

ADF Code Corner

Oracle JDeveloper OTN Harvest 02 / 2011



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 knowledge exchange into an interesting read for developers who enjoy harvesting little nuggets of wisdom.

<http://blogs.oracle.com/jdevotnharvest/>

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
28-FEB-2011

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>*

February 2011 Issue – Table of Content

Does ADF Faces work with JavaScript disabled?.....	2
How to set a default activity in an unbounded task flow	2
How to set the initial component focus	3
How to deploy global managed beans.....	3
getRow(key) and findByKey(key,1) inconsistency	4
How to protect UI components using OPSS Resource Permissions...5	
How-to change the required field indicator location	10
How to launch LOV and Date dialogs using the keyboard	11
How to filter tree node child data	12
How to ensure serverListener events fires before action events.....	18
Best practices for good performance in ADF	19

Does ADF Faces work with JavaScript disabled?

No. ADF Faces requires browsers to have JavaScript enabled.

How to set a default activity in an unbounded task flow

An unbounded task flow does not have clearly defined boundaries and thus does not have a default activity to start an application. The unbounded task flow definition is the equivalent of the JavaServer Faces `faces-config.xml` file and users may request any of its defined view activities from a browser URL. To enforce a defined entry to an application, developers can

- Use a phase listener that redirects a request to a start page if the request type is GET and the requested view id is not the view to start with
- Only add a single view activity definition to the unbounded task flow and then use a task flow activity to continue with bounded task flows. In this case the **URL Invoke** property of all task flows should be set to *url-invoke-disallowed* so task flows are not directly accessible from browsers.
- Use bounded task flows only and set the **URL Invoke** property of all task flows but one to *url-invoke-disallowed*. This allows direct browser GET access for a single bounded task flow only. The unbounded task flow definition is empty

How to set the initial component focus

In ADF Faces, you use the `af:document` tag's *initialFocusId* to define the initial component focus. For this, specify the **id** property value of the component that you want to put the initial focus on. Identifiers are relative to the component, and must account for NamingContainers. You can use a single colon to start the search from the root, or multiple colons to move up through the NamingContainers - `":"` will pop out of the component's naming container and begin the search from there, `":":` will pop out of two naming containers and begin the search from there. Alternatively you can add the naming container IDs as a prefix to the component Id, e.g. `nc1:nc2:comp1`.

http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_document.html

To set the initial focus to a component located in a page fragment that is exposed through an ADF region, keep in mind that ADF Faces regions – `af:region` – is a naming container too. To address an input text field with the *id* "it1" in an ADF region exposed by an `af:region` tag with the *id* `r1`, you use the following reference in `af:document`:

```
<af:document id="d1" initialFocusId="r1:0:it1">
```

Note the "0" index in the client Id. Also, make sure the input text component has its *clientComponent* property set to **true** as otherwise no client component exist to put focus on.

How to deploy global managed beans

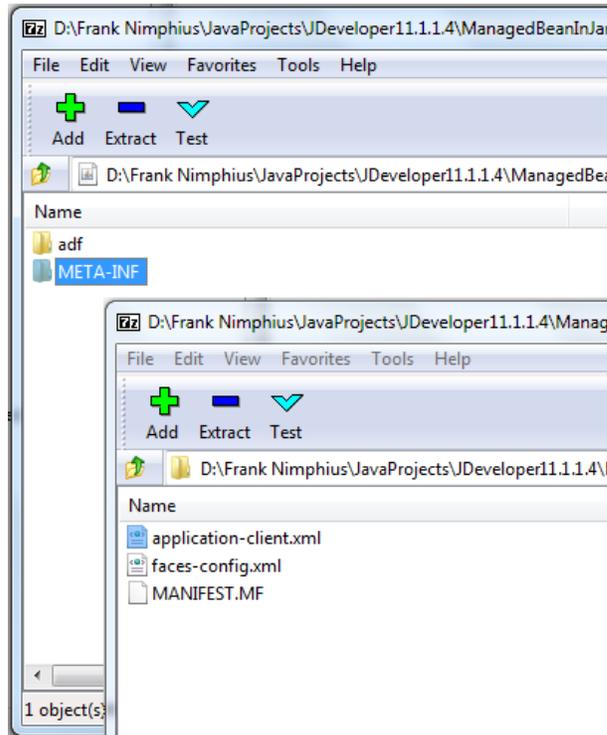
"Global managed" beans is the term I use in this post to describe beans that are used across applications. Global managed beans contain helper – or utility – methods like or instead of JSFUtils and ADFUtils. The difference between global managed beans and static helper classes like JSFUtils and ADFUtils is that they are EL accessible, providing reusable functionality that is ready to use on UI components and – if accessed from Java – in other managed beans.

For example, the ADF Faces page template (`af:pageTemplate`) allows you to define attributes for the consuming page to pass in object references or strings into it. It does not have method attributes that allow command components contained in a template to invoke listeners in managed beans and the ADF binding layer, or to execute actions. To create templates that provide global button or menu functionality, like logon, logout, print etc., an option for developers is to deploy managed beans with the ADF Faces page templates. To deploy a managed bean with a page template, create an ADF library from the project containing the template definition and import this ADF library into the target project using the Resource

palette. When importing an ADF library, all its content is added to the project, including page template definitions, managed bean sources and configurations.

More about page templates

http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_pageTemplate.html



Using a globally configured managed bean allows you to use Expression Language in the UI to access common functionality but also use Java in application specific managed beans. Storing the faces-config.xml file in the JAR file META-INF directory automatically makes it available when the JAR file is found in the class path of an application.

