ADF Code Corner

59. How-to filter ADF bound tables by date range

twitter.com/adfcodecorner

Abstract:

A reoccuring requirement in ADF applications is to filter tables by date range, which is not provided by default for Oracle ADF bound tables. This blog article outlines a solution for implementing table filtering by date range based on ADF Business Components View Criteria. At runtime, users will select from two date pickers to define the start and end date of a range to query, where the query could be designed to be in memory or RDBMS based.

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

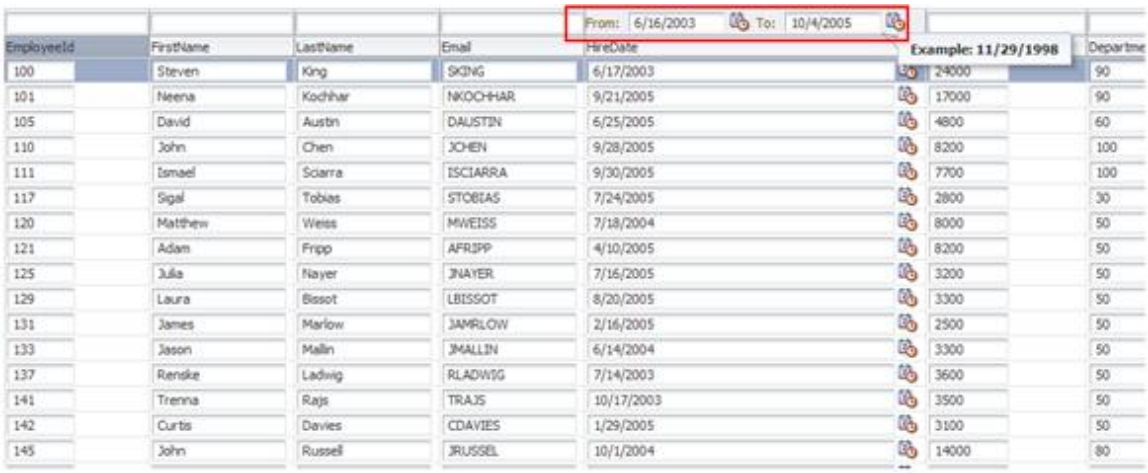
Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

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

To filter ADF bound tables by date range, this solution uses a View Criteria that filters the query of a View Object if a start date and end date is provided through bind variables. The bind variables are set from a custom table query listener, which decorates the default listener, using method bindings in the PageDef file. The result at runtime looks as shown in the image below.



The filter fields also have format converters and validators defined that reference the underlying ADF BC attribute to ensure correct user input. To build this sample, the following steps are needed

- A view criteria needs to be defined on the View Object to filter the query for a start date and end date passed in as bind variables.
- The view criteria needs to be added to the View Object instance in the ADF BC data model, which is defined on the Application Module
- Setter methods need to be exposed on the ADF BC client interface so method bindings can be created
- A table needs to be created from the View Object with the table filter option enabled

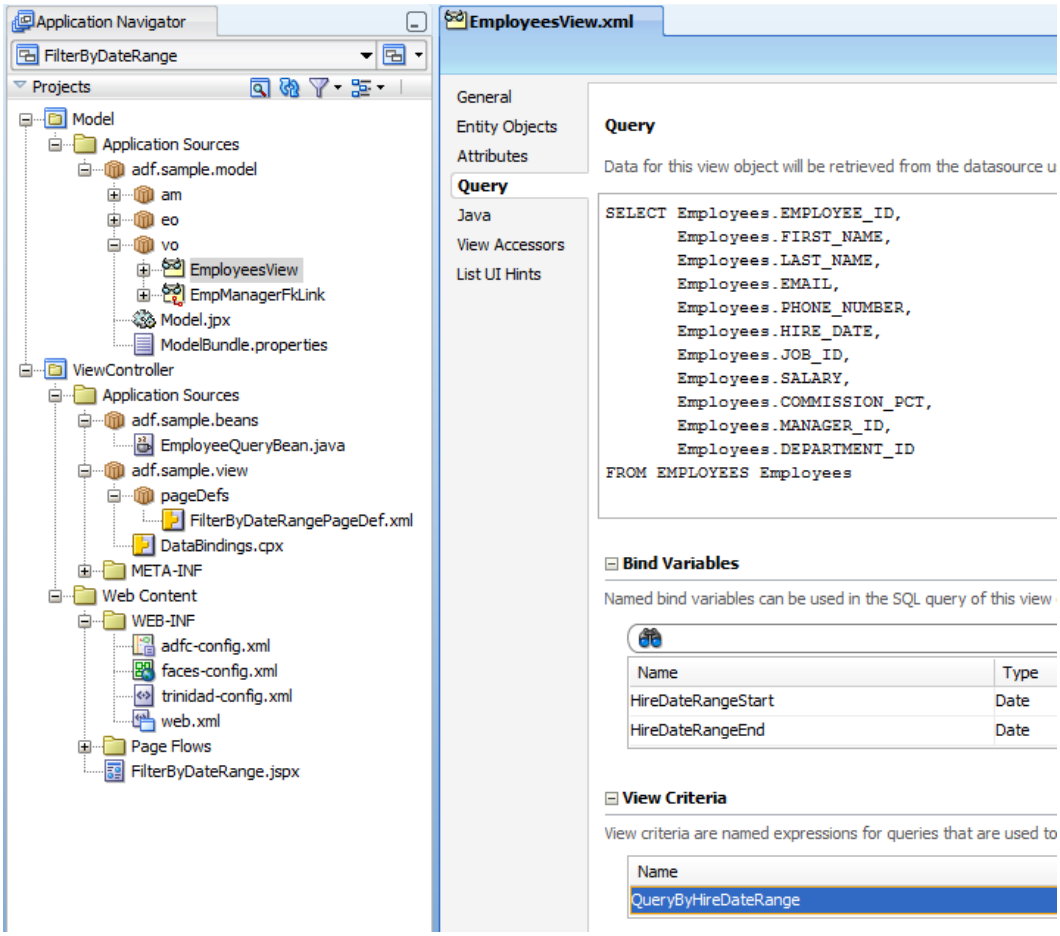
- The filter for the date field, HireDate in the sample, needs to be customized with a second date picker. Both date pickers reference temporary filter criteria attributes
- An attribute binding needs to be created for the date field (Hiredate) so the validation information can be obtained from the table filter date pickers

