ADF Code Corner

026. How-to access the selected row data in an ADF bound ADF Faces TreeTable or Tree

Abstract:

The ADF Faces af:treeTable component displays hierarchical data in a combined tree and table structure. In this, the leading column is displayed as a tree, while subsequent columns, which could represent different hierarchy levels, are displayed in a table format. If you use ADF to bind the af:treeTable to business data, then a tree binding definition in the PageDef file is used by the ADF binding layer to provides the CollectionModel instance that is required by the af:treeTable as a component model.

The question answered in this blog is how developers use Java in a managed bean to determine the hierarchical data that represents the af:treeTable row that a user selected.

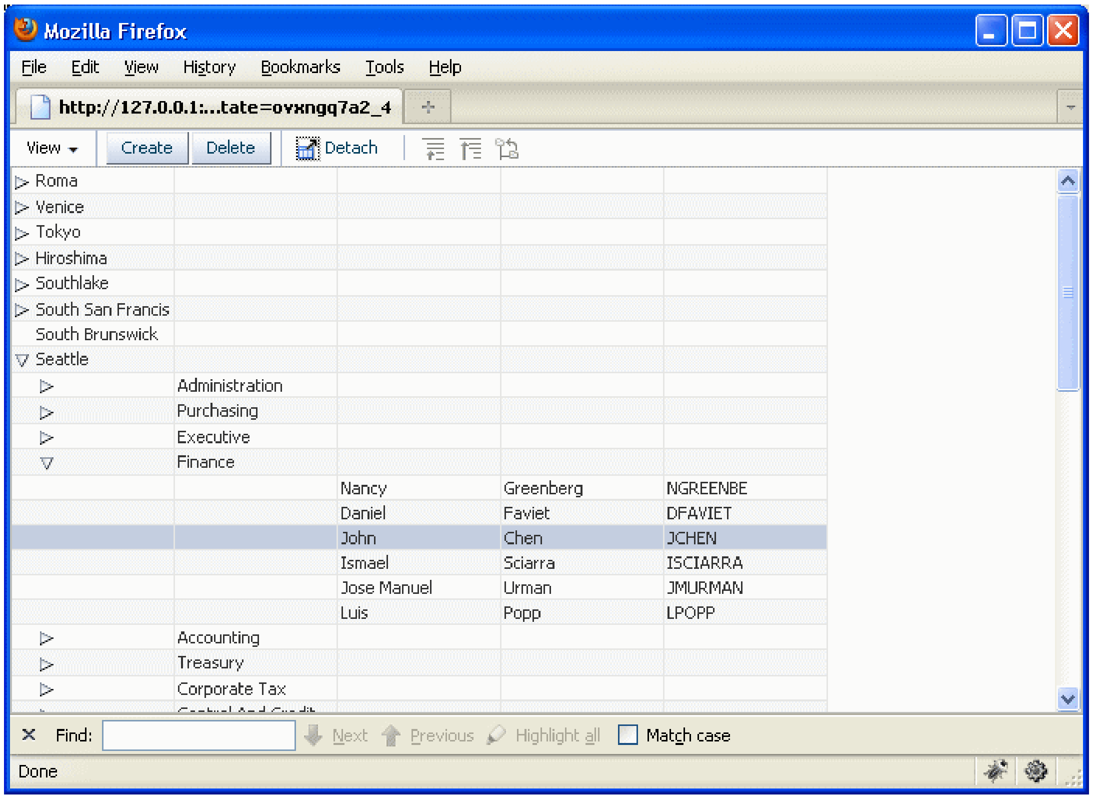
TreeTables and trees share the same bindings, which is why the code examples shown in this article can be used with both, ADF Faces tree and treeTable components.