Another use-case for globally configured managed beans is for creating helper methods to be used in many applications. Instead of creating a base managed bean class that then is extended by all managed beans used in applications, you can deploy a managed bean in a JAR file and add the faces-config.xml file with the managed bean configuration to the JAR's META-INF folder as shown below.

getRow(key) and findByKey(key,1) inconsistency

Calling `getRow(key)` on a `RowSet` seems to return different results than using `findByKey(key,1)` on the same set. The `getRow(Key)` API iterates the rows in the `RowSet` (fetching rows from the DB as needed) until the key matches. In the worst case, if your VO sorts data and the key you are looking for is the last row in the row set, you fetch all rows from the database just to find the one you're looking for.

Also, if you have a View with `ORDER BY <ATTRIBUTE>` and you call `setMaxFetchSize(5)` on the VO at runtime, it will only fetch 5 rows and the `getRow(key)` may not find a result because it is not in the first five result rows in sorted order. In this case, the `getRow(key)` method will return a different result than the `findByKey(key,1)` method searching the same `RowSet`.

In contrast, `findByKey(key, 1)` formulates a targeted query using the key attribute(s) and retrieves the row from the database if it cannot already find the row in the cache.

The use of `findByKey` is recommended to use unless you have a use case that demands using the `getRow` method

(answered by Steve Muench)

How to protect UI components using OPSS Resource Permissions

ADF security protects ADF bound pages, bounded task flows and ADF Business Components entities with framework specific JAAS permissions classes (`RegionPermission`, `TaskFlowPermission` and `EntityPermission`). If used in combination with the ADF security expression language and security checks performed in Java, this protection already provides you with fine grained access control that can also be used to secure UI components like buttons and input text field. For example, the EL shown below disables the user profile panel tabs for unauthenticated users:

```
<af:panelTabbed id="pt1" position="above">
  ...
  <af:showDetailItem
    text="User Profile" id="sdi2"
    disabled="#{!securityContext.authenticated}">
  </af:showDetailItem>
  ...
</af:panelTabbed>
```

The next example disables a panel tab item if the authenticated user is not granted access to the bounded task flow exposed in a region on this tab:

```
<af:panelTabbed id="pt1" position="above">
  ...
  <af:showDetailItem text="Employees Overview" id="sdi4"
    disabled="#{!securityContext.taskflowViewable
      ['/WEB-INF/EmployeeUpdateFlow.xml#EmployeeUpdateFlow']}">
  </af:showDetailItem>
  ...
</af:panelTabbed>
```

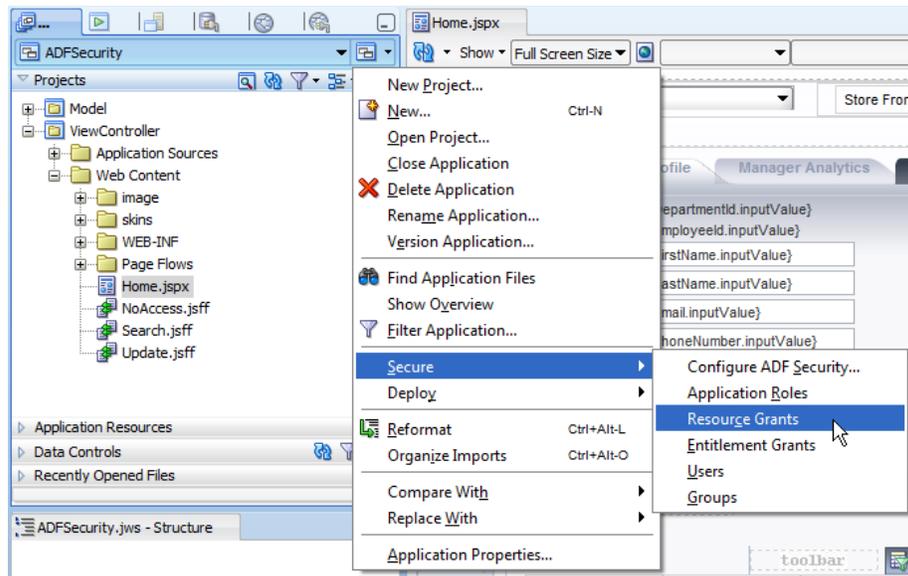
Security expressions like shown above allow developers to check the user permission, authentication and role membership status before showing UI components. Similar, using Java, developers can use code like shown below to verify the user authentication status:

```
ADFContext adfContext = ADFContext.getCurrent();
SecurityContext securityCtx = adfContext.getSecurityContext();
boolean userAuthenticated = securityCtx.isAuthenticated();
```

Note that the Java code lines use the same security context reference that is used with expression language.

But is this all that there is? **No !** The goal of ADF Security is to enable all ADF developers to build secure web application with JAAS (Java Authentication and Authorization Service). For this, more fine grained protection can be defined using the `ResourcePermission`, a generic JAAS permission class owned by the Oracle Platform Security Services (OPSS). Using the `ResourcePermission` class, developers can grant permission to functional parts of an application that are not protected by page or task flow security.

For example, an application menu allows creating and canceling product shipments to customers. However, only a specific user group – or application role, which is the better way to use ADF Security – is allowed to cancel a shipment.



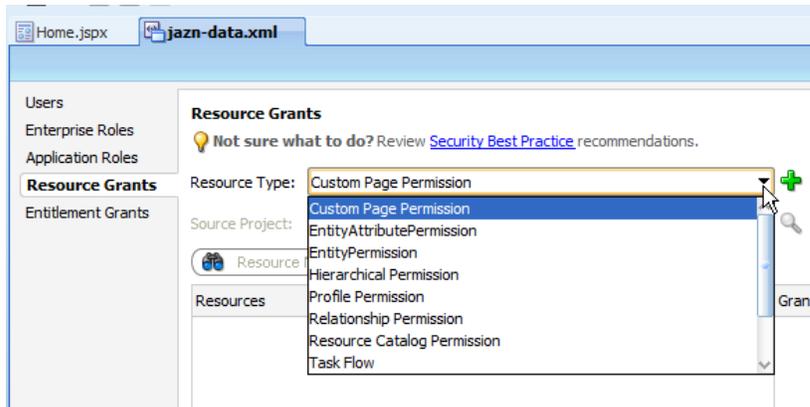
To enforce this rule, a permission is needed that can be used declaratively on the UI to hide a menu entry and programmatically in Java to check the user permission before the action is performed.

