ADF Code Corner

96. How to invoke a table selection listener from Java

twitter.com/adfcodecorner

Abstract:

Component listeners like action, select and query are invoked in response to user interaction with the UI.

However, there might be a use case in which you need the component listener to fire as if the change was triggered by a user. An example for this is to invoke the selection listener on a table in response to navigating the parent collection using a navigation bar. In this article I show how you can implement such a use case using ADF Faces and the ADF binding layer

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
01-FEB-2012

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

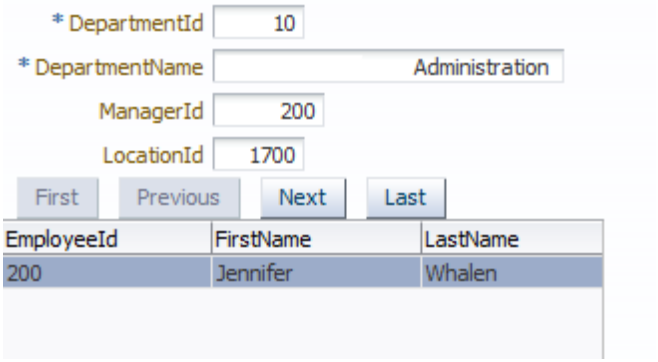
Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

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 sample code in this article invokes a `SelectionEvent` on an ADF Faces table each time the parent collection is navigated using the navigation commands.

By default, selection events in tables are only produced when the users selects a different row than the current selected row.

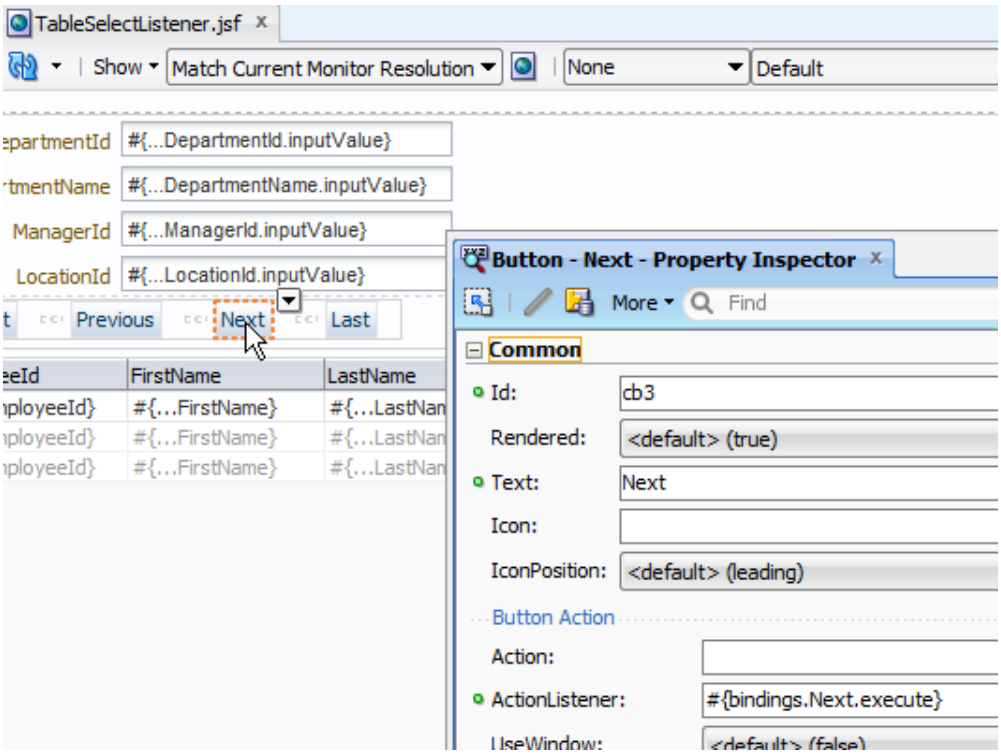


Note: An alternative solution to manual invocation of a component event is to directly call the code executed by the listener from Java. By default the selection listener added by ADF when dragging a collection from the DataControls panel and dropping it as a table is to set the clicked row as the current row in the underlying ADF itertor. This one for example would be an easy one to do in Java from an action event invoked by a command button.

