# ADF Code Corner

## 61. How-to search in a rendered ADF bound tree


twitter.com/adfcodecorner

**Abstract:**

This blog article explains how to search in a rendered, ADF bound tree component. The use case is to find and expand tree nodes that match a specific search pattern in their tree node attributes.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
14-OCT-2010

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

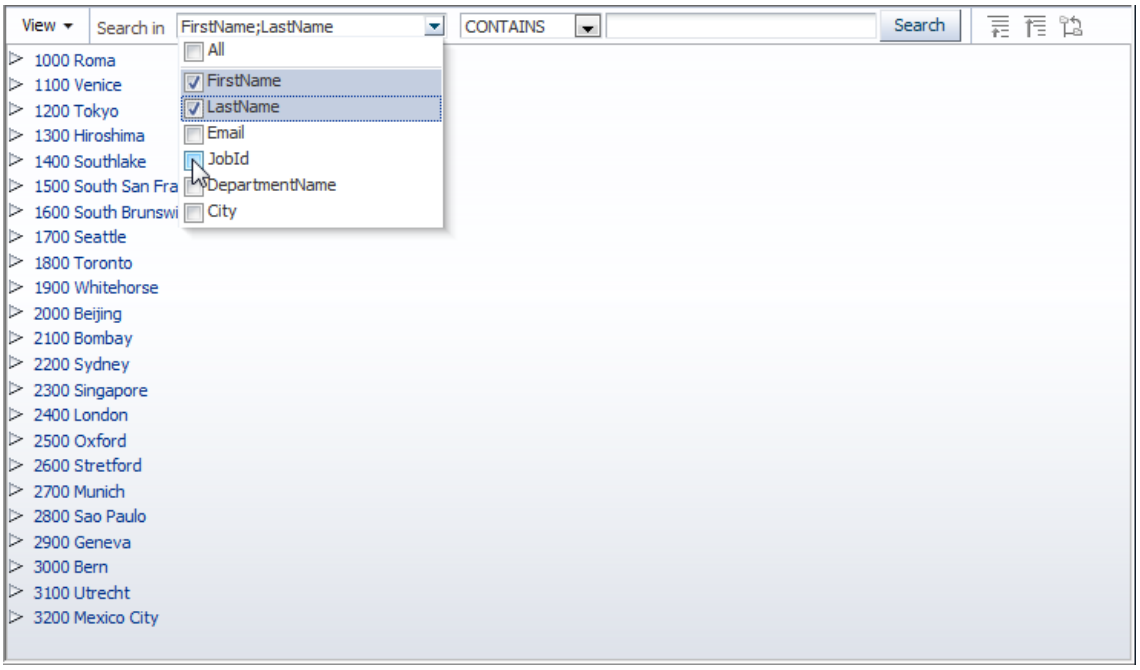
*Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.*

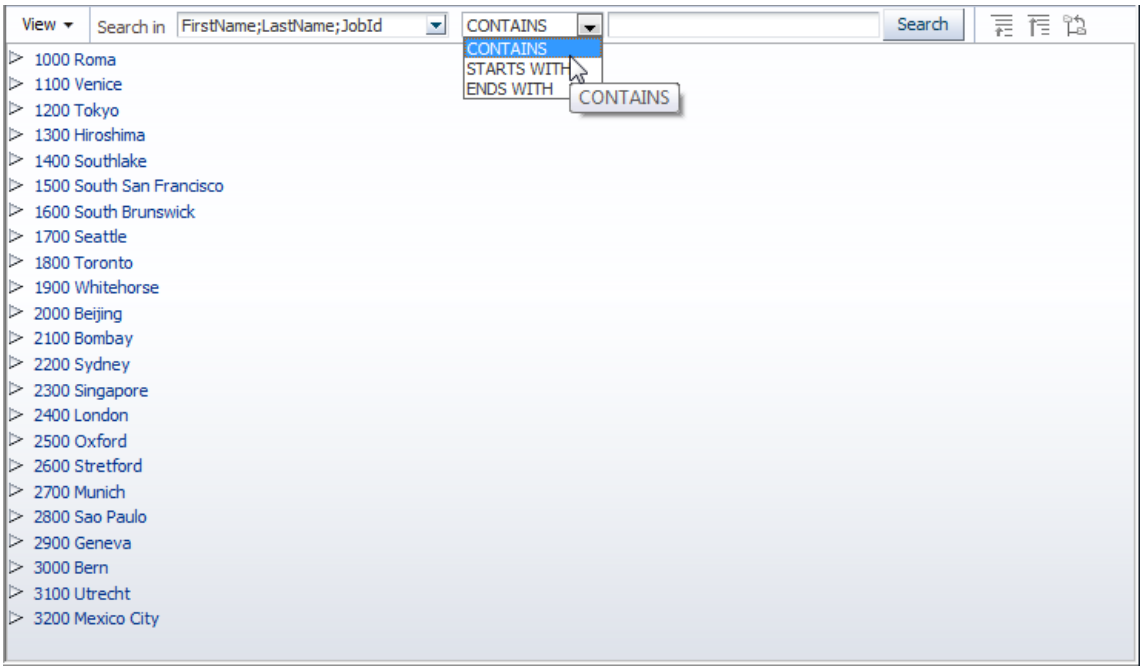
*Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83*