Note that multiple lines of defense are what you should implement in your application development. Don't just rely on UI protection through hidden or disabled command options.

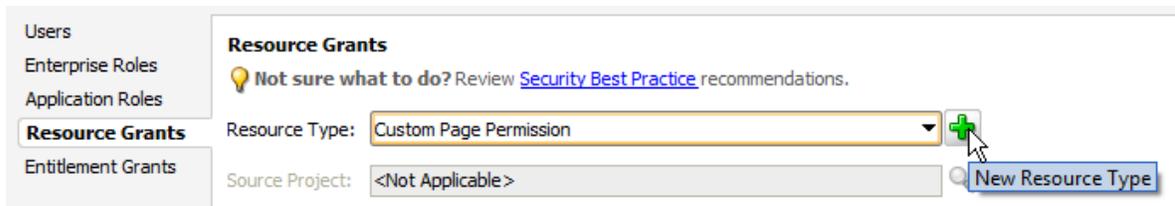
To create menu protection permission for an ADF Security enable application, you choose **Application | Secure | Resource Grants** from the Oracle JDeveloper menu.

The opened editor shows a visual representation of the `jazn-data.xml` file that is used at design time to define security policies and user identities for testing. An option in the **Resource Grants** section is to create a new **Resource Type**.

A list of pre-defined types exists for you to create policy definitions for. Many of these pre-defined types use the `ResourcePermission` class.

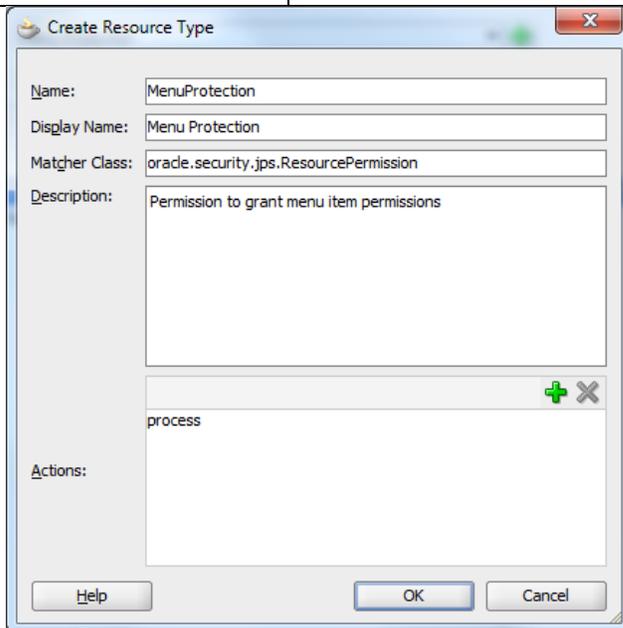


To create a custom **Resource Type**, for example to protect application menu functions, you click the green plus icon next to the **Resource Type** select list.



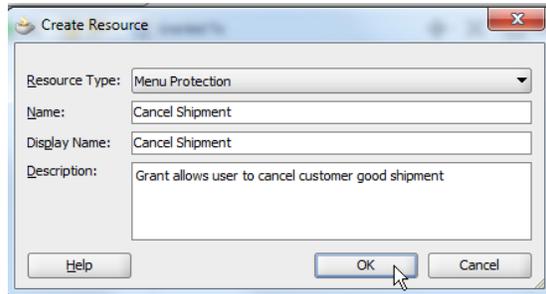
The **Create Resource Type** editor that opens allows you to add a name for the resource type, a display name that is shown when granting resource permissions and a description. The `ResourcePermission` class name is already set. In the menu protection sample, you add the following information:

Name:	MenuProtection
Display Name:	Menu Protection
Description:	Permission to grant menu item permissions



OK the dialog to close the resource permission creation.

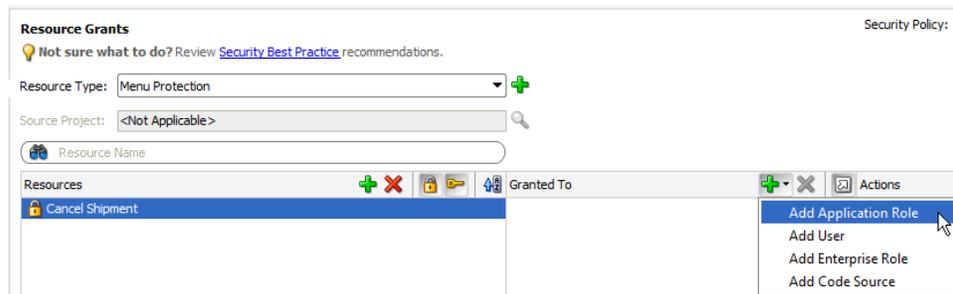
To create a resource policy that can be used to check user permissions at runtime, click the **green plus** icon in the **Resources** section of the **Resource Grants** section.



In the **Create Resource** dialog, provide a name for the menu option you want to protect. To protect the **cancel shipment** menu option, create a resource with the following settings

Resource Type:	Menu Protection
Name:	Cancel Shipment
Display Name:	Cancel Shipment
Description:	Grant allows user to cancel customer good shipment

A new resource **Cancel Shipment** is added to the **Resources** panel. Initially the resource is not granted to any user, enterprise or application role. To grant the resource, click the green plus icon in the **Granted To** section, select the **Add Application Role** option and choose one or more application roles in the opened dialog.



