# ADF Code Corner

## The Oracle JDeveloper Forum Harvest 10 / 2010


twitter.com/adfcodecorner

**Abstract:**

The Oracle JDeveloper forum is in the Top 5 of the most active forums on the Oracle Technology Network (OTN). The number of questions and answers published on the forum is steadily increasing with the growing interest in and adoption of the Oracle Application Development Framework (ADF).

The ADF Code Corner "Oracle JDeveloper OTN Harvest" series is a monthly summary of selected topics posted on the OTN Oracle JDeveloper forum. It is an effort to turn random knowledge exchange into an interesting read for developers who enjoy harvesting little nuggets of wisdom.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
30-OCT-2010

*Oracle ADF Code Corner OTN Harvest is a monthly blog series that publishes how-to tips and information around Oracle JDeveloper and Oracle ADF.*

*Disclaimer: ADF Code Corner OTN Harvest is a blogging effort according to the Oracle blogging policies. It is not an official Oracle publication. All samples and code snippets are provided "as is" with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.*

*If you have questions, please post them to the Oracle OTN JDeveloper forum:*
*http://forums.oracle.com/forums/forum.jspa?forumID=83*

## Table of Contents

# OTN Harvest

Below is a summary – or call it a premium selection – of OTN topics posted to the OTN forum this month. The answers provided in this document may not be the only solution possible to a problem; however, they give you some ideas and code snippets to play with.

## Oracle JDeveloper 10.1.3 documents are back on OTN

Due to an infrastructure change on OTN, content URLs have changed for previously published materials. In addition, some documents didn't make the automated migration and therefore needed to be uploaded again. The documentation team took on this challenge and successfully recovered the Oracle JDeveloper 10.1.3 documents:

The index page:
http://download.oracle.com/docs/cd/B25221_05/web.1013/e18745/toc.htm

With the documentation link:
http://download.oracle.com/docs/cd/B25221_05/web.1013/e18745/toc.htm#Documentation

The Javadoc:
http://download.oracle.com/docs/cd/B25221_05/web.1013/e18745/apidocs/index.html

The Core tag doc:
http://download.oracle.com/docs/cd/B25221_05/web.1013/e18745/tagdoc/core/imageIndex.html

The HTML tag doc:
http://download.oracle.com/docs/cd/B25221_05/web.1013/e18745/tagdoc/html/index.html

In addition, the 10.1.3.5 ADF Faces component demo war file can be downloaded from here

http://www.oracle.com/technetwork/testcontent/adffaces1013-096465.html

## How-to get help for OAF framework

Developers use to post questions about the Oracle Applications Framework (OAF) on the Oracle JDeveloper forum on OTN, either to find help or a new version of Oracle JDeveloper with OAF on board. OAF however is not related to Oracle JDeveloper and instead has its own forum to post questions to:

http://forums.oracle.com/forums/forum.jspa?forumID=210

## Oracle Fusion Reference Architecture (OFRA)

Many Oracle Fusion Middleware and ADF customers are eager to learn from Oracle Fusion Applications how to setup Fusion Middleware at runtime and design time for building large scale business applications. The Oracle Fusion Reference Architecture (OFRA) is an initiative to release such information authorized by the Oracle Applications group. Just for you to know the scope of the reference architecture initiative

has broadened and as such the name has been changed from OFRA to ORA (Oracle Reference Architecture). A landing pad has been prepared on OTN, waiting for content to become available.

www.oracle.com/goto/itstrategies

## Dynamically Adding a where-clause to ADF BC

Use cases may require to dynamically adding where clauses to a View Object instance, for which the ADF BC API provides the following options for View Objects:

**addWhereClause** - public void addWhereClause(java.lang.String whereClauseToAppend)

*Appends an additional WHERE clause to the query. The new WHERE clause is appended to the current WHERE clause.*

*This method does not interpose "AND" or any other conjunctive between the old and new clauses: the application must provide it. The modified WHERE clause does not take effect until executeQuery() is called.*

**setWhereClause -** public void setWhereClause(java.lang.String whereClause)

*Sets the query's WHERE clause. The new WHERE clause does not take effect until executeQuery() is called. Note that calling setWhereClause() does not clear the previous settings of WHERE clause parameters.*

*To reset WHERE clause parameters, call setWhereClauseParams explicitly with a null value. For example: vo.setWhereClauseParams(null);*

In addition to setting the WHERE clause, you could also use ViewCriteria, to dynamically define and apply query filter conditions.

**applyViewCriteria** - public void applyViewCriteria(ViewCriteria criteria)

*Applies the view criteria to this view object. Invoking this method will erase all the previously applied view criteria on the view object.*

**applyViewCriteria -** public void applyViewCriteria(ViewCriteria criteria, boolean bAppend)

*Applies the view criteria to this view object. If bAppend is true view criteria is appended to the list of already applied view criteria. If bAppend is false the applied view criteria list is cleared prior to applying the passed view criteria.*

View Criteria can be pre-defined for a View Object and dynamically added by a named reference. View Criteria can be defined declaratively in the View Object editor (double click on the View Object definition).

To obtain a handle to a defined ViewCriteria, call getViewCriteriaManager()on the View Object instance. The `ViewCriteriaManager` then allows you to obtain the view criteria to apply

```
viewCriteriaManager.getViewCriteria(java.lang.String name)
```

As a best practice, no matter which approach you choose, make sure bind variables are used to define the dynamic part of the where clause condition instead of re-constructing and re-applying the whole where clause for each new condition. This reason for this recommendation is performance as the database does not need to parse the query string again for each where clause condition.

## How-to add ADF binding based auto suggest behavior

The Oracle ADF Faces component set in Oracle JDeveloper 11g provides an af:autoSuggestBehavior tag for developers to declaratively add suggest behaviors to their ADF Faces applications. An example of the auto suggests feature using an ADF bound data sources is released as sample #62 on ADF Code Corner:

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

## How-to access a hidden attribute from a table row

ADF bound tables may not display all of the View Object attributes as a table column. However, the variable "row" that is used by default to stamp the table rows contains a reference to it if the attribute is part of the ADF tree binding populating the table. Using #{row.bindings.hiddenAttrName} developers can access the hidden value, for example from an af:setPropertyListener tag.

To build the "hidden" attribute binding, it's the easiest to create the attribute as a column when creating the table and then to remove the page content for this column.

## How-to add JavaScript code to an ADF Faces pages

To use JavaScript on an ADF Faces view, it is either added to the page source or referenced in an external library file. To ensure that the JavaScript sources are efficiently loaded to the client, ADF Faces provides the *af:resource* component tag, which must be added as a direct descendent of the *af:document* tag.

```
<af:document>
  <af:resource type="javascript" source="/customJsCode.js"/>
  …
</af:document>
```

```
<af:document>
  <af:resource type="javascript">
    function customJsFunction(){ … }
  </af:resource>
  …
</af:document>
```

**Note**: To add JavaScript to ADF Faces page fragments exposed by an ADF region, you use the *trh:script* tag of the Apache MyFaces Trinidad component set. Because page fragments don't have an *af:document* tag, they cannot use the *af:resource* tag.

## How-to launch a popup programmatically from Java

Starting Oracle JDeveloper 11.1.1.3, instance of af:popup can be opened using a Java method on the `RichPopup` object. The `RichPopup` class is the bean implementation of the af:popup ADF Faces tag. To launch a popup from a managed bean, you first need to get a hold on to a rich popup instance, which you can do by searching it on the UIViewRoot or by referencing a JSF component binding – which you create using the "binding" property of the af:popup component tag. To open the popup you then call

```
RichPopup.PopupHints ph = new RichPopup.PopupHints();
popup.show(ph);
```

The popup hints can be used to set the aligned, the component the popup is opened next to, and the alignment type. To close a popup, just call

```
popup.show(hide);
```

## Best option for accessing an attribute value of an iterator binding

A benefit of working with ADF is that it abstracts developers away from the underlying business service implementation. For example, data that is queried from the business service ends up in an ADF iterator that is accessed from a control binding like JUCtrlAttrsBinding or JUCtrlHierBinding.