Note: The sample requires Oracle JDeveloper 11g R1 PS3 (11.1.1.4) or later because of a rendering of the table header in previous releases. As the time of writing, Oracle JDeveloper 11.1.1.4 is not publicly available, but is expected to be available shortly.

Preparing the ADF Business Component model

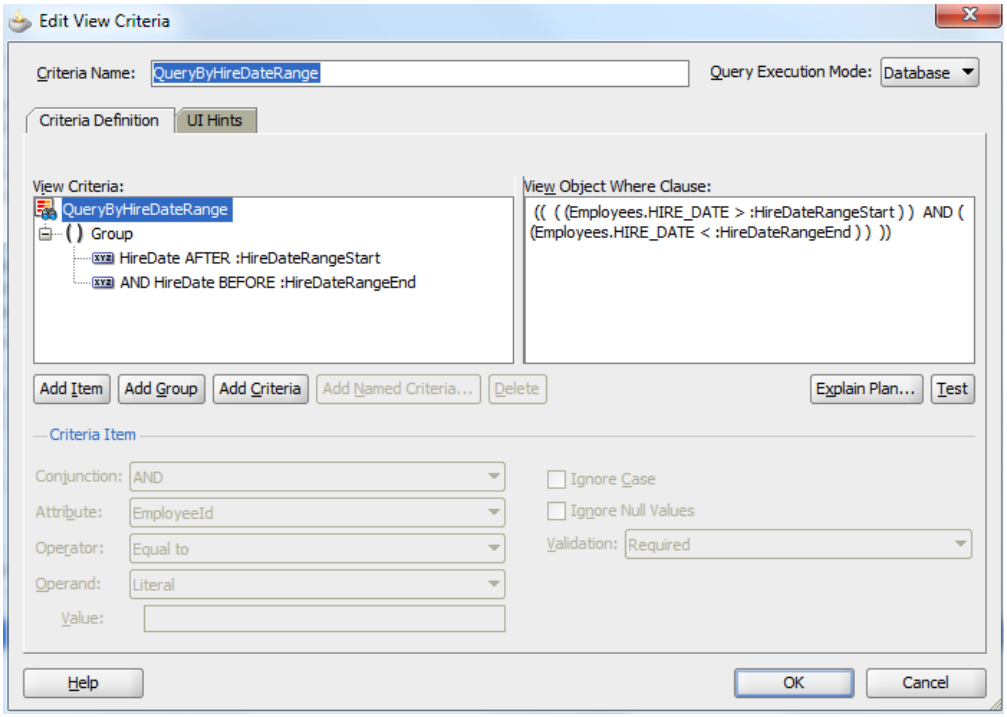
The idea behind this sample is to use a View Criteria to filter a data range if a start and end date is provided by the bind variables. The View Criteria itself is applied to an instance of the View Object using the Application Module data model editor for View Object instances.

To create a View Criteria, open the View Object with a double click and choose the "Query" category. In the View Criteria section, press the green plus icon to define the new filter condition (which at runtime will be applied as a where clause).