Finally, you click the **process** action to define the policy. Note that permission can have multiple actions that you can grant individually to users and roles. The cancel shipment permission for example could have another action "view" defined to determine which user should see that this option exist and which users don't.

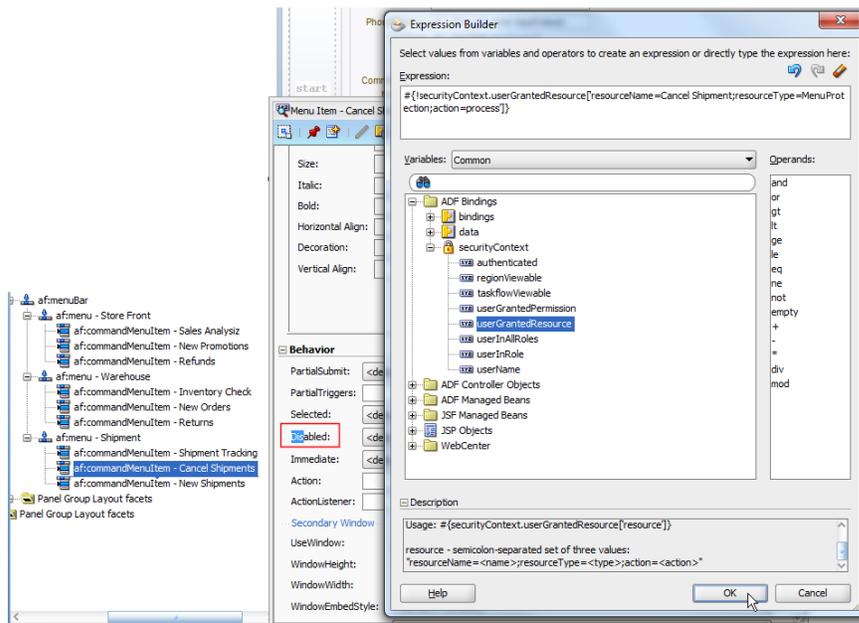


To use the **cancel shipment** permission, select the **disabled** property on a command item, like `af:commandMenuItem` and click the arrow icon on the right. From the context menu, choose the **Expression Builder** entry. **Expand the ADF Bindings | securityContext** node and click the **userGrantedResource** option.

Hint: You can expand the **Description** panel below the EL selection panel to see an example of how the grant should look like.

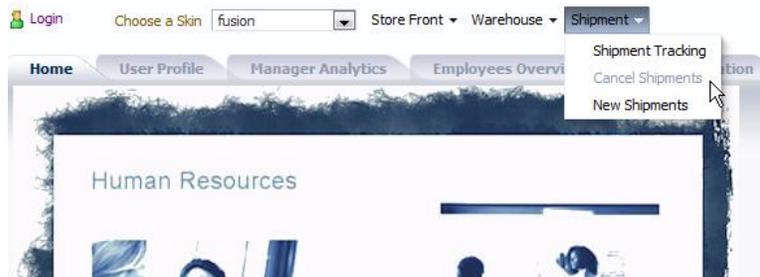
The EL that is created needs to be manually edited to show as

```
#{securityContext.userGrantedResource[
    resourceName=Cancel Shipment;resourceType=MenuProtection;action=process']}
```



OK the dialog so the permission checking EL is added as a value to the **disabled** property. Running the application and expanding the **Shipment** menu shows the **Cancel Shipments** menu item disabled for all users that don't have the custom menu protection resource permission granted.

Note: Following the steps listed above, you create a JAAS permission and declaratively configure it for function security in an ADF application. Do you need to understand JAAS for this? No! This is one of the benefits that you gain from using the ADF development framework.



To implement multi lines of defense for your application, the action performed when clicking the enabled "Cancel Shipments" option should also check if the authenticated user is allowed to use process it. For this, code as shown below can be used in a managed bean

```
public void onCancelShipment(ActionEvent actionEvent) {
    SecurityContext securityCtx =
        ADFContext.getCurrent().getSecurityContext();
    //create instance of ResourcePermission(String type, String name,
    //String action)
    ResourcePermission resourcePermission =
        new ResourcePermission("MenuProtection","Cancel Shipment",
            "process");
    boolean userHasPermission =
        securityCtx.hasPermission(resourcePermission);
    if (userHasPermission){

        //execute privileged logic here
    }
}
```

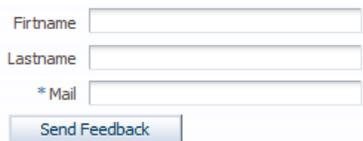
Note: To learn more about ADF Security, visit

http://download.oracle.com/docs/cd/E17904_01/web.1111/b31974/adding_security.htm#BGBGJEAH

How-to change the required field indicator location

By default, the required field indicator that you specify for an input field is left aligned to the component label as shown in the image below.

```
<af:inputText label="Mail" id="it3"
    requiredMessageDetail="You must provide your mail address to send feedback to the forum"
    required="true"/>
```



Firstname

Lastname

* Mail

To change the icon position to show to the right of the label, you use skinning as shown below

```
.AFRequiredIconStyle{
    float:right;
}
```

