object that is based on the HR schema's "Departments" table.

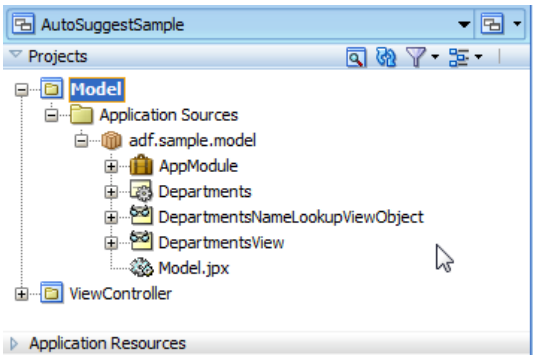
To qeuery the suggest strings, a second View Object is created. Using other – more complex data models than HR – it is more likely that the suggest data is queried from a lookup table. However, to demonstrate how to implement auto suggest using ADF Business Components queried data, this simple sample does it well enough.



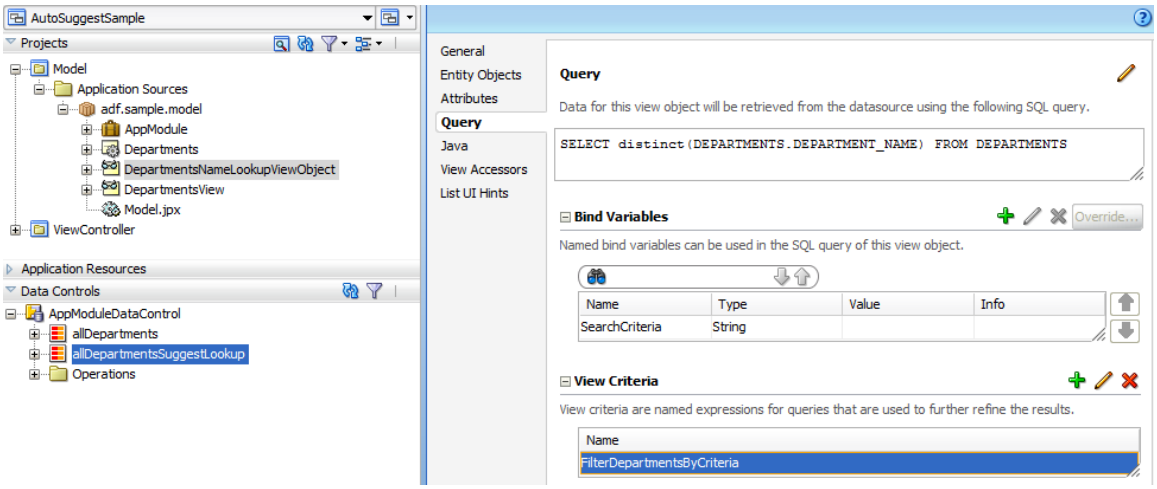
As shown in the image above, as soon as the user starts typing into the DepartmentName field, a popup is opened with suggestions queried from the database or – as it can be implemented using the appraoch outlined in this paper – from already fetched data in memory.

Building the Business Service

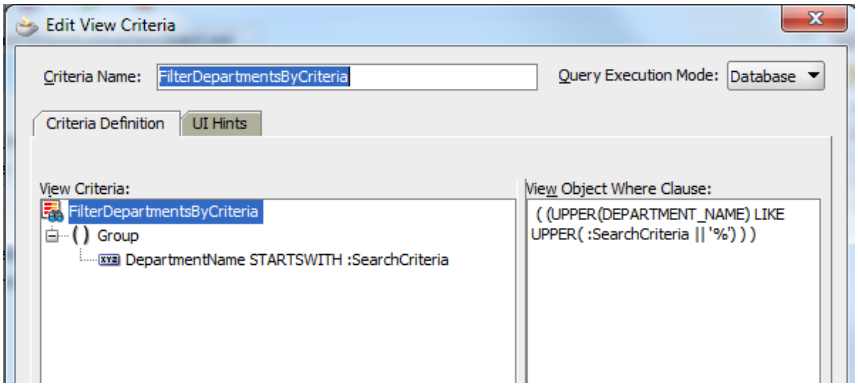
Two View Objects are needed for the auto suggest example: One that accepts data, and one that queries the data for possible completion strings. In the example, both View Objects query the same database table. In reality the two View Objects may as well query different tables.



The "DepartmentsView" View Object is the data source to update through the suggest component. The "DepartmentsNameLookupViewObject" is a View Object that provides the suggest items and that suffers from a German developer (me) trying to define a good name for it.