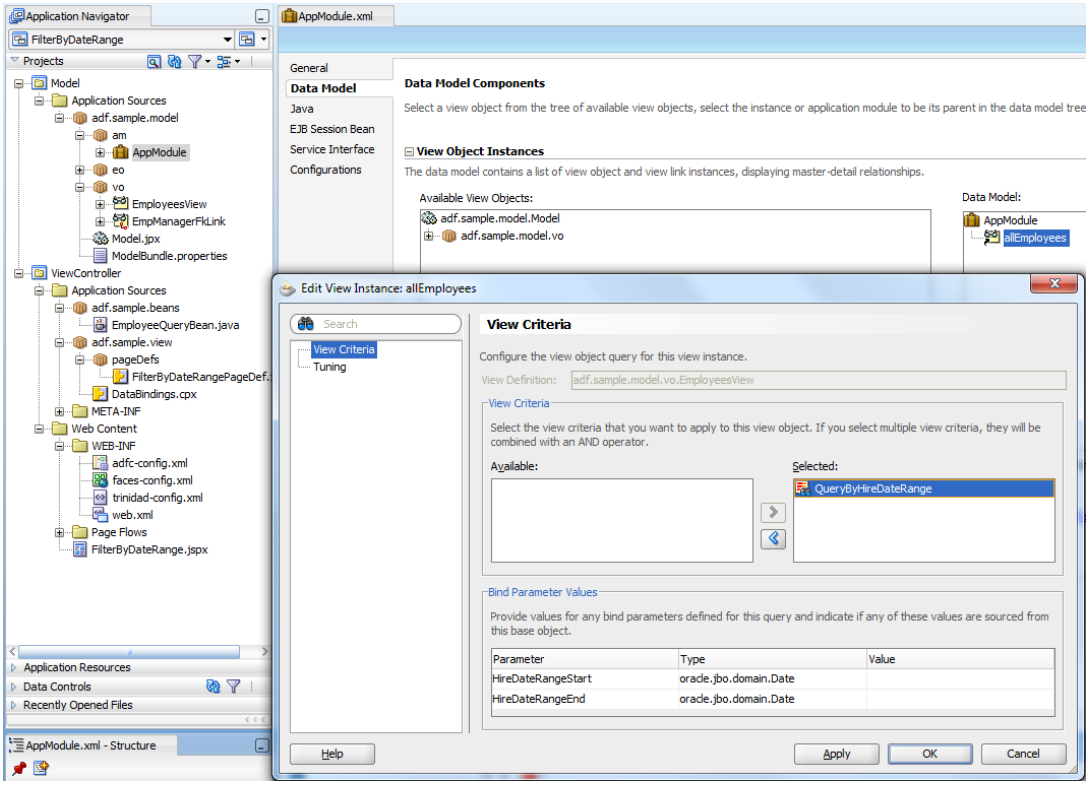


The View Criteria is defined on the Employees View in the sample and uses two bind variables of type Date to filter the employee query by HireDate. Note the use of AFTER and BEFORE in the View

Criteria definition to build the range. The bind variables are optional as it must be possible to run the View Object without specifying a date range.



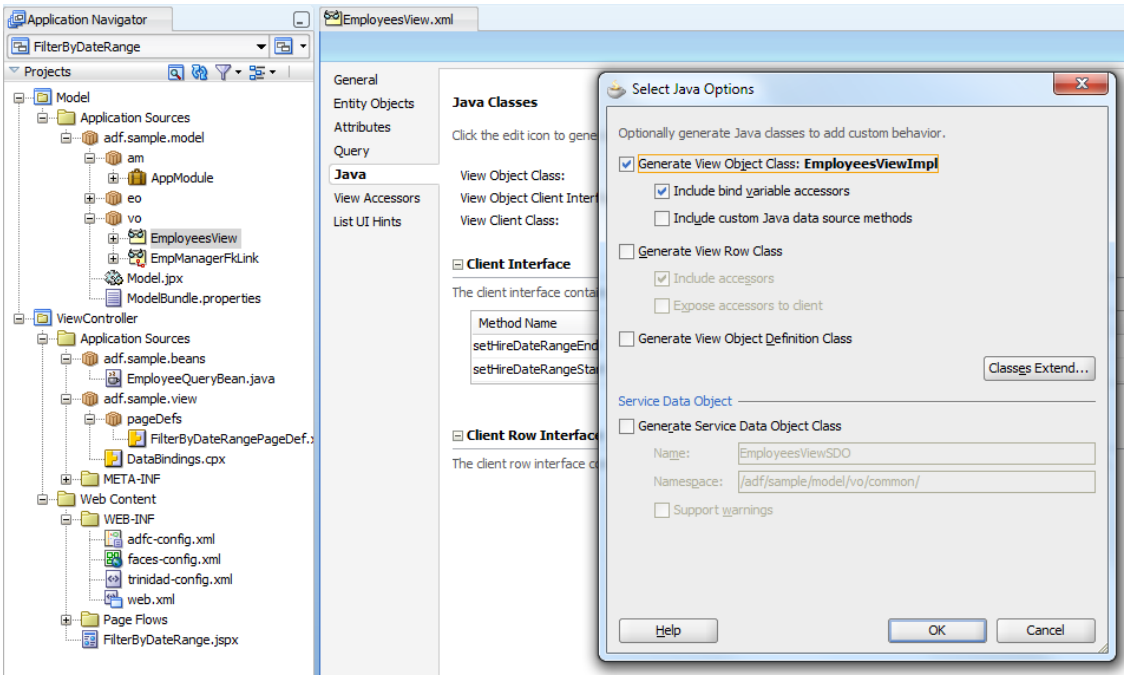
Note that using the "Query Execution Mode" list box, you can define the ViewCriteria to filter data that already is queried and available in memory or data queried from the database. In the screen shot above, the View Criteria filters data queried from the database.



