ADF Code Corner

Oracle JDeveloper OTN Harvest 08 / 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
30-August-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

August 2011 Issue – Table of Contents

New OTN Harvest Feature: OTN Harvest Spotlight

The ADF community is growing at an exponential rate. New developers joining the community will find that they are joining a family of like-minded ADF practitioners, many whom personally know each other and are motivated to help others where they can. To make it easier for new developers to find their way around the ADF community, a new feature column, the **OTN Harvest Spotlight** starts this month.

The OTN Harvest Spotlight introduces members of the ADF community, allowing us all to learn about individuals in the community, their company background, experiences, tips, tricks and the motivation behind their work in the ADF community.

While the intention is not to focus on the most prominent community members, the first featured column spotlights Chris Muir, Oracle ACE director and the founder and driving force behind the ADF Enterprise Methodology Group (EMG). Read about Chris Muir in the first OTN Harvest Spotlight published at the end of this issue.

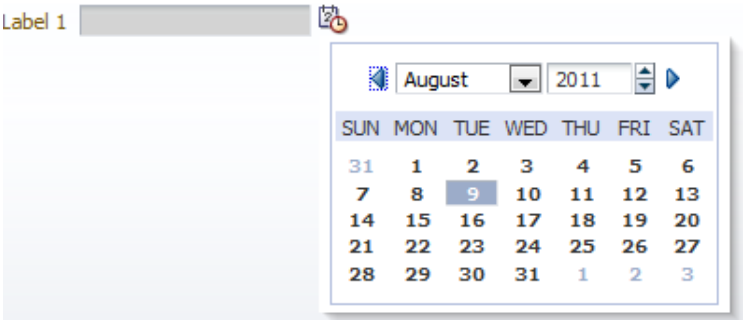
How to learn and where to start learning ADF

A frequent question on the OTN forum for Oracle JDeveloper and ADF is "how do I learn ADF". Shay Shmeltzer from the Oracle Product Management team did publish a comprehensive summary on his blog in 2010, which you can read from

http://blogs.oracle.com/shay/entry/how_do_i_start_learning_oracle_adf_and_jdeveloper

Disabling keyboard input on af:inputDate

Setting the **ReadOnly** property on an `af:inputDate` component to true does not only make the input field read-only but also hides the calendar icon.



If the use case is to force users to always select an input date from the popup calendar and to prevent keyboard input, as show in the image above, you can use a JavaScript based solution as shown below:

```
<af:resource>
    function disableEntry(evt){
          evt.cancel();
     }
 </af:resource>
…
```

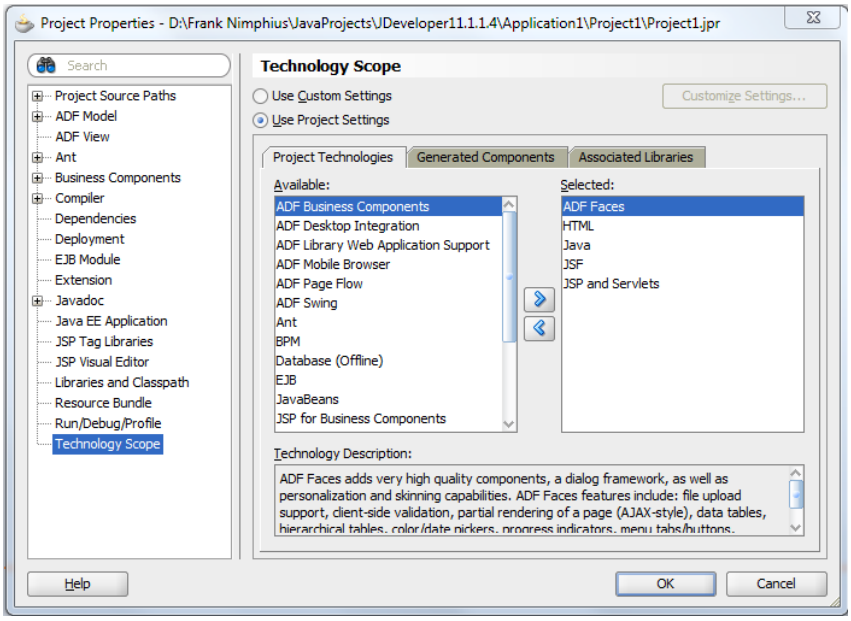
```
<af:inputDate label="Label 1" id="id1" readOnly="false"
        contentStyle="background-color:lightgray;">
        <af:clientListener method="disableEntry" type="keyDown"/>
</af:inputDate>
```

The **contentStyle** attribute sets the `af:inputDate` field background color to light gray to indicate a read-only field.