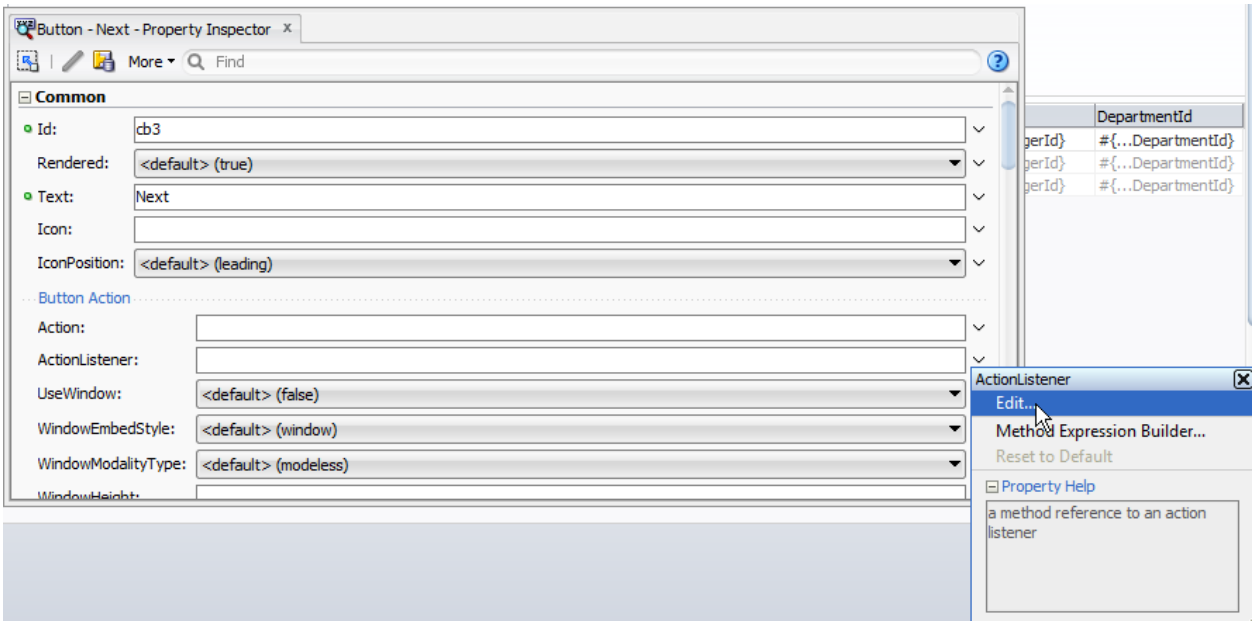
How it works!

Let's use the **Next** button as the starting point for the navigation of the parent collection (Departments) that will result in a selection event invoked on the employee table.

