ADF Code Corner

033. How-to open a Bounded Task Flow in a new Browser Tab

twitter.com/adfcodecorner

Abstract:

Web browsers allow users to browse web pages in tab so they don't need to open new browser windows.

Because tab browsing is so common these days, you may have the requirement to allow users to do the same in your ADF application. In this how-to article I show how to achieve tab browsing as close as it gets using the Oracle Fusion development stack and ADF.

The browsed page is wrapped in a bounded task flow to allow developers to define the transaction and data control scope for the opened pages.

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

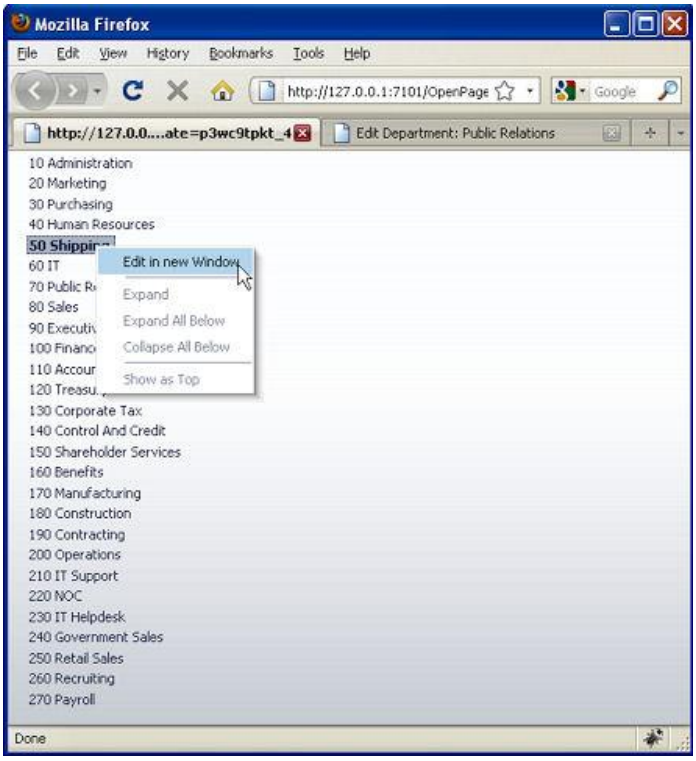
Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

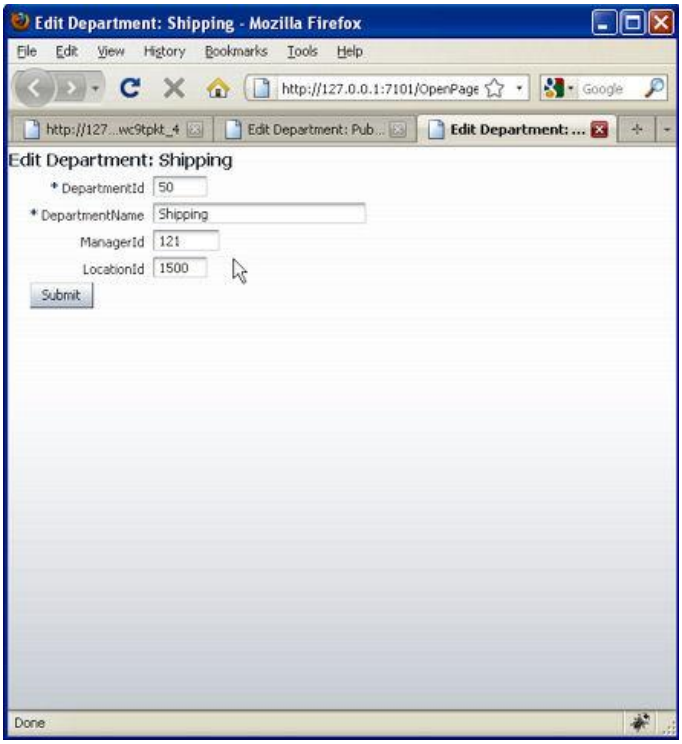
Tab browsing is very popular and a common user pattern. The benefit tab browsing provides is that you can open multiple views for comparison or quick access. In JavaSever Faces, the easiest approach for tab browsing is to use goLinks to render links on a page. This link then needs to have all the request parameters added to query the page so the intended content displays. In application development, you however need to go more complex, for example to show a list of queried database rows in a table or tree to then make these accessible for tab browsing. In this article I use a hierarchical tree for displaying department record. To edit a department, you use the right mouse button to launch a new browser tab from the context menu, displaying the requested content.