A question on OTN was about the best option to access an attribute value of a current row from Java in a managed bean. The options that are available to access an attribute are

1. Create an attribute binding for the row attribute to read and use EL in Java to read its value
2. Create an attribute binding for the row attribute to read and access it directly from Java
3. Access the ierator binding, an instance of DCIteratorBinding, and read the value from the current row

The recommendation – as you would expect – depends on what information else you need to get access to. For example, if access is only for a single attribute value and if – optionally – this also needs to be EL accessible (for example to reference from a task flow input parameter or method binding) then creating an attribute binding is the better solution to use. Still however, I would recommend avoiding the use of EL in managed bean Java code because of the unnecessary overhead and assumption it makes about the binding layer structure. So code like shown below will do

```
BindingContainer vBindingContainer =
           BindingContext.getCurrent().getCurrentBindingsEntry();
ControlBinding vControlBinding =
           vBindingContainer.getControlBinding("attributeBindingName");
AttributeBinding vAttributeBinding = (AttributeBinding)vControlBinding;
Object value =  vAttributeBinding.getInputValue();
```

If however the use case is to access multiple attribute values, then reading the iterator directly seems to be a better solution as it only requires a single binding layer access.

```
BindingContainer vBindingContainer =
           BindingContext.getCurrent().getCurrentBindingsEntry();
DCBindingContainer vDCBindingContainer =
           (DCBindingContainer)vBindingContainer;
DCIteratorBinding vDCIteratorBinding =
           vDCBindingContainer.findIteratorBinding("theIterator");
Row vRow = vDCIteratorBinding.getCurrentRow();
Object attr1Value1 = vRow.getAttribute("attrName1");
Object attr1Value2 = vRow.getAttribute("attrName2");
...
```

Option 2 and 3 are the solutions to recommend to developers who want to access the ADF binding layer from Java.

**Note**: if you use ChangeEventPolicy with a value of "ppr" to refresh ADF Faces components in response to data changes on the binding layer, please read on.

## Strange component refresh behavior using ADF auto-ppr

Related to the above question and answer, the following observation appears to be a strange behavior when using auto-ppr (changeEventPolicy set to "ppr" for the iterator in the ADF binding layer).

- txt1 bound to managed bean method1 that accesses an ADF attribute binding using EL in Java to return the value of an attribute
- txt2 bound to managed bean method2 that accesses an ADF attribute binding from Java to return the value of an attribute
- txt3 bound to managed bean method3 that accesses an iterator in ADF to read from the current row's attribute

The iterator has its changeEventPolicy set to PPR (to enable auto-ppr). However, navigating the data with command buttons bound to ADF binding operations ("partialSubmit" set to true) refresh only txt1 and txt2 - not txt3.

The reason for this behavior is to explain with the way the auto-ppr functionality works. When the change event policy is set to "ppr" on the ADF binding, then all changes in the binding layer – like row currency changes – are reported to the client component by adding the component reference to the list of partial targets to refresh in response to a request, which is pressing the navigation button.

Iterators are bound to UI components through control bindings, like attribute, list or tree bindings. Any component that directly or indirectly – through a managed bean method referenced from the component value property – accesses a control binding is added to the list of partial targets to refresh. Components that access a managed bean method that accesses the iterator binding directly are not added to this list and therefore not refreshed.

A component that binds to the ADF iterator through a managed bean method bypasses the ADF Faces binding layer. This means that registration with the active model (auto-ppr) does not happen. If you need a component to refresh though it accesses the iterator binding directly through a managed bean method, then set up its PartialTriggers property to point to the component that starts the change (the navigation button in the example).

As a rule of thumb: auto-ppr refresh with ADF works for components that directly or indirectly reference a control binding in the "Bindings" section of the PageDef file.

**Note:** A thank you goes to John Stegeman and Jan Vervecken for their contributions to OTN thread http://forums.oracle.com/forums/thread.jspa?messageID=7423464

## How-to keep selected row after rollback

A frequent requirement posted on OTN is to keep the current selected row state after commit or rollback. The good news is that this functionality is easy to implement – and almost declarative. To do so, follow the steps below

- Drag the Rollback operation from the Data Controls panel and drop it as a button
- The "Rollback" button has its "disabled" property pointing to the action binding in the PageDef file. Either remove this reference or ensure the rollback button is refreshed when the submit button is pressed.

- Double click onto the "Rollback" button, which opens a dialog for you to create a managed bean and a method in this bean. Optional the action method can be created in an existing bean.
- The action method in the bean gets created as shown below, where "onRollback" is the name for the action that we chose for the new method

```
public String onRollback() {
 BindingContainer bindings = getBindings();
 OperationBinding operationBinding =
                 bindings.getOperationBinding("Rollback");
  Object result = operationBinding.execute();
  if (!operationBinding.getErrors().isEmpty()) {
        return null;
   }
     return null;
}
```

The behavior of the method is the same as before. The difference is that instead of EL, which is used after the drag and drop of the Rollback operation, Java is used to perform the rollback action.

- Change the generated method above as shown below to read the row key for a master and detail relation

```
public String onRollback() {
    BindingContainer bindings = getBindings();
    //get rowKey for parent and child
    DCIteratorBinding parentIter =
          (DCIteratorBinding)bindings.get("DepartmentsView1Iterator");
   DCIteratorBinding childIter =
          (DCIteratorBinding) bindings.get("EmployeesView3Iterator");
   Key parentKey = parentIter.getCurrentRow().getKey();
   Key childKey = childIter.getCurrentRow().getKey();
   // perform rollback operation
   OperationBinding operationBinding =
                       bindings.getOperationBinding("Rollback");
   Object result = operationBinding.execute();
   if (!operationBinding.getErrors().isEmpty()) {
     return null;
   }
   //user row key back to preserve row selection
   parentIter.setCurrentRowWithKey(parentKey.toStringFormat(true));
   childIter.setCurrentRowWithKey(childKey.toStringFormat(true));
   return null;
}
```

**Note**: In JDeveloper 11.1.1.3 chances are that the Rollback action binding is deleted when you create the managed bean reference. In this case the rollback action needs to be added manually in

the PageDef file. This does not reproduce in the upcoming JDeveloper 11.1.1.4 release

## How-to programmatically queue a selection event

A common misconception is about when component events like action, valueChange or selection are raised by the framework. The brief summary is that component events are raised in response to a user action. A user pressing a button causes an action event to be fired. A user selecting a row in a table causes a selection event to be fired. However, a user deleting a table row by pressing a delete button, which then implicitly changes the selected row, does **not** cause a select event to be fired. Though the primary action is initiated by the user – pressing the delete button – the resulting change of the underlying row currency is not caused by the user but implicit.

So what if you have mandatory program code in a select listener that – no matter what should be executed. The two options I see are

- Option 1 – Delete the table row from Java and call a Java method that executed the same logic that you have invoked by the select event.
- Option 2 – Create a select event after deleting the table row and queue it for the table component. This way the row currency change appears as if initiated by the user

Below is a sample code, assuming the row delete operation is performed using a button created by dragging the "Delete" operation of a View Object from the Data Control panel. To create the managed bean method to do the same in Java, just double click on the button and then follow the dialog that opens. The action method code below highlights the generated code lines in bold.

```
public String onDelete() {
  //we deleted the column, so the old value doesn't exist anymore
  RowKeySet oldSet = new RowKeySetImpl();
  //create a new set
  RowKeySet newSet = new RowKeySetImpl();

  //delete row the ADF way
  BindingContainer bindings = getBindings();
  OperationBinding operationBinding =
                      bindings.getOperationBinding("Delete");
  Object result = operationBinding.execute();
  if (!operationBinding.getErrors().isEmpty()) {
    return null;
  }
  //get a reference to the table component either through a JSF
  //component binding or by looking up the component in the UI
  //view root. From the table you get a generic access to the
  //DCIteratorBinding
  CollectionModel model = (CollectionModel) table1.getValue();
  JUCtrlHierBinding tableBinding =
      (JUCtrlHierBinding) model.getWrappedData();
  DCIteratorBinding dciter = tableBinding.getDCIteratorBinding();
  //get the current row key
```

```
  Key k = dciter.getCurrentRow().getKey();
  //keys in a table are List types
  ArrayList newSelectedRowKey = new ArrayList();
  newSelectedRowKey.add(k);
  newSet.add(newSelectedRowKey);
  //create and queue the selection event
  SelectionEvent se = new SelectionEvent(oldSet,newSet,table1);
  se.queue();
  return null;
}
```

Note: This of course also works for other events, like ValueChange etc.

