ADF Code Corner

009. How-to Configure the ADF Faces Carousel Component with ADF

twitter.com/adfcodecorner

Abstract:

The af:carousel component is new in Oracle JDeveloper 11g R1 PatchSet 1. Though the carousel component is one of the choices in the context menu when dragging a View Object collection from the Data Controls palette onto a page, its not fully configured when selected. Instead, to setup the carousel to show ADF bound data, you need to explicitly add the databound content to the component's nodestamp facet. To implement a master-detail behavior where the carousel component represents the master row set, you need to implement the CarouselSpinListener using Java in a managed bean to mark the selected carousel item as the current selection in the binding layer. In this how-to article we provide the detailed steps of how to build a master-detail implementation of an ADF bound carousel and also give you a generic implementation of the CarouselSpinListener a hand that works with any ADF bound carousel. You can copy and paste into your custom development or make it part of a library.

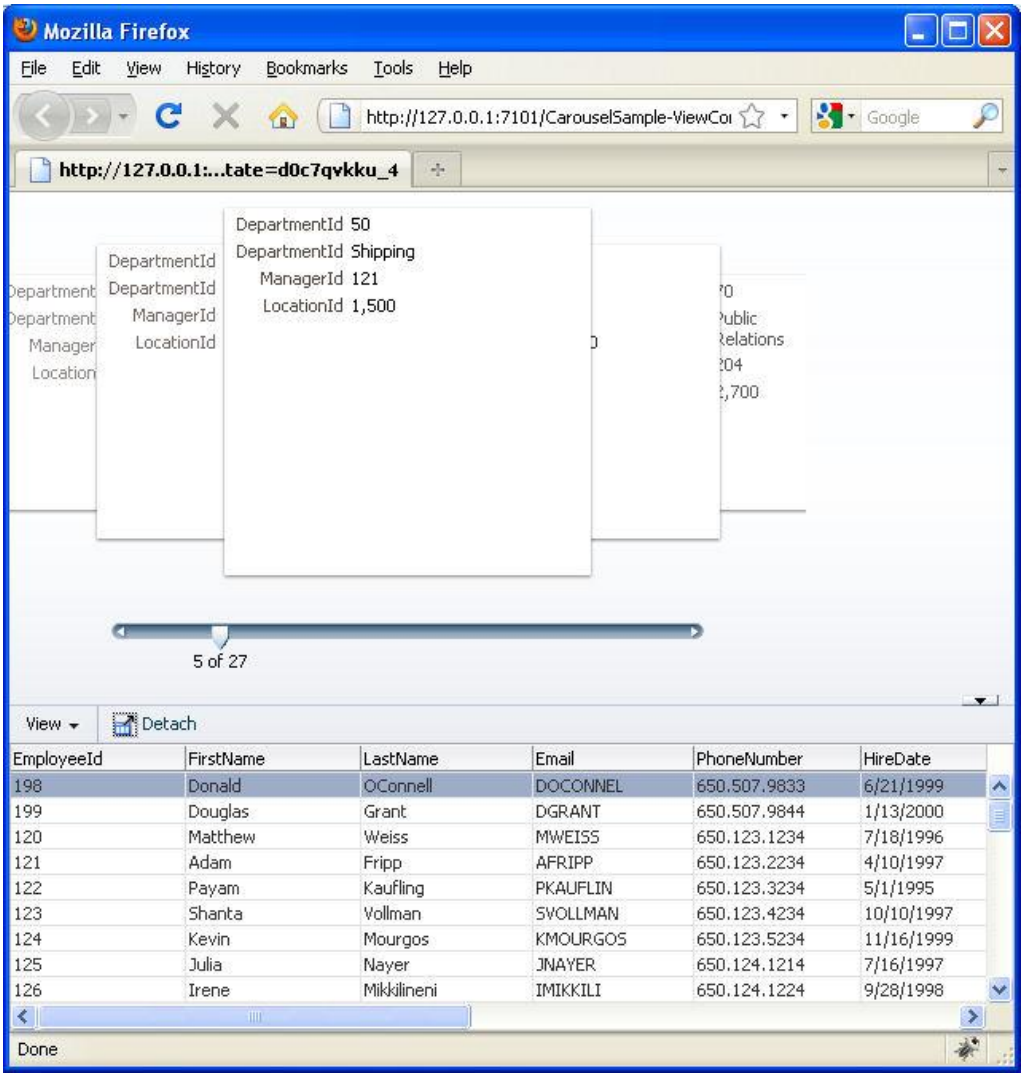
Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
01-JUL-2011