What you learn in this article:

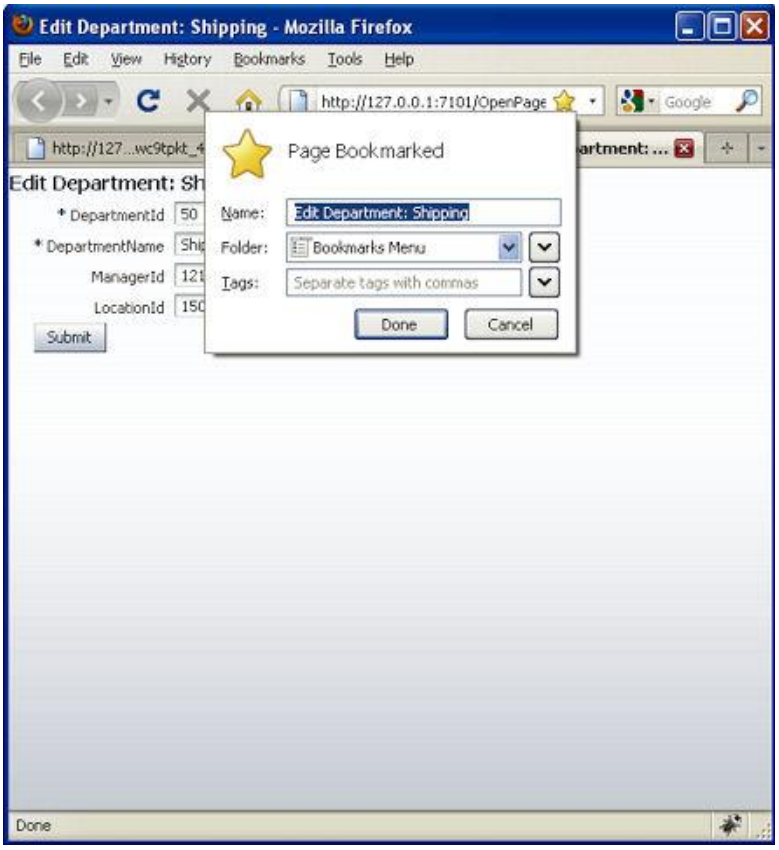
- Making a bounded task flow URL accessible
- Launching a new browser window from a managed bean
- Defining the bounded task flow request URL in Java
- Passing input parameters to a task flow using a HashMap
- Use the hierarchical tree context menu to invoke an action on right mouse click
- Adding JavaScript to a managed bean response



Selecting "Edit in new Window" opens a new browser tab to edit the selected record. Note that the browser tab contains a title that includes the name of the department that is edited (though hardly visibly in the above figure).



As you will see when I go to explain how to build this sample, tab browsing in ADF Faces allows you to control the transaction context when using the ADF controller. Because a redirect is performed for editing the selected department, it is also possible to bookmark the page.



Once done with editing the page, you press submit to commit the change, which leads you to a confirmation page

Figure 3: Submitting the edit page allows you to continue with a task flow, like for showing a confirmation dialog

Creating the af:tree

The hierarchical tree component uses a ADF to access the tree data from the business service layer. The tree provides the following out of the box functionality that makes this solution possible

- Context menu handling
- Setting the selected node as current in the ADF binding

The context menu is added to the af:tree contextMenu facet. The contextMenu facet is an invisible area of the tree that is displayed when selecting the tree with the right mouse button. It expects an af:popup component as a child component.