## How to configure a project to use Trinidad components

If you want to create a view layer project for Trinidad, you start with building a Generic project in Oracle JDeveloper. The steps are as outlined below:

- Create a new "Generic Project" and choose JSF as the Technology Scope. Don't use any ADF Faces or PageFlow scope.
- Create a JSPX page – which is the easiest option to open the component palette, which gives you a quick access to configuring the Trinidad tag libraries
- Select the Component Palette and right mouse click into it
- Choose "Edit Tag Libraries" and select the Trinidad components. Move them to the "Selected Libraries" section and Ok the dialog.
- The first time you drag a Trinidad component to a page, the web.xml file is updated with the required filters

**Note**: Shay Shmeltzer recorded a video about this:
http://blogs.oracle.com/shay/2009/02/using_trinidad_in_jdeveloper_1.html

## How-to enable the commit button on ADF BC based input form

The command button that you create by dragging the ADF BC commit operation onto a form is disabled by default. To enable the button, you need to either submit data changes, or remove the button "disabled" property value, an Expression Language string that points to the binding

The commit operation is enabled within the model if data changes are detected, which happens in response to a form submit. So if you use autosubmit="true" on an input component or users press a submit button, then the commit button can be enabled by setting its "PartialTriggers" attribute to reference the Id of the submit button or the id of the UI component with the autosubmit="true" set.

## PartialTriggers on command item confusion

Command components – like buttons – have a PartialTriggers attribute that allow developers to trigger its refresh in response to an action performed on or by another component. The following use case however may cause confusion:

1. Drag a CreateInsert operation as a command button on a page. The CeateInsert operation is listed under "Operations" for a View Object in the DataControls panel.
2. Darg the View Object and drop it as an editable table. This creates the table but also adds a PartialTrigger entry to the command button.

What's confusing with this is that you would expect the table to have a PartialTriggers property entry pointing to the command button. Instead the PartialTriggers property of the button points to the table. Why is this?

1. The command button is set to partialSubmit="false" by default. This means that pressing the command button performs a full page refresh, which also leads to the table getting refreshed. Only if the partialSubmit property of the command item is set to "true" you need to set the PartialTriggers property on the table to refresh when the button is pressed
2. The command button has the "disabled" property pointing to the ADF binding layer to determine whether or not it is enabled: #{!bindings.CreateInsert.enabled}. Chances are that an operation on the table changes the button state, which is why the button has its PartialTriggers property pointing to the table. Keep in mind that table actions are performed by partial submits issued by the component itself.

So the confusion actually comes from the IDE being smart, doing the right thing without telling. In general, of course, PartialTriggers only need to be set if a component requires refresh in response to a partial submit performed on another component.

## Why isn't my custom skin showing for the login pages?

Developers who use skinning in Oracle ADF Faces or Apache MyFaces Trinidad applications, may experience a problem with skinning on a login page – assuming the login page is built with JSF as well. Usually the reason for this is that the security constraint in the web.xml file is defined for the application root, the public_html folder.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>allPages</web-resource-name>
      <url-pattern>/</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>valid-users</role-name>
  </auth-constraint>
</security-constraint>
```

The above constraint in web.xml protects all resources – not only pages – of an application, making it accessible for authenticated users only. If such an application uses form based authentication, in which the form is contained in a JSF page, then this page comes without any style applied to it – and most likely without images too – because of the root level protection.

To avoid this problem, either use ADF Security based authorization, store page content in a sub folder under the public_html directory – to the use /pages/* as the authorization pattern – or use an explicit mention of the file extension to protect (e.g. *.jspx). However note that if the page is served as a controller reference, then it is accessed by its viewActivity name, which does not use any file extension at all. Applying web based security to JSF applications, which implies ADF Faces and MyFaces Trinidad, is

not easy to do and proves to be error prone. If you are an ADF user then using ADF Security is the best option to go with. A recording about ADF Security and how to set it up is available on the ADF Insider website: http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adfinsider-093342.html

### How-to hide the slider bar of the af:carousel component

If you want to only allow programmatic navigation within a af:carousel component, your requirement is to hide the slider bar that allows users to change the displayed carousel detail item by dragging it to the left or right. As there is no API exposed on the af:carousel component to hide the slider, skinning comes to the rescue. En example skin definition is shown below:

```
af|carousel::spin-bar{
    visibility: hidden;
}
af|carousel::spin-h-previous-icon-style{
    visibility: hidden;
}
af|carousel::spin-h-next-icon-style{
  visibility: hidden;
}
af|carousel::spin-info{
    visibility: hidden;
}
```

### How-to filter tables case insensitive

By default, values entered in a table filter cell filter the result set case sensitive. This behavior can be changed declaratively setting the "FilterFeatures" property on the af:column component representing the table column. The FilterFeatures property flags how this column should be filtered in a query-by-example (QBE) case. Currently the only values supported are "caseSensitive" and "caseInsensitive".

If not specified, the case sensitivity is model-dependent. The features are hints for a collection model, which supports filtering, to perform special filtering operations (such as case-sensitive/insensitive filtering).

See. http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12419/tagdoc/af_column.html

**Note:** Early versions of Oracle JDeveloper 11g did not support this column feature; it is supported in recent releases though.

### ADF Faces page doesn't open in inline frame

For security reasons, ADF Faces pages don't render in inline frames to avoid clickjacking attacks in which a sites wraps another page in an inline frame to attack the users – e.g. trying to get the password. If you are in control of the ADF Faces application and the site that attempts to display this page in an inline frame, then you can disable the protection by setting a context parameter to switch the framebursting feature off, or to weaken it.

To allow ADF Faces pages to run in inline frames, you e.g. set the FRAME_BURSTING web.xml context parameter to "always"