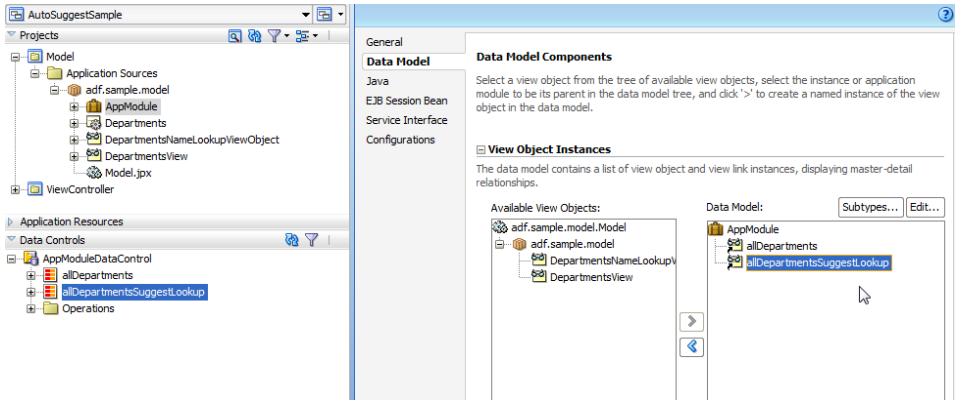


The "DepartmentsNameLookupViewObject" lookup View Object has a ViewCriteria defined that uses a bind variable "SearchCriteria" to filter its query by matching department names.

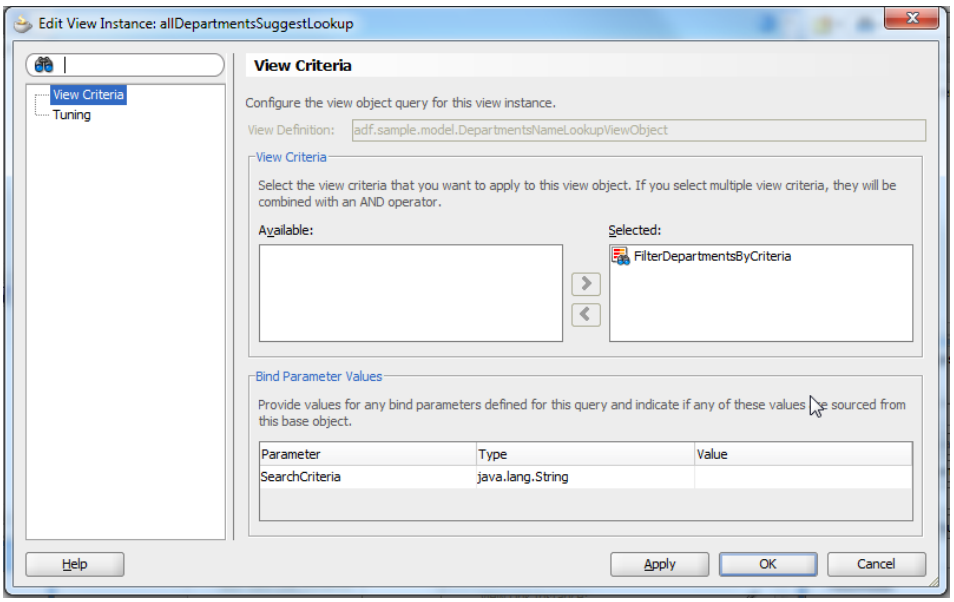


Note: The View Criteria dialog has a "Query Execution Mode" list that determines if the criteria filters including data in the database, or only data that has been queried before and that is cached for a user.

The ViewCriteria query defines the filter for all department names that start with the search string in the bind variable. The View Criteria is created by pressing the green plus icon for the View Object query category. The bind variable can be created on the fly when building the criteria.



