ADF Code Corner

81. How-to create master-detail behavior using af:panelTabbed and DVT graph components

twitter.com/adfcodecorner

Abstract:

Oracle ADF Faces tree, tree table and table components keep track of their row (node) selection state, allowing developers to access the selected data rows, for example, to fetch dependent detail data or similar operations. The `af:iterator` component behaves like trees or table in that it stamps it child components, but does not keep track of the selected data row. This sample shows how you can build better user interfaces for master-detail data display using the `af:iterator` component while still keeping track of the current row selection.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
01-MAY-2011

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

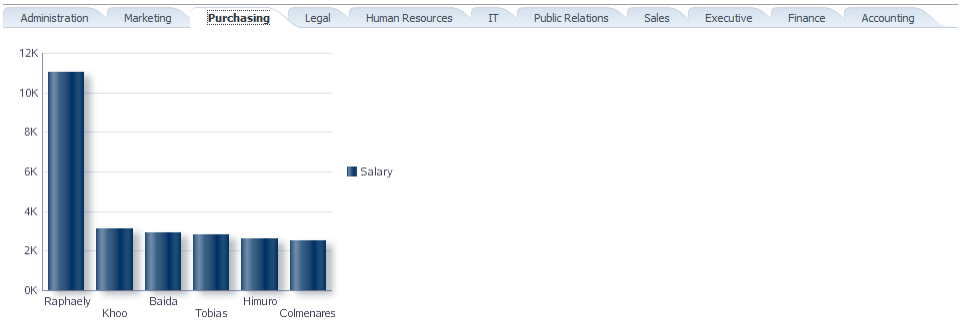
Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

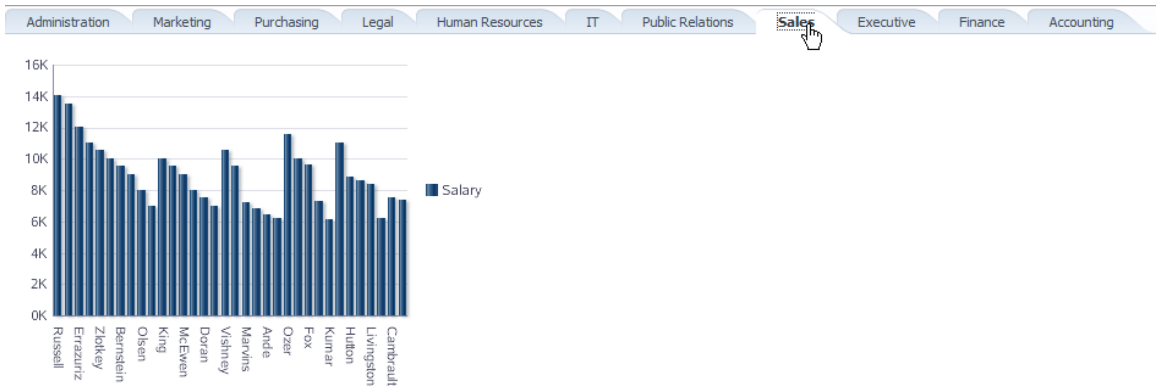
Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

The `af:iterator` component and the `af:forEach` component both iterate over a data set to display data values in custom arranged user interfaces. The `af:iterator` component behaves like an ADF Faces table component, without the table header, column, row and border chrome. It works well with collection models, which are exposed by the ADF tree binding in the PageDef file. In general, using `af:iterator` is a lower footprint on the client than using `af:forEach`. The `af:forEach` tag substiutes the JSTL `c:forEach` tag, which does not support the "varStatus" reference that give developers access to the iteration statelike. the index of the current printed row. The `af:forEach` component does not stamp it children but creates object representations and renders list data. It does not support collections. As `af:forEach` is index based, items should not be removed after the list is rendered. A use case for this component is to dynamically render entries of an `af:selectOneChoice` list.

In this article, I explain how you use the `af:iterator` to iterate over a master detail set to render its rows as tabs in an `af:panelTabbed` component. Each tab contains an ADF bound graph that displays dependent detail data.