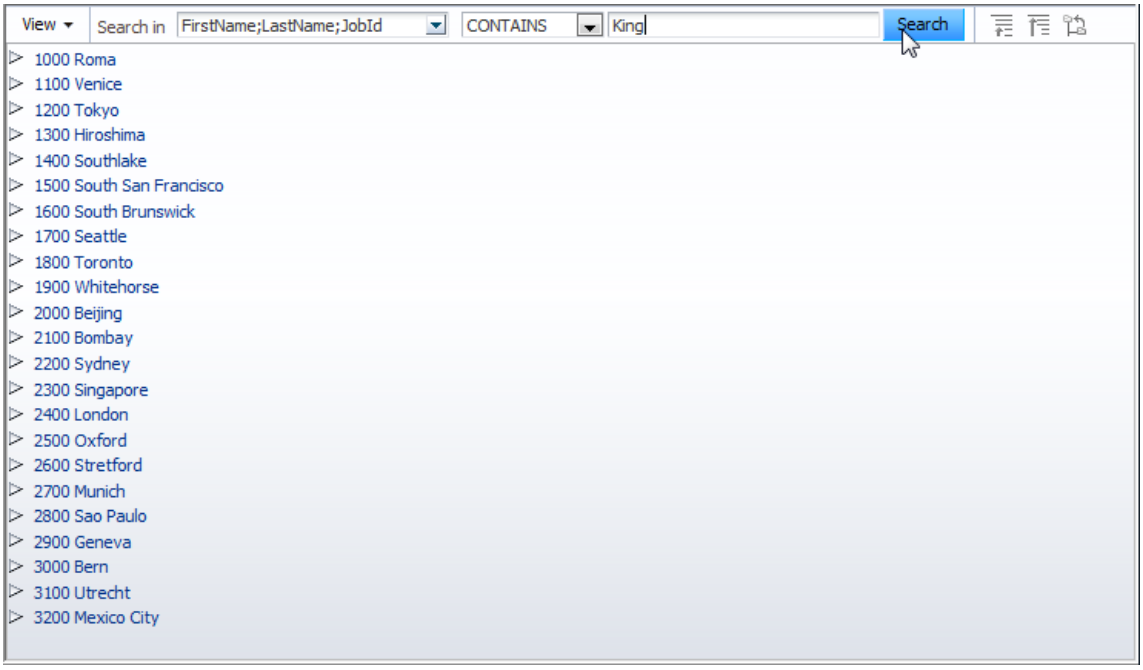
## Introduction

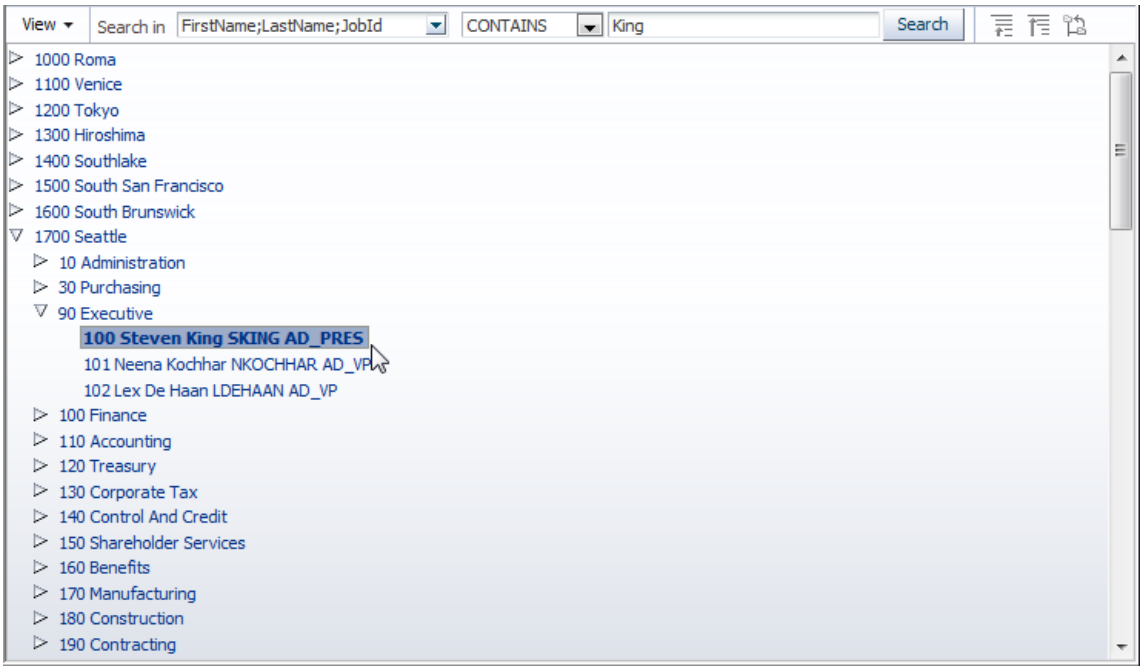
The sample code listed in this blog article allows users to search within a rendered tree component to find node or leaf entries that match a given search criteria. The tree configuration is to allow multi row (node) selection so that developers can perform additional operations on the tree nodes found.

The images below show a sample search. Users can select the node attributes to find a specific search string in. The search in the sample is for attribute values that start with, end with or contain the search string. The search is performed in a single ot multiple attributes.




View
Search in
FirstName;LastName;JobId
CONTAINS
CONTAINS
STARTS WITH
ENDS WITH
CONTAINS
Search
1000 Roma
1100 Venice
1200 Tokyo
1300 Hiroshima
1400 Southlake
1500 South San Francisco
1600 South Brunswick
1700 Seattle
1800 Toronto
1900 Whitehorse
2000 Beijing
2100 Bombay
2200 Sydney
2300 Singapore
2400 London
2500 Oxford
2600 Stretford
2700 Munich
2800 Sao Paulo
2900 Geneva
3000 Bern
3100 Utrecht
3200 Mexico City


View
Search in
FirstName;LastName;JobId
CONTAINS
King
Search
1000 Roma
1100 Venice
1200 Tokyo
1300 Hiroshima
1400 Southlake
1500 South San Francisco
1600 South Brunswick
1700 Seattle
1800 Toronto
1900 Whitehorse
2000 Beijing
2100 Bombay
2200 Sydney
2300 Singapore
2400 London
2500 Oxford
2600 Stretford
2700 Munich
2800 Sao Paulo
2900 Geneva
3000 Bern
3100 Utrecht
3200 Mexico City