twitter.com/adfcodecorner

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
09-NOV-2009

Introduction

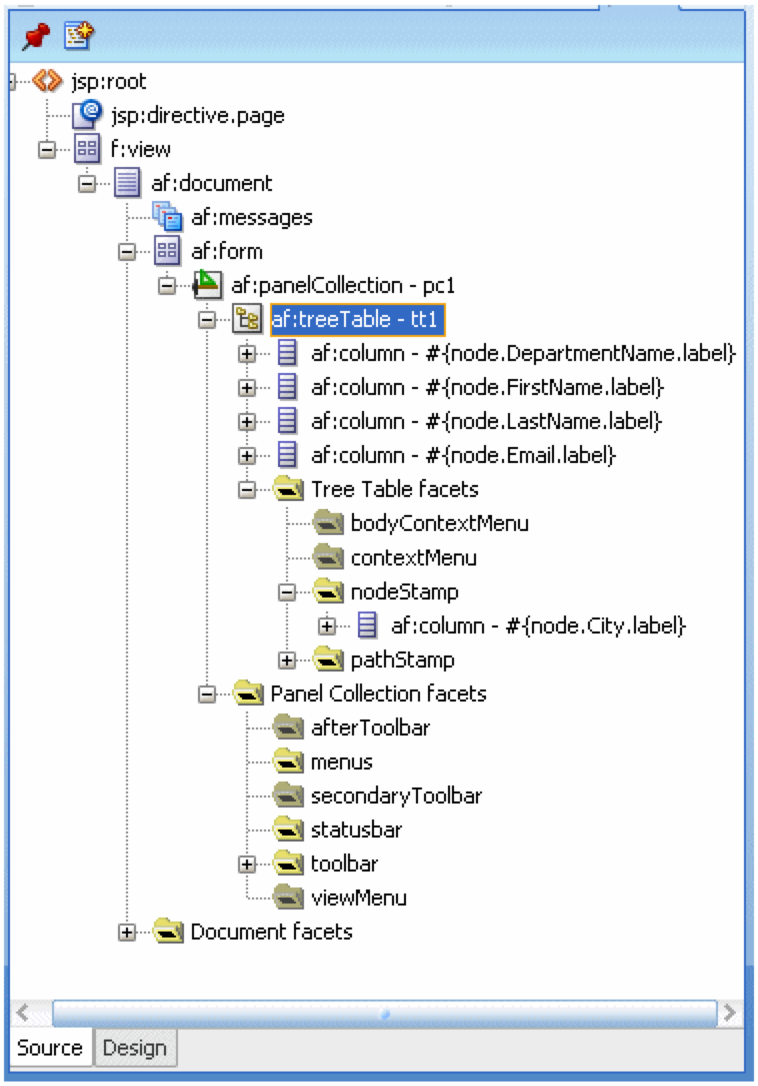
In ADF, ADF Faces table, tree and tree table components are bound to the tree binding in the ADF binding layer. At runtime, the binding definition file, build as the page's PageDef file, becomes the content of the binding container. The binding container does not operate on XML metadata, but Java objects, which are instances of the FacesCtrl* bindings if using ADF Faces as the user interface. All FacesCtrl* classes extend the generic binding classes, which for the tree binding are JUCtrlHierBinding and JUCtrlHierNodeBinding. So for developers to know what the selected data node is when the user selects a row in a tree table, these two classes are important to know about at least to know they exist. In the following we provide you example code that gives you access to the selected treeTable data.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:...tate=ovxngq7a2_4
View Create Delete Detach
Find: Next Previous Highlight all Match case
Done

▷ Roma				
▷ Venice				
▷ Tokyo				
▷ Hiroshima				
▷ Southlake				
▷ South San Francis				
South Brunswick				
▽ Seattle				
▷	Administration			
▷	Purchasing			
▷	Executive			
▽	Finance			
		Nancy	Greenberg	NGREENBE
		Daniel	Faviet	DFAVIET
		John	Chen	JCHEN
		Ismael	Sciarra	ISCIARRA
		Jose Manuel	Urman	JMURMAN
		Luis	Popp	LPOPP
▷	Accounting			
▷	Treasury			
▷	Corporate Tax			

Note: The screenshot above is taken using Oracle JDeveloper 11g R1 Patch Set 1, which by the time of writing is not yet publicly available. However, the code examples contained in this article work with any release of Oracle JDeveloper 11g.