```
oracle.adf.view.rich.security.FRAME_BUSTING
```

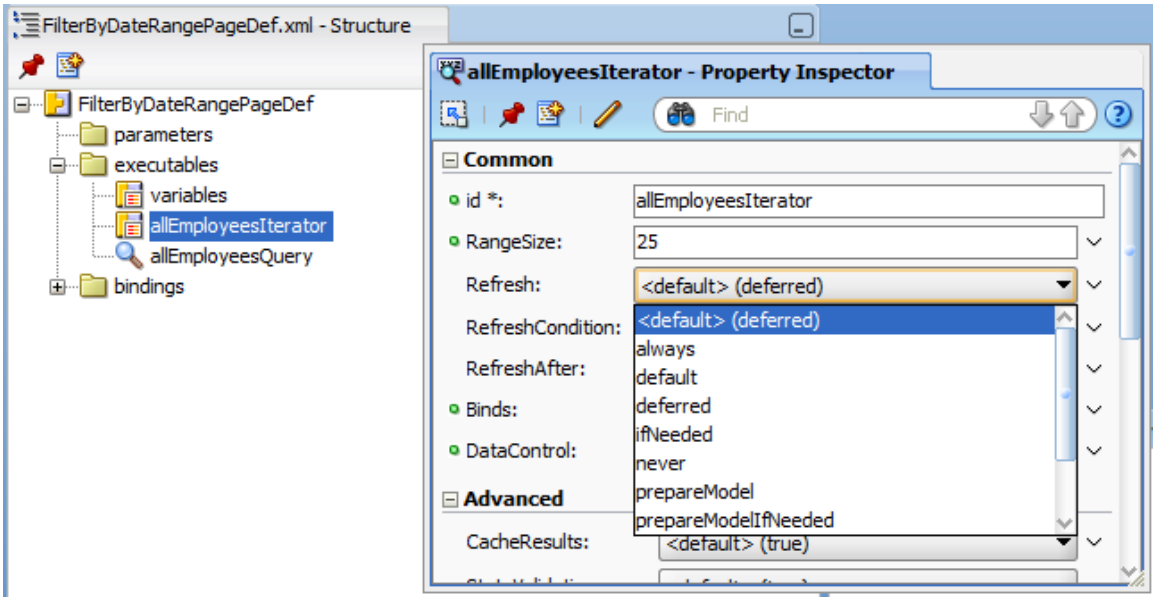
Valid values are

- **always**: The page will show an error and redirect whenever it attempts to run in a frame.
- **differentDomain**: The page will show an error and redirect only when it attempts to run in a frame on a page that originates in a different domain (the default).
- **never**: The page can run in any frame on any originating domain. (the default)

See: http://download.oracle.com/docs/cd/E15523_01/web.1111/b31973/ap_config.htm#BABDHGEJ

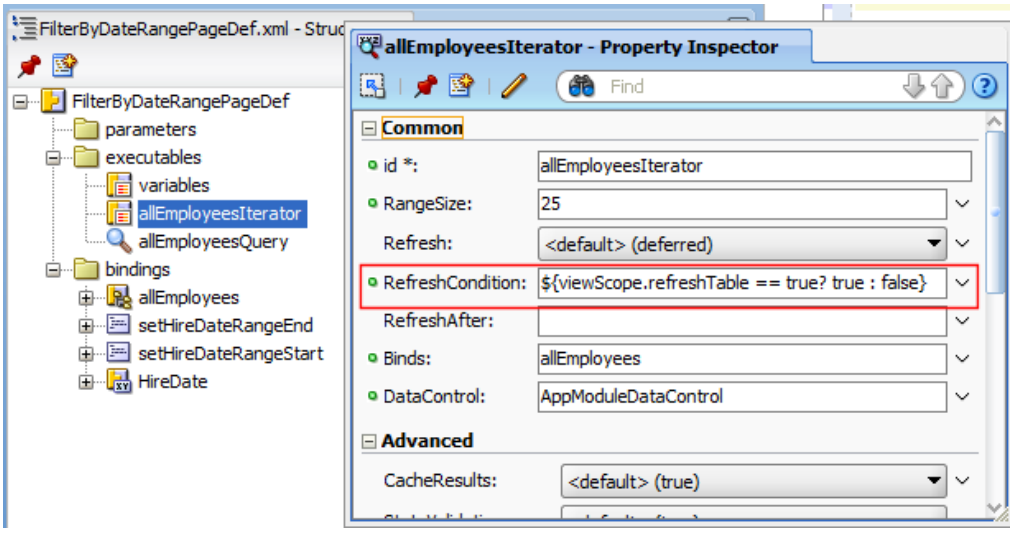
## How to prevent table population at page rendering

The table execution in Oracle ADF is determined by the refresh option defined on the iterator binding used by the table (note that the table is bound to a tree binding, which references the iterator). By default, the refresh option is set to "deferred", which means that the data is refreshed during the JavaServer Faces render response phase. In addition, a RefreshCondition property exists that can be used with EL to check a specific condition.




If you want avoid table execution on the initial page rendering, then you can check for a memory attribute, for example ${viewScope.refreshTable == true? false : true}. This returns false if the attribute is not set or set to false, in which case the iterator is not refreshed and the table is empty. Using a command item action, or an af:setPropertyListener, the memory attribute can be set to true. The next time the table refreshes, the data then is shown.

```
<af:commandButton text="Submit" id="cb1" partialSubmit="true">
  <af:setPropertyListener from="#{true}"
                          to="#{viewScope.refreshTable}"
                          type="action"/>
</af:commandButton>
```




Note that using the viewScope memory scope, which is an ADF Controller specific scope, ensures the memory attribute is cleaned up when users navigate off the page. It's all about scope: So chances are that storing the attribute in a viewScope does not 100% meet your needs, in which case you may want to reach out to pageFlowScope or sessionScope.

Another use case is in which you don't want the table to refresh at all until you said so. In this case, set the iterator refresh option to "never". This then however requires you to refresh the table from Java in a managed bean.

```
String iterator = "myIteratorName";
BindingContext bctx = BindingContext.getCurrent();
BindingContainer bindings = bctx.getCurrentBindingsEntry();
DCIteratorBinding dciter = (DCIteratorBinding)bindings.get(iterator);
dciter.executeQuery();
dciter.refresh(DCIteratorBinding.RANGESIZE_UNLIMITED);
```

**Note**: If JPA is used, you need to re-execute the finder method exposed in the binding

## Configuring ADF Security for custom Servlet

The ADF Security declarative authorization framework protects ADF bound pages and ADFc bounded Task Flow using JAAS. In addition, it provides security expression language vocabulary (EL) that allows developers to declaratively check for user granted permissions, authentication state and role memberships from the ADF Faces UI layer, ADFc metadata and the ADF binding. Of course, all you can do using EL us also available from Java using the SecurityContext that you can access from the ADFContext class.

A question on OTN was how to protect a custom servlet with ADF Security so that access is granted to users only that are in a specific application role with the permission granted to execute the servlet. This requirement is not new and a similar question was asked two years ago for a servlet to access the binding layer using Oracle JDeveloper 10.1.3. Not sure we are walking on supported grounds here, but from the design and architecture, custom servlets can be configured to integrate with the ADF binding layer and also to leverage ADF Security.

For a servlet to integrate with the ADF binding layer – and to leverage ADF Security framework protection – the servlet needs to have a PageDef file defined and registered in the web project's DataBindings.cpx metadata file.

To configure a custom servlet with ADF and to protect it with ADF Security, do as follows

1. Create new PageDef file as a copy of an existing PageDef file. The PageDef file does not need to have bindings defined and could be empty for this reason. All that it needs is the valid PageDef XML syntax and structure so the ADF binding framework knows how to parse it
2. Assuming ADF Security is enabled for the web project, use the ADF Security policy editor to protect the Servlet PageDef file with an application role (which then is granted permission for accessing the PageDef file (and thus the servlet)
3. Associate the custom servlet request path with the pageDef ID in DataBindings.cpx file. Its best to first have a look how other pages and PageDef files are configured before messing with this registry file.
4. Map the ADFBindingFilter with the servlet in the web.xml file

```
<pageMap>
   <page path="/servlet/MyServlet" usageId="myPageDef"/>
    ...
</pageMap>
<pageDefinitionUsages>
   <page id="myPageDef" path="view.pageDefs.MyPageDef"/>
     ...
</pageDefinitionUsages>
```

5. To ensure ADF Security works well with this configuration, configure the custom servlet to the JpsFilter. Note that the JpsFilter mapping **must come before** the adfBinding filter mapping shown next in step 6.