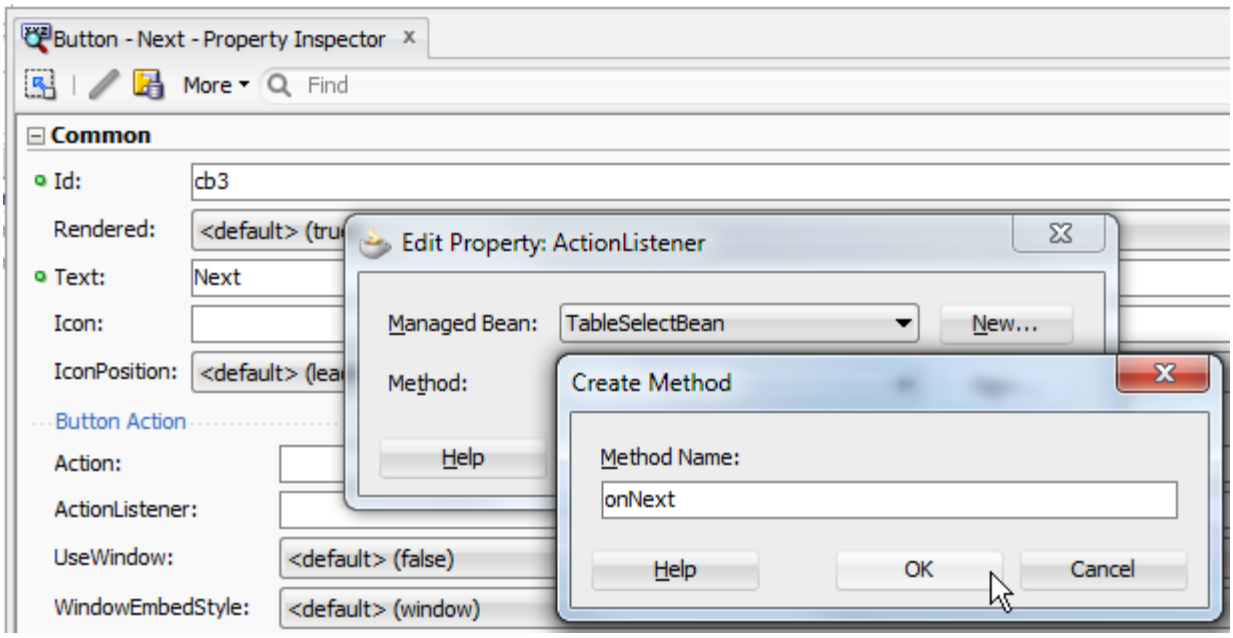
First, you need to replace the default **ActionListener** configuration that is created when dragging the **Next** operation as a button to the page, or when creating an ADF form. Instead of directly executing the **Next** operation in the binding layer when the command button is pressed, we take a little detour involving a managed bean.



The managed bean and method reference can be created declaratively using the **Edit** menu option on the **ActionListener** property.



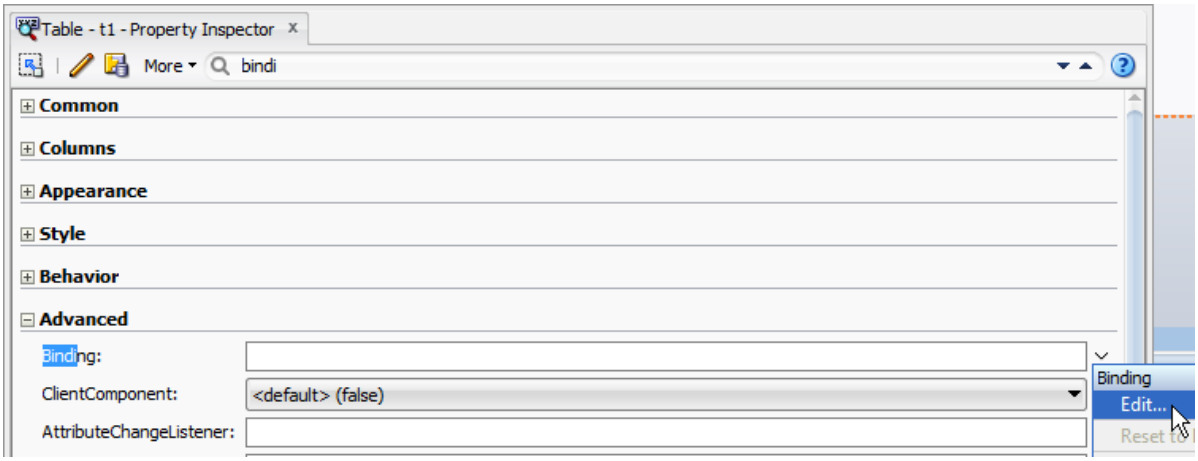
In the example, a method **onNext** is created as shown in the image below.