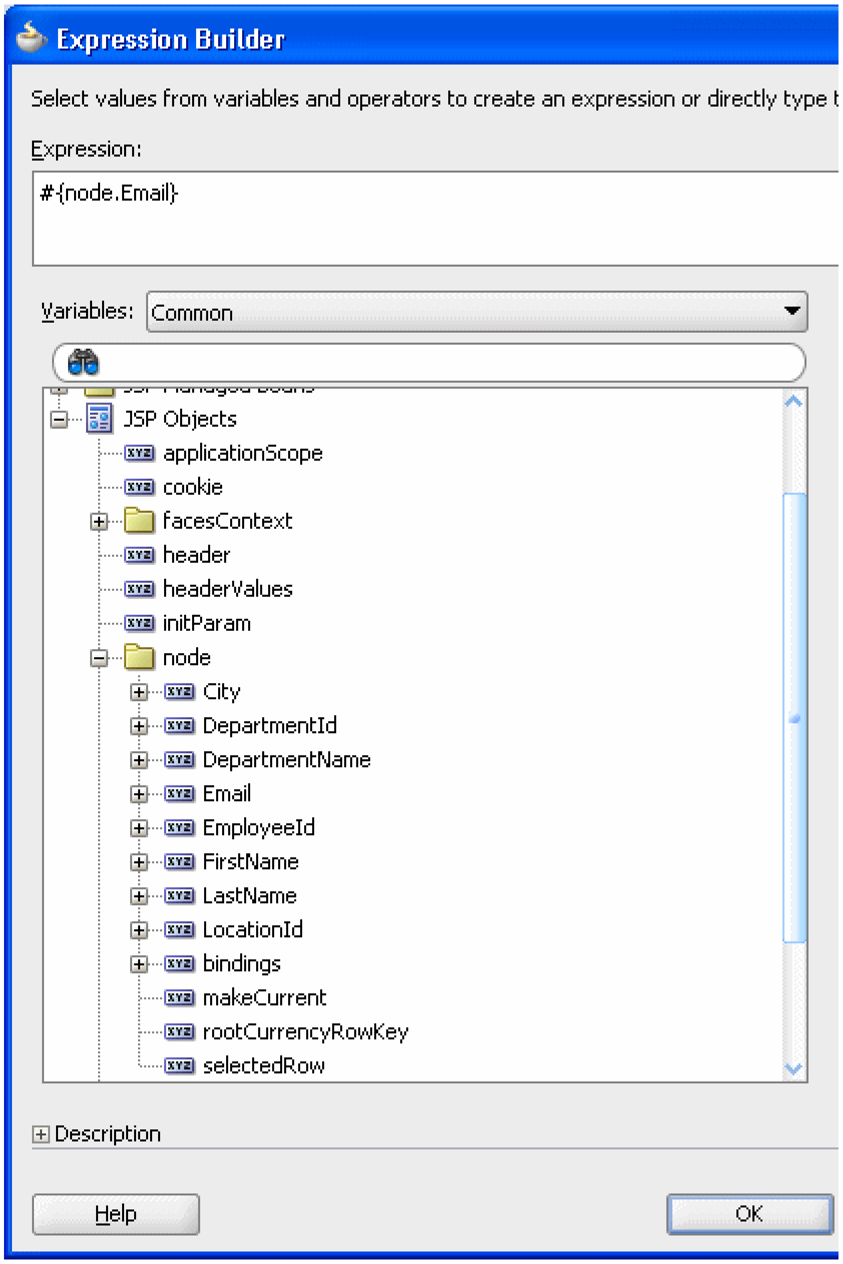
There is a minor difference between working with ADF Business Components and working with POJOs as the business service, which is why we keep the use cases separated. In both cases, you start by creating the treeTable by dragging a Collection from the DataControls palette to the JSF page to then create the tree table. Note that by default, the tree table is created with a single column that is populated by the nodePath facet. To make a tree table out of the initial design, you need to create af:column instances within the af:treeTable and add components that render the cell value into them, as shown in the image below



```
jsp:root
  jsp:directive.page
  f:view
    af:document
      af:messages
      af:form
        af:panelCollection - pc1
          af:treeTable - tt1
            af:column - #{node.DepartmentName.label}
            af:column - #{node.FirstName.label}
            af:column - #{node.LastName.label}
            af:column - #{node.Email.label}
            Tree Table facets
              bodyContextMenu
              contextMenu
              nodeStamp
                af:column - #{node.City.label}
              pathStamp
          Panel Collection facets
            afterToolbar
            menus
            secondaryToolbar
            statusbar
            toolbar
            viewMenu
      Document facets
```

The tree table above contains one column definition in the nodeStamp facet, which is the name of the city we are looking at. The other four columns represent the departments and associated employees . The visual display at runtime is shown in the image on top.