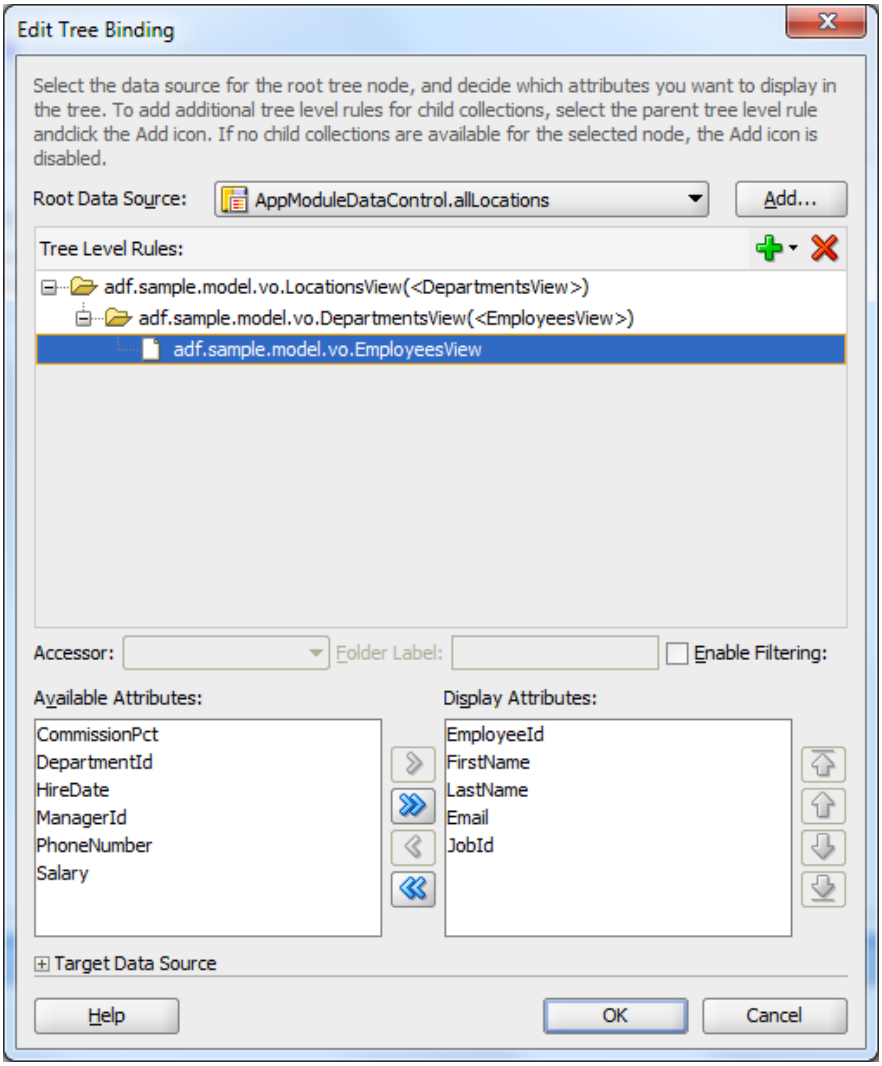

Using the ctrl-key, users can change the list of selected tree nodes by clicking on a node.

**Note:** See sample #026 "How-to access the selected row data in ADF bound TreeTable and Tree" on ADF Code Corner (http://www.oracle.com/technetwork/developer-tools /adf/ learnmore/ index-101235.html) for code samples that explain how to further process tree nodes selected by a user.

**Note:** The sample code is an excerpt from the McGraw Hill book "Oracle Fusion Developer Guide". The book samples are not publicly available for download. The plan is and was – over time - to release selected samples, updated for the current version of Oracle JDeveloper, here on ADF Code Corner.

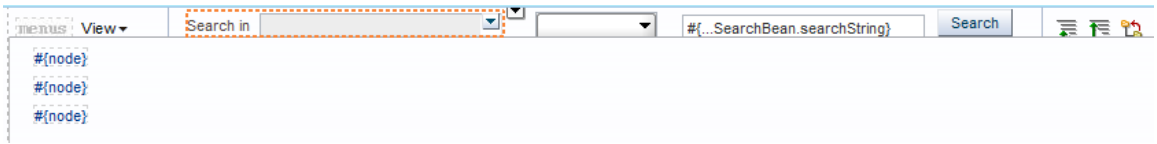
## Building the Tree

There is nothing specifial in building the tree. Just drag a collection – in the sample I used a View Object for the HR Locations table with detail references to Departments and Employees.




