

ADF Code Corner

90. Filtering ADF bound lists

ORACLE
CODE CORNER



twitter.com/adfcodecorner

Abstract:

Falling into the category of dependent lists, is the use case of reducing list entries based on previous user entries. The ADF `JUCtrlListBinding` class exposes a method `filterList` that when called constructs a view criteria for the underlying list collection.

This blog article shows two examples for using this API. One sample filters the available value list in a select many shuttle component. The other sample filters a single select list based on a previous user entry in an input text field.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
30-OCT-2011

Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

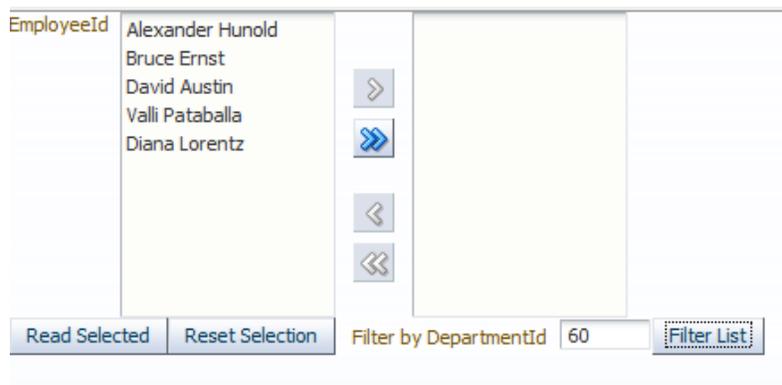
Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: <http://forums.oracle.com/forums/forum.jspa?forumID=83>

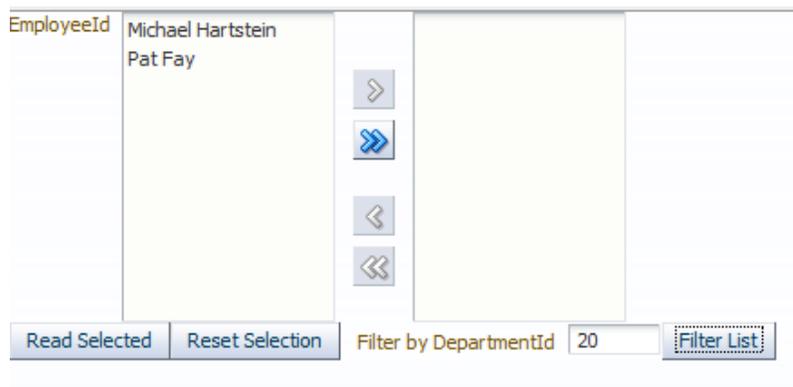
Introduction

A common use case is to filter list data dependent on user provided data entries. This article explains how to filter the `af:selectManyShuttle` and `af:selectOneChoice` list.

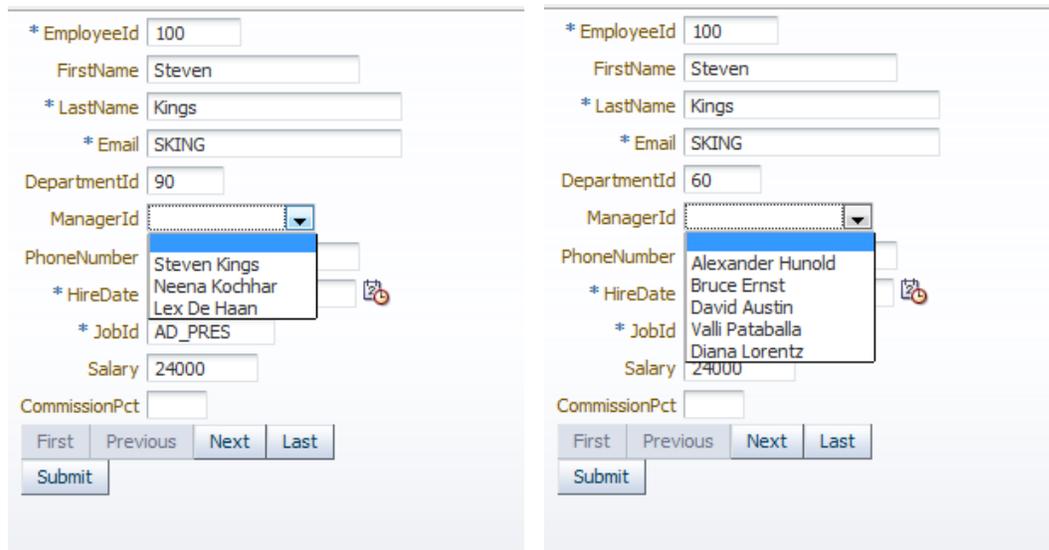
The first sample shows an input text field that if you provide a valid department Id in, filters the list of all employees shown in the `af:selectManyShuttle` accordingly. In the image below, the list is filtered for employees in department 60.