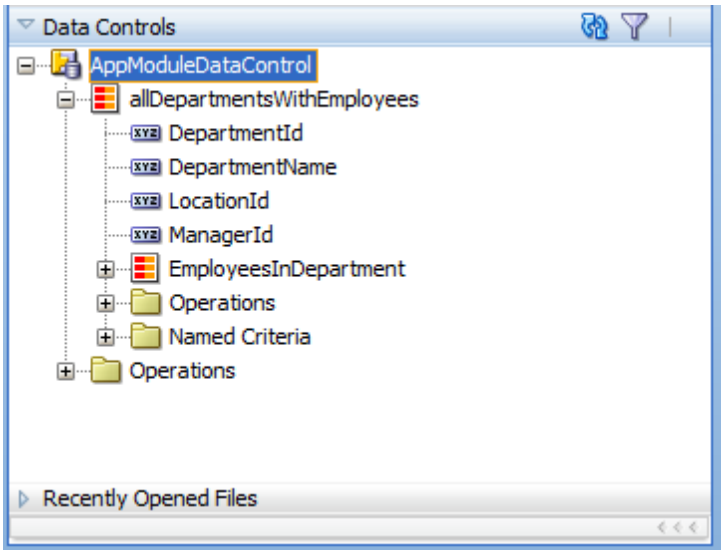




The ADF BC model

The ADF Business Components model is a master-detail structure consisting of a DepartmentsView view object and an EmployeesView view object.

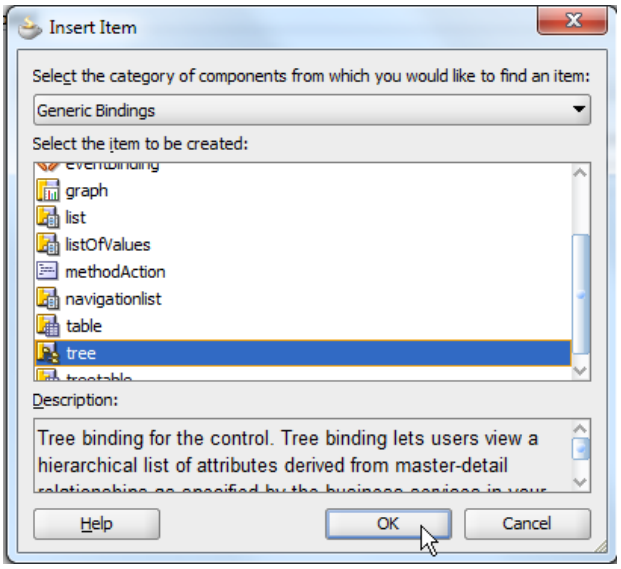
The data model defined in the ADF BC Application Module exposes an instance of departments and employees that is connected by a ViewLink so that selecting a department automatically synchronizes the dependent employee detail records. For demonstration purposed only, the **allDepartments** instance is filtered by a ViewCriteria to only show departments having employees. The image below shows the data model in the Data Controls panel view.



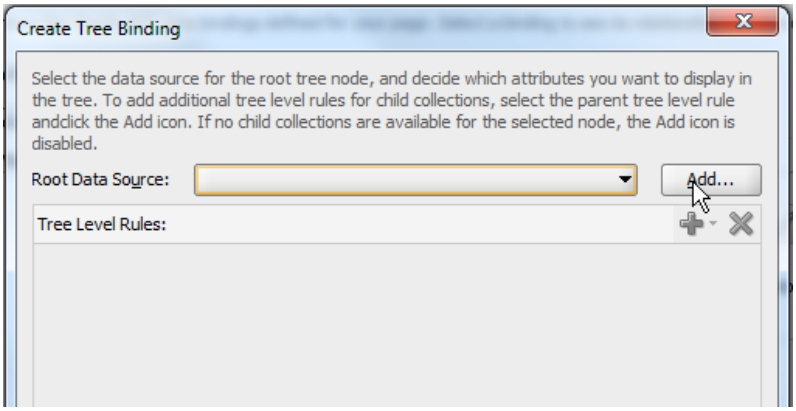
ADF Faces implementation

To implement the solution, you create an ADF tree binding in the PageDef file of the view that should contain the master-detail display. To create an ADF tree binding, right mouse click onto the page or page fragment and choose **Go to Page Definition**. If no ADF binding file exists, it can be created on the fly. With the binding file open, click the green plus icon in the **Bindings** section