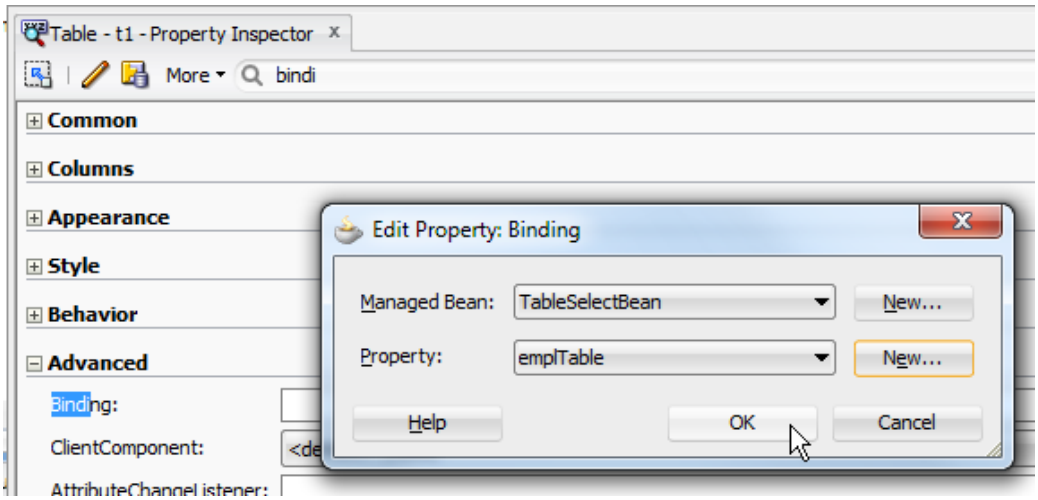


To access the table instance in the managed bean, the **Edit** menu item next to the **Binding** property is used. This turns the managed bean into a backing bean as now there is a direct relation created between a component instance on the page and the bean.

Note: There are other options to look up components from a managed bean. For example, you can look up the `UIViewRoot` on the `FacesContext` and perform a `findComponent()` call to find the table instance.

For simplicity – and to stay focus on the solution explained in this example – I used the component binding using the **Binding** property.





The managed bean code for the **onNext** method invoked by the Next button is shown below.

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

import javax.faces.event.ActionEvent;
import oracle.adf.model.BindingContext;
import oracle.adf.model.OperationBinding;
import oracle.adf.model.binding.DCIteratorBinding;
import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.adf.view.rich.component.rich.nav.RichCommandButton;

import oracle.binding.BindingContainer;
import oracle.jbo.Key;
import oracle.jbo.Row;

import org.apache.myfaces.trinidad.event.SelectionEvent;
import org.apache.myfaces.trinidad.model.RowKeySet;
import org.apache.myfaces.trinidad.model.RowKeySetImpl;
…

 public void onNext(ActionEvent actionEvent) {
  //preserve default behavior #{bindings.Next.execute}
  //in code below
  BindingContext bctx = BindingContext.getCurrent();
  BindingContainer bindings = bctx.getCurrentBindingsEntry();
  OperationBinding nextButton =
     (OperationBinding) bindings.get("Next");
      nextButton.execute();
   //API: new Selection(RowKeySet oldSelection, RowKeySet newSelection,
   //UIComponent table)
  DCIteratorBinding employeeIterator =
            (DCIteratorBinding) bindings.get("employeesIterator");
  Row currentRow = employeeIterator.getCurrentRow();
```