```
<filter-mapping>
  <filter-name>JpsFilter</filter-name>
  <servlet-name>MyServlet</servlet-name>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

6. Configure the ADF binding filter with the servlezt

```
<filter-mapping>
  <filter-name>adfBindings</filter-name>
  <servlet-name>MyServlet</servlet-name>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

The JPS filter is configured in web.xml is required to setup the OPSS (Oracle Platform Security Service) policy provider for an application.

**Note:** Credits go to Dimitar Dimitrov, who tested and confirmed this setting in the following OTN thread: http://forums.oracle.com/forums/thread.jspa?threadID=1772911&start=0&tstart=0

## JpsFilter changes in Oracle JDeveloper 11.1.1.4

Only because we mentioned the JpsFilter in the previous section: In the upcoming Oracle JDeveloper 11.1.1.4 release, the configuration of the JpsFilter and some of its default settings will change. First of all, the JpsFilter no longer is required to be in the web.xml file and instead becomes a system filter in WLS.

You only set the JpsFilter manually in the application web.xml file if you want to change the default configuration. The default configuration in Oracle JDeveloper 11.1.1.4 is changed for the following settings:

| JpsFilter Init Parameter | New default setting in 11.1.1.4 | Description |
|---|---|---|
| enable.anonymous | true | Set it to false if you want to disable the anonymous subject with anonymous user and anonymous-role role |
| remove.anonymous.role | false | Set it to true if you want the anonymous-role removed from the authenticated subject. |

Other properties – that haven't changed their default value – are:

**application.name**

The deployed application name, an application ID that must be the same in jazn-data.xml and the JpsFilter if configured. In general this property is not needed to be set. If this property is not set, the deployed application name will be used instead. This property only needs to be set if multiple applications should share the same application ID defined in the policy store.

**add.application.roles**

Adds application roles to the authenticated Subject. By default both, application roles and enterprise roles are added to the authenticated subject from where they can be checked e.g. using the ADFSecurity security expression. The default setting is "true".

**add.authenticated.role**

Adds the "authenticated" role to the user Subject. The default setting is "true" and the authenticated role is added to the Subject for all users that successfully authenticated to OPSS.

**oracle.security.jps.jaas.mode**

By default, this is set to "doAsPrivileged" . Other options are "doAs", which however is a rare use case for an ADF application.

## Why? ADF BC bound table references View Object definition

Configuring trees or tables from a View Object collection is one option provided in Oracle JDeveloper when dragging a View Object instance from the Data Contols panel to a JSF page or page fragment. No matter what you choose, table or tree, the ADF binding definition added to the PageDef file is the "tree" binding, which at runtime is presented by the JUCtrlHierBinding object.

For ADF programming and debugging consistency, the table, tree and tree table components bind to same ADF binding type, which may be confusion for developers who worked with ADF in JDeveloper 10.1.3 in where a separate table binding existed. This however is not as confusing as seeing the View Object definition (the interface) referenced from the tree binding node definition, where you would only expect the View Object instance to be referenced.

```
<bindings>
    <tree IterBinding="EmployeesView1Iterator" id="EmployeesView1">
      <nodeDefinition DefName="otn.oracle.example.model.EmployeesView">
        <AttrNames>
          <Item Value="EmployeeId"/>
          <Item Value="FirstName"/>
```

In the example above,the View Object instance is defined as "EmployeesView1", whereas the interface definition is defined as the DefName attribute:

The presence of the DefName="adf.sample.model.EmployeesView" parameter supports polymorphic View Object. A tree binding can have node definitions for different types and subtypes for which the ADF model binding can automatically make a different set of attributes available for subtypes. If you have a View Object with just a single type of rows, the DefName attribute is effectively optional.

The DefName property is also defined for other DataControls, describing the type of object being rendered by a tree node.

In a tree – form samples posted to ADF Code Corner (see sample #038), the selected node type is determined using EL referencing the hierTypeBinding.

The example below shows the af:switcher component used in the sample to display an input-form dependent on the selected tree node.

```
<af:switcher facetName="#{node.hierTypeBinding.structureDefName}"
        defaultFacet="adf.pojo.entities.Locations">
      <f:facet name="adf.pojo.entities.Locations"> …  </f:facet>
      <f:facet name="adf.pojo.entities.Employees"> ... </f:facet>
</af:switcher>
```

In summary, the tree binding defName attribute is optional up to when you actively start using it or – in the case of ADF BC – use polymorphic View Objects.

## Multi row selection table returns single row key

A common problem with tables in ADF is that users change the row selection from "single" to "multiple" to – for example – delete multiple rows when user press a delete button. Surprisingly it may happen that only a single row – always the last row a user selects – gets removed. The code to delete multiple rows looks okay as shown below:

```
RichTable table = <get your table handle here>;
RowKeySet rowKeys;
rowKeys = table.getSelectedRowKeys();
```

```
// iterate over selected rows
Iterator selectedRows;
selectedRows = rowKeys.iterator();
while (selectedRows.hasNext()){
  //key is an instance of ArrayList
  Object key = selectedRows.next();
  //make row current
  table.setRowKey(key);
  Object o = table.getRowData();
  JUCtrlHierNodeBinding rowData = (JUCtrlHierNodeBinding) o;
  //get underlying row from ADF binding
  Row row = rowData.getRow();
  // remove the row
  row.remove();
}
```

As said, the code to delete the rows looks correct – and it not only looks so. The problem is cause by the rowKeySet only having a single entry, which happens when the selection listener, which by default is added when creating a single row select table, still exists.

```
<af:table value="#{bindings.allEmployees.collectionModel}"
          var="row" rows="#{bindings.allEmployees.rangeSize}"
          ...
         selectionListener=
              "#{bindings.allEmployees.collectionModel.makeCurrent}"
         rowSelection="multiple" ... >
```

Removing the line highlighted in bold fixes the problem so that all selected rows are deleted. Note that the ADF binding layer only supports a single row to be current, which is why there is no equivalent listener setting for the multi row select case.

## How-to programmatically invoke a clientListener

The af:clientListener tag allows developers to respond to component events on the client using JavaScript.

```
<af:commandLink id="cl1" text="click here" partialSubmit="true">
  <af:clientListener method="onLinkPressed" type="action"/>
</af:commandLink>
```

When the application user clicks or hits enter on this command link, a JavaScript function is called before the event is passed to the server to execute associated actions and action listeners. The JavaScript function signature is shown below

```
Function onLinkPressed(evt){ … }
```

The "evt" argument is the ADF Faces event object that is passed to the function. It allows developers to cancel the event – thus to suppress server side execution of logic associated with the action or action listener property – know about and access the event origin (the link component) and more. In an OTN post, the question was how to invoke the clientListener event on a component from JavaScript.

Because the clientListener only listens for a component event, the solution to this is to raise the event, which can be done using JavaScript as shown below:

```
var link = AdfPage.PAGE.findComponentByAbsoluteId( 'cl1');
AdfActionEvent.queue(link, link.getPartialSubmit());
```

This then in turn invokes the clientListener and – if the event is not cancelled – the server side logic associated with the command link component.

## Closing a popup from HTML in an inline frame

Though breaking encapsulation of content displayed in an inline frame, the ADF Faces client architecture provides JavaScript APIs to access ADF Faces components on the containing page. For example, to close a popup dialog from an embedded HTML page, you use the AdfPage.PAGE static method as shown below

```
var popup = AdfPage.PAGE.findComponent('<yourPopupClientId>');
if (popup.isPopupVisible()) {
  popup.hide();
}
```

The AdfPage.PAGE access to an ADF Faces view also becomes handy for integration between 3rd party UI rendering technologies like Flash or JQuery and ADF Faces.

## How-to manually deploy and test applications in JDeveloper?

Yes. The integrated WebLogic server in Oracle JDeveloper behaves like a stand-alone server. To deploy applications to the integrated WLS and then to test it by manually adding the request URL to a browser window, follow the steps outlined below

1. Choose Run | Start Server Instance from the JDeveloper menu
2. Choose View | Application Server Navigator
3. It should show an entry IntegratedWebLogicServer. If not, create one
4. Create a WAR file deployment profile for the web project (which may already exist), then an EAR file (Application | Application Properties | Deploy)
5. Select project and deploy to integrated WLS

**Note** The port number is 7101 (not 7001), which is an Oracle JDeveloper specific configuration. The WebLogic console, for example, is accessible from http://localhost:7101/console

## Browser support for Oracle JDeveloper 11.1.1.3

Especially when experiencing problems with a specific browser, it is worth checking the support of the current Oracle JDeveloper version for this particular browser. For example, Oracle JDeveloper supports the following browser versions / OS combinations

| | |
|---|---|
| Firefox 2.0 **, 3.0, 3.5+ [2] | Windows, Linux, Mac OS, Solaris |
| Internet Explorer 7, 8 [2] | Windows |
| Safari 3.2, 4.0 [2] Windows, Mac OS | |
| Safari 3.0 (mobile) [2] | iPhone OS |
| Chrome 1.0+ [1] Windows | |

** The minimum required Firefox version is 2.0.0.20+

1. Support or certification added in 11g Release 1 (11.1.1.1.0). Does not apply to 11g versions prior to 11.1.1.1.0.
2. Support or certification added in 11g Release 1 Patch Set 1 (11.1.1.2.0). Does not apply to 11g Release 1 versions prior to 11.1.1.2.0.
3. Support or certification added in 11g Release 1 Patch Set 2 (11.1.1.3.0). Does not apply to 11g Release 1 versions prior to 11.1.1.3.0.

For more information, visit:
http://www.oracle.com/technetwork/developer-tools/jdev/index-091111.html

## How-to access the username from ADF BC?

To access the username of an authenticated web session in ADF Business Components, use the security context as shown below