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



Apple made the carousel component popular as a dynamic menu on Mac computers and many companies and websites followed. The ADF Faces carousel component allows developers to display dynamic

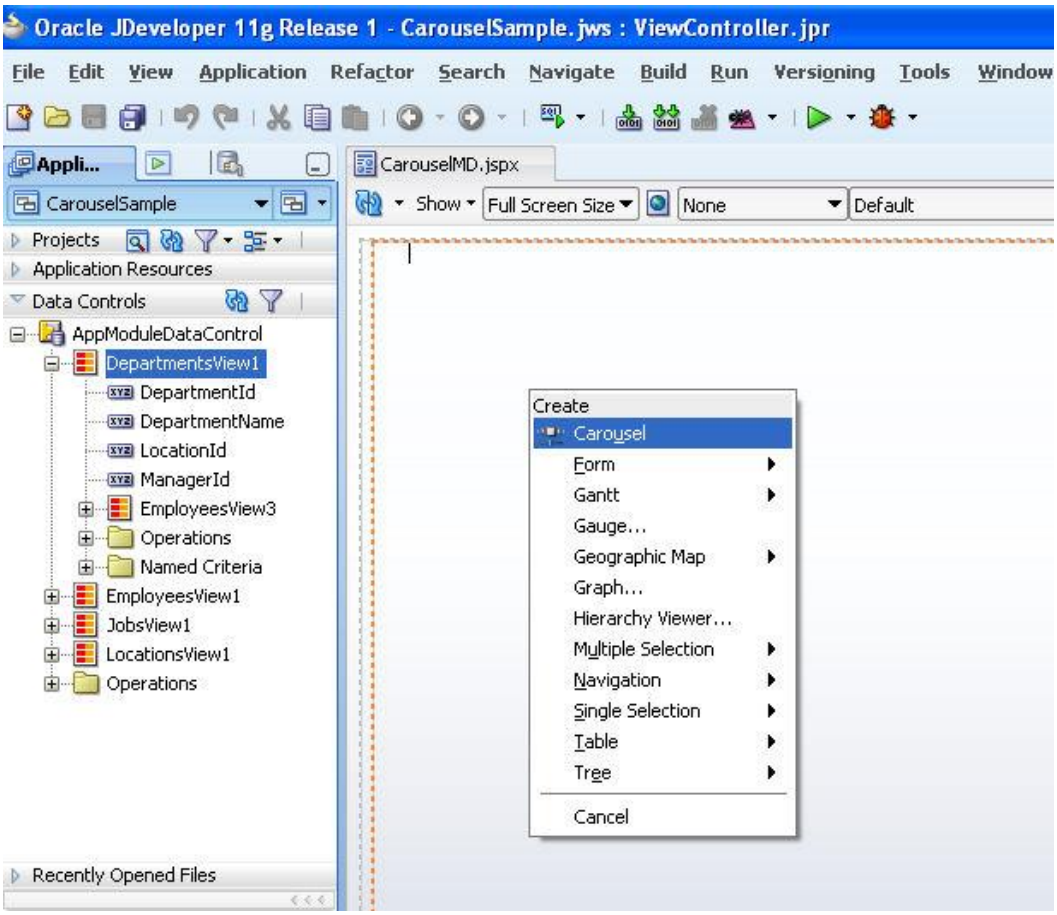
content, images and or text content, in rotating index cards. Though the carousel component can be used with the ADF tree binding, the implementation is only half-automatic and manual setup is required using Java and JSF. In this blog article we explain how to create an ADF based carousel that displays entries of the HR Departments table as the content of the carousel index cards and entries of the Employees table in a dependent ADF Faces table. Finishing this how-to, the result looks as shown below:

Note: We used a two column quick layout template to build the page layout.