Firtname

Lastname

Mail*

More about skinning in ADF Faces:

http://download.oracle.com/docs/cd/E14571_01/web.1111/b31973/af_skin.htm

How to launch LOV and Date dialogs using the keyboard

Using the ADF Faces JavaScript API, developers can listen for user keyboard input in input components to filter or respond to specific characters or key combination. The JavaScript shown below can be used with an `af:clientListener` tag on `af:inputListOfValues` or `af:inputDate`. At runtime, the JavaScript code determines the component type it is executed on and either opens the LOV dialog or the input Date popup.

```
<af:resource type="javascript">
  /**
   * function to launch dialog if cursor is in LOV or
   * input date field
   * @param evt argument to capture the AdfUIInputEvent object
   */

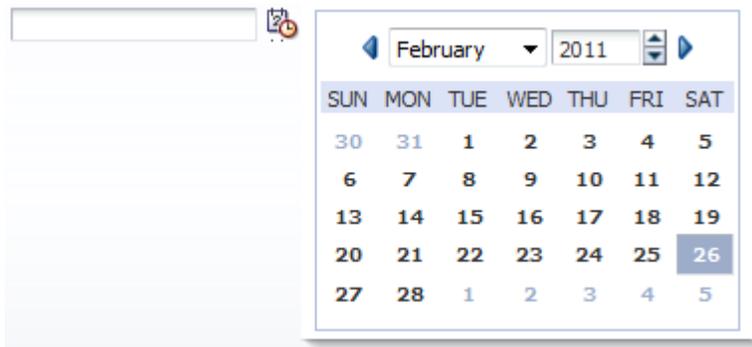
  function launchPopUpUsingF8(evt) {
    var component = evt.getSource();
    if (evt.getKeyCode() == AdfKeyStroke.F8_KEY) {
      //check for input LOV component
      if (component.getTypeName() == 'AdfRichInputListOfValues') {
        AdfLaunchPopupEvent.queue(component, true);
        //event is handled on the client. Server does not need
        //to be notified
        evt.cancel();
      }
      //check for input Date component
      else if (component.getTypeName() == 'AdfRichInputDate') {
        //the inputDate af:popup component ID always is ::pop
        var popupClientId = component.getId() + '::pop';
        var popup = component.findComponent(popupClientId);
        var hints = {align : AdfRichPopup.ALIGN_END_AFTER,
                    alignId : component.getAbsoluteLocator()};
        popup.show(hints);
        //event is handled on the client. Server does not need
        //to be notified
        evt.cancel();
      }
    }
  }
}
```

```
    }  
  }  
</af:resource>
```

The `af:clientListener` that calls the JavaScript is added as shown below.

```
<af:inputDate label="Label 1" id="id1">  
  <af:clientListener method="launchPopUpUsingF8" type="keyDown"/>  
</af:inputDate>
```

As you may have noticed, the call to open the popup is different for the `af:inputListOfValues` and the `af:inputDate`. For the list of values component, an ADF Faces `AdfLaunchPopupEvent` is queued with the LOV component passed as an argument. Launching the input date popup is a bit more complicated and requires you to lookup the implicit popup dialog and to open it manually. Because the popup is opened manually using the `show()` method on the `af:popup` component, the alignment of the dialog also needs to be handled manually. For this, the popup component specifies alignment hints, that for the `ALIGN_END_AFTER` hint aligns the dialog at the end and below the date component. The `alignId` hint specifies the component the dialog is relatively positioned to, which of course should be the input date field. At runtime, the popup opens as shown below.



The ADF Faces JavaScript API and how to use it is further explained in the *Using JavaScript in ADF Faces Rich Client Applications* whitepaper available from the Oracle Technology Network (OTN)

<http://www.oracle.com/technetwork/developer-tools/jdev/1-2011-javascript-302460.pdf>

An ADF Insider recording about JavaScript in ADF Faces can be watched from here

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/adf-insider-javascript/adf-insider-javascript.html

How to filter tree node child data

When you drag an ADF Business Components View Object from the Data Controls panel in Oracle JDeveloper and drop it as an ADF bound ADF Faces tree component, two things happen implicitly:

1. A tree binding is created that allows you to configure the hierarchical structure of the data presented in the tree.

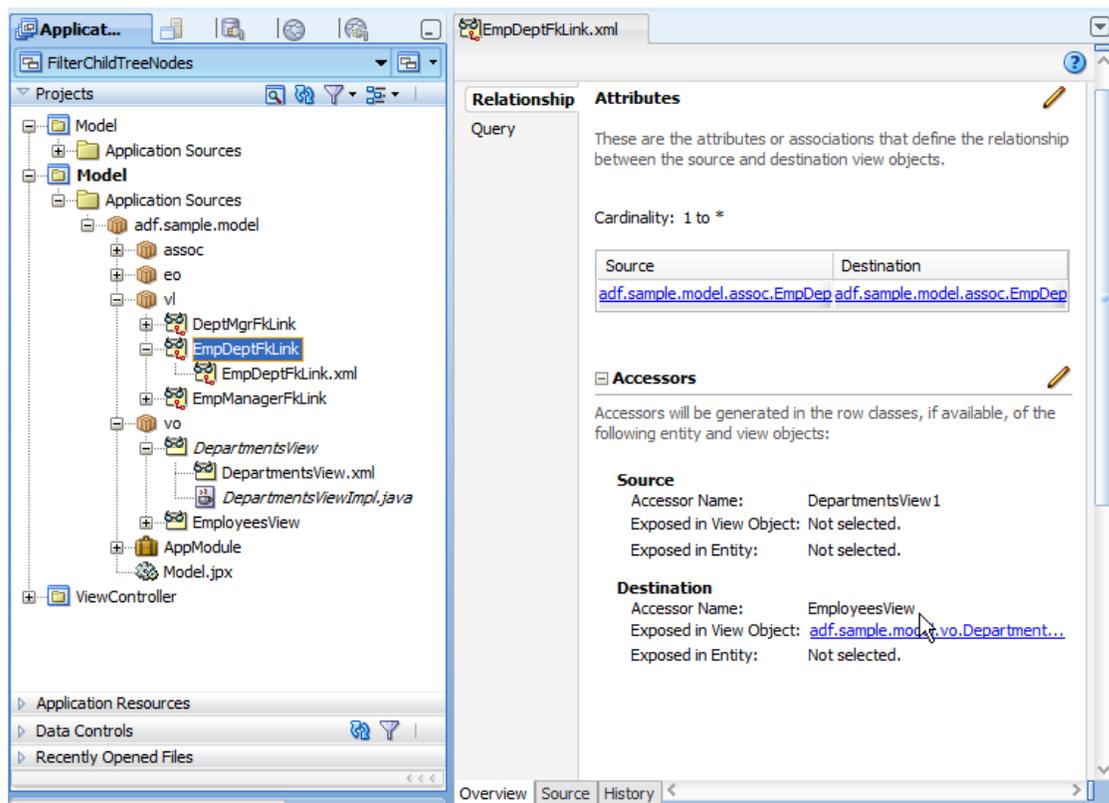
- An iterator binding is created for the View Object that represents the top level node, the root node. No additional iterators are created for the dependent View Objects that populate the tree nodes