```
ADFContext adfctx = ADFContext.getCurrent();
String user = adfctx.getSecurityContext().getUserPrincipal().getName();
```

The security context is also accessible from Groovy in ADF Business Components, for example to define a bind variable used in a View Criteria that queries View Object data in the context of the authenticated user

```
adf.context.securityContext.userName
```

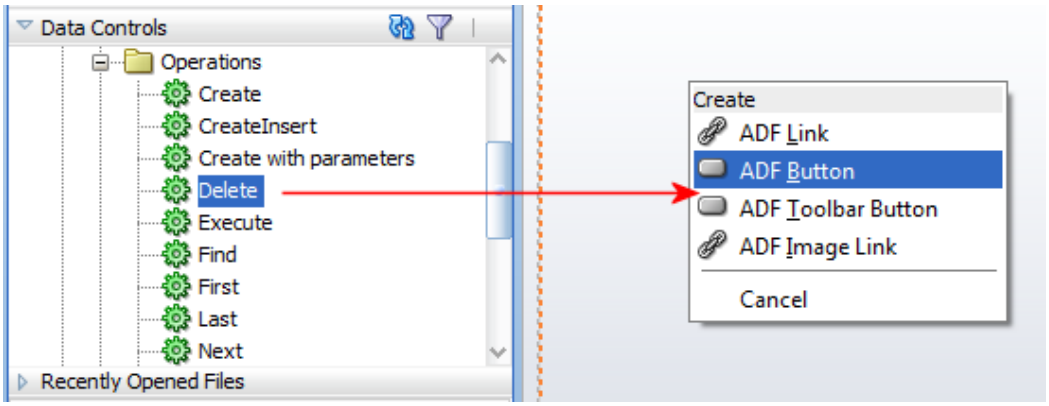
The SecurityContext is not available in Oracle JDeveloper releases before 11g. In older releases, the authenticated username is accessible from a call to getUserPrincipalName() on the Application Module Impl class. This "old" API is still supported but discouraged in favor of the SecurityContext accessed from the ADF context.

## How-to delete a row and commit with a single button press

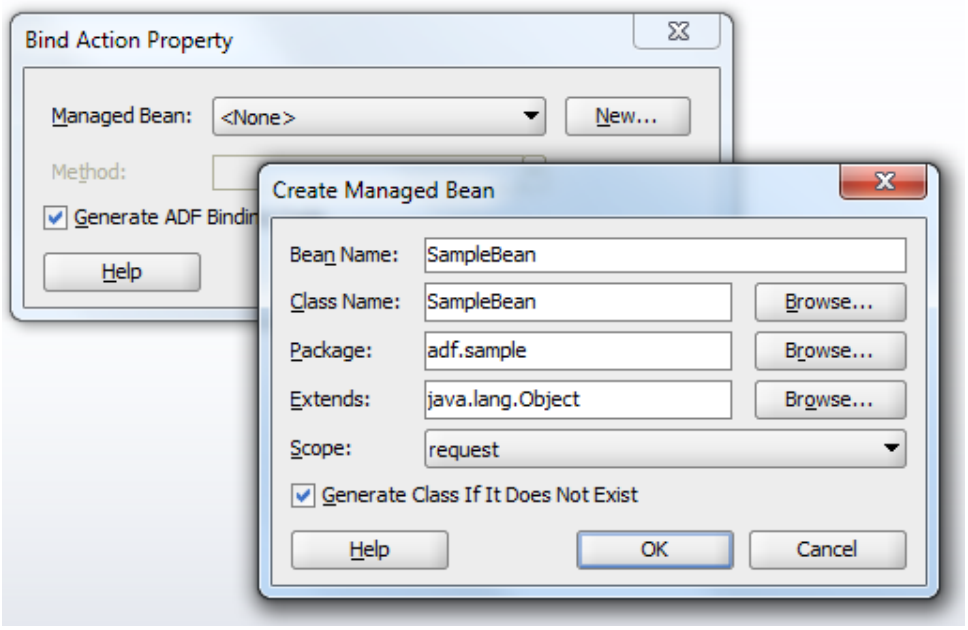
The default behavior in ADF Business Components is that CRUD operations are saved in the user session's entity cache until it is committed to the database. A frequent question is how to change this behavior so that pressing a single button a row – for example the delete button – first delete the current row and then immediately commits the transaction.

The solution to this question is to "line up" the two actions using a managed bean. This "line up" can be done half declarative, half programmatically as outlined below.

- Drag the operation binding – for example DELETE – from the View Object's operations node in the Data Controls panel and drop it as a command button onto your page. When pressing the button at runtime, the ADF binding is called to remove the current row (#{bindings.Delete.execute}) from the collection




- Double click the command button in the visual editor to open the dialog for creating a managed bean method for the command action




- After creating the class, give a meaningful name to the action method that is getting created and OK the dialog
- Select the "Bindings" tab at the bottom of the visual editor of the page
- In the "Bindings" section, press the green plus icon to create a new "action"
- Select the Application Module entry (root entry) and choose the "Commit" operation. This creates an action binding "Commit" when you OK the dialog
- Open the managed bean you created before and Change the action method from

```
public String deleteAction() {
  BindingContainer bindings = getBindings();
  OperationBinding operationBinding =
    bindings.getOperationBinding("Delete");
```

```
    Object result = operationBinding.execute();
    if (!operationBinding.getErrors().isEmpty()) {
          return null;
    }
    return null;
  }
```

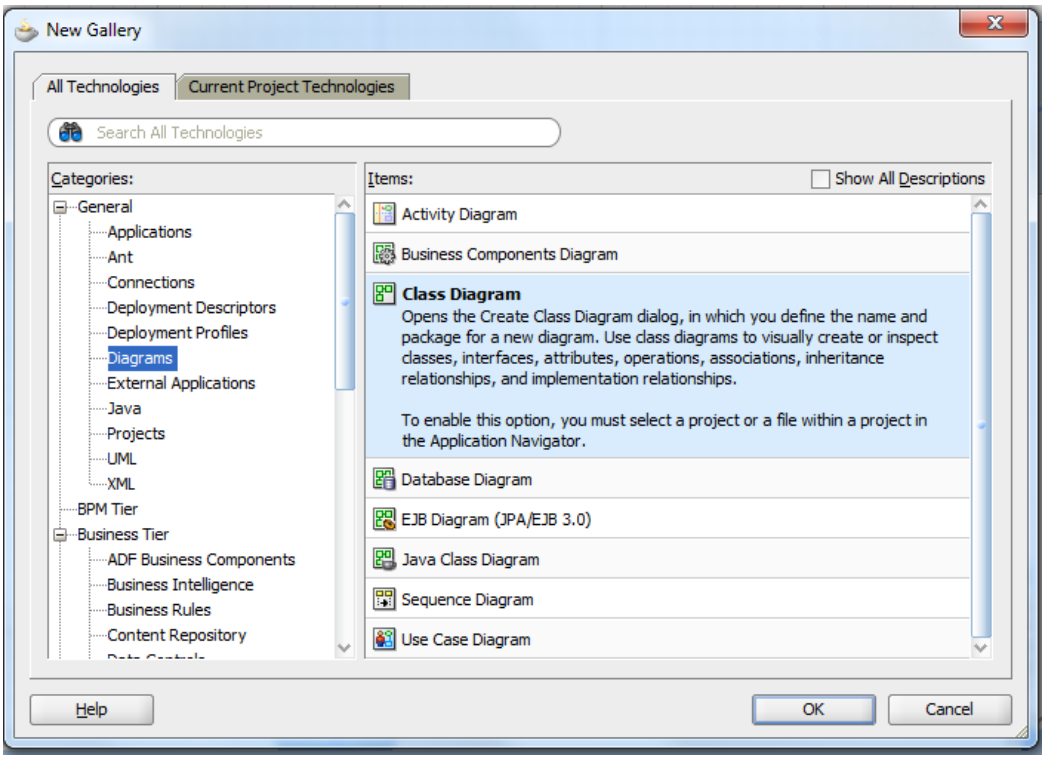
to (added lines highlighted in bold)