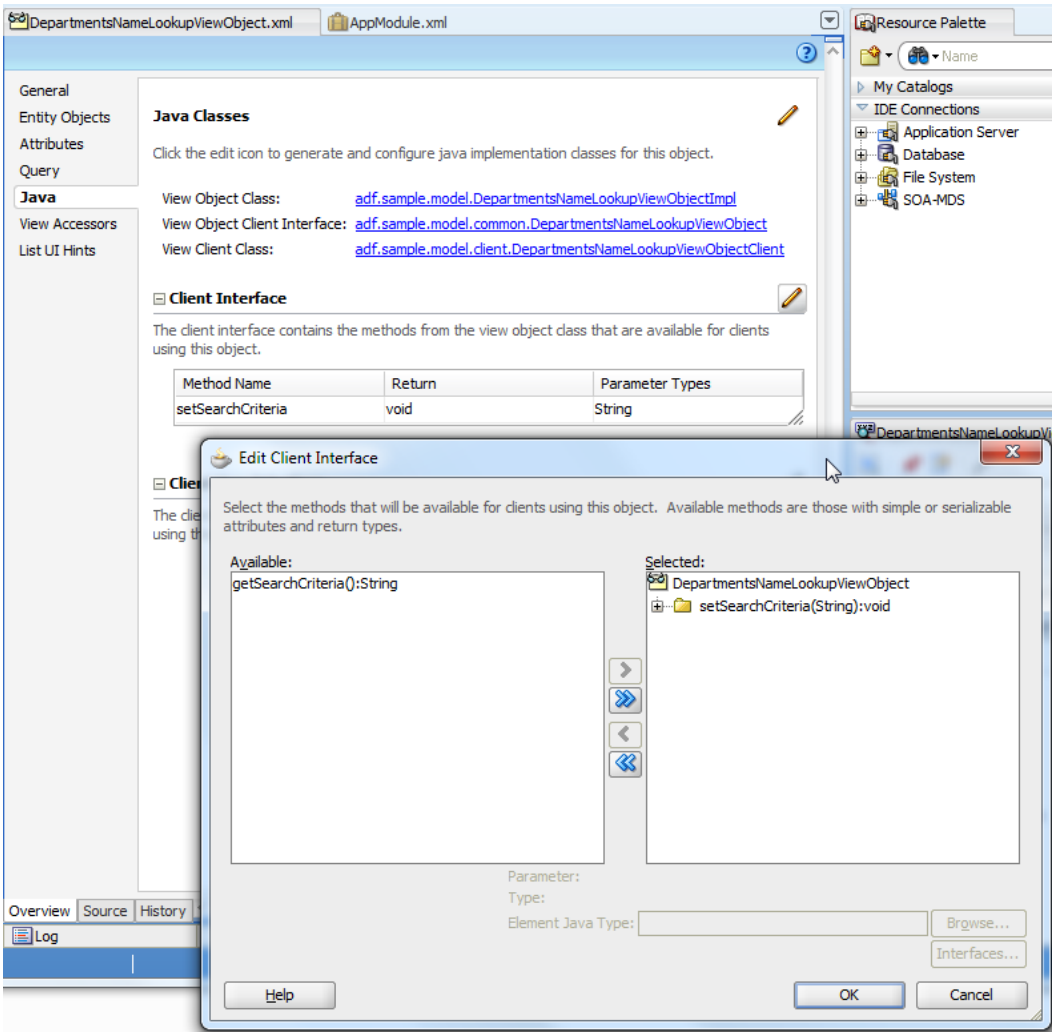
Double click the Application Module (AppModule in this example) to rename the View Object instance so they better describe what they do. In the example, the instance based on the DepartmentView View Object is renamed to "allDepartments" as it queries and updates all departments, and the other is renamed to "allDepartmentsSuggestLookup" to indicate that this is a lookup service.



Select the " allDepartmentsSuggestLookup" View Object instance and choose the Edit option to apply the ViewCriteria to the instance. This way the View Object instance is always filtered by the ViewCriteria. The bind variable value can be left empty as this will be dynamically passed in.

Finally, expose the setter method for the bind variable on the View Object client interface so it becomes accessible from ADF. Open the View Object editor with a double click and select the Java category.

Click the pencil icon next to "Java Classes" to create an Impl class for the View Object. In the opened dialog, check the "Create View Object class" option and ensure the "Include bind variable accessors" option is checked too. This creates the View Object implementation class and you can now Ok the dialog. Still in the Java category, next, click the pencil icon next to the "Client Interface" section. The dialog shown below is opened for you to select the setter method for the bind variable and move it to the list of selected client methods.