The tree component is surrounded by an af:panelCollection, which not only is my favorite ADF Faces component but also provides the toolbar area for the search component input fields.

However, be aware that the panel collection is a naming container, which needs to be taken into account when dynamically looking up the tree component in the view. The component id of the tree is prefixed with the id of the naming container, separated by a colon, ":"



```
<af:panelCollection id="pc1">
  <f:facet name="menus"/>
  <f:facet name="toolbar">
  <af:toolbar id="t2">
    <af:selectManyChoice label="Search in" id="sml1"
                         value="#{viewScope.SearchBean.searchAttributes}">
      <af:selectItem label="FirstName" value="FirstName" id="si4"/>
      <af:selectItem label="LastName" value="LastName" id="si2"/>
      <af:selectItem label="Email" value="Email" id="si6"/>
       <af:selectItem label="JobId" value="JobId" id="si1"/>
       <af:selectItem label="DepartmentName" value="DepartmentName"  id="si5"/>
        <af:selectItem label="City" value="City" id="si3"/>
       </af:selectManyChoice>
     <af:selectOneChoice label="" id="soc1"
                         value="#{viewScope.SearchBean.searchType}">
       <af:selectItem label="CONTAINS" value="CONTAIN" id="si7"/>
       <af:selectItem label="STARTS WITH" value="START" id="si8"/>
       <af:selectItem label="ENDS WITH" value="END" id="si9"/>
    </af:selectOneChoice>
    <af:inputText label="Search" id="it1" simple="true"
                  value="#{viewScope.SearchBean.searchString}"/>
    <af:commandToolbarButton text="Search" id="ctb1"
                             actionListener="#{viewScope.SearchBean.onSearch}"/>
   </af:toolbar>
  </f:facet>
  <f:facet name="statusbar"/>
  <af:tree value="#{bindings.allLocations.treeModel}" var="node"
           rowSelection="multiple" id="tree1" partialTriggers="::ctb1">
    <f:facet name="nodeStamp">
      <af:outputText value="#{node}" id="ot1"/>
    </f:facet>
    <f:facet name="pathStamp">
      <af:outputText value="#{node}" id="ot2"/>
   </f:facet>
 </af:tree>
</af:panelCollection>
```

## Managed Bean Search Code

The managed bean code to access the tree and parse its node values for the search string is provided as a reference. An Oracle JDeveloper sample workspace can be downloaded from the ADF Code Corner web site for you to try. Comments are added inline to explain the functionality in the managed bean.