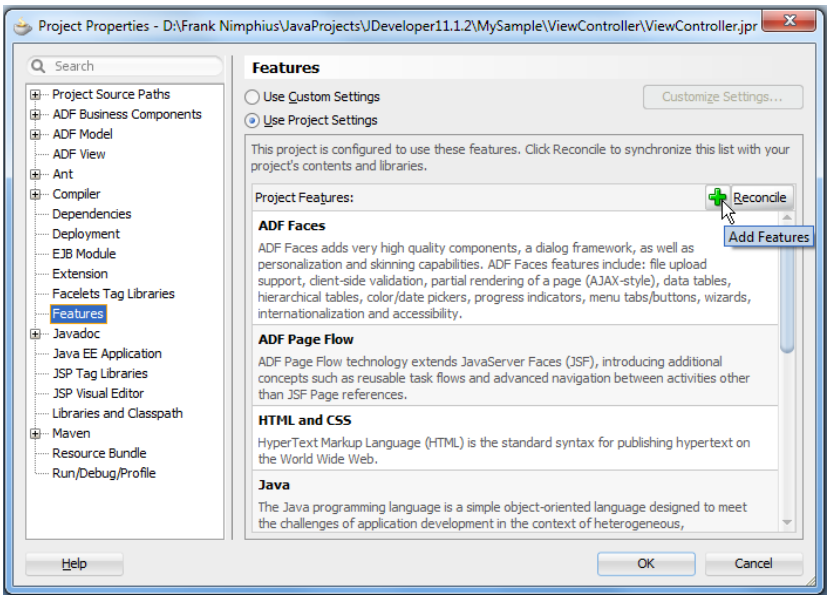
Note: If the read-only setting is for all instances `af:inputDate` then, instead if using the **contentStyle** attribute, you use skinning **af|inputDate::content{background-color:lightgray;}**

Technology Scope in 11g R1 vs. Features in 11g R2

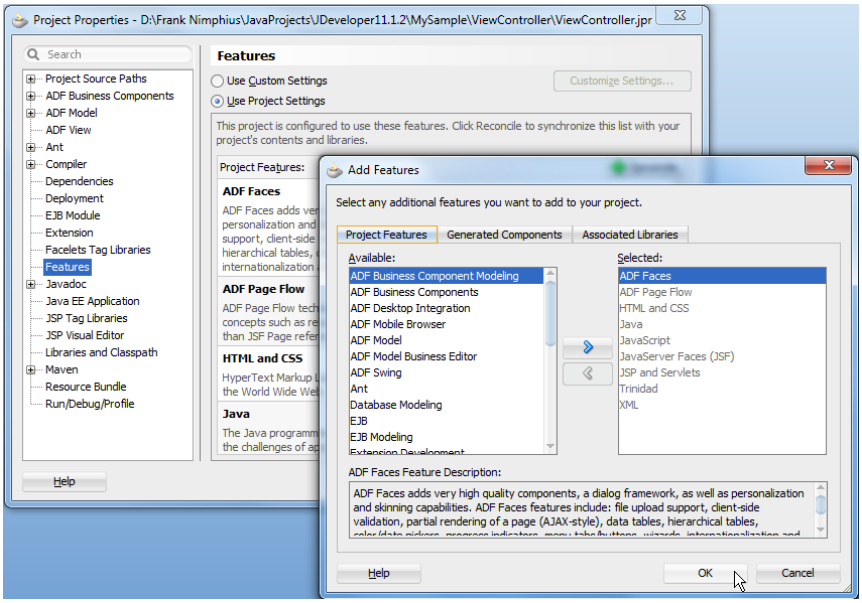
The **Technology Scope** feature of Oracle JDeveloper 11g R1 has been renamed and restructured to **Features** in Oracle JDeveloper 11g R2. In Oracle JDeveloper 11*g* R1 (11.1.1.x), the Technology Scope settings of a project determines the library and project configuration. You access the **Technology Scope** setting with a mouse double click on the project node, or by choosing **Project Properties** in the right mouse context menu of a project.