Implementation

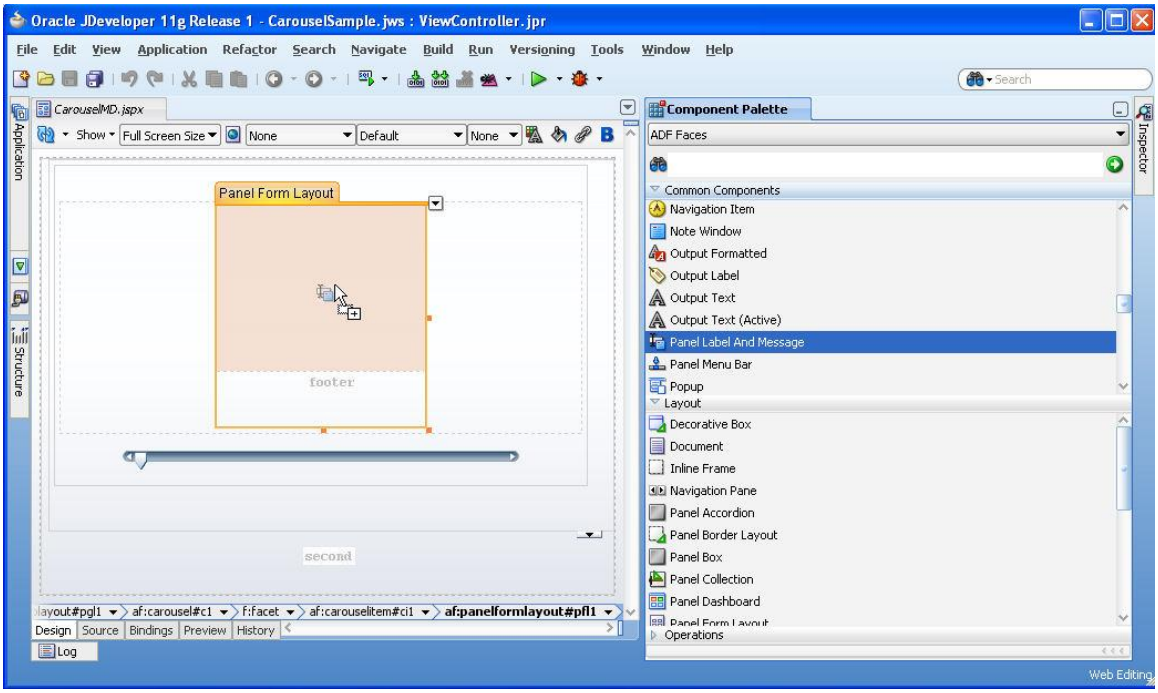
Starting from an existing ADF Business Component model, you drag and drop the Departments View Object from the Data Controls palette to the JSP page. In the opened context menu, select the Carousel entry to create the component page instance. The component is created with a single af:carouselItem child component added to the nodestamp facet.

The value property of the af:carousel component references the tree binding of the ADF binding layer using Expression language. To access the rows of the Departments View Object through the created ADF tree binding, the af:carousel has its var attribute set to "item". The "item" variable is EL accessible when the component is rendered. You use this variable to populate the carousel index card with data content.



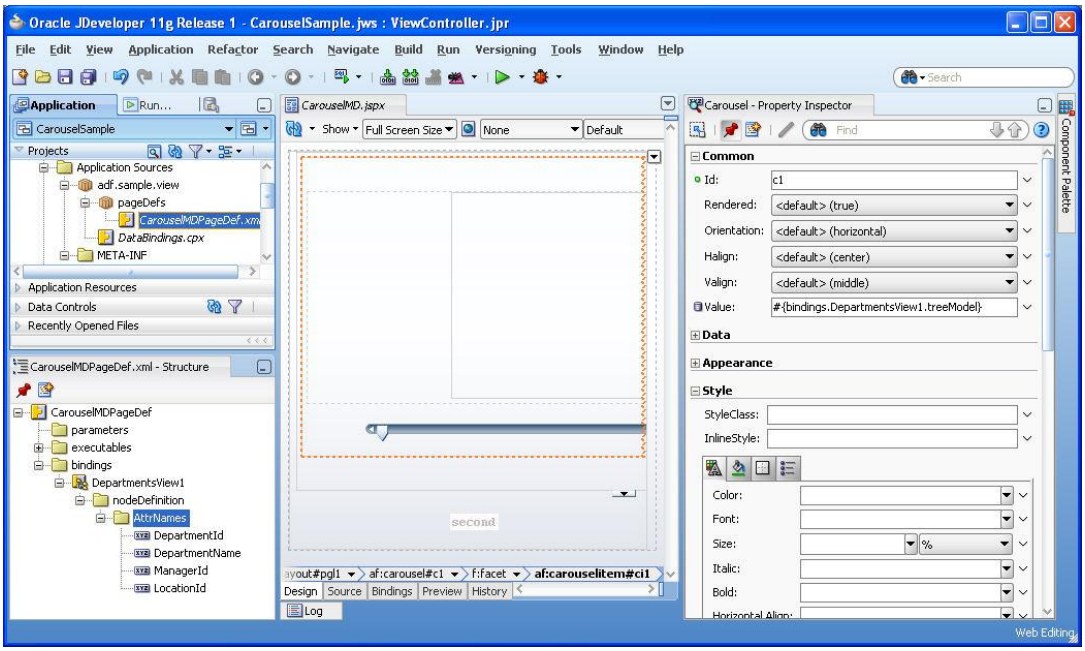
In our example, we display the Departments row data in an af:panelFormLayout component using af:outputText components. First, drag and drop an af:panelFormLayout component from the Component Palette onto the carousel item. Then drag and drop 4 instance of the Panel Label and Message component into the af:panelFormLayout component.