```
 public String deleteAction() {
   BindingContainer bindings = getBindings();
   OperationBinding deleteBinding =
      bindings.getOperationBinding("Delete");
   OperationBinding commitBinding =
      bindings.getOperationBinding("Commit");
    Object result = deleteBinding.execute();
    if (!deleteBinding.getErrors().isEmpty()) {
       //add error handling here
       return null;
    }
    Object commitResult = commitBinding.execute();
    if (!commitBinding.getErrors().isEmpty()) {
       //add error handling here
       return null;
    }
    return null;
  }
```

Next time the button is pressed, it deletes the current row and – if this could be done without errors – commits the transaction.

## Where to get ADF binding and ADF BC class diagrams from

Class diagrams provide a great inside in how software artifacts relate to each other. The more you get into developing with the ADF framework, the more you want to know about the framework classes, in which case a class diagram is a good start. The Oracle product documentation does not contain class diagrams, which is understandable as the whole framework diagram would have the size of a wallpaper that would hardly fit in a readable size on a printable page. However, there is a solution to this, which is to create the diagram you are interested in yourself. For this you use the Oracle JDeveloper class diagrammer, which you open by choosing New | General | Diagrams (with the All Technologies tab selected).

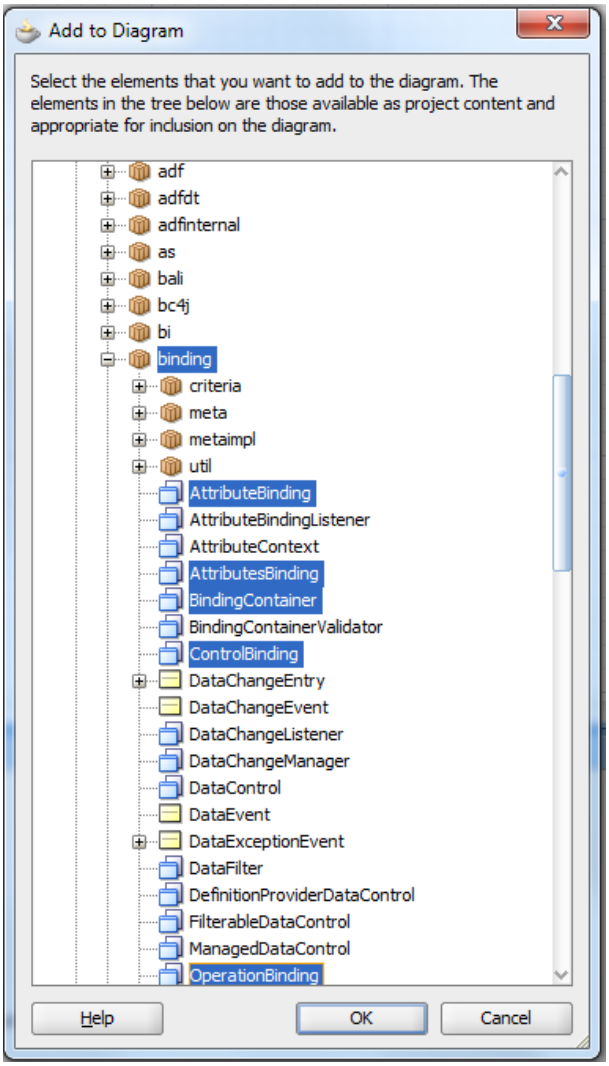



Once open, use the right mouse context menu option "Add to Diagram" to browse the class path (or source path) for the classes you are interested seeing in their relation to each other.

As shown in the image below, the class overview also makes sure you get the classes from the correct packages, thus avoiding accidental use of internally packaged classes. No source code is required for building your own class diagrams of the ADF framework classes. Just make sure the project you create the diagram in is configured for ADF so it has the libraries set.

The image below shows classes from the "binding" package, which contains the interfaces. Concrete implementation classes, like DCBindingContainer, DCIterator etc. are locatedin the adf | model | binding package.

To see the classes in a compact view, select them all using ctrl+A or the mouse and choose "View As" from the context menu to select the "compact" option. Open the context menu again and choose "Optimize Shape Size | height and width" to reduce the diagram entries in their size.




## Can I use a managed bean as a DataControl?

The use case behind this question is to keep track of the current selected object. For example, a POJO serving a list of departments should also become aware of the selected POJO so the select state can be set on a table. There are two answers with the same "no" response to this question

1. From the perspective of separation of duty, the current row of a component is handled by the business service and set through the JSF component model. Because not all business services maintain an active data model that keeps track of row selection, the ADF binding layer provides this functionality for its iterators. If however the binding layer state is not preserved between two view requests (e.g. because the binding container is refreshed or iterators are re-queried, it is the responsibility of the application developer to set previous selection state back. This then however is not through configuring the Data Control as a managed bean, but a separate managed bean or a memory scope attribute that hold the previous selected row key.
2. A managed bean is handled by the JavaServer Faces framework, whereas the Data Control is managed by ADF. The two are not sharing instances of a POJO, which means that you create

two separate instances of an object if a POJO is configured as a managed bean and the Data Control.

### PPR on a component with a previous render state set to "false"

A use case that developers struggle the most is to display a component that has its "rendered" property set to false in response to a change on another component. Attempts to set the rendered property to "true" and issuing a partial refresh for the component, declaratively or programmatically, usually fail. The reason for this is that components that are not rendered on a view - having their rendered property set to false - don't have an object representation in the JSF server side memory tree and therefore cannot participate in the ADF Faces partial refresh. To make the use case working, set the rendered property to "true" either using an EL reference or a managed bean component access and partially refresh the parent component.

### How-to programmatically determine BTF input parameters

If for logging purpose you need programmatic access to bounded task flow (BTF) parameters and input values without hard coding the input parameter names, you can define a design guideline for developers to write input parameter values to a Map in memory (pageFlowScope).

So instead of mapping input parameters of a bounded task flow to #{pageFlowScope.param1}, #{pageFlowScope.param2}, #{pageFlowScope.param2} … developers would use the following: #{pageFlowScope.btfinputparams.param1}, #{pageFlowScope.btfinputparams.param2} etc.

Now, to access the input parameters from the task flow, you access #{pageFlowScope.btfinputparams}, which provides you with a Map of input parameters and values. So code like shown below grant you access to the input parameters provided for a bounded task flow

```
AdfFacesContext adfFacesCtx = AdfFacesContext.getCurrentInstance();
Map pageFlowScope = adfFacesCtx.getPageFlowScope();
Map btfInputParameters = (Map) pageFlowScope.get("btfinputparams");
…
```

**Note**: There exists a second – very elegant – option using the framework internal `MetadataService` class. However, the use of internal classes is not recommended and therefore an enhancement request was filed to provide a public API for developers to analyze bounded task flow metadata.

### How-to access the bounded task flow document information

To determine the task flow name and id of the current task flow, the code shown below can be used