A frequent requirement of developers is to manipulate the tree node child query to filter data – for example employee records – displayed in the tree.

Example 113 on Steve Muench's *Not Yet Documented ADF Sample Applications* blog explains how to filter View Objects that are exposed as child nodes in an ADF bound tree. Though the sample is written with and for Oracle JDeveloper 10.1.3, it is still valid. View Objects are hierarchically connected through *View Links*. *Accessors* exposed on the View Links provide developers a handle to the child collection.

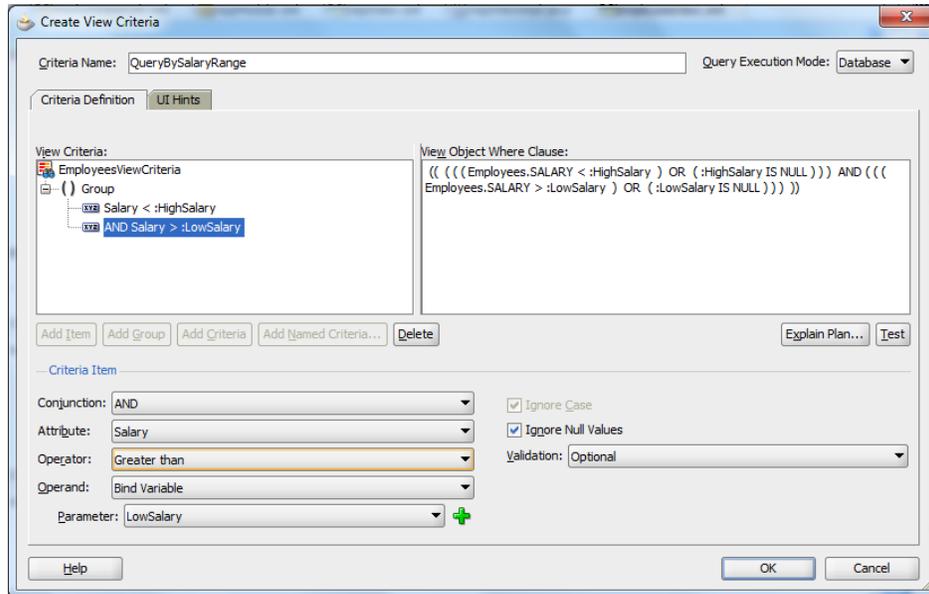
In the following, I explain a solution that closely follows Steve Muench's original example, except that it uses a View Criteria to filter the dependent EmployeesView rows. The sample uses the **Departments** and **Employees** table of the Oracle RDBMS HR sample schema. To create the Model project, just run the *Create Business Components from Table* wizard in the Oracle JDeveloper New Gallery. I did later refactor the generated View Objects and Entity Objects to reside in their own packages. I did the same for view links and entity associations.

The relationship between a Department and its Employees is represented by the *EmpDeptFKLink* view link.

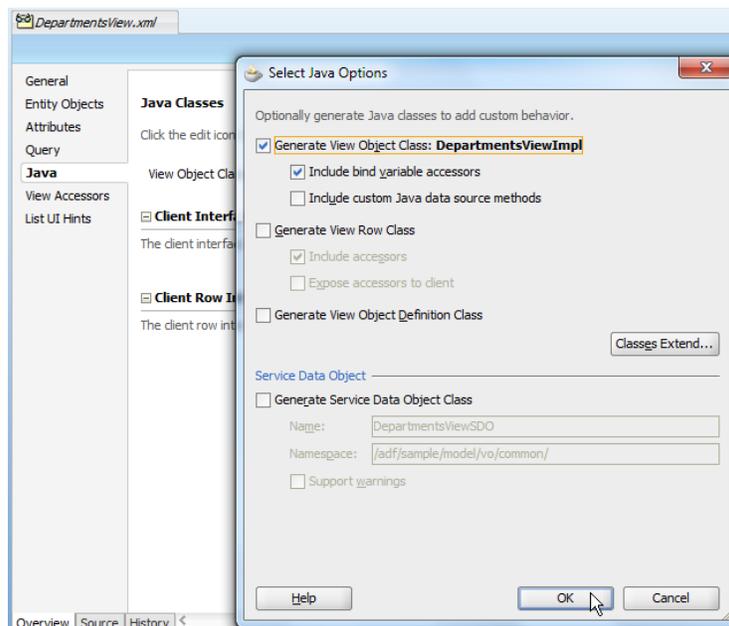


The association from the DepartmentsView parent collection to the EmployeesView detail collection is represented by the *EmployeesView* accessor name

To filter employees data by the salary range a user provides in the web tier, a View Criteria is created on the *EmployeesView* view object. Double click onto the View Object and select the *Query* panel in the opened editor. Create the View Criteria as shown below.