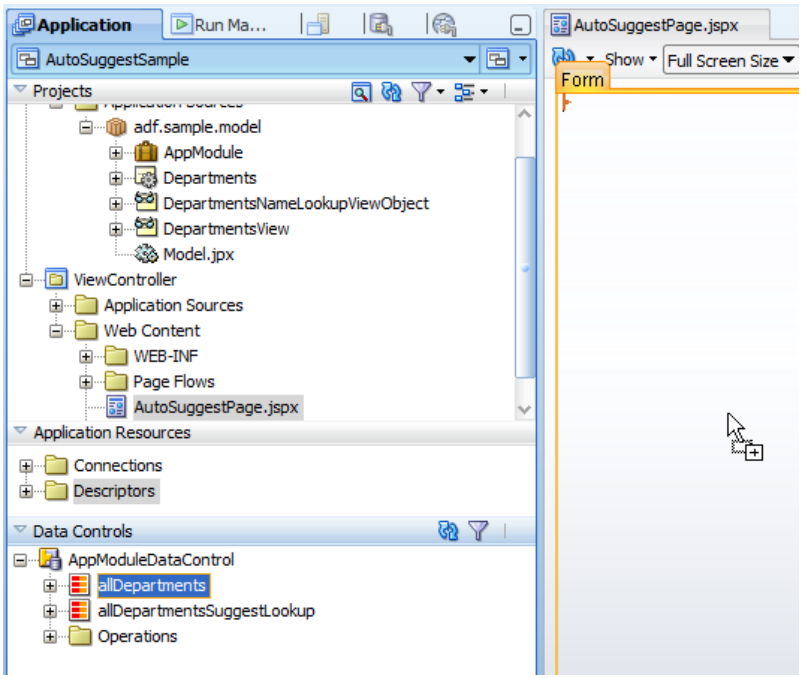


Note: If you have the DataControls panel open, hit the refresh icon to ensure it shows the latest and greatest state of the ADF Business Component model.

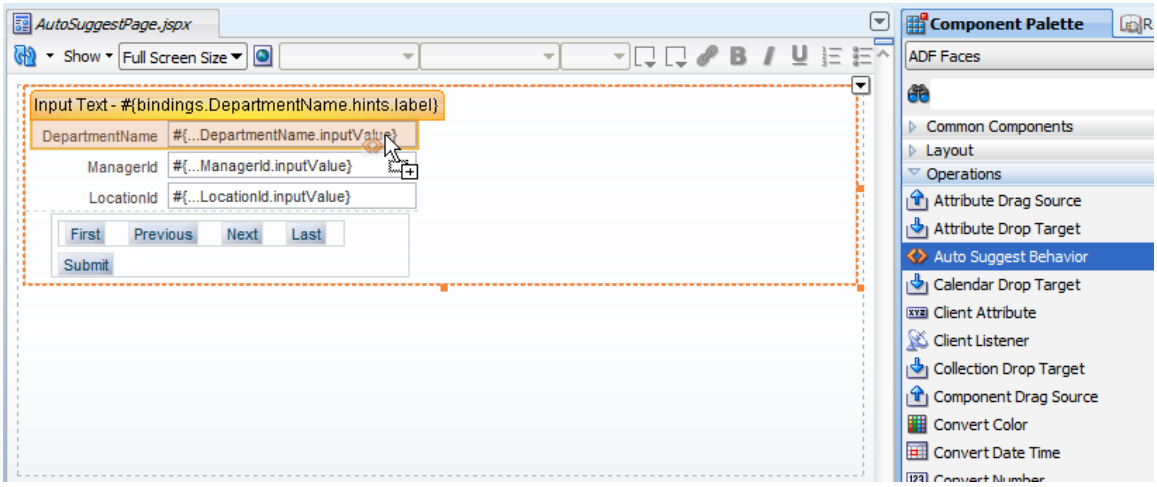
Building the View Layer

With the business model in place, it is time to build the input form with the suggest behavior. From the Data Controls panel, drag and drop the "allDepartments" View Object as a form onto the JSF page you created in the project.

