ADF Code Corner

025. How-to build a generic Selection Listener for ADF bound ADF Faces Trees and ADF BC models

twitter.com/adfcodecorner

Abstract:

When creating an ADF bound ADF Faces tree by dropping a collection from the Data Controls palette onto a JSF page, Oracle JDeveloper adds the following EL to the tree's SelectionListener property.

#{bindings.<tree binding>.treeModel.makeCurrent}.

The string references the "makeCurrent" method in the FacesCtrlHierBinding class, which is an internal class developers should not use directly in their application development.

However, if you are not supposed to use it, what if you need to add custom pre- or -post processing instructions to the execution of the selection listener. One option - the most obvious - is to call the EL string added by Oracle JDeveloper from a MethodExpression in a managed bean. This works fine and we could stop this article here. However, what if you want a generic solution in which you don't hard code the tree binding string indicated by the <tree binding> in my example above?

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
28-MAR-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

Lets start with an **appology**: The content in this blog article is advanced and you probably wont understand what I am writing about without ADF binding experience. If right away you understand what this blog is all about - welcome to the red carpet club of ADF experts.

Trees and tables are identical in ADF as they both use the ADF tree binding to connect to its business service. In a past article, I wrote about building a generic listener to handle selection in a table that if you try it out with a tree most likely will fail to meet your expectations. The reason is that a table, though exposing structured information, is a single level tree structure, whereas the tree component has one to many tree levels to deal with.

This said, if you use the table listener with the tree, you would be able to synchronize the top level node with the user selection.

So there is something special about trees, which is that i) trees have multiple levels that have no iterator of its own but use the internal accessors of the business service and ii) developers can synchronize an independent iterator binding to synchronize with the tree node selection, using the Target Data Source entry in the tree node editor. In fact "Target Data Source" would better be named "Target Iterator". So to build a generic tree listener, we have some code to write. Good that generic means "write once use everywhere".

Note: Because the code sample in this blog article uses the row key string to set the current row on the target iterator, the limitation is that it only works with ADF BC models. If you use other models like EJB or POJO, you still need to use EL from Java to execute the selection listener defined on the ADF binding.

Code Sample

I am not going to run a full tutorial on building multi level trees that synchronize with independent iterators at runtime. This I plan to release such an example at a later time on ADF Code Corner so you see how the generic listener I am discussing today looks when put into action.

Fort this article, I just dump the code lines that make the generic listener as I made sure code comments guide you the way to understanding how it works. The following method is stored in a managed bean and referenced from an ADF tree's SelectionListener

```
/**
 * Custom managed bean method that takes a SelectEvent input argument
 * to generically set the current row corresponding to the selected row
 * in the tree. Note that this method is a way to replace "makeCurrent"
 * EL expression (#{bindings.<tree binding>. treeModel.makeCurrent}that
 * Oracle JDeveloper adds to the tree component SelectionListener
 * property when dragging a collection from the Data Controls panel.
 * Using this custom selection listener allows developers to add pre-
 * and post processing instructions. For example, you may enforce PPR
 * on a specific item after a new tree node has been  selected. This
 * methods performs the following steps
 *
 * i.   get access to the tree component
 * ii.  get access to the ADF tree binding
 * iii. set the current row on the ADF binding
 * iv.  get the information about target iterators to synchronize
 * v.   synchronize target iterator
 *
 * @param selectionEvent object passed in by ADF Faces when configuring
 *  this method to become the selection listener
 *
 * @author Frank Nimphius
*/
public void onTreeSelect(SelectionEvent selectionEvent) {

 /* custom pre processing goes here */
 /* --- */

//get the tree information from the event object
RichTree tree1 = (RichTree) selectionEvent.getSource();
//in a single selection case ( a setting on the tree component ) the
//added set only has a single entry. If there are more then using this
//method may not be desirable. Implicitly we turn the multi select in a
//single select later, ignoring all set entries than the first
RowKeySet rks2 = selectionEvent.getAddedSet();
//iterate over the contained keys. Though for a single selection use
//case we only expect one entry in here
Iterator rksIterator = rks2.iterator();

//support single row selection case
if (rksIterator.hasNext()){
  //get the tree node key, which is a List of path entries describing
  //the location of the node in the tree including its parents nodes
```

```
    List key = (List)rksIterator.next();
    //get the ADF tree  binding to work with
    JUCtrlHierBinding treeBinding = null;
    //The Trinidad CollectionModel is used to provide data to trees and
    //tables. In the ADF binding case, it contains the tree binding as
    //wrapped data
    treeBinding = (JUCtrlHierBinding)
                  ((CollectionModel)tree1.getValue()).getWrappedData();
    //find the node identified by the node path from the ADF binding
    //layer. Note that we don't need to know about the name of the tree
    //binding in the PageDef file because
    //all information is provided
    JUCtrlHierNodeBinding nodeBinding = nodeBinding =
                             treeBinding.findNodeByKeyPath(key);
    //the current row is set on the iterator binding. Because all
    //bindings have an internal reference to their iterator usage,
    //the iterator can be queried from the ADF binding object
    DCIteratorBinding _treeIteratorBinding = null;
    _treeIteratorBinding = treeBinding.getDCIteratorBinding();
    JUIteratorBinding iterator = nodeBinding.getIteratorBinding();
    String keyStr = nodeBinding.getRowKey().toStringFormat(true);
    iterator.setCurrentRowWithKey(keyStr);

    //get selected node type information
    JUCtrlHierTypeBinding typeBinding =
                                nodeBinding.getHierTypeBinding();

    //The tree node rule may have a target iterator defined. Target
    //iterators are configured using the Target Data Source entry in
    //the tree node edit dialog
    //and allow developers to declaratively synchronize an independent
    //iterator binding with the node selection in the tree.
    //
    String targetIteratorSpelString =
                              typeBinding.getTargetIterator();

    //chances are that the target iterator option is not configured. We
    //avoid NPE by checking this condition

    if (targetIteratorSpelString != null &&
        !targetIteratorSpelString.isEmpty()) {

      //resolve SPEL string for target iterator
      DCIteratorBinding targetIterator =
                   resolveTargetIterWithSpel(targetIteratorSpelString);
      //synchronize the row in the target iterator
      Key rowKey = nodeBinding.getCurrentRow().getKey();
      targetIterator.setCurrentRowWithKey(rowKey.toStringFormat(true));
    }

    /* custom post processing goes here */
    /* --- */

  }
}
```

```
/**
 * Helper method to resolve EL expression into DCIteratorBinding
 * instance
 * @param spelExpr the SPEL expression starting with ${...}
 * @return DCIteratorBinding instance
*/
private DCIteratorBinding resolveTargetIterWithSpel(String spelExpr){
 FacesContext fctx = FacesContext.getCurrentInstance();
 ELContext elctx = fctx.getELContext();
 ExpressionFactory elFactory =
                fctx.getApplication().getExpressionFactory();

 ValueExpression valueExpr = elFactory.createValueExpression(
                                elctx, spelExpr,Object.class);
 DCIteratorBinding dciter = (DCIteratorBinding)
                                   valueExpr.getValue(elctx);
 return dciter;
}
```

As you see, no assumption is made in this code about the current binding container in regards to what the name of the tree binding is or what the name of the iterator binding is. If a tree node has the Target Data Source set to an iterator in the binding layer - using SPEL like ${reference to iterator} - then this information too is dynamically read.