Note: Because the tree references the child employee collection through the accessor, it does not make sense to configure the View Criteria on a View Object instance in the Application Module data model. Instead, it needs to be dynamically added to the accessor View Object.

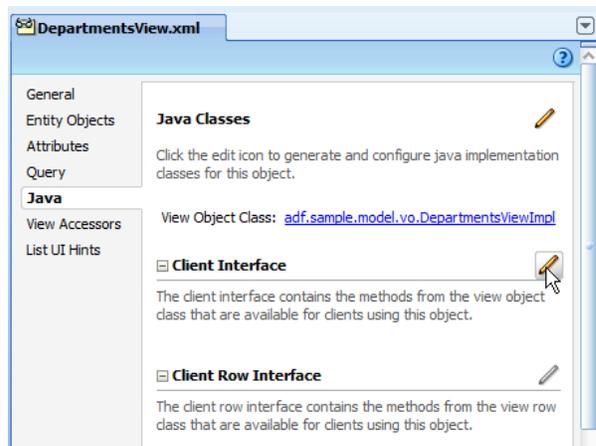


To set the bind variables defined in the View Criteria and to apply the View Criteria, a method needs to be created in the *Departments View Object Impl* class and exposed as a client method. Double click the *DepartmentsView* View Object and choose the *Java* panel in the opened editor. Click the pencil icon next to the *Java Classes* header. As shown in the image above, select the *Generate View Object Class* option and press

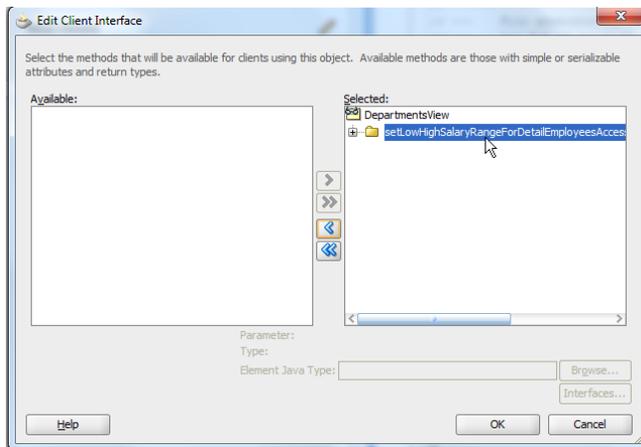
OK. Open the created View Object Impl class and provide the following method, a slight variation of Steve Muench's original. Note that the data type `Number` is `oracle.jbo.domain.Number`. It's the same data type used for the Bind Variables in the View Criteria.

```
public void setLowHighSalaryRangeForDetailEmployeesAccessorViewObject
    (Number lowSalary,
     Number highSalary) {
    Row r = getCurrentRow();
    if (r != null) {
        RowSet rs = (RowSet)r.getAttribute("EmployeesView");
        if (rs != null) {
            ViewObject accessorVO = rs.getViewObject();
            accessorVO.ensureVariableManager();
            accessorVO.getVariableManager().setVariableValue(
                "LowSalary", lowSalary);
            accessorVO.getVariableManager().setVariableValue(
                "HighSalary", highSalary);
            ViewCriteriaManager vcm = accessorVO.getViewCriteriaManager();
            ViewCriteria vc = vcm.getViewCriteria("QueryBySalaryRange");
            accessorVO.applyViewCriteria(vc);
        }
        executeQuery();
    }
}
```

Next, open the Departments View object editor – if not already open - and again select the Java category. Click the pencil icon next to the **Client Interface** entry.

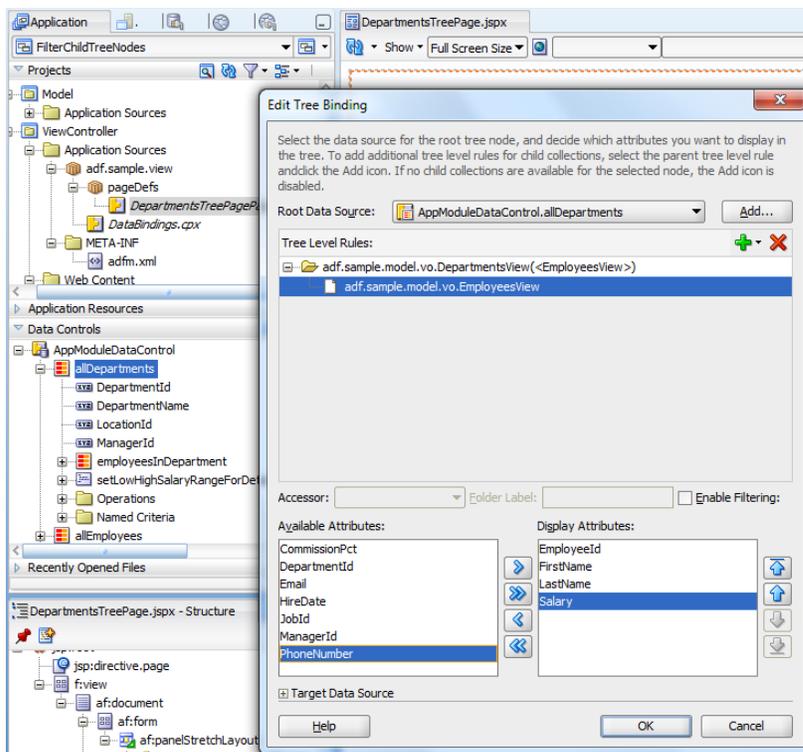


Shuttle the method in the `DepartmentsViewImpl` class to the list of *Selected* interface methods