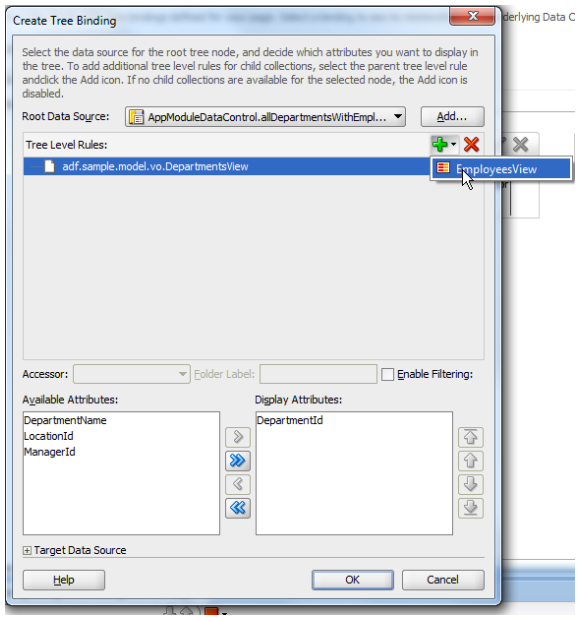
From the list of ADF bindings, choose the **tree** binding



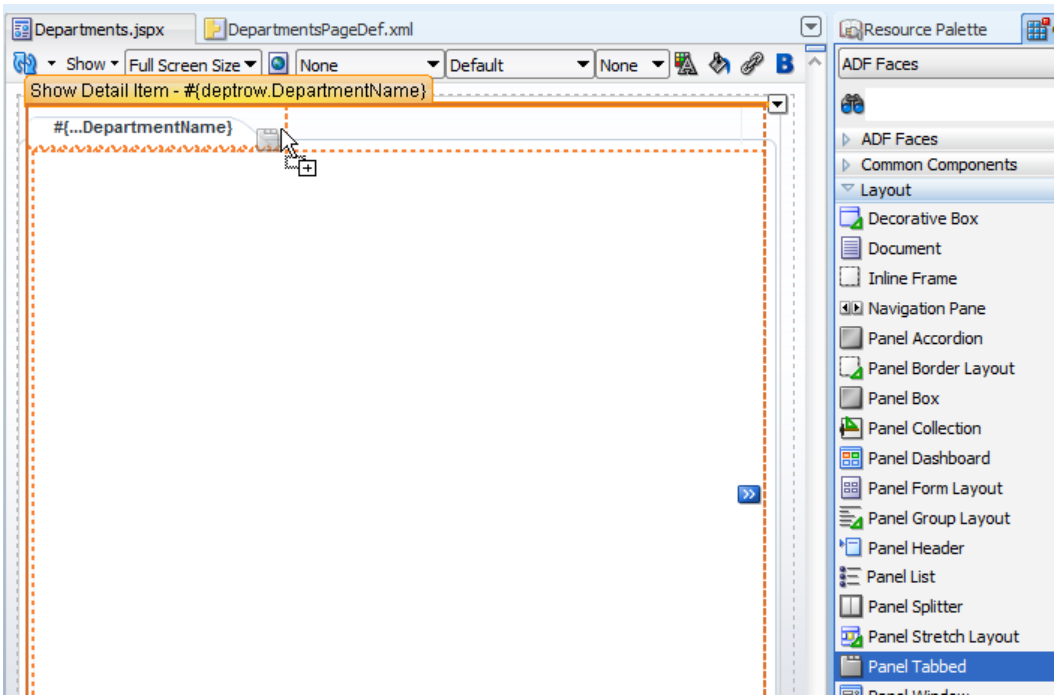
Use the **Add** button in the **Create Tree Binding** dialog to create a new iterator to base the tree binding on. In the image below, the iterator is created for the **allDepartmentsWithEmployees** View Object instance.