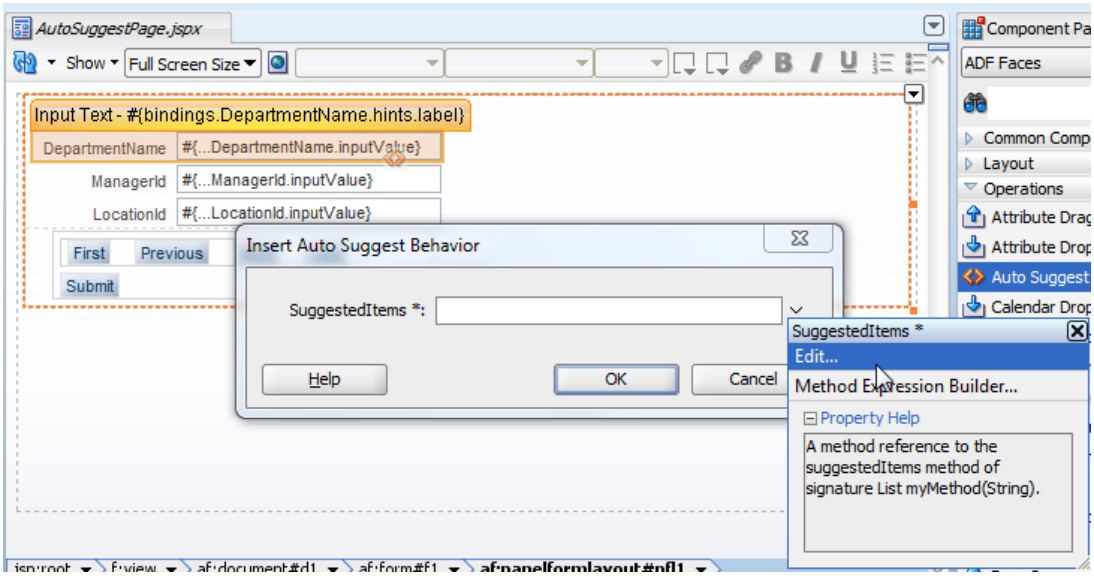
Note: Just in case you are new to Oracle JDeveloper, to create a new JSF page, with the View Layer project selected in the Application Navigator, choose File | New | Web Tier | JSF | JSF Page from the menu and New Gallery.



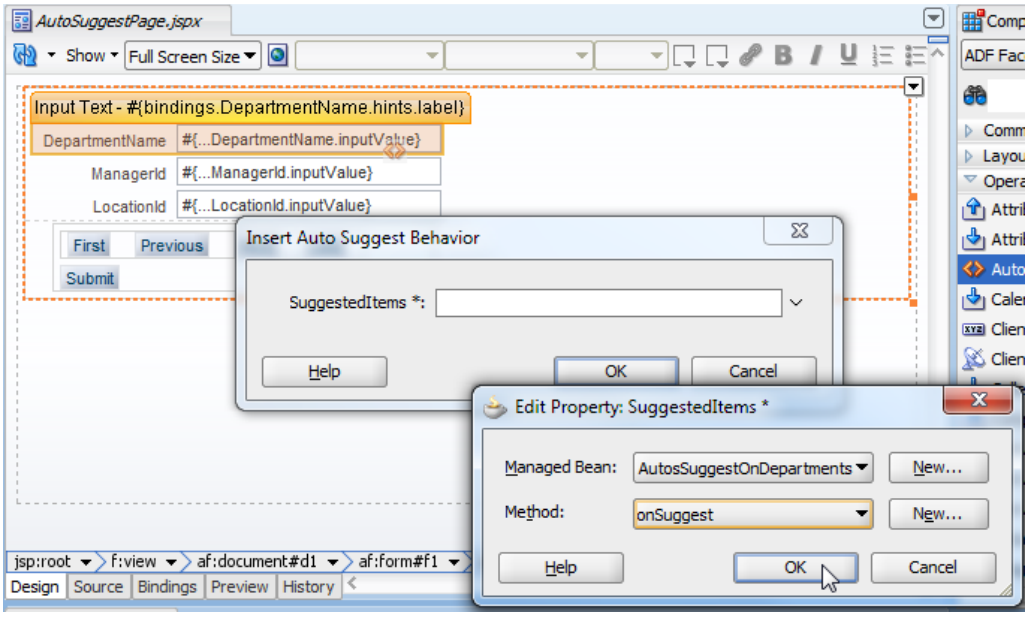
When the form is built, open the Component Palette and expand the "Operations" node as shown in the image below. The "Auto Suggest Behavior" tag is shown as the 3rd entry in the list. Drag the suggest tag from the component palette and drop it on the "DepartmentName" field as shown in the image below.



In the opened dialog press the "down arrow" icon next to the "SuggestItems" field and choose the "Edit" option.



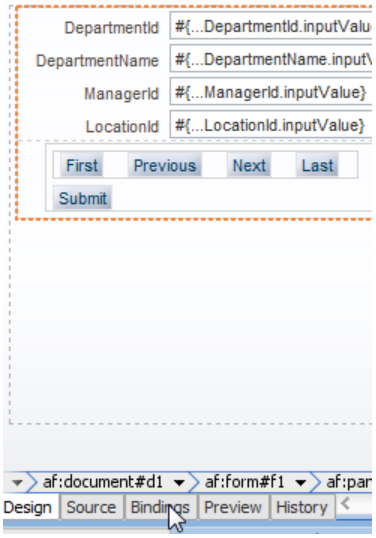
Either create a new managed bean and method, or select an existing bean and method to populate the suggest list. The method signature is expected to accept a single String argument. The method is supposed to return List<SelectItem>. In the managed bean, use the input argument to filter the list of return items.