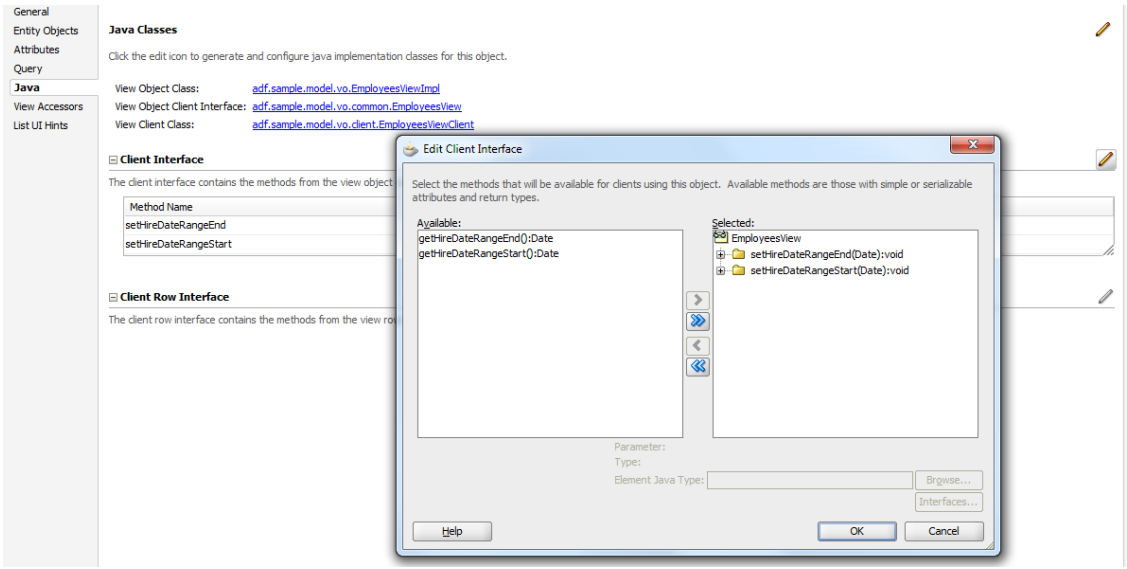
To assign the View Criteria to a View Object instance, select and double click the Application Module definition in the Application Navigator, which opens the AM editor. Select the View Object instance – allEmployees in the sample – to add the View Criteria to and press the Edit button. In the opened dialog, select the view criteria and toggle it to the "Selected" list.

Note: You don't need to provide values for the bind variables unless you want to hard code a date range, which is not what we want for this use case.

Again, open the View Object and select the Java option in the opened dialog. Check the option to create a Impl file for the View Object and then, in a second step, Create a Client Interface method for the bind variable setter methods.



The image below shows the client interface dialog to specify the bind variable values. Exposing the bind variable setter methods allows us to pass the start and end date values from the ADF client.

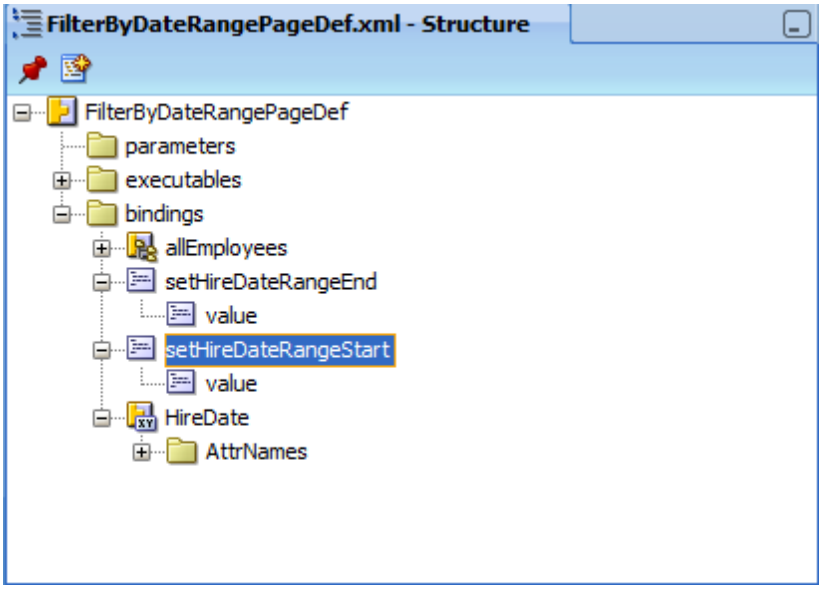


The model is prepared and can be tested using the ADF Business Component tester.