You use the **Technology Scope** to select ADF or Java EE technologies for a project. Oracle JDeveloper makes sure that the required JAR files are added to the project class path and meta-data components, for example, `web.xml` or `adfc-config.xml`, are created and added to the project.

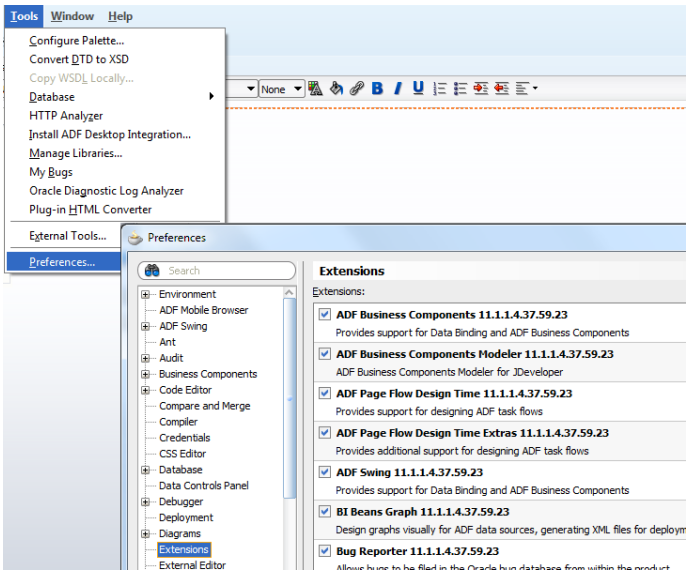


In Oracle JDeveloper 11g R2 (11.1.2.x), the Technology Scope feature has been renamed and restructured to **Features**. The functionality is the same, and the access path similar: You access the **Features** setting with a mouse double click on the project node, or by choosing **Project Properties** in the right mouse context menu of a project. To change the **Features** configuration, you need to click the green plus icon, which then shows a dialog similar to the one in Oracle JDeveloper 11g R1.



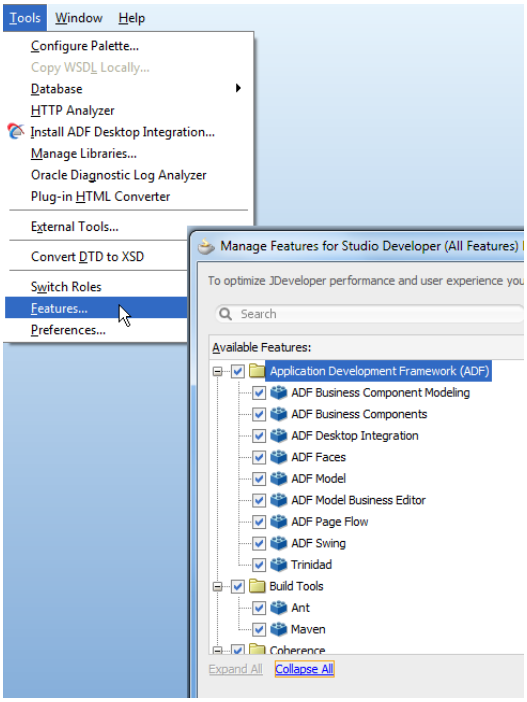
How to tell which JDeveloper extensions are installed

In Oracle JDeveloper 11g R1 (11.1.1.x), extensions can be added, which you usually do with **Help | Check for Updates**, viewed and disabled, which you do by selecting **Tools | Preferences | Extensions** from the Oracle JDeveloper menu menu.



For example, you may want to disable extensions for technologies you don't use in your application development to improve the IDE performance after a re-start.

In Oracle JDeveloper 11g R2, the extensions view has been moved and renamed. You now find them in the **Tools | Features** menu



Note that in Oracle JDeveloper 11g R2, extensions are loaded lazily, which means that you don't get the same IDE performance benefit from disabling extensions.

row.attributeName vs. row.bindings.attributeName

ADF uses two different types of EL to reference the binding layer in `af:tree`, `af:treeTable` and `af:table` components. For example, if a table is read-only, using `af:outputText` to render the table column values, the cell values are referenced as #{row.attributeName}. For editable tables, the EL is #{row.bindings.attributeName.inputValue}. Using #{row.attributeName}, the value object is read-only and of type String, whereas Using #{row.bindings.attributeName.inputValue}, the object is updateable and of type `FacesCtrlAttrsBinding`, an internal class that extends `JUCtrlAttrsBinding`, which implements `AttributeBinding`. So whenever table cell values are updateable, or if other attribute information should be accessed, the EL used is #{row.bindings.attributeName.inputValue}.