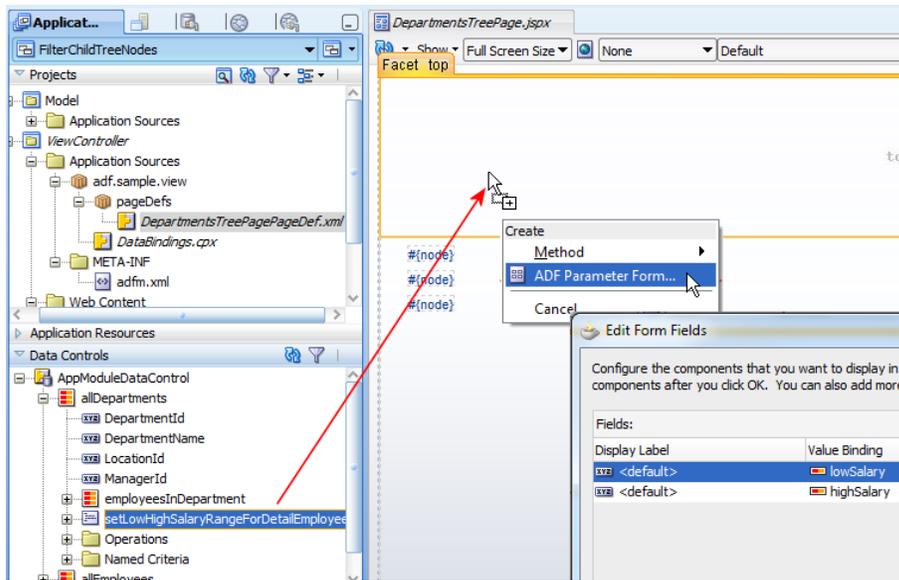


In the ViewController project, create a new ADF Faces page and drag and drop the *allDepartments* View Object instance as an ADF tree.

Note: For better readability, I renamed the default View Object instance names in the ApplicationModule data model. For this, double click the Application Module entry and select the *Data Model* panel. Select the View Object instance to change the name for and choose *Rename* from the right mouse context menu.



When the tree is created, drag the *allDepartments* View Object instance client method that is exposed in the Data Control panel onto the JSF page and drop it as an ADF Parameter form.



Change the submit button to "Filter Employees by Salary" and configure the tree component's *Partial Triggers* property to point to the parameter form command button ID. Ensure the button's *PartialSubmit* property is set to **true**.

Change the prompts of the parameter form fields to "High Salary" and "Low Salary"



At runtime, the page comes up as shown below



Providing values for the *Low Salary* and the *High Salary* field and pressing the command button queries the detail node (Employee) data before issuing a partial refresh of the tree component. In the example above, the tree is cleared from employee records that have salary values less than 2000 or more than 5000.

Note: You find Steve Muench's sample page at: <http://blogs.oracle.com/smuenchadf/examples/>

How to ensure serverListener events fires before action events

Using JavaScript in ADF Faces you can queue custom events defined by an `af:serverListener` tag. If the custom event however is queued from an `af:clientListener` on a command component, then the command component's action and action listener methods fire before the queued custom event. If you have a use case, for example in combination with client side integration of 3rd party technologies like HTML, Applets or similar, then you want to change the order of execution.

The way to change the execution order is to invoke the command item action from the client event method that handles the custom event propagated by the `af:serverListener` tag. The following four steps ensure your successful doing this

1. Call `cancel()` on the event object passed to the client JavaScript function invoked by the `af:clientListener` tag
2. Call the custom event as an immediate action by setting the last argument in the custom event call to `true`

```
function invokeCustomEvent(evt) {
    evt.cancel();
    var custEvent = new AdfCustomEvent(
        evt.getSource(),
        "mycustomevent",
        {message:"Hello World"},
        true);
    custEvent.queue();
}
```

3. When handling the custom event on the server, lookup the command item, for example a button, to queue its action event. This way you simulate a user clicking the button. Use the following code

```
ActionEvent event = new ActionEvent(component);
event.setPhaseId(PhaseId.INVOKE_APPLICATION);
event.queue();
```

The *component* reference needs to be changed with the handle to the command item which action method you want to execute.

4. If the command component has behavior tags, like `af:fileDownloadActionListener`, or `af:setPropertyListener`, defined, then these are also executed when the action event

is queued. However, behavior tags, like the file download action listener, may require a full page refresh to be issued to work, in which case the custom event cannot be issued as a partial refresh.

File download action tag:

http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_fileDownloadActionListener.html

" Since file downloads must be processed with an ordinary request - not XMLHttpRequest AJAX requests - this tag forces partialSubmit to be false on the parent component, if it supports that attribute."

To issue a custom event as a non-partial submit, the previously shown sample code would need to be changed as shown below

```
function invokeCustomEvent (evt) {  
    evt.cancel ();  
    var custEvent = new AdfCustomEvent (  
        evt.getSource (),  
        "mycustomevent",  
        {message:"Hello World"},  
        true);  
    custEvent.queue (false) ;  
}
```

To learn more about custom events and the `af:serverListener`, please refer to the tag documentation:

http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_serverListener.html

Best practices for good performance in ADF

A reoccurring question on the OTN forum is the one about performance best practices, which various community members has blogged about in the past. Hints for best performance however are also well documented in the *Oracle Fusion Middleware Performance and Tuning Guide*:

http://download.oracle.com/docs/cd/E14571_01/core.1111/e10108/adf.htm

Another source of information, if you use ADF Business Components as a business service, are chapters 40 and chapter 41 of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* about ADF BC Application Module state management and tips for module and connection pooling

http://download.oracle.com/docs/cd/E17904_01/web.1111/b31974/bcstatemgmt.htm#sm0318

http://download.oracle.com/docs/cd/E17904_01/web.1111/b31974/bcampool.htm#sm0299

<input type="checkbox"/>	
<input type="checkbox"/>	
<input type="checkbox"/>	