Building the table filter

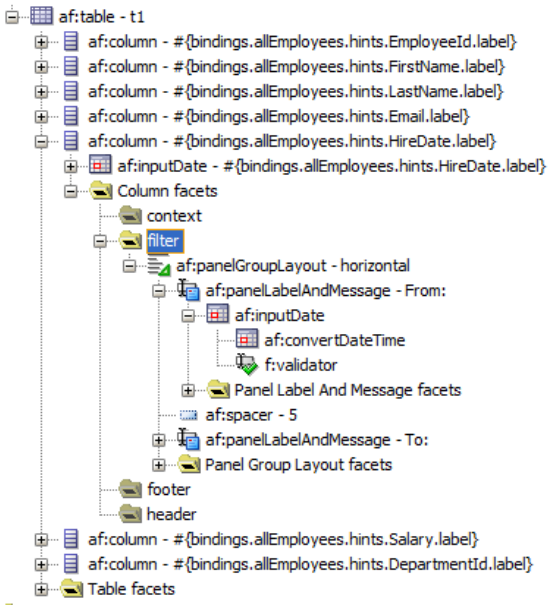
In the following, the steps to create the table filter for the date range on the HireDate attribute is outlined. To build the filter, the following needs to be done

- Create a table with the filter option selected. This creates a search binding in the PageDef executable section, which ADF uses to pass the filter query to the model
- Add an additional date picker to the existing HireDate filter
- Use additional layout components to align the two date pickers.
- Customize the QueryListener defined on the table

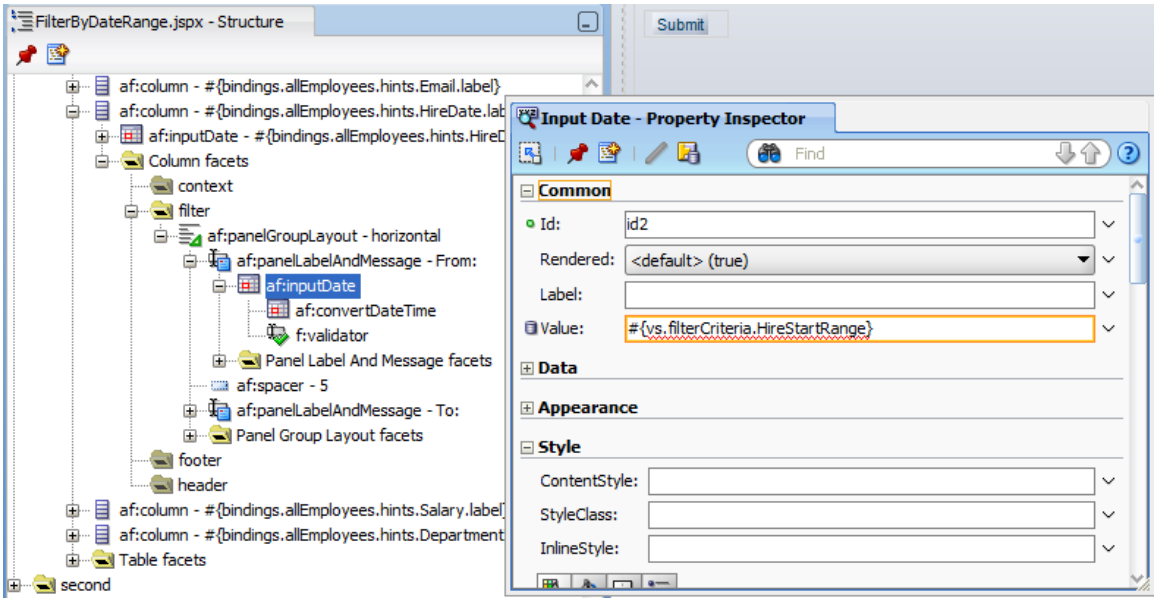


As shown in the image above, you need to create two method bindings for the setter methods of the bind variables. For this, select the "Bindings" tab on the JSPX visual editor window and press the green plus icon in the Binding section. Choose "method action" from the list of generic bindings and select the View Object that exposes the client methods. Choose the client method and ok the dialog, which then creates the method binding with an entry for the argument – which by default has an argument name of "value" – to pass to the client method.

The attribute binding shown in the image above is needed so we can add a validator tag to the date picker that ensures correct data entry by the user.