Changing the department Id to 20, as shown below, changes the list of employees to only show employees that are in department 20.

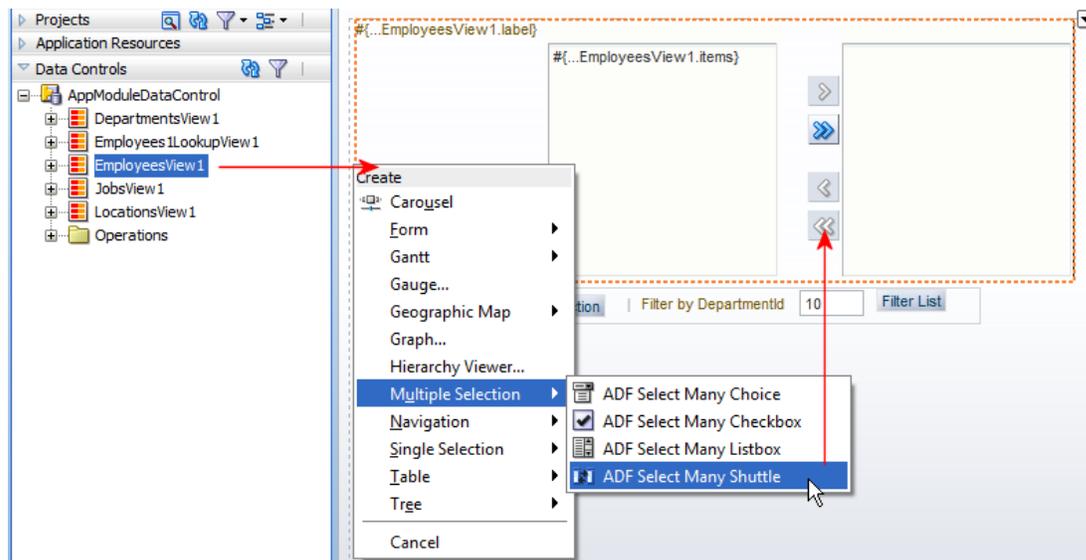


The second example uses the **autoSubmit** functionality on an input text field to reduce the list of employees shown in the select one choice component.



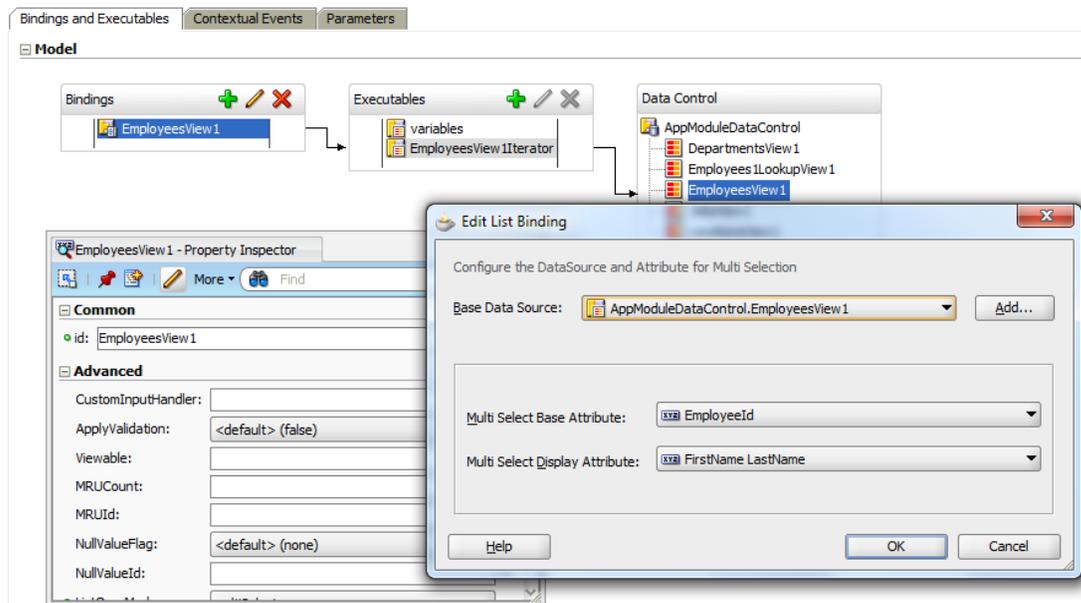
Filtering the ADF Faces Select Many Shuttle