The af:popup component then contains the menu to display. In the source code below, then menu contains a single menu item, which is the "Edit in new Window" option. The menu item invokes an action listener method that is defined in a managed bean in request scope. As shown in figure 1, the context menu has more menu entries than the one defined in the page source. The other context menu entries are automatically added by the ADF Faces component and allow users to manipulate the tree.

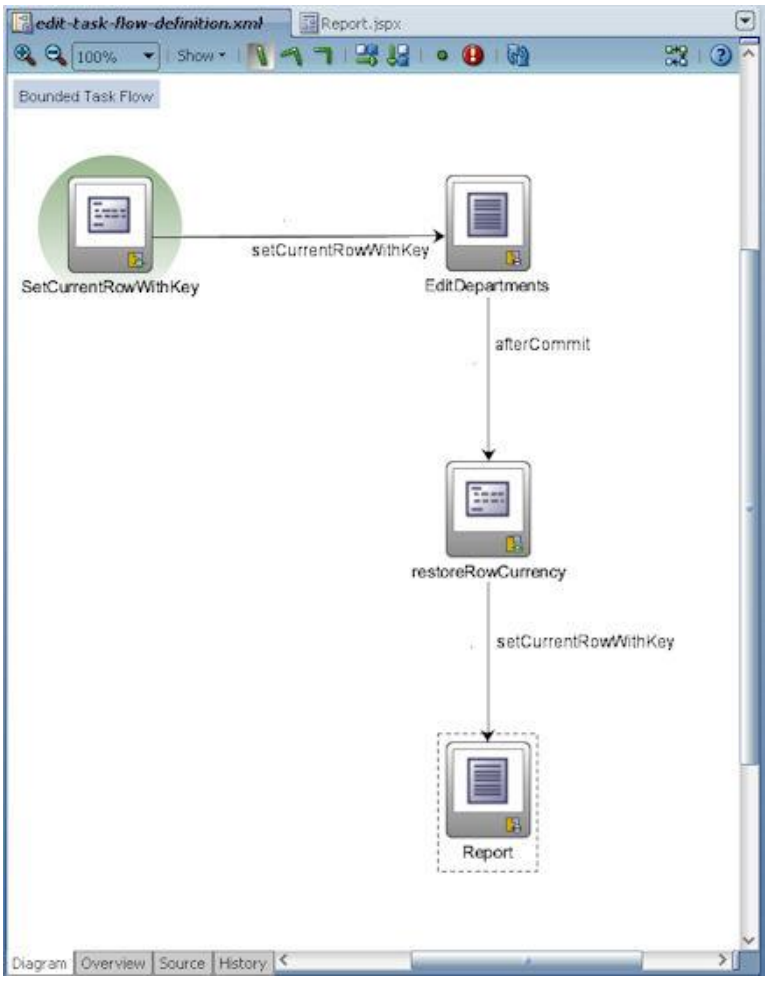
```
<af:tree value="#{bindings.DepartmentsView1.treeModel}" var="node"

selectionListener="#{bindings.DepartmentsView1.treeModel.makeCurrent}"
    rowSelection="single" id="t1">
  <f:facet name="nodeStamp">
    <af:outputText value="#{node}" id="ot1"/>
  </f:facet>
  <f:facet name="contextMenu">
    <af:popup id="p1" contentDelivery="lazyUncached">
      <af:menu text="menu 1" id="m2">
        <af:commandMenuItem text="Edit in new Window" id="cmi1"
           actionListener=
               "#{BrowseDepartmentsBacking.onLaunchForEdit}"/>
      </af:menu>
    </af:popup>
  </f:facet>
</af:tree>
```

Note the use of selectionListener and rowSelection attribute, which are configured by default, ensuring that the selected tree node is synchronized with the underlying model, which is the ADF binding layer in this example. Not explicitly set is the contextmenuSelect attribute that allows you to configure the table such that a right mouse click on a node makes this node the selected row in the tree. This selection behavior is the default selection behavior in the tree and therefore does not need to be explicitly set.