This is because the Departments View Object has four attributes, DepartmentId, DepartmentName, ManagerId and LocationId. The af:panelLabelAndMessage component allows you to associate labels with the af:outputText component that is used to display the department information. Unlike the af::inputText component, the af:outputText component does not have implicit labels. Drag and drop one af:outputText component into each of the four Panel Label and Message components.



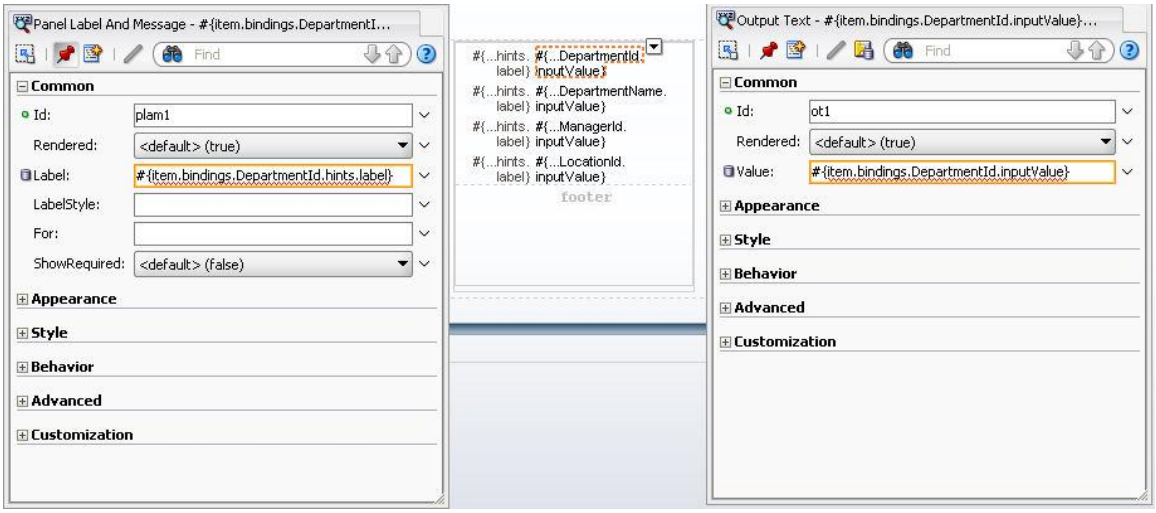
As shown below, the value property of the af:carousel component references the treeModel of the ADF tree binding.

The treeModel is an instance of the Apache Trinidad CollectionModel, which is a useful information we need when programming the master-detail behavior. The row level information is exposed through the af:carousel "var" property, which by default has a value of "item". When the af:carousel component renders, the "item" reference is used to populate the index cards.