To add ADF bound ADF Faces select many shuttle components to a view, you drag a collection – EmployeesView1 in this sample – from the Data Controls panel onto the page and choose **Multiple Selection | ADF Select Many Shuttle** from the popup menu.



Implicitly this creates a list binding in the **PageDef** binding file of the view. The list binding is configured on the `af:selectManyShuttle` component to produce the list and also hold the selected values.

To update the business service with the user selection, you usually listen for a value change event or program the submit button action to access the list binding's `JUCtrlListBinding` instance. The image below shows the list binding configuration used in this sample. The employee first name and last name is displayed to represent **EmployeeId** as the list data.



Though not in the focus of this article, the action code associated with the **Read Selected** command button shows how to access the user selected data. Similar, the **Reset Selection** shows how to unset the user selection, for example to refresh a form for the next user selection.

The method `filterListAction` shown below is invoked by the **Filter List** button. It access the list binding – named "EmployeesView1" – in the PageDef file and creates a `HashMap` with the filter keys and data. The key in this example is "DepartmentId" to indicate that the department id attribute is used for filtering the list. The value of the filter is provided through the user entry. Having configured the sample to access the HR schema, you want to try 20, 30, 50, 60, 80 (**Note** that for the sample I did not implement validation for the user entry. So **garbage in produces garbage out**)

When applying the `HashMap` content as a filter to the `JUCtrlListBinding` instance, ADF creates a view criterion that then is applied to the list query.

(Code below is taken from `SelectManyShuttleBean.java` of this sample)

```
public String filterListAction() {
    BindingContext bctx = BindingContext.getCurrent();
    BindingContainer bindings = bctx.getCurrentBindingsEntry();
    JUCtrlListBinding listBinding =
        (JUCtrlListBinding)bindings.get("EmployeesView1");
    HashMap m = new HashMap();
    try {
        m.put("DepartmentId", new Number(departmentId));
    } catch (SQLException e) {
        //TODO handle exception gracefully
        e.printStackTrace();
    }
}
```

```

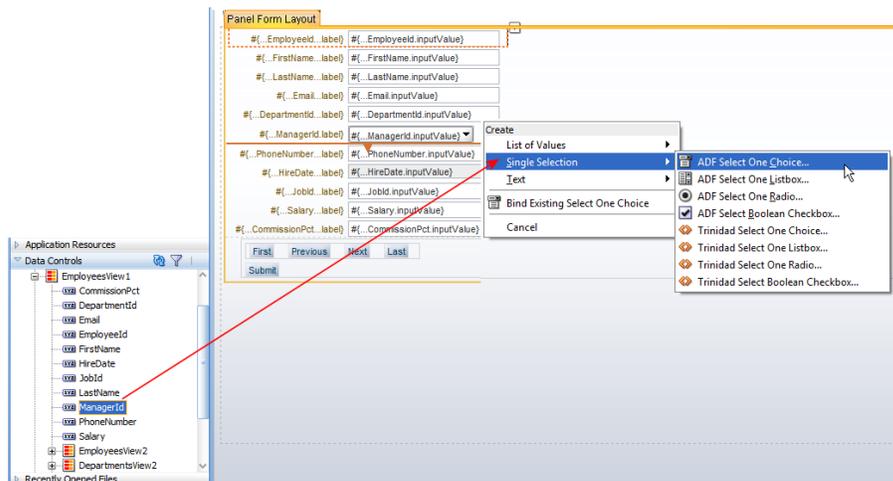
}
listBinding.getDCIteratorBinding().getViewCriteria().clear();
//remove selected values before filtering
listBinding.clearSelectedIndices();
listBinding.filterList(m);
return null;
}