Create rules for the tree nodes, like you are familiar with when building trees from dragging a collection from the Data Control panel.



Drag and drop the `af:panelTabbed` component from the ADF Faces component palette onto the view. This automatically adds a default Show Detail Item to represent a tab.



Next, surround the `af:showDetailItem` component with an `af:iterator` tag, using the source editor. Of course, there is a declarative and visual way of doing the same, but over time I found the use of the source editor for this to be quick and efficient, especially because of the tag and syntax help Oracle JDeveloper provides.

<af:panelTabbed id="pt1">

```
<af:iterator var="deptrow" value="#{bindings.allDepartmentsWithEmployees.collectionModel}">
  <af:showDetailItem text="#{deptrow.DepartmentName}" id="sdi1"
      disclosureListener="#{graphBean.onTabDisclosure}">
    <f:attribute name="rowKey" value="#{deptrow.keyPath}"/>
    …
    </af:showDetailItem>
  </af:iterator>
</af:panelTabbed>
```

Note the content highlighted in bold! The **var** attribute defines an EL accessible object during rendering, which gives you access to the current rendered data row. This variable is used in an **f:attribute** tag, which is the JSF way of extending existing components with additional attributes. In the use case above, the key path of the current row is added as a new "rowKey" attribute to the `af:showDetail` component. The rowKey value is different for each tab generated for a department. Also note the reference of the ADF tree binding (`allDepartmentsWIthEmployees.collectionModel`) from the value attribute of the `af:iterator` tag. A disclosure listener is defined on the `af:dowDetailItem` component to handle the master-detail correlation. Handling the master-detail correlation in this use case means to access the selected detail item, read the key path value from the **rowKey** attribute and make this key the current selected row in the **allDepartmentsWithEmployees** iterator.

The managed bean containing the disclosure listener, is shown below

```
public class GraphBean {

//bean is in request scope. So accessing FacesContext here is
//okay as there is no risk of using stale context data
FacesContext fctx = FacesContext.getCurrentInstance();
ELContext elctx = fctx.getELContext();
ExpressionFactory expressionFactory =
fctx.getApplication().getExpressionFactory();

public GraphBean() {
   super();

}

//called by each tab with a changed disclosure state
public void onTabDisclosure(DisclosureEvent disclosureEvent) {
  RichShowDetailItem tab =
              (RichShowDetailItem) disclosureEvent.getSource();
  if (tab.isDisclosed()) {
    //read the custom attribute "rowKey" from the showDetailItem
    //component reference. Note that the key is the key of the data
    //in the collection model and thus is a List
```