Creating the bounded task flow

The edit page in this example is launched in the context of a bounded task flow. Using a bounded task flow allows us to reuse the edit page in the running application, isolate the bounded task flow instance from the data control usage and to open a new transaction context, allowing us to perform commit and

rollback of the edited values without impacting other parts of the application. The process flow of the bounded task flow in this example is shown below.



The default activity of the bounded task flow is a method activity that is created by dragging the *SetCurrentRowWithKey* operation from the Data Controls palette onto the task flow diagram.

The required argument, the key to set as current, is passed to the bounded task flow as a task flow parameter. I made the task flow parameter required so that any attempt to run the task flow without providing a parameter leads to an error. When the current row is set, the edit page is called. The input form of the edit page is created by dragging the departments View Object from the Data Controls palette to the page and choosing Editable Forms from the context menu. The submit button of the edit form (make sure you select the option in the form creation dialog) does not yet commit the update. To do this, drag and drop the "Commit" operation from the Data Controls palette to the submit button. In the opened dialog, make sure that all settings of the button are kept as they are except the actionListener property that should be changed to contain the EL reference to the "Commit" operation in the binding.

```
<af:commandButton text="Submit" id="cb1"
  actionListener="#{bindings.Commit.execute}"
  action="afterCommit"/>
```

The "afterCommit" setting of the *action* attribute forces navigation to the confirmation page. Before this however, I set the selected row back to the edited row. This is important because commit and rollback operations reset the iterator. Because the input parameter that gets passed when requesting the bounded task flow is accessible for the duration of the task flow processing, it is no big deal to re-create the row selection.