The table filter for the HireDate component is created out of an af;panelGroupLayout (horizontal) container, two af:panelLabelAndMessage component to show the labels "From" and "To", two date picker components to select the start and end date of the range, as well as a "convert date" and "validator" tags.



The two date picker components write their values to the filterCtriteria, which is accessible from the "vs" variable "filterCriteria" entry. When you open the Expression Language builder for the "Value" property, you can see the "vs" entry under the "JSP Object" node.

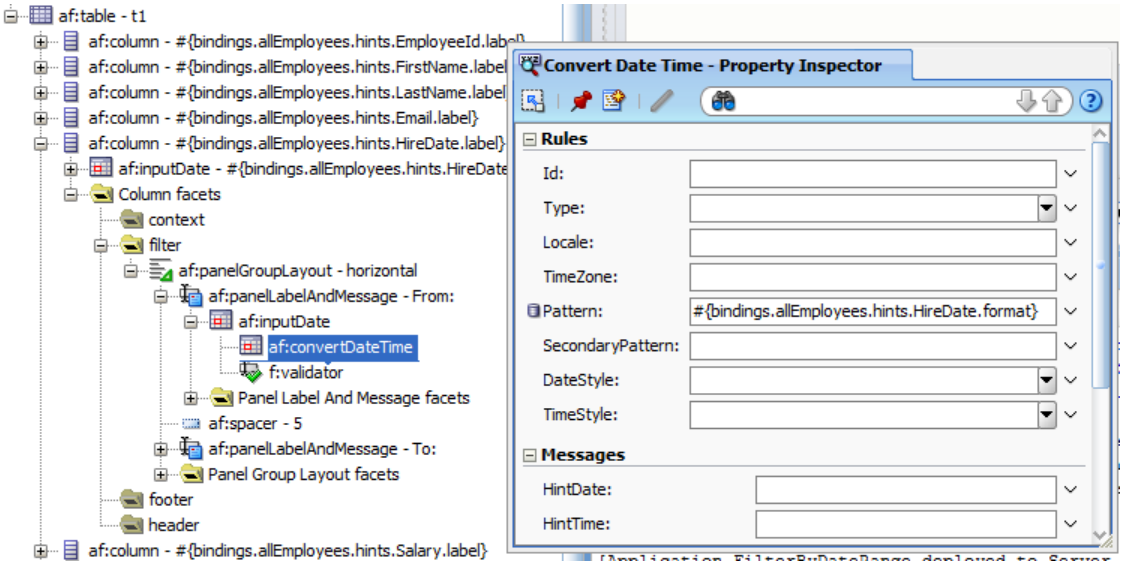
The "filterCriteria" option shows all the queryable attributes defined on the View Object, including one for HireDate. However, we can ignore the HireDate entry and instead use our own attribute names. Yes, we can make them up.

The filter criteria is a map that can take any attribute you want to pass in. For the date range filter use case, we define two custom attributes that don't exist as attributes on the View Object: "HireStartRange"

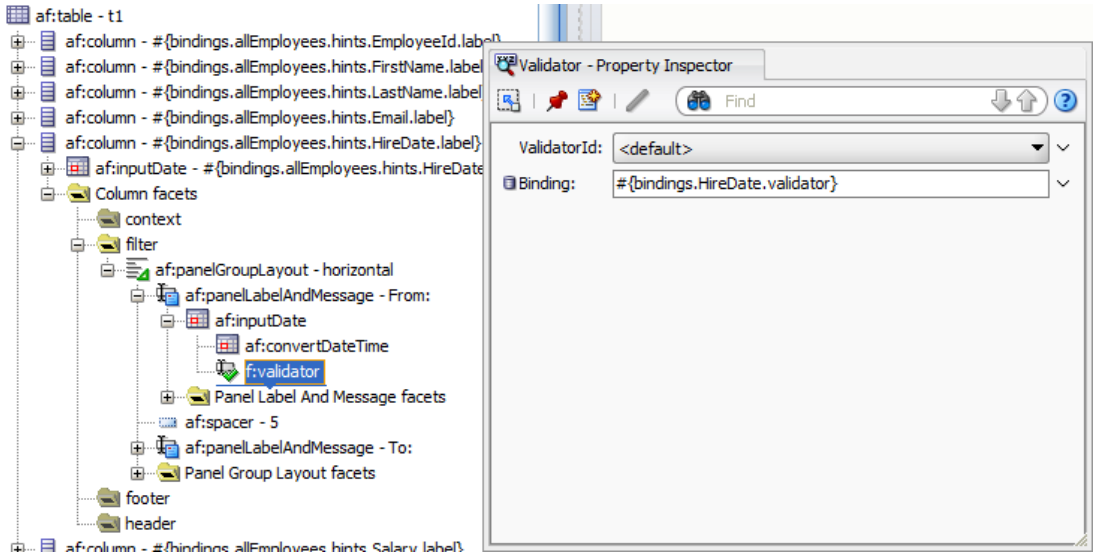
and "HireEndRange". In the custom query listener, we need to remove the two custom attributes from the map before the filter criteria is passed to the ADF Business Components model. If the two attributes are not removed from the filter criteria, a NullPointerException is thrown when the ADF search binding is invoked.