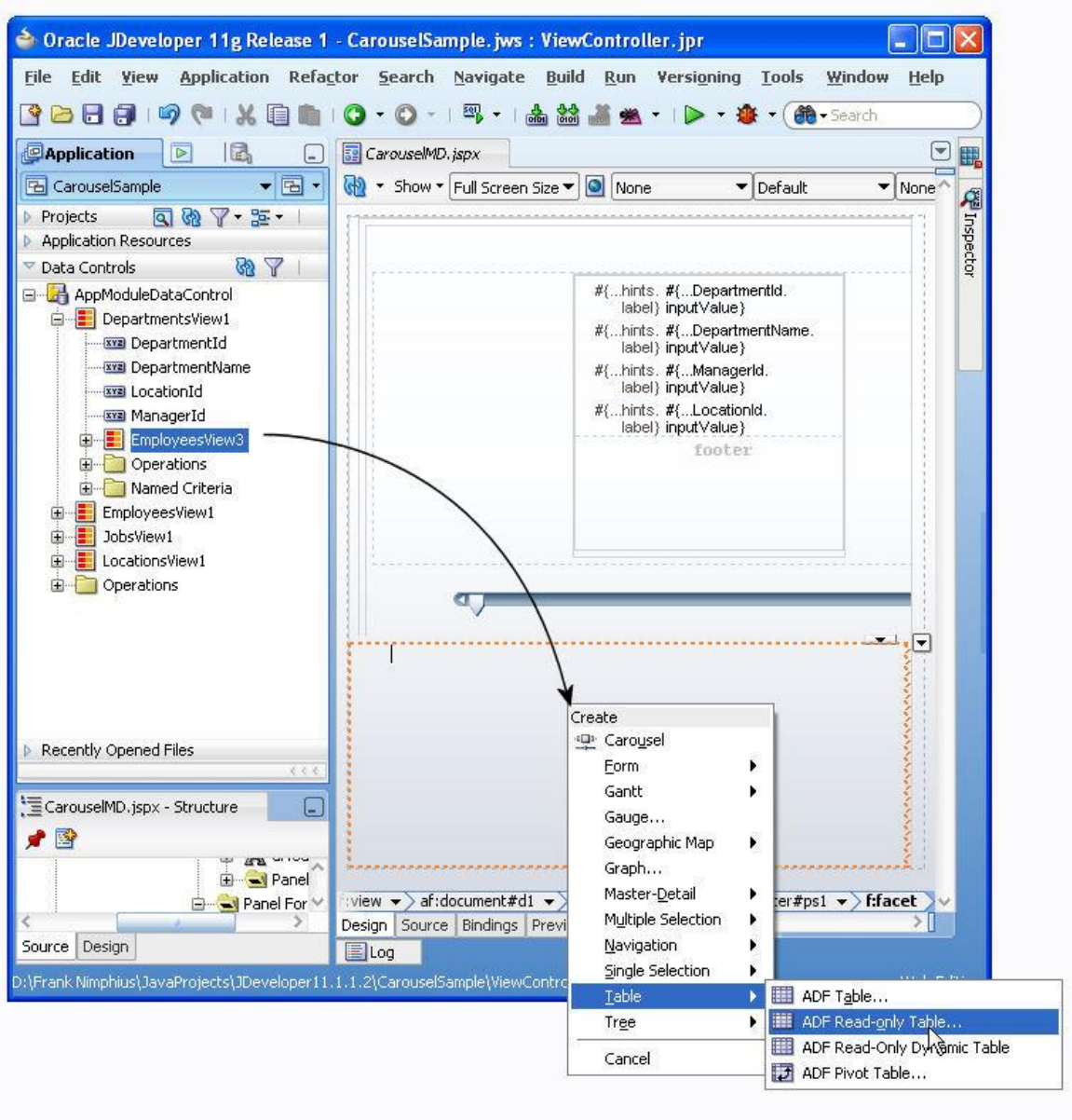


To access the binding layer from the af:panelLabelAndMessage component, which displays the label, and the af:outputText component, you reference the "item" variable of the af:carousel component using Expression Language. For example, to read the label of the DepartmentId, you use the following EL: #{item.bindings.DepartmentId.hints.label}. This reads the label of the DepartmentId attribute from the Departments View Object through the ADF binding layer.

The label can be internationalized using the Control Hint property of the View Object attribute. Similar, to read the DepartmentId value into the af:ouputTextField, add the following EL to the output component value property: #{item.bindings.DepartmentId.inputValue}. The image below shows the EL for the label property of the af:panelLabelAndMessage component and the EL of the af:outputText component value property.