```
  //build the table rowKeySet
  List rowKeyList = new ArrayList();
  //check if there is child data at all
  if (currentRow != null) {
    //add primary key as jbo key
    Key jboKey = new Key(new Object[] {
                  currentRow.getAttribute("EmployeeId") });
    rowKeyList.add(jboKey);
    //add key to RowKeySet. For table multi row select usecases
    //you would add multiple row keys like this
    RowKeySet newRowKeySet = new RowKeySetImpl();
    newRowKeySet.add(rowKeyList);
    //create SelectioNEvent that pretends users has selected first row
    //in table
    SelectionEvent selectEvent =
                  new SelectionEvent(emplTable.getSelectedRowKeys(),
                                     newRowKeySet, emplTable);
    //queue event for execution
    selectEvent.queue();
  }
}
```

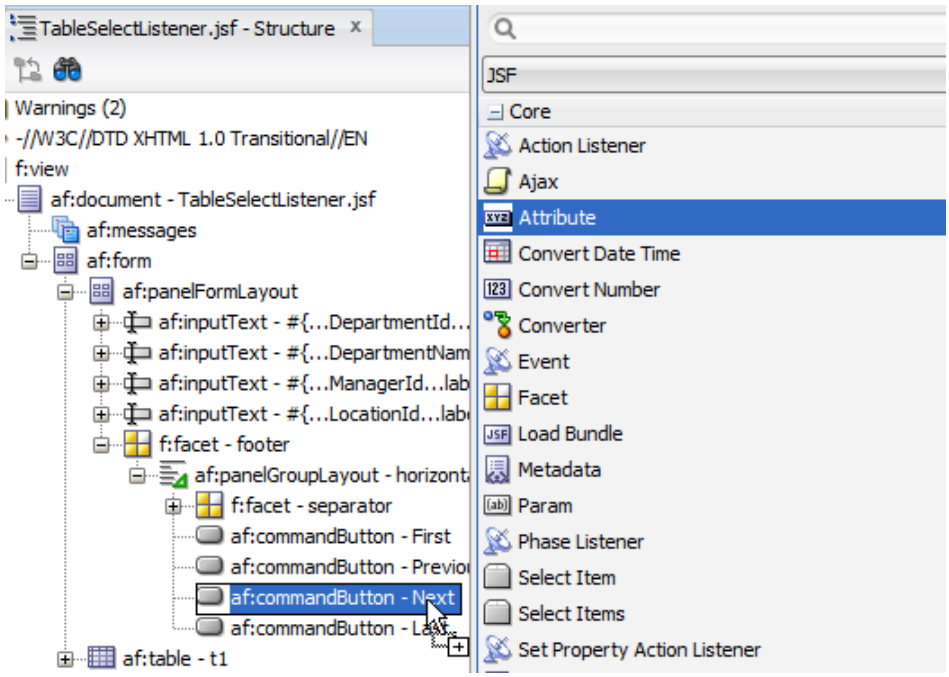
Note: If you change the default ADF selection listener and instead point the table **SelectionListener** property to a managed bean method that prints a statement when called, you have it easy to tell that the code works.

Don't repeat yourself

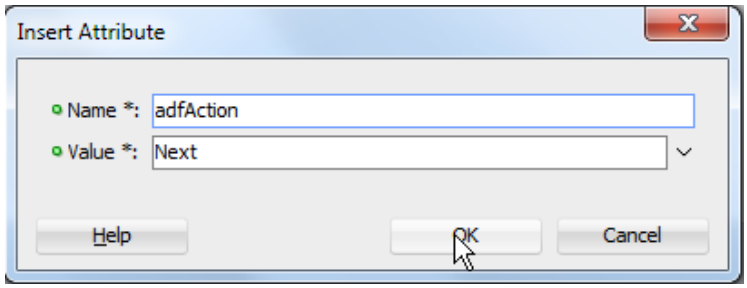
There are more buttons than just the **Next** button to invoke the selection event. One way to make the above code generic is to pass the ADF action to invoke as an attribute with the command button action.

Note: The Next, Last, Previous and First method bindings need to be created or exist in the PageDef file. This can be achieved by creating an ADF input form with the navigation bar option selected or manually in the PageDef file directly.