**Note:** The managed bean is configured in viewScope for performance reason

```
/**
 * Managed bean configured in viewScope. It looks up the UI view root
 * for the tree component and keeps the handle until the bean moves out
 * of scope. The bean holds the values for the search attributes, the
 * position to search from STARTS, CONTAINED or ENDS, and the search
 * string.
 * When the search button is pressed, the tree component is iterated
 * and each node is tested for the search attributes and search string.
 * If attribute and search string match, the node is stored in a list
 * of rowKeys to select in the tree. At the same time, a list of
 * disclosed keys is generated that include the parent nodes.
*/
public class SearchBean {
 private String searchString = "";
 private String searchType = "CONTAIN";
 private List searchAttributes = new ArrayList();
 private RichTree tree1 = null;

 //log information to the ADF logger
 ADFLogger logger = ADFLogger.createADFLogger(this.getClass());

 public SearchBean() {
  super();
 }

 public void onSearch(ActionEvent actionEvent) {
   JUCtrlHierBinding treeBinding = null;

  //get handle to tree if it does not exist. If tree component
  //cannot be found in view, exit this function
  if (tree1 == null) {
    this.findTreeInView();
    if (tree1 == null) {
     //tree not found
       log("The tree component could not be found in the view. "+
           "Check for naming containers. Search function cancelled");
       return;
    }
  }

  //Get the JUCtrlHierbinding reference from the PageDef
  CollectionModel model = (CollectionModel)tree1.getValue();
  treeBinding = (JUCtrlHierBinding)model.getWrappedData();

  //Read the attributes to search in from the SelectManyChoice
  //component
```

```
  String searchAttributeArray[] = null;
  searchAttributeArray[] = (String[])searchAttributes.toArray(
                            new String[searchAttributes.size()]);

  //Define a node to search in. In this example, the root node
  //is used
  JUCtrlHierNodeBinding root = treeBinding.getRootNodeBinding();

  //However, if the user used the "Show as Top" context menu option to
  //shorten the tree display, then we only search starting from this
  //top mode

  List topNode = (List)tree1.getFocusRowKey();
    if (topNode != null) {
      //make top node the root node for the search
      root = treeBinding.findNodeByKeyPath(topNode);
  }

 //Select the tree items that match the search criteria and expand the
 //tree to display them. The resultRowKeySet contains the list of nodes
 //to select. The disclosedRowKeySet contains the keys of the selected
 //rows and their parent nodes
  RowKeySet resultRowKeySet =  searchTreeNode(root,
       searchAttributeArray,searchType, searchString);
  //define the row key set that determines the nodes to disclose.
  RowKeySet disclosedRowKeySet =
       buildDiscloseRowKeySet(treeBinding, resultRowKeySet);
  tree1.setSelectedRowKeys(resultRowKeySet);
  tree1.setDisclosedRowKeys(disclosedRowKeySet);
  //refresh the tree after the search
  AdfFacesContext.getCurrentInstance().addPartialTarget(tree1);

}

/**
 * Method that parses an ADF bound ADF Faces tree component to find
 * search string matches in one of the specified attribute names.
 * Attribute names are ignored if they don't exist in the search node.
 * The method performs a recursiv search and returns a RowKeySet with
 * the row keys of all nodes that contain the search string
 * @param  node The JUCtrlHierNodeBinding instance to search
 * @param  searchAttributes An array of attribute names to search in
 * @param  searchType defines where the search is started within the
 * text. Valid values are
 * START, CONTAIN, END. If NULL the "CONTAIN" is set as the default
 * @param  searchString  The search condition
 * @return RowKeySet row keys
*/
  private RowKeySet searchTreeNode(JUCtrlHierNodeBinding node,
```

```
                    String[] searchAttributes,
                    String searchType, String searchString){
 RowKeySetImpl rowKeys = new RowKeySetImpl();
 //set default search
 String _searchType = searchType == null ? "CONTAIN" :
                searchType.length() > 0 ? searchType : "CONTAIN";
 //Sanity checks
 if (node == null) {
   log("Node passed as NULL");
   return rowKeys;
 }
 if (searchAttributes == null || searchAttributes.length < 1) {
   log(node.getName() +
         ": search attribute is NULL or has a ZERO length");
   return rowKeys;
}

if (searchString == null || searchString.length() < 1) {
  log(node.getName() + ": Search string cannot be NULL or EMPTY");
  return rowKeys;
}
Row nodeRow = node.getRow();
 if (nodeRow != null) {
  for (int i = 0; i < searchAttributes.length; i++) {
    String compareString = "";
    try {
      Object attribute =
            nodeRow.getAttribute(searchAttributes[i]);
      if (attribute instanceof String) {
         compareString = (String)attribute;
      } else {
       //try the toString method as a simple fallback