When you configure the cell render component of the af:column component with a data reference, note that you reference a treeTable variable and not the "bindings" object. by default the tree table variable is defined as "node". The variable name, which you can change in the Property Inspector, also shows in the Oracle JDeveloper 11g Expression Builder under the JSP Objects node. This provides you with an error free configuration of the EL required to display tree table data. The attributes that are displayed beneath the "node" entry in the Expression Builder represent all the attributes that you selected for access in the tree binding dialog (Since this is an advanced topic we cover here, our assumption is that you know how to create tree rules that build the tree structure in the ADF tree binding dialog)



```
#{node.Email}
```

Once you built the treeTable, select the component's "binding" property in the PropertyInspector and create a managed bean reference for it. This allows your managed bean to access the instance of the af:treeTable component at runtime. Note that another way to access the treeTable component at runtime is to search for it in JavaServer Faces UIViewRoot. This latter option allows you to write more generic code and does not require to create a page dependency to a managed bean. If the client logic you execute on the selected treeTable row value is initiated by an event raised by the treeTable, then the handle to the treeTable component instance can be obtained from the event object by calling getSource() on it.

ADF Business Components use case

The source code below assumes that the user pressed a command button to perform action on the selected af:treeTable node. However, as you see, the actionEvent object of the command button action listener implementation is not used, which means that the same code can be used elsewhere.

```
import java.util.Iterator;
import java.util.List;
import javax.faces.event.ActionEvent;
import oracle.adf.view.rich.component.rich.data.RichTreeTable;
import oracle.adfdt.model.objects.TreeTable;
import oracle.jbo.uicli.binding.JUCtrlHierBinding;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;
import org.apache.myfaces.trinidad.model.CollectionModel;
import org.apache.myfaces.trinidad.model.RowKeySet;

public class TreeTableHandlerBean {
 private RichTreeTable treeTable1;
 public TreeTableHandlerBean() {}

//user action handling
public void onRowCreateAction(ActionEvent actionEvent) {
  RichTreeTable treeTable = this.getTreeTable1();
  //get selected row keys. This could be a single entry or
  //multiple entries dependent on whether the tree table is
  //configured for multi row selection or single row selection
  RowKeySet rks = treeTable.getSelectedRowKeys();

  Iterator rksIterator = rks.iterator();

  //assuming single select use case. Otherwise, this is where you
  //need to add iteration over the entries in the multi select use case

  if (rksIterator.hasNext()){
    //a key is a list object that containe the node path information
    //for the selected node
    List key = (List)rksIterator.next();
    //determine the selected node. Note that the tree table binding is
    //an instance of JUCtrlHierBinding
    JUCtrlHierBinding treeTableBinding = null;
    //We can get the binding information without using EL in our Java,
    //which you always should try to do. Using EL in Java is good to
    //use, but only second best as a solution
    treeTableBinding = (JUCtrlHierBinding)
            ((CollectionModel)treeTable.getValue()).getWrappedData();
  //the row data is represented by the JUCtrlHierNodeBinding class at
  //runtime. We get the node value from the tree binding at runtime
  JUCtrlHierNodeBinding nodeBinding =
               treeTableBinding.findNodeByKeyPath(key);
  //the JUCtrlHierNodeBinding object allows you to access the row data,
  //so if all you want is to get the selected row data, then you are
  //done already. However, in many cases you need to further
  //distinguish the node, which is what we can do using the
  //HierTypeBinding
  String nodeStuctureDefname =
             nodeBinding.getHierTypeBinding().getStructureDefName();

   //determine the selected node by the ViewObject name and package

   String departmentsDef = "adf.sample.model.DepartmentsView";
   String employeesDef = "adf.sample.model.EmployeesView";
   String locationsDef = "adf.sample.model.LocationsView";

   if (nodeStuctureDefname.equalsIgnoreCase(locationsDef)){
    //work with location node data
   }
   else if (nodeStuctureDefname.equalsIgnoreCase(departmentsDef)){
     //work with departments node data
   }
   else if (nodeStuctureDefname.equalsIgnoreCase(employeesDef)){
     //work with location node data
   }
   else{
    //what the heck did the user click on? Ask him ;-)
   }
  }
 }

  ... access to the treeTable instance ...
}
```

Note: If the child node is a read only View Object, make sure the View Object has a primary key defined. If the View Object queries multiple tables then the key need to be set for more than one attribute. Open the VO, select an attribute and check the "key" checkbox. If you don't do this, a NPE is thrown