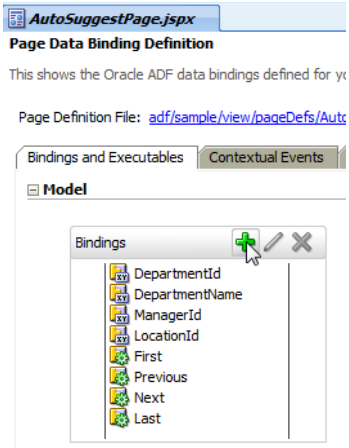
Building the ADF binding layer

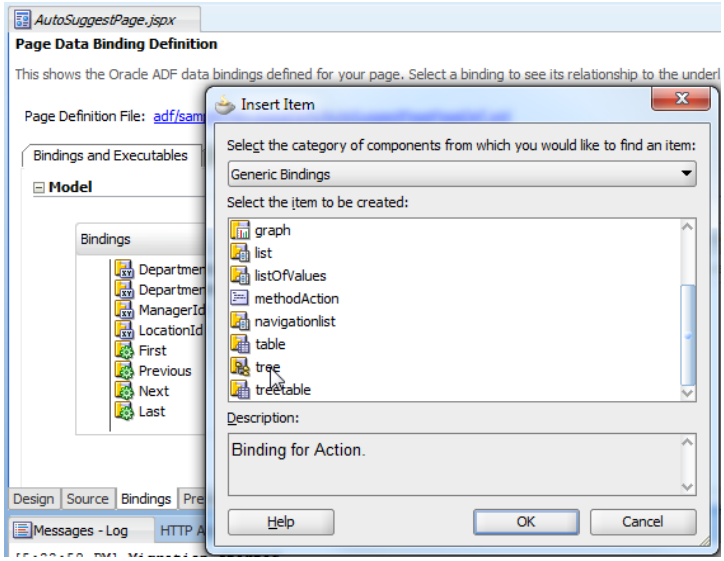
To build the suggest list through the ADF binding layer, we need to create a tree binding to populate the list and a method binding to set the bind variable value to the String argument passed into the managed bean method.

To access the binding layer, click the "Bindings" tab at the bottom of the visual page editor.

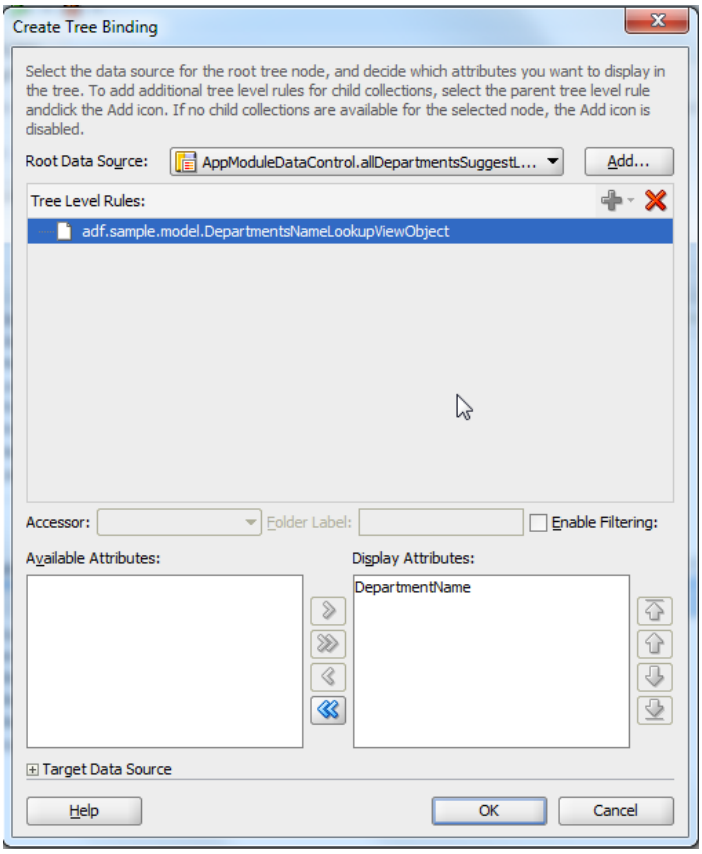


In the "Bindings" section, press the green plus icon and choose "tree" as the binding artifact to create.