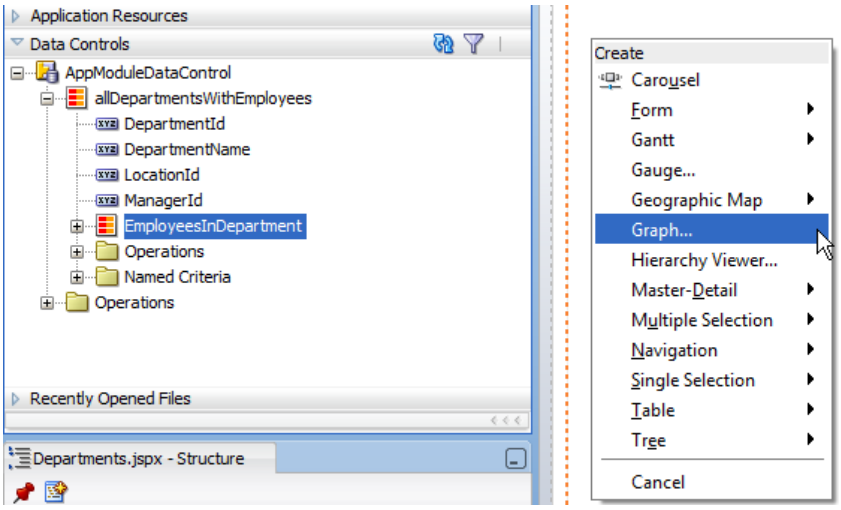
```
    List key = (List)tab.getAttributes().get("rowKey");
    JUCtrlHierBinding treeBinding = this.getTreeBinding();
    //get the ADF tree binding node that corresponds with the
    //key path in the Collection Model
    JUCtrlHierNodeBinding currentNode =
                                  treeBinding.findNodeByKeyPath(key);
    //make the current node the current row in ADF
    makeCurrent(currentNode);
    }
  }

  /*
   * Make row current in underlying iterator
   */
   private void makeCurrent(JUCtrlHierNodeBinding node){
     //get the key of the row represented
     //by the node binding
     Row rw = node.getRow();
     Key key = rw.getKey();
     //make this row the current row in the binding layer
     //so master/detail behavior works correctly
     JUCtrlHierBinding treeBinding = this.getTreeBinding();
     DCIteratorBinding iterator = treeBinding.getDCIteratorBinding();
     iterator.setCurrentRowWithKey(key.toStringFormat(true));
   }

    private JUCtrlHierBinding getTreeBinding(){
      BindingContext bctx = BindingContext.getCurrent();
      BindingContainer bindings = bctx.getCurrentBindingsEntry();
      //get reference to tree binding by its ID in the PageDef
      JUCtrlHierBinding treeBinding =
       (JUCtrlHierBinding) bindings.get("allDepartmentsWithEmployees");
      return treeBinding;
    }
}
```

Making the selected tab's keyPath value the current row in ADF, synchronizes the employees data to show employees for the selected department. If the business service was not ADF Business Components but Web Services, you would take the key path value, or a primary key value, and call a method exposed on the service. Only change to this sample is that the **Refresh** attribute on the iterators should be set to **ifNeeded** instead of deferred, which is used in this sample.

To display the Graph, you select the EmployeesInDepartment collection from the DataControls panel and drag and drop it into the **showDetailItems** component.



The completed page source then looks as shown below

```
<af:panelTabbed id="pt1">
    <af:iterator var="deptrow" value="#{bindings.allDepartmentsWithEmployees.collectionModel}">
      <af:showDetailItem text="#{deptrow.DepartmentName}" id="sdi1"
                         disclosureListener="#{graphBean.onTabDisclosure}">
        <f:attribute name="rowKey" value="#{deptrow.keyPath}"/>
        <dvt:barGraph id="barGraph1"
                      value="#{bindings.EmployeesInDepartment.graphModel}"
                      subType="BAR_VERT_CLUST">
          <dvt:background>
            <dvt:specialEffects/>
          </dvt:background>
          <dvt:graphPlotArea/>
          <dvt:seriesSet>
            <dvt:series/>
          </dvt:seriesSet>
          <dvt:o1Axis/>
          <dvt:y1Axis/>
          <dvt:legendArea automaticPlacement="AP_NEVER"/>
        </dvt:barGraph>
      </af:showDetailItem>
```

```
    </af:iterator>
</af:panelTabbed>
```

Download

The Oracle JDeveloper 11.1.1.4 workspace can be downloaded as sample #81 from the ADF Code Corner website.

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

Configure the database connection in the workspace to point to a database with the HR schema installed and run the JSPX file. The steps explained in this sample should also work with upcoming JDeveloper 11g releases (11.1.1.5 and 11.1.2), as well as with previous 11g versions.

RELATED DOCOMENTATION

⊠	ADF Faces af:iterator tag documentation: http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_iterator.html
⊠	ADF Faces af:forEach tag documentation http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_forEach.html
⊠	