Pojo use case

The code you use for the POJO use case is slightly different in the way the row data is accessed. Also, since we are working with POJOs, we determine the node level by checking the node object instance against the POJO objects. The sample code below actually determines the selected node / row value in an ADF Faces tree. The source code though is the same as when working with an af:treeTable component since the underying ADF bindingstructure is identical. This also is a great example of how learning a single API allows to you program against different business services and UI components.

```
import java.util.List;
import javax.faces.event.ActionEvent;
import oracle.adf.model.bean.DCDataRow;
import oracle.adf.view.rich.component.rich.data.RichTree
import oracle.jbo.uicli.binding.JUCtrlHierBinding;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;

//my custom POJO entities
import oracle.pojo.entities.Departments;
import oracle.pojo.entities.Employees;
import oracle.pojo.entities.Locations;


//trinidad classes

import org.apache.myfaces.trinidad.model.CollectionModel;
import org.apache.myfaces.trinidad.model.RowKeySet;

public class TreeBean {
  //this example uses a tree. However, the data access is the same as
  //for tree tables using an ADF tree binding
  private RichTree locationTree;

  public TreeBean() {
  }

  //method executed when the user presses the command button in our
  //example
  public void onRowCreateAction(ActionEvent actionEvent) {

  //get access to the Tree Collection Model. The tree component
  //instance is accessed through its binding property reference
  //to this managed bean

  CollectionModel treeModel = null;

  //access the tree component from a JSF component binding, a search on
  //the UIViewRoot, or if the method is invoked from a tree event, from
  //the getSource() call
  treeModel = (CollectionModel) this.getTree().getValue();
  //The CollectionModel is of type FacesModel, which is an inner class.
  //To get to the ADF tree binding, we can call wrappedData on the
  //Collection Model
  JUCtrlHierBinding treeBinding =
            (JUCtrlHierBinding) treeModel.getWrappedData();
  //get the selected tree nodes
  RowKeySet rks = locationTree.getSelectedRowKeys();
  //If there is a tree node selected ...
  if(!rks.isEmpty()){
  //get first selected node as we assume this code to execute in a
  //single row selection use case. Iterate over the whole iterator if
  //you are in a multi node selection case

  List firstSet = (List)rks.iterator().next();

  //get the ADF node binding. If you want to access sub nodes, make
  //sure that you set the primary key attribute for the entity
  //representing this node.
```

```
    //If you don't do this, the next line throws an NPE

   JUCtrlHierNodeBinding node =
     treeBinding.findNodeByKeyPath(firstSet);

    //The Row is of type DCDataRow, a binding class not often used,
    //especially if you are used to the Fusion development environment

    DCDataRow rw = (DCDataRow) node.getRow();

    //The data provider is the object - the entity - that is used to
    //render the node.
    Object entity = rw.getDataProvider();

    //determine the object so you can type cast it if needed.

    if (entity instanceof Locations){
     //do some location work here
     return;
    }
    if (entity instanceof Departments){
     //do some department work here
     return;
    }
    if (entity instanceof Employees){
     //do some employee work here
     return;
    }
   }
  }
... access to the tree instance ...
}
```

In this example, note the use of the DCDataRow class to access the entity that represents the selected node/row. Note that the DCDataRow object is another ADF binding class. Also note how similar the code shown above is compared to the ADF Business Components example where we used the af:treeTable component.

Disclaimer

The content of this blog article is an advanced topic and is not representative for working with ADF. If you are a beginner with ADF, don't be too impressed with the content - or shocked because you don't understand it in detail. As a beginner your initial look should be at the developer guides published in the documentation section on OTN and www.oracle.com, as well as the Oracle by Example tutorials. Sooner or later you will come back to this example, fully understanding what it does and enjoying the power that ADF provides beyond drag and drop development.

This blog example explains how to access the selected tree node or tree table node data from Java, which a use business use case may require you to do. There also is a strategy to get the same information declaratively using Expression Language. For example, using ADF Business Components, you create iterators for the View Objects that have data representations in the tree structure. The tree binding can be configured to make sure that the selected node always synchronizes the iterator. Creating attribute bindings for the iterators and reference them from ADF Faces input components, allow to easily and declaratively building input forms for the selected tree nodes. After all, ADF is there to simplify Java EE development using declarative development gestures. However, if the use case demands it that you need to access the selected hierarchical structure value in Java, then no you know.

RELATED DOCOMENTATION

☒	
☒	
☒	