To create the detail view, drag the dependent Employee View Object (EmployeeView3 in this example) from the Data Controls palette to the second panel splitter facet and choose the read only table option. Optionally, you can surround the af:table component with an af:panelCollection component (this is what we did in the example screen shot at the beginning of this article). To surround the table with a Panel Collection, select the af:table component and choose "Surround with" from the right mouse context menu.

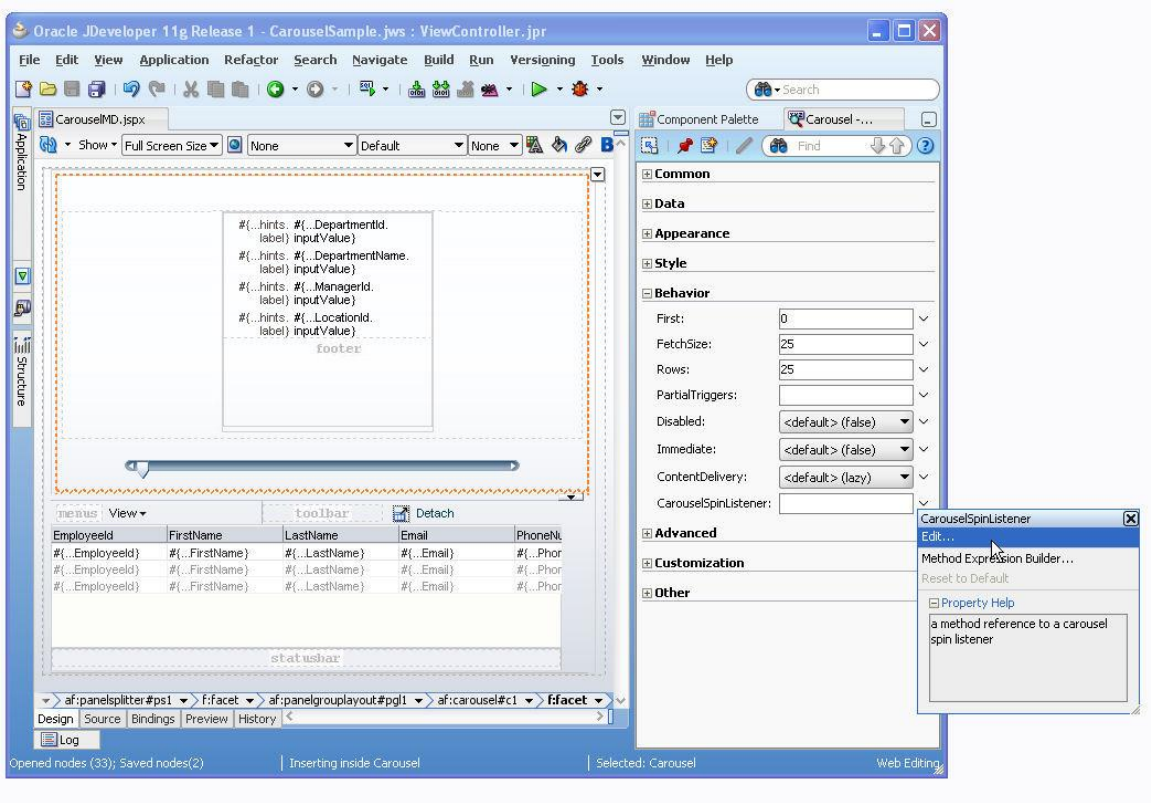


Now it is getting interesting: The master-detail synchronization

Now that the master and detail views are constructed, its time to implement the master-detail behavior. Using ADF Business Components, the active model automatically synchronizes the detail view to match the selected parent row. So all that we need to do to implement a master-detail behavior between the

af:carousel and the dependent table is to implement the CarouselSpinListener property of the af:carousel component in a managed bean so that whenever the carousel spins, a new parent row is set as the current row in the binding

Reference the af:carousel component from the af:table PartialTriggers property



To create a managed bean and the CarouselSpinListener handler method, click the arrow icon next to the CarouselSpinListener property and choose Edit from the popup menu.

In the opened dialog, provide information about the managed bean to create. Since the managed bean code for the CarouselSpinListener doesn't need to keep any state, the Scope can also be set to None, in which case the bean is dismissed as soon as the method is executed. In our example, we choose request scope as the lifetime duration of the bean. All that the managed bean configurtion is used for is to make the listener method EL accessible.