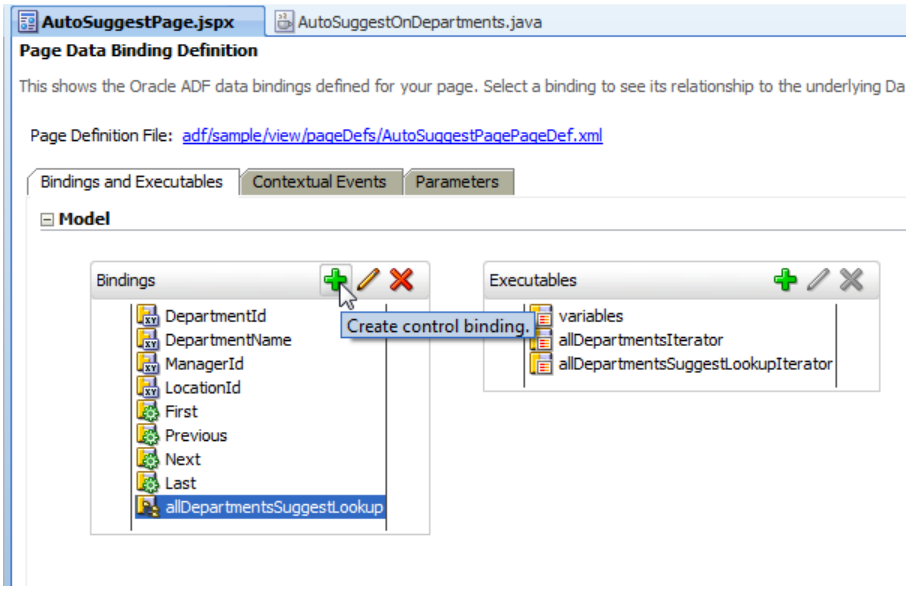


Select the "allDepartmentsSuggestLookup" View instance after pressing the "Add" button and then press the green plus icon (its grayed out in the image below) to create a tree binding level. The only attribute to choose is the attribute that should be displayed as the label in the suggest list.

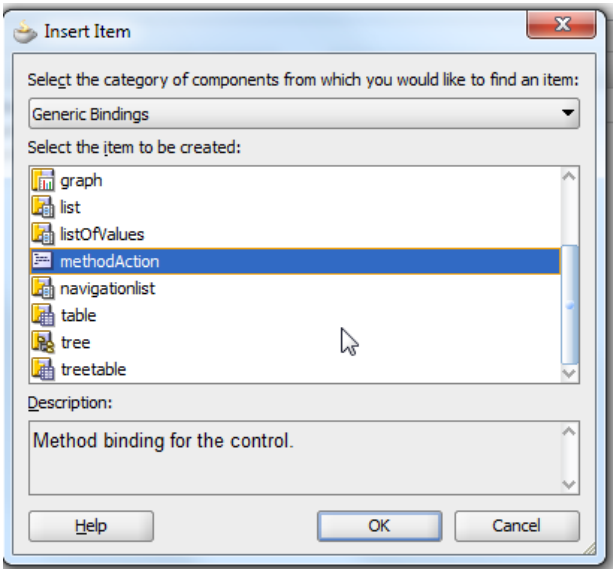


In the same "Bindings" section, press the green plus icon again to create a method binding for the client method you created to set the bind variable value.