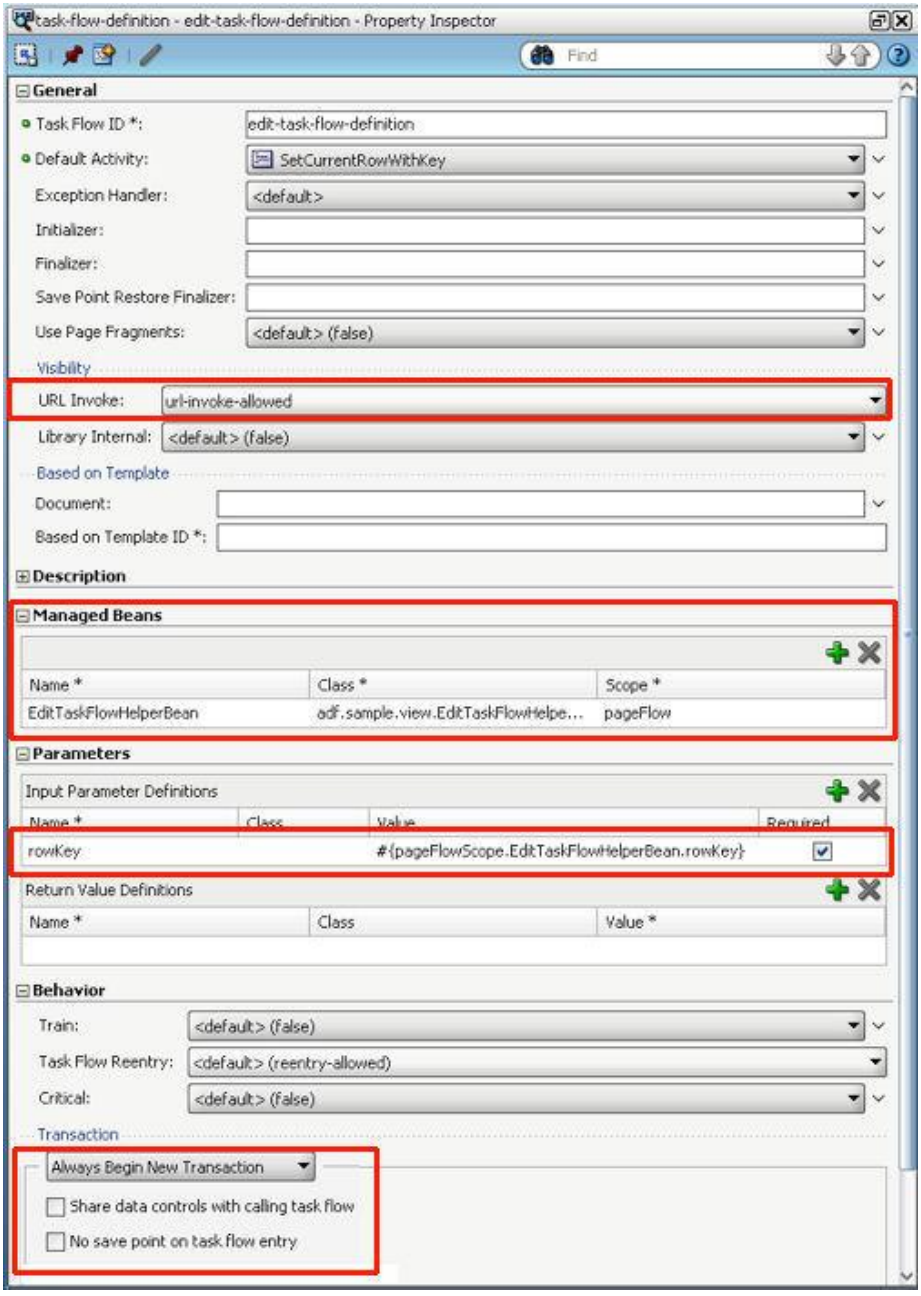


Figure 5 highlights the settings that make this use case work. First of all, and new in JDeveloper 11g R1 PS1, a bounded task flow cannot be called directly from a browser URL field. This security related configuration is defined by the "URL Invoke" property that needs to be set to "url-invoke-allowed" to enable direct GET requests for a bounded task flow.

To store the task flow input parameter value for later use during task flow processing, I chose a managed bean setter/getter pair. The alternative to using a managed bean is to write the value directly to a memory scope (e.g. #{pageFlowScope.rowKey}). Using an in memory attribute is good to use if you don't want to use the Expression Builder in JDeveloper to access the input parameter and if you don't need type checking of the parameter value that is passed in. Another benefit of the use of managed beans is code documentation. It is easy to document the meaning of an input parameter in a Java class, but you can't do the same if using a memory attribute (just in case you work in teams and other people need to understand what you did in your coding). In my example, I could have used both but decided for the managed bean approach.

The row key of the row to edit must be passed into the task flow. I made the rowKey parameter required so the task flow wont execute if this parameter is missing. Note however that it does not check if the parameter value is null, it only checks if the parameter itself is missing from the request.

Last but not least, I isolated the bounded task flow from the application data control instance and the transaction, so that it runs completely independent. Note however that this setting also is the most expensive setting because a new data control frame and transaction context needs to be created for each opened task flows instance of this kind.

ActionListener referenced by the menu item

The last bit of information for me to explain is the content of the action listener method that is referenced by the "Edit in new Window" menu item in the context menu. The managed bean is a different one than the bean I use to handle the task flow input parameter.