```
ControllerContext cctx = ControllerContext.getInstance();
ViewPortContext currentViewPort = cctx.getCurrentViewPort();
TaskFlowContext taskFlowCtx = currentViewPort.getTaskFlowContext();
TaskFlowId taskFlowId = taskFlowCtx.getTaskFlowId();
String taskFlowName = taskFlowId.getFullyQualifiedName();
```

### Dynamic Region does not work for second task flow

Dynamic regions reference a managed bean from the task flow binding in the parent page's PageDef file to determine the TaskFlowId to display. When you build the dynamic region declaratively using Oracle JDeveloper, then the managed bean is defined in "backingBean" scope. Backing bean scope however has

a duration of request, which means that at the end of the request the bean status is set back to the default, which usually is the TaskFlowId of the first task flow dropped on a page as a dynamic region. This causes a second task flow to show on a page when switched to, but not function correctly. There are two options for you to fix this

1. Change the bean scope from backingBean scope to viewScope so the managed bean is not reset after the request. When you do this, be aware that the backingBean scope is used in the EL references on the page (the links or tabs to switch between dynamic region views) and the task flow binding in the PageDef file. All of these references need to be changed to use "viewScope" in their EL #{viewScope…}
2. The second option is to change how the managed bean keeps track of the current TaskFlowId. Before serving the current task flow Id, you check a memory attribute in viewScope for the current TaskFlowId value. If this attribute value returns as null – which is does on the initial call – you return the TaskFlowId of the first task flow dropped as a dynamic region. This value then is saved in the viewScope attribute so the next time the attribute value is not null. Whenever the user switches between task flows, you write the TaskFlowId of the selected task flow to the memory attribute. This way the managed bean scope does not need to be changed

```
public class DynRegionBean {
   private static final String CURRENT_TASKFLOWID =
                       "__CURRENT_TF_IN_VIEWSCOPE__";
    private String taskFlowId =
      "/WEB-INF/shopping-cart-task-flow-definition.xml"+
      "#shopping-cart-task-flow-definition";

    public DynRegionBean() {}

    public TaskFlowId getDynamicTaskFlowId() {
        ADFContext adfctx = ADFContext.getCurrent();
        Map viewScope = adfctx.getViewScope();
        if (viewScope.get(CURRENT_TASKFLOWID)==null){
          viewScope.put(CURRENT_TASKFLOWID, taskFlowId);
          return TaskFlowId.parse(taskFlowId);
        }
        else{
         String tfid = (String)viewScope.get(CURRENT_TASKFLOWID);
         return TaskFlowId.parse(tfid));
        }
    }

    public String shoppingcarttaskflowdefinition() {
        taskFlowId =
         "/WEB-INF/shopping-cart-task-flow-definition.xml"+
         "#shopping-cart-task-flow-definition";
        ADFContext adfctx = ADFContext.getCurrent();
```

```
            Map viewScope = adfctx.getViewScope();
            viewScope.put(CURRENT_TASKFLOWID, taskFlowId);
            return null;
        }
    public String detailjspxtaskflowdefinition() {
         taskFlowId =
            "/WEB-INF/detail-jspx-task-flow-definition.xml"+
            "#detail-jspx-task-flow-definition";
        ADFContext adfctx = ADFContext.getCurrent();
        Map viewScope = adfctx.getViewScope()
        viewScope.put(CURRENT_TASKFLOWID, taskFlowId);
     return null;
     }
```

## How-to unlock WLS if locked in exclusive mode

It could happen that a deployment from Oracle JDeveloper 11g fails because the integrated WebLogic Server instance is locked by another instance. One reason for when this may happen is if the WLS Java process previously got killed from the command line. If this happens, then the following error message is displayed in the Oracle JDeveloper message console for a failed deployment attempt

Error is: '[Deployer:149164]The domain edit lock is owned by another session in exclusive mode - hence this deployment operation cannot proceed.'

To fix this problem, open the WLS admin console in a browser by typing http://hostname:7101/console in the browser URL field and connecting as weblogic /weblogic1 in the login form that displays after the console is initialized. In the console, you should see a command button pair in the left upper area to undo or confirm recent changes. As this is a development environment, undoing recent changes appears to be a good choice.

## Stress testing of ADF applications

Of course, ADF applications should not only be easy and fast to build, they should also behave like this at runtime. To verify that your application development scales and that it is able to handle the envisioned user load, you may stress test your application, following hints provided by the links below

http://www.oracle.com/technetwork/oem/app-test/index.html
http://www.testingreflections.com/node/view/8538
http://www.connotea.org/user/jdeveloper/tag/jmeter
http://www.connotea.org/user/jdeveloper/tag/tuning

http://one-size-doesnt-fit-all.blogspot.com/2009/06/stress-load-testing-web-applications.html
http://one-size-doesnt-fit-all.blogspot.com/2010/04/configuring-apache-jmeter-specifically.html

http://www.yenlo.nl/harryvanoosten/2010/03/10/load-testing-an-adf-11g-application/

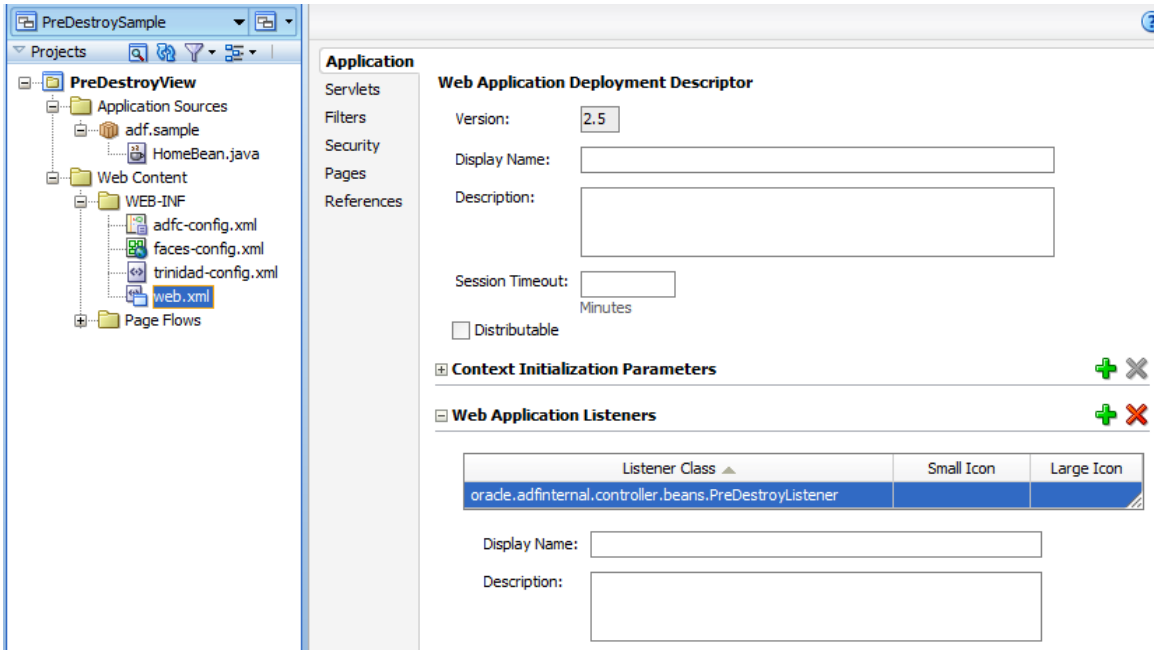
## How-to make @PreDestory work with ADFc

Developers may want to invoke a managed bean method in response to managed bean instantiation or destroy. For this, the JSF lifecycle supports the @PostConstruct and @PreDestroy annotations. ADF

Controller supports JSF lifecycle annotations for managed beans in standard scopes, which are request, session and application. While @PostConstruct works out of the box, @PreDestroy needs requires an entry in the web.xml file

```
<listener>
  <listener-class>
    oracle.adfinternal.controller.beans.PreDestroyListener
  </listener-class>
</listener>
```

The listener declaration in web.xml needs to be created between the filter-mapping and the servlet section. If you are unsure of where this is, double click the web.xml file entry in the Application Navigator and use the "Web Application Listeners" section in the "Application" category.




**RELATED DOCOMENTATION**

| | |
|---|---|
| ☒ | ADF Code Corner<br>http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html |
| ☒ | |
| ☒ | |