The method that is created in the next step creates the listener skeletton in the managed bean. All you need to do next is to use the available information, the event object to determine the row key of the current index card of the carousel and set the ADF iterator binding's current row to the same row.

This will then change the dependent View Object, the Employees View Object, to show the details of the selected department (only if you setup the partial refresh as mentioned earlier).

At the end, the CarouselSpinListener property property contains the following EL entry: #{SpinHelper.onSpin}.

The generated managed bean method skeletton looks as shown below

```
public void onSpin(CarouselSpinEvent carouselSpinEvent) { ... }
```

The complete implementation code for the master-detail behavior used in our example is shown below. Please also read the code comments to learn about what this code is doing. Its a great example of building reusable code in ADF.

```
import java.util.List;
import oracle.adf.model.binding.DCIteratorBinding;
import oracle.adf.view.rich.component.rich.data.RichCarousel;
import oracle.adf.view.rich.event.CarouselSpinEvent;
import oracle.jbo.Key;
import oracle.jbo.uicli.binding.JUCtrlHierBinding;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;
import org.apache.myfaces.trinidad.model.CollectionModel;
```

```
...
/*
 * The action handler below is generic and can be used with any
 * ADF bound implementation of the af:carousel component. The method
 * is invoked from the CarouselSpinListener property using EL and
 * ensures that the selected item is set as the current row of the
 * underlying ADF tree binding. This allows developers to build
 * master/detail forms with the carousel as the master data set.
 *
 * This code is generic and does not store any state, it can be stored
 * in a managed bean with a scope of NONE.
 *
 * Make sure that you setup a PartialTriggers property reference
 * between the child component (e.g. a form or table, and the carousel
 * so that the detailis refreshed when the carousel spins
 *
 * Note that this code accesses the ADF binding layer without any
 * knowledge about namings in the PageDef file. It also works with the
 * JU* binding classes, avoiding any use of the FacesCtrl* private
 * classes.
*/
public void onSpin(CarouselSpinEvent carouselSpinEvent) {
  //get the selected item key - an instance of java.util.List
  List currentSelectedKey = (List) carouselSpinEvent.getNewItemKey();
  //get a handle to the carousel component instance. We need this to
  //generically access the binding layer
  RichCarousel carousel = (RichCarousel) carouselSpinEvent.getSource();
  //get the Trinidad CollectionModel for this component
  CollectionModel componentModel =
                             (CollectionModel) carousel.getValue();
  //get the ADF tree binding from the CollectionModel. This code also
  //works with tables and trees and thus is worth to remember
  JUCtrlHierBinding carouselTreeBinding =
                 (JUCtrlHierBinding) componentModel.getWrappedData();
  //get the selected node
  JUCtrlHierNodeBinding selectedCarouselItemNode =
           carouselTreeBinding.findNodeByKeyPath(currentSelectedKey);
  //get the row key to make it the current key in the ADF iterator
  //Key currentCarouselItemKey = selectedCarouselItemNode.getRowKey();
  //You can access the iterator binding from the tree binding so that
  //you don't need any knowledge about the namings in the PageDef file
  //- Cool, he ?
  DCIteratorBinding  dcIterBinding =
                           carouselTreeBinding.getIteratorBinding();
  //make the row the current row
  dcIterBinding.setCurrentRowWithKey
                 (currentCarouselItemKey.toStringFormat(true));
}
```

This is it. All you need to do next is to setup the partial refresh of the table when the index card of the carousel component changes because of spinning. Select the af:table and press the arrow icon next to its PartialTriggers property. In the popup menu, click the "Edit" option. In the opened dialog, navigate to and select the carousel component. This configures the partial trigger for the table component.

Download

A sample application can be downloaded as sample #09 from ADF Code Corner.

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

You need to configure the database connection used in the model project to point to a database with the HR schema installed. Run the JSPX file in the sample to see it working,

RELATED DOCOMENTATION

⊠	af:carousel - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12419/tagdoc/af_carousel.html
⊠	af:carouselItem - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12419/tagdoc/af_carouselItem.html
⊠	Oracle Fusion Developer Guide (McGraw Hill), Frank Nimphius, Lynn Munsinger http://www.mhprofessional.com/product.php?cat=112&isbn=0071622543