With the custom filter criteria attributes, the selected dates for the start and end of the query range become available in the custom query listener, from where it is passed to the bind variables used in the ViewCriteria defined on the Employees View Object.

The convertDateTime tag can be copied from the HireDate table cell renderer. The EL references the tree binding in the PageDef file, which is good to use in the table filter as well.

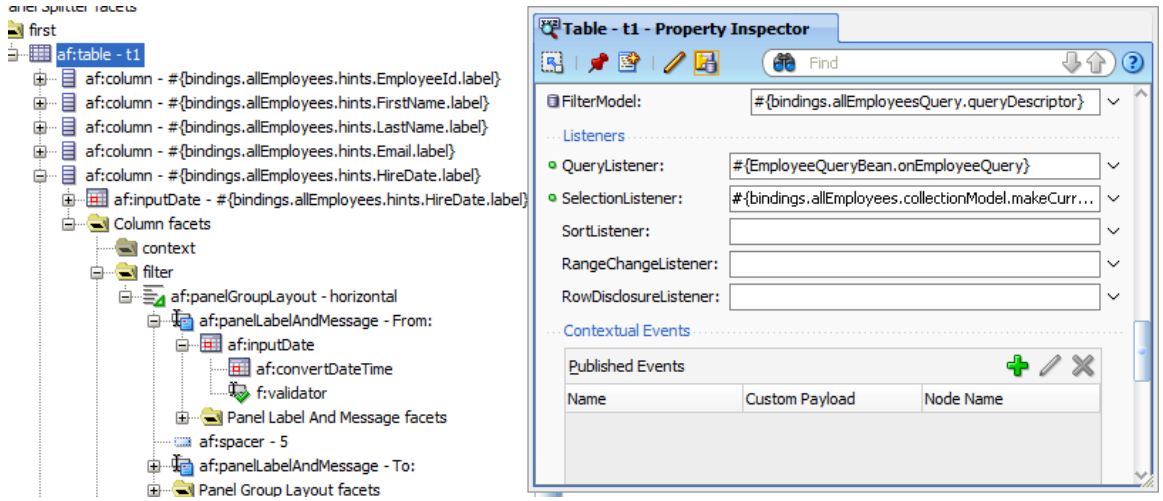


The validator tag also can be copied from the HireDate cell renderer component but must be changed to point to the HireDate attribute binding created earlier. By default, the validator setting is accessed through #{row.bindings …}, which is not an option to use in the table filter. The change in the EL binding is shown in the image below.



The last declarative step is to change the QueryListener reference of the table component from the ADF binding layer to a managed bean. Before creating a new managed bean by clearing the EL expression from the property and using the "Edit" context menu option, make sure you keep a copy of the existing query listener reference. The managed bean method only decorates the call to the search binding in the ADF PageDef file, it does not replace it.

To decorate the default query listener configuration, we use the copied EL reference in Java. The image below shows the custom QueryListener configuration in the sample table.



Query Filter Code

The custom query filter is called whenever the table is queried. It allows developers to access the query filter criteria, which in this sample, is used to access the custom filterCriteria attributes "HireStartRange" and "HireEndRange". Note that these two parameters only exist on the client. They don't exist on the business service and therefore must be removed before the filter criteria are passed on the ADF search binding. Before removing the attributes, the attribute values are read and stored in a local variable.

The attribute values of the "HireStartRange" and "HireEndRange" filters are set as bind variable values on the ViewCriteria using the method bindings created earlier.

Once the temporary filter attributes are removed, the search binding is referenced and executed using the previously saved EL expression.

The query filter code below contains comments for the important parts of its execution.

```
import java.util.Map;
import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;
import javax.faces.context.FacesContext;
import oracle.adf.model.BindingContext;
import oracle.adf.model.binding.DCBindingContainer;
import oracle.adf.view.rich.event.QueryEvent;
import oracle.adf.view.rich.model.FilterableQueryDescriptor;
import oracle.binding.OperationBinding;
```