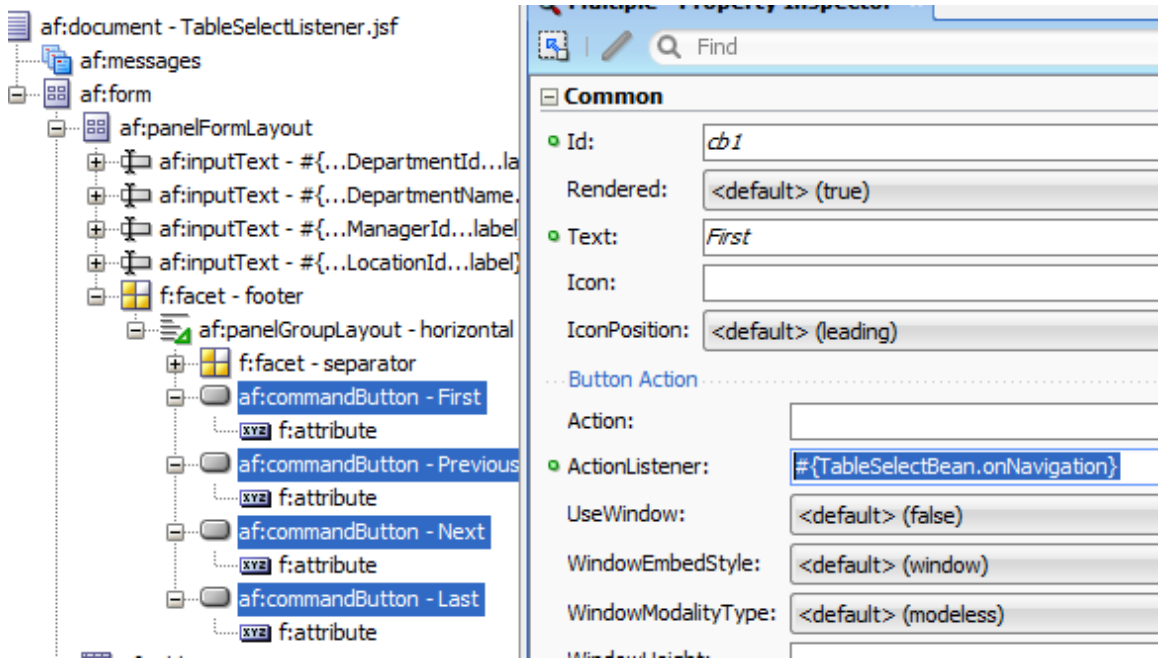
To define a custom attribute to the command button, drag and drop the **Attribute** element from the ADF Faces component palette (in the JSF category) onto the command button.



In the opened dialog, define **adfAction** as the attribute name and **Next**, **Previous**, **Last**, **First** respectively for the command buttons. The **Value** property name must match the name of the ethod binding in the PageDef file (so pay extra attention here)



Note that in the managed bean I refactored the **onNext** method name to **onNavigation**. All command buttons now have an `f:attribute` tag added and their **ActionListener** property pointing to the **onNavigation** method in the managed bean.



The modified code in the `onNavigation` method is shown below

```
public void onNavigation(ActionEvent actionEvent) {
  RichCommandButton navButton =
        (RichCommandButton) actionEvent.getSource();
  //TODO: add null check in case component does not have
  //f:attribute set
  String adfAction =
        (String)navButton.getAttributes().get("adfAction");

  //preserve default behavior #{bindings.Next.execute}
  //in code below
  BindingContext bctx = BindingContext.getCurrent();
  BindingContainer bindings = bctx.getCurrentBindingsEntry();
  //TODO: check if binding exists. OperationBinding would be null if
  //binding does not exist
  OperationBinding nextButton =
                      (OperationBinding) bindings.get(adfAction);
  nextButton.execute();
…
```

RELATED DOCOMENTATION

☒	
☒	
☒	