Another example is #{node.attributeName}, which is used as a value reference in read-only tree components. "node" and "row" are EL variables defined on the tree or table component variable property.

Suggested skin editor workflow

Following the work flow outlined below helps you to create a custom skin for your AD Faces application using the integrated or stand alone skin editor.

Integrated Skin Editor

The integrated skin editor is good to use for projects that are migrated or newly created in Oracle JDeveloper 11g R2.

1. **Create a skin project**. Skins are reusable components and as such should be created in their own project or workspace.
2. **Create a new skin by extending an existing skin**. Oracle ADF Faces provides simplified default skins, like fusionFx-simple-v2.desktop, that are specifically designed to be extended by custom skins. Note that Fx-simple-v2 is not available for releases before Oracle JDeveloper 11g R2. If you are on an 11.1.1.x version of Oracle JDeveloper, use the stand-alone editor. The stand-alone editor helps with which default skins are available for the target version you build the skin for.
3. **Analyze the application for the components to skin**. Identify the components in a page that you want to skin and map them to skin selectors using the ADF Faces skin editor.
4. **Identify global alias selectors to skin first**. Alias selectors are the 20% of work that color 80% of the components in an application.
5. **Define custom :alias definitions:** Custom :alias definitions simplify maintenance of style definitions used on multiple components. Custom :alias definitions are applied using the *–tr-rule-ref* selector,
6. **Generate images and icons**. Use the ADF Faces skin editor to generate custom colored versions of ADF Faces component images and icons.
7. **Edit the skin CSS file using the skin editor**. Use the skin editor to discover skin selectors for the components and behavior you want to skin and apply the changes using the Properties Inspector.
8. **Deploy the skin in an ADF library**. Skins can and should be reused. For this, create an ADF library deployment profile from the skin project and deploy the skin in a JAR file.

9. **Frequently test your custom skin in a browser**. Reference the skin JAR file from the ViewController project of the target application. Set the custom skin family name in the skin-family element of the target application `trinidad-config.xml` file.
10. **Deploy the skin as a shared library to WebLogic server**. Optionally, if a custom skin is shared across applications, deploy the skin ADF library JAR file as a shared library to Oracle WebLogic server and edit the applications' `weblogic-application.xml` file to reference it.

Stand-alone Skin Editor

The stand-alone skin editor allows you to build custom skins for projects build with Oracle JDeveloper 11.1.2, 11.1.1.5 and 11.1.1.4. If you build sins for JDeveloper versions prior to 11.1.1.4, just make sure you either don't extend an existing skin or extend one that exists in the target JDeveloper version. Note that the FusionFx simple skin is not available for all version of JDeveloper.

You could extend the "fusion" skin instead, which however is more difficult to skin as it isn't reduced to using global alias selectors as the simple skins do.

The work flow to follow for the stand alone editor is comparable to using the integrated editor