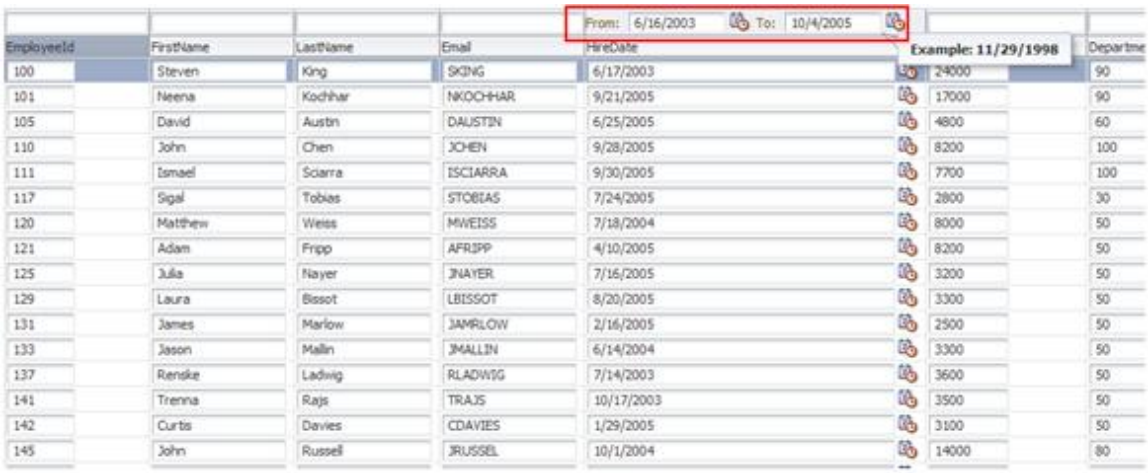
```
public class EmployeeQueryBean {
  public EmployeeQueryBean() {
}

public void onEmployeeQuery(QueryEvent queryEvent) {
  //default EL string created when dragging the table
  //to the JSF page
  //#{bindings.allEmployeesQuery.processQuery}
  FilterableQueryDescriptor fqd =
           (FilterableQueryDescriptor) queryEvent.getDescriptor();
  Map map = fqd.getFilterCriteria();
  BindingContext bctx = BindingContext.getCurrent();
  DCBindingContainer bindings =
              (DCBindingContainer) bctx.getCurrentBindingsEntry();
 //access the method bindings to set the bind variables on the
 //ViewCriteria
 OperationBinding rangeStartOperationBinding =
            bindings.getOperationBinding("setHireDateRangeStart");
 OperationBinding rangeEndOperationBinding   =
              bindings.getOperationBinding("setHireDateRangeEnd");
 //get the temporary, transient attributes from the filter map.
 //Note that the attributes exist no-where in the business service
 //but only in the map that represents the filter criteria

 Object hireStartRange = map.get("HireStartRange");
 Object hireEndRange = map.get("HireEndRange");

//set the start and end date of the range to search. If the attribute
//values are not set then NULL is passed to the biding variables, which
//more or less resets the View Criteria to allow all data to be queried
 rangeStartOperationBinding.getParamsMap().put(
                                          "value", hireStartRange);
 rangeEndOperationBinding.getParamsMap().put("value", hireEndRange);

//remove temporary attributes as they don't exist in the
//business service and would cause a NPE if passed with
//the query.
map.remove("HireStartRange");
map.remove("HireEndRange");

//set bind variable on the business service. Note that this does not
//yet query the View Object
rangeStartOperationBinding.execute();
rangeEndOperationBinding.execute();
```

```
  invokeMethodExpression("#{bindings.allEmployeesQuery.processQuery}",
                         Object.class,QueryEvent.class,queryEvent);

  //put filter values back so search filter is not empty after table is
  //queried
  map.put("HireStartRange", hireStartRange);
  map.put("HireEndRange", hireEndRange);
}


/*
 * The code below should be in a Utility class
*/
public Object invokeMethodExpression(
    String expr, Class returnType, Class[] argTypes, Object[] args) {

  FacesContext fc = FacesContext.getCurrentInstance();
  ELContext elctx = fc.getELContext();
  ExpressionFactory elFactory =
                   fc.getApplication().getExpressionFactory();
  MethodExpression methodExpr = elFactory.createMethodExpression(
                                  elctx, expr, returnType, argTypes);
  return methodExpr.invoke(elctx, args);
}
public Object invokeMethodExpression(
       String expr, Class returnType, Class argType, Object argument) {
       return invokeMethodExpression(expr, returnType,
                         new Class[]{argType},new Object[]{argument});
    }
}
```

Sample Download

The sample shown in this article requires Oracle JDeveloper 11g R1 PS3 (11.1.1.4), which by the time of writing is not yet publicly available. You can open this sample in previous versions of Oracle JDeveloper 11g, which however have a known rendering problem at runtime for the date picker in the table header.

At runtime, the table date range filter allows users to define both, a lower and upper range end, just a start range or an end range only. Note that a possible enhancement to this solution is to handle the case in which users provide a lower value for the range end date. The current sample would just return an empty table in this case,



The sample application uses the HR schema of the Oracle XE and enterprise database to query the sample data. This needs to be configured on the ADF BC model or choosing the database view in Oracle JDeveloper. A zip file with the workspace is available for downloaded on ADF Code Corner on OTN: http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html.

RELATED DOCOMENTATION

☒	030. How-to intercept and modify table filter values (ADF Code Corner) - http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html
☒	
☒	