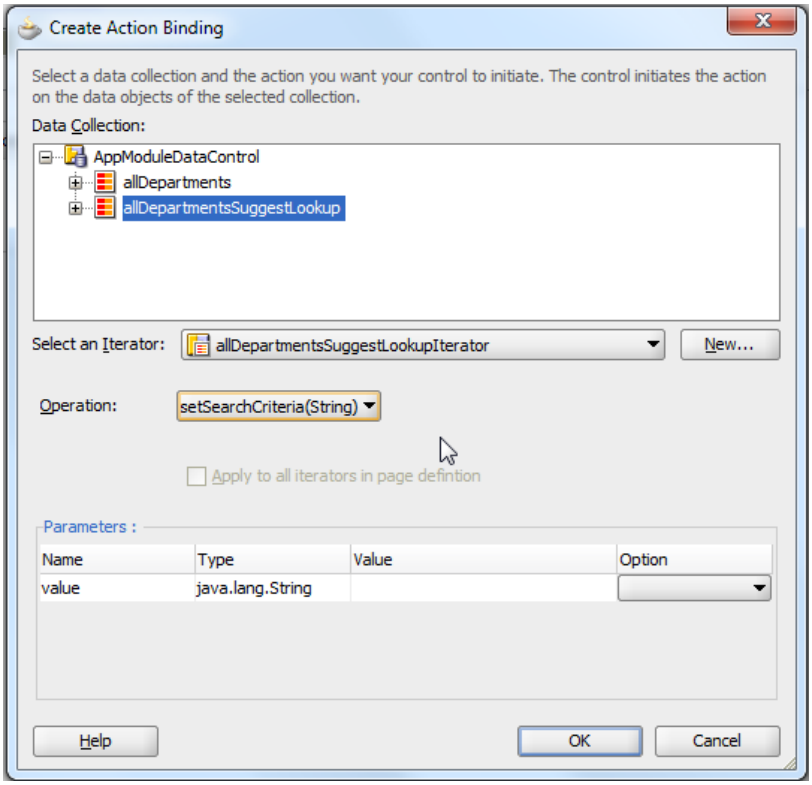
Note: by default the iterator that is created for the tree binding fetches 25 rows. You may want to set the iterator's RangeSize property to -1 to see a larger list of suggest items.



Choose "methodAction" in the list of bindings.



Select the "allDepartmentsSuggestLookup" entry under the Data Collection structure and then select "setSearchCriteria(String)" in the "Operation" list. The method parameter can be left empty. Just note that he parameter name is "value", information needed when implementing the managed bean method.



Managed Bean Auto-Suggest Handling

The managed bean method that is referenced from the af:autoSuggestBehavior component on the DepartmentName field is shown below. The source code is commented for you to understand what it does.

```
import java.util.ArrayList;
import java.util.List;

import javax.faces.model.SelectItem;

import oracle.adf.model.BindingContext;
import oracle.binding.BindingContainer;
import oracle.binding.OperationBinding;

import oracle.jbo.Row;
import oracle.jbo.uicli.binding.JUCtrlHierBinding;
import oracle.jbo.uicli.binding.JUCtrlValueBindingRef;

public class AutoSuggestOnDepartments {
  public AutoSuggestOnDepartments() {
    super();
 }


 public List onSuggest(String string) {
   //get access to the binding context and binding container at runtime
   BindingContext bctx = BindingContext.getCurrent();
   BindingContainer bindings = bctx.getCurrentBindingsEntry();

   //set the bind variable value that is used to filter the View Object
   //query of the suggest list. The View Object instance has a View
   //Criteria assigned
   OperationBinding setVariable =
          (OperationBinding) bindings.get("setSearchCriteria");
   setVariable.getParamsMap().put("value", string);
   setVariable.execute();

   //the data in the suggest list is queried by a tree binding.
   JUCtrlHierBinding hierBinding =
       (JUCtrlHierBinding) bindings.get("allDepartmentsSuggestLookup");
   //re-query the list based on the new bind variable values
   hierBinding.executeQuery();

   //The rangeSet, the list of queries entries, is of type
   //JUCtrlValueBndingRef.
   List<JUCtrlValueBindingRef> displayDataList =
                                      hierBinding.getRangeSet();


   ArrayList<SelectItem> selectItems = new ArrayList<SelectItem>();
   for (JUCtrlValueBindingRef displayData : displayDataList){
```

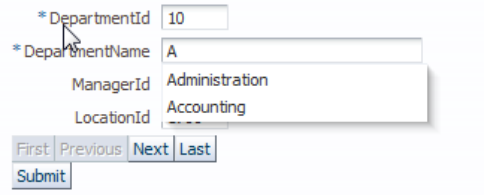
```
      Row rw = displayData.getRow();
      //populate the SelectItem list
      selectItems.add(new SelectItem(
                              (String)rw.getAttribute("DepartmentName"),
                              (String)rw.getAttribute("DepartmentName")));
       }
   return selectItems;
 }
}
```

Sample Download

You can download the sample shown in the screen shots from the ADF Code Corner website, where it is sample #62. The workspace was developed with Oracle JDeveloper 11.1.1.3 and requires you to change the database connection to point to a local database with the HR schema installed and enabled. ADF Code Corner can be accessed from here

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

Run the JSPX page, clear the DepartmentName field and start typing to see the suggest list.



RELATED DOCOMENTATION

☒	
☒	
☒	