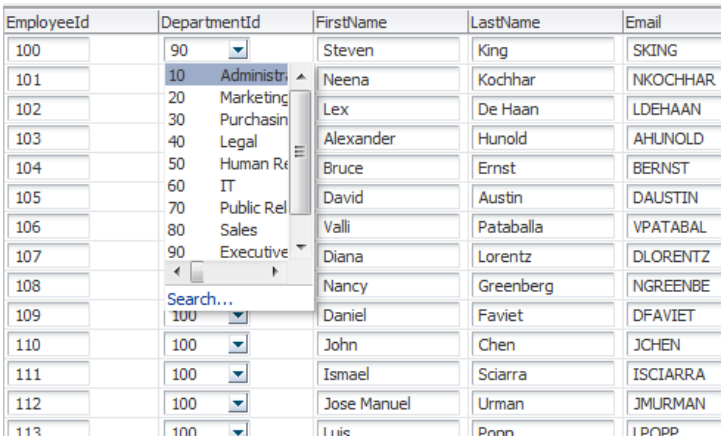
1. **Create a skin project**. Skins are reusable components and as such should be created in their own project or workspace. Never open a JDeveloper 11.1.1.x project directly in the stand alone skin editor as it is JDeveloper 11.1.2 based and thus will migrate the project to JDeveloper 11.1.2 and JSF 2.0.
2. **Create a new skin by extending an existing skin**. Oracle ADF Faces provides simplified default skins, like fusionFx-simple-v2.desktop, that are specifically designed to be extended by custom skins. When creating a new skin project, select the target version of Oracle JDeveloper you build the skin for, which then automatically updates the list of available default skins for this platform.
3. **Analyze the application for the components to skin**. Identify the components in a page that you want to skin and map them to skin selectors using the ADF Faces skin editor.
4. **Identify global alias selectors to skin first**. Alias selectors are the 20% of work that color 80% of the components in an application.
5. **Define custom :alias definitions:** Custom :alias definitions simplify maintenance of style definitions used on multiple components. Custom :alias definitions are applied using the *–tr-rule-ref* selector,
6. **Generate images and icons**. Use the ADF Faces skin editor to generate custom colored versions of ADF Faces component images and icons.
7. **Edit the skin CSS file using the skin editor**. Use the skin editor to discover skin selectors for the components and behavior you want to skin and apply the changes using the Properties Inspector.
8. **Deploy the skin in an ADF library**. Skins can and should be reused. For this, create an ADF library deployment profile from the skin project and deploy the skin in a JAR file.
9. **Frequently test your custom skin in a browser**. Reference the skin JAR file from the ViewController project of the target application. Set the custom skin family name in the skin-family element of the target application `trinidad-config.xml` file.

10. **Deploy the skin as a shared library to WebLogic server**. Optionally, if a custom skin is shared across applications, deploy the skin ADF library JAR file as a shared library to Oracle WebLogic server and edit the applications' `weblogic-application.xml` file to reference it.

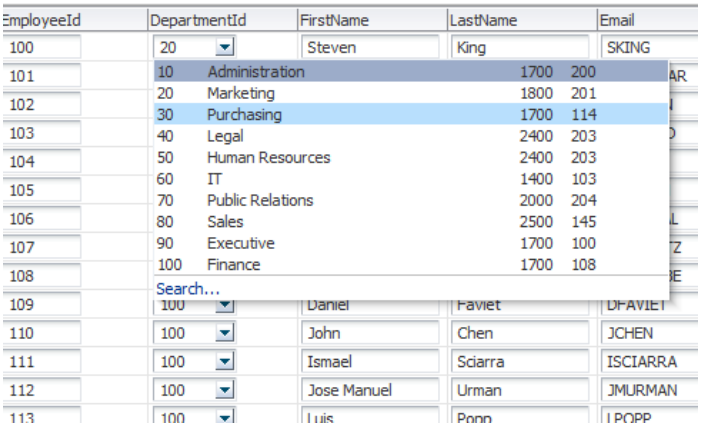
Note: In its November / December 2011 edition, Oracle Magazine publishes an article where I provide more information and hands-on related information for you to try.

Stretching af:inputComboboxListOfValues lists in a table

The default behavior of the `af:inputComboboxListOfValues` component select list is to take the size of the column it updates when added to a table. At runtime, the list then shows like in the image below



Note that in the image above, the `af:inputComboboxListOfValues` list is cropped to the size of the table column the component resides in. If the list content doesn't fit into this view, vertical and horizontal scrollbars are shown. Though this nicely aligns the list with the column, it is not liked by everyone, which is why you may want to stretch the list to its maximum size so it looks like in the image shown below



You achieve this stretching through 1) skinning and 2) an alternative option explained later in this section.