       compareString = attribute.toString();
     }
    //not all nodes have all attributes. In this case an exception
    //is thrown that we don't need to handle as it is expected
  } catch (oracle.jbo.JboException attributeNotFound) {
    //node does not have attribute. Exclude from search
  }
//compare strings case insensitive.
if (_searchType.equalsIgnoreCase("CONTAIN") &&
 compareString.toUpperCase().indexOf(searchString.toUpperCase())> -1)
{
  //get row key
  rowKeys.add(node.getKeyPath());
```

```
  } else if (_searchType.equalsIgnoreCase("START") &&
    compareString.toUpperCase().startsWith(searchString.toUpperCase()))
  {
    //get row key
    rowKeys.add(node.getKeyPath());
  } else if (_searchType.equalsIgnoreCase("END") &&
     compareString.toUpperCase().endsWith(searchString.toUpperCase())){
     //get row key
     rowKeys.add(node.getKeyPath());
  }
 }
}

 List<JUCtrlHierNodeBinding> children = node.getChildren();
 if (children != null) {
   for (JUCtrlHierNodeBinding _node : children) {
    //Each child search returns a row key set that must be added to the
    //row key set returned by the overall search
    RowKeySet rks = searchTreeNode(_node, searchAttributes,
                                   this.searchType, searchString);
    if (rks != null && rks.size() > 0) {
      rowKeys.addAll(rks);
     }
   }
 }
 return rowKeys;
}
/**
  * Helper method that returns a list of parent node for the RowKeySet
  * passed as the keys argument. The RowKeySet can be used to disclose
  * the folders in which the keys reside. Node that to disclose a full
  * branch, all RowKeySet that are in the path must be defined
  *
  * @param  treeBinding ADF tree binding instance read from the PageDef
  *         file
  * @param  keys  RowKeySet containing List entries of oracle.jbo.Key
  * @return RowKeySet of parent keys to disclose
  */
  private RowKeySet buildDiscloseRowKeySet(
           JUCtrlHierBinding treeBinding,
           RowKeySet keys) {
    RowKeySetImpl discloseRowKeySet = new RowKeySetImpl();
    Iterator iter = keys.iterator();
    while (iter.hasNext()) {
      List keyPath = (List)iter.next();
      JUCtrlHierNodeBinding node =
```

```
            treeBinding.findNodeByKeyPath(keyPath);
          if (node != null && node.getParent() != null &&
            !node.getParent().getKeyPath().isEmpty()) {
            //store the parent path
            discloseRowKeySet.add(node.getParent().getKeyPath());
            //call method recursively until no parents are found
            RowKeySetImpl parentKeySet = new RowKeySetImpl();
            parentKeySet.add(node.getParent().getKeyPath());
            RowKeySet rks =
               buildDiscloseRowKeySet(treeBinding, parentKeySet);
               discloseRowKeySet.addAll(rks);
          }
       }
        return discloseRowKeySet;
      }

      private void log(String message) {
        logger.log(ADFLogger.WARNING, message);
      }

      public void setSearchString(String searchString) {
        this.searchString = searchString;
      }
      public String getSearchString() {
        return searchString;
      }

      public void setSearchType(String searchType) {
        this.searchType = searchType;
      }

      public String getSearchType() {
        return searchType;
      }

      public void setSearchAttributes(List searchAttributes) {
        this.searchAttributes = searchAttributes;
      }

      public List getSearchAttributes() {
        return searchAttributes;
      }

      //To learn what the code below is doing, please see Sample #58 on ADF
      //Code Corner
      //http://www.oracle.com/technetwork/developer-tools/
      //                  adf/learnmore/index-101235.html


    private void findTreeInView() {
```

```
  FacesContext fctx = FacesContext.getCurrentInstance();
  UIViewRoot root = fctx.getViewRoot();
  //hard coding tree component Id with its surrounding naming container
  //ID PanelCollection
  root.invokeOnComponent(fctx, "pc1:tree1", new ContextCallback() {
     public void invokeContextCallback(FacesContext facesContext,
                                       UIComponent uiComponent) {
     tree1 = (RichTree)uiComponent;
    }
  });
 }
}
```

## Sample Download

The Oracle JDeveloper 11.1.1.3 workspace is available for download as sample #061 from the ADF Code Corner website:
http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

Configure the model project to connect to a local database with the HR schema installed and enabled. Run the JPX page, choose attributes to search in and provide a search string. Then hit the search button.

**RELATED DOCOMENTATION**

| | |
|---|---|
| ⊠ | Chapter 9: Oracle Fusion Developer Guide – McGraw Hill Oracle Press, Frank Nimphius, Lynn Munsinger<br>http://www.mhprofessional.com/product.php?cat=112&isbn=0071622543 |
| ⊠ | Sample 026 "How-to access the selected row data in ADF bound TreeTable and Tree" on ADF Code Corner<br>http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html |
| ⊠ | |