The managed bean is configured in the adfc-config.xml file and I recommend to always ensure that managed beans are configured in the task flow configuration definition in which process scope it is used. The managed bean code is shown below

```
package adf.sample.view;
import java.util.HashMap;
import java.util.Map;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;
import oracle.adf.controller.ControllerContext;
import oracle.adf.controller.TaskFlowId;
import oracle.adf.model.BindingContext;
import oracle.binding.BindingContainer;
import oracle.jbo.uicli.binding.JUCtrlHierBinding;
import org.apache.myfaces.trinidad.render.ExtendedRenderKitService;
import org.apache.myfaces.trinidad.util.Service;

public class BrowseDepartmentsBacking {

  //task flow document definition and ID. The task flow document name
  //starts from the WEB-INF directory and includes the '.xml' extension
  private final String taskflowId=
                         "edit-task-flow-definition";
  private final String taskflowDocument=
                         "/WEB-INF/edit-task-flow-definition.xml";
```

```
    public BrowseDepartmentsBacking() {
        super();
    }

    /*
     * method called by the command menu item on the context menu.
     */
    public void onLaunchForEdit(ActionEvent actionEvent) {
      //access the ADF binding layer and access the tree binding that
      //populates the table
        BindingContext bctx = BindingContext.getCurrent();
        BindingContainer bindings =
                                bctx.getCurrentBindingsEntry();
        //access the tree binding defined in the ADF binding layer
        JUCtrlHierBinding model =
               (JUCtrlHierBinding) bindings.get("DepartmentsView1");
        //get the current selected row key from the iterator binding
        //referenced from the table binding (tree binding). Of course,
        //I could have used the iterator name directly in the binding
        //lookup, but looking up the tree binding instead allows to
        //change the tree binding iterator dependency to a later point
        //in time without breaking this functionality. Its all about
        //"weak" dependencies that give you flexibility when coding ADF
        String rwKeyString =
               model.getDCIteratorBinding().getCurrentRowKeyString();
        launchWindow(rwKeyString);
    }

    private void launchWindow(String rwKey) {
      FacesContext fctx = FacesContext.getCurrentInstance();
      String rowKeyString = rwKey;
      //task flow parameters can be passed as input arguments
      //configured on a task flow call activity or ADFregion, or they
      //can be provided in a hash map
      Map<String, Object> params = new HashMap<String, Object>();
      params.put("rowKey",rowKeyString);
      String taskflowURL =
            ControllerContext.getInstance().getTaskFlowURL(false,
            new TaskFlowId(taskflowDocument,taskflowId),params);
      ExtendedRenderKitService erks =
           Service.getRenderKitService(fctx,
                                       ExtendedRenderKitService.class);
      StringBuilder script = new StringBuilder();
      script.append("window.open(\""+taskflowURL+"\");");
      erks.addScript(FacesContext.getCurrentInstance(),
                     script.toString());
    }

}
```

The action listener method accesses the current row key from the iterator binding in the ADF binding. The rowKey then is appended to the task flow request URL, which is composed out of the task flow document name and the task flow ID. Using the Trinidad ExtendedRenderKitService class, we use JavaScript in the action listener response to open the browser window.

Important - please read !: There is no JavaScript API to open a browser tab. The way this sample works is that I omit any further information about the opened window, like location, window title, to show or not to show the URL field etc. It is dependent on the browser configuration whether a new window is opened as a tab or separate window. In Firefox it appears that the default setting is to open it in a tab. On IE, I had to change the new window behavior by setting Tools | Internet Options | Tabs > Settings | Always open popups in a new tab. So if you are in a corporate network, you may have the chance to ask users to change the IE settings accordingly. If not - pray for the user browser to be FF ;-)

Download Sample

You can download the zip file that contains the JDeveloper 11g R1 PS1 workspace. The business service layer uses ADF Business Components. For this to work, make sure you change the database configuration to point to a database near you that has the HR schema installed and unlocked. **Download from ADF Code Corner:**

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

RELATED DOCOMENTATION

⊠	Product Documentation – Managing Task Flow Transactions http://download.oracle.com/docs/cd/E15523_01/web.1111/b31974/taskflows_complex.htm#CHDFJDCE
⊠	Calling JavaScipt in ADF Faces http://download.oracle.com/docs/cd/E15523_01/web.1111/b31973/af_event.htm#BABHIBGF
⊠	Oracle Fusion Developer Guide – McGraw Hill Oracle Press, Frank Nimphius, Lynn Munsinger http://www.mhprofessional.com/product.php?cat=112&isbn=0071622543