```

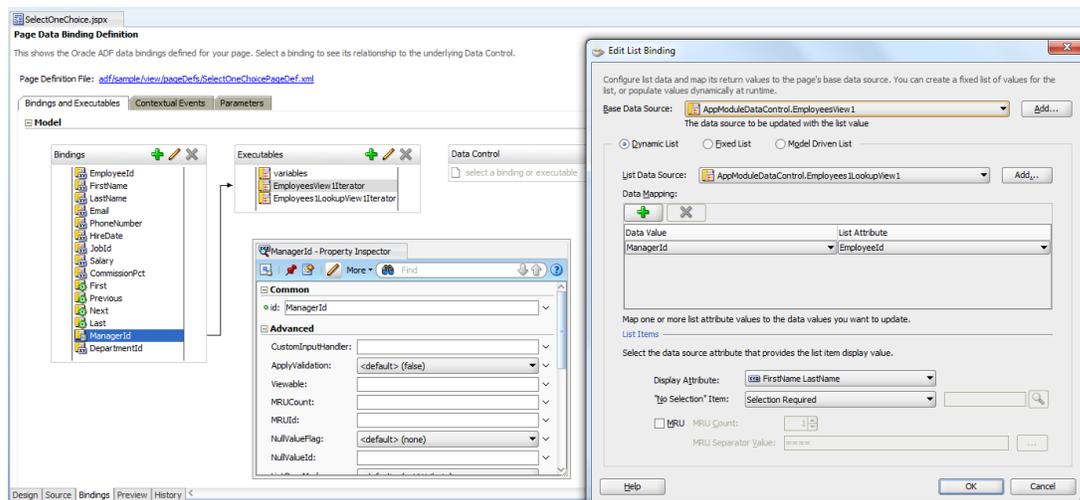
Note: The code above clears the view criteria added to a View object before applying the filter. If the View Object has an existing view criteria applied that should be kept, change this code to only remove the criteria added last.

Filtering the ADF Faces Select One Choice

The same filtering used with ADF Faces select many components bound to ADF can be used for single select bindings. In the image below, an ADF Faces select one choice is dropped into an Employee edit for to replace the **ManagerId** input field with an `af:selectOneChoice`.



The list is chosen to be a **Dynamic List** (see image below) and wired up to a View Object that queries the Employees table.



The **DepartmentId** field **autoSubmit** property is set to **true** so that a change in the department value automatically updates the model. The **ManagerId** `af:selectOneChoice` **PartialTriggers** property is configured to point to the **DepartmentId** field ID to establish PPR refresh on value change. Set the **immediate** property of the

Note: Check the **ChangeEventPolicy** property of the iterator binding the forms is bound to. If the setting is "ppr", set it back to "none".

When the department id value is changed by a user, the new department value is used to filter the list displayed in the `af:selectOneChoice` component. The code is exactly the same as used in the select many shuttle sample.

(Code Below is taken from **SelectListBean.java** of the Sample)

```
private String filterListAction(String departmentValue) {
    BindingContext bctx = BindingContext.getCurrent();
    BindingContainer bindings = bctx.getCurrentBindingsEntry();
    JUCtrlListBinding listBinding =
        (JUCtrlListBinding) bindings.get("ManagerId");
    //clear selection to avoid selectd values that don't match the
    //filter criteria
    HashMap m = new HashMap();
    try {
        m.put("DepartmentId",
            new oracle.jbo.domain.Number(departmentValue));
    } catch (SQLException e) {
        e.printStackTrace();
    }

    listBinding.getDCIteratorBinding().getViewCriteria().clear();
    listBinding.filterList(m);
    return null;
}
```

Note: The sample also uses a JSF component binding ("Binding" attribute reference to a managed bean) to trigger the initial filtering of the list after page load. The code is contained in the **SelectListBean.java** source.

Sample Download

The Oracle JDeveloper 11.1.1.4 workspace for the sample shown in this article can be downloaded as sample #90 from ADF Code Corner:

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

Configure the database connection of the sample to access the Oracle database HR schema before running **SelectManyShuttle.jspx** or **SelectOneChoice.jspx** from the adfc-config.xml task flow.

Known Issues

Oracle JDeveloper 11g R2 has a known issue with this solution, which I filed for development investigation. So for now, use this sample with Oracle JDeveloper 11g R1 (11.1.1.x).

RELATED DOCUMENTATION

<input type="checkbox"/>	
<input type="checkbox"/>	
<input type="checkbox"/>	