1) Skinning

http://download.oracle.com/docs/cd/E21764_01/web.1111/b31973/af_skin.htm#BAJFEFCJ

The steps are as follows:

1. Create trinidad-skins.xml file in the WEB-INF directory of the CiewController project's web folder (public-html). The trinidad-skins.xml configuration file is the skin registry in ADF Faces and is not there by default. Oracle JDeveloper 11.1.1.x (11g R1) releases you create this file manually. In JDeveloper 11.1.2 (11g R2) this file can be created for you using the integrated skin editor **(New | Web Tier | JSF | ADF skin)**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
  <skin>
    <id>your-skin-name.desktop</id>
    <family> your-skin-name </family>
    <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
    <style-sheet-name>
      css/skins/ your-skin-name.css
    </style-sheet-name>
      <extends>fusion.desktop</extends>
  </skin>
</skins>
```

2. Configure trinidad-config.xml file to reference the custom skin name, "your-skin-name" in the example above, in the <skin-family> name element. This is not required in Oracle JDeveloper 11.1.2 if you use the skin editor to create the skin.

3. Create a CSS file, a text file with the extension CSS to contain the skin definition. The location of the CSS file should be in a folder structure under public_html, e.g. css/skins in this sample. Note that this is done for you in JDeveloper 11.1.2 using the integrated skin editor.

Skins are used to change the look and feel of ADF Faces components but also to change the behavior of some of the components.

In this example, the `inputComboboxListOfValues` list should be stretched so all value are visible without horizontal scrolling. Add the following selector to the CSS file

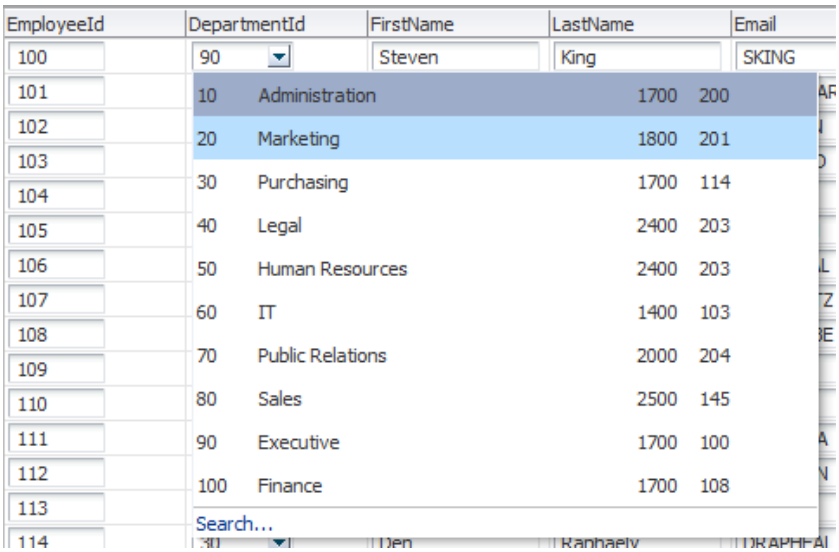
```
af|inputComboboxListOfValues{
  -tr-stretch-dropdown-table:true;
}
```

Note that you can also define how many values are displayed vertically before ascrollbar is shown. By default this value is set to 19, but it can be changed like shown below

```
af|inputComboboxListOfValues{
  -tr-stretch-dropdown-table:true;
  -tr-dropdown-number-of-rows: 30;
}
```

If in addition you want to increase the vertical space for each row in the list, e.g. from 16 px (default) to 25 px, you use

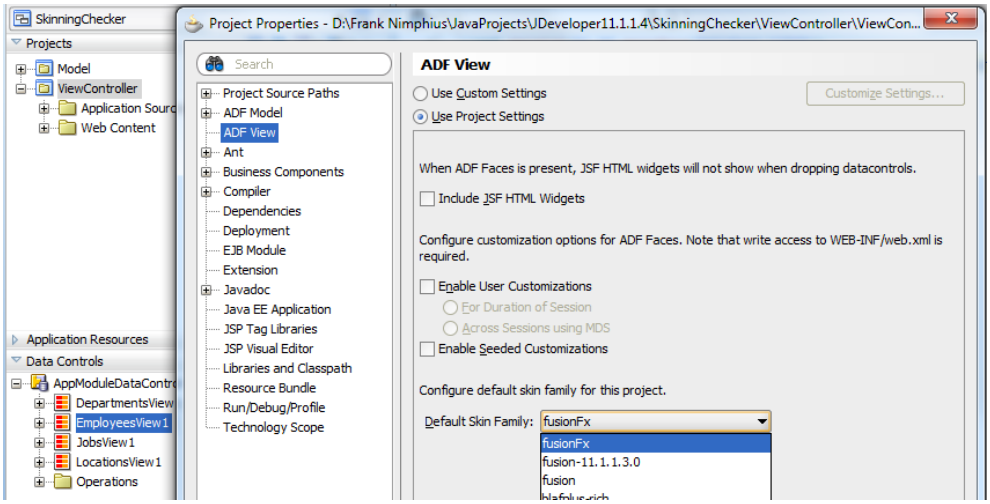
```
af|inputComboboxListOfValues{
  -tr-stretch-dropdown-table:true;
  -tr-dropdown-number-of-rows: 30;
  -tr-dropdown-row-height:25;
}
```



2) Alternative Solution:

Another option to show the `af|inputComboboxListOfValues` list extended is to use a default skin family, or extend from a default skin family, with an "FX" in the name. For example, JDeveloper 11.1.1.4 ha the "fusionFX" skin that contains the settings explained above (only the stretchable behavior)

To choose this string, double click the ViewController project to bring up its property dialog. Select the **ADF View** entry and in here select the "fusionFX" skin, or any other skin with "FX" in the name (later versions of JDeveloper 11g R1 have more choice available). This will configure the **fusionFX** skin in the skin-config.xml file.



Getting database connect information in ADF

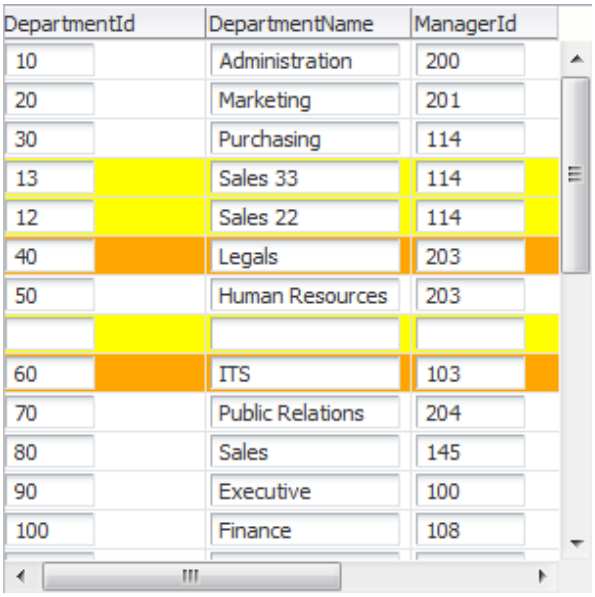
To get the database connect information of an ADF BC model in ADF, expose the Java method below in on the `ApplicationModule` Impl class.

```
public String getDatabaseInformation(){
  DBTransaction dbTransaction = (DBTransaction) this.getTransaction();
  //statement is only created, not send to the RDBMS
  PreparedStatement prepStatement =
      dbTransaction.createPreparedStatement("select * from dual", 0);
  try {
    String dbSchema =
           prepStatement.getConnection().getMetaData().getUserName();
    String connectURL =
                prepStatement.getConnection().getMetaData().getURL();
     //returns schema_name@host:port:sid
     return dbSchema + connectURL.substring(connectURL.indexOf("@")-1);
   } catch (SQLException e) {
       e.printStackTrace();
   }
     return null;
 }
```

Expose this method on the AM client interface if you need this to be accessible from the ADF client, fro example a method binding in a PageDef file or a managed bean.

Highlighting new and uncommitted data changes in a table

A common requirement is to visually indicate data rows in a table that have been changed but not yet committed by users. The ADF Unleashed blog, run by the Oracle JDeveloper and ADF QA team, has a clever solution for this, which I want to further explore here. The image below shows an ADF Faces table with new rows created by the user (yellow background), as well as rows that contain uncommitted data changes (orange background).



DepartmentId	DepartmentName	ManagerId
10	Administration	200
20	Marketing	201
30	Purchasing	114
13	Sales 33	114
12	Sales 22	114
40	Legals	203
50	Human Resources	203
60	ITS	103
70	Public Relations	204
80	Sales	145
90	Executive	100
100	Finance	108

You would think that the code behind this is complex, which it is not as this can be achieved with a single line of EL added to the **InlineStyle** property of the `af:column` component.

```
<af:table …>
  <af:column  headerText="#{bindings.DepartmentsView11.hints.DepartmentId.label}" …
    inlineStyle="#{row.row.entities[0].entityState == 0?'background-color:yellow;' :
                   row.row.entities[0].entityState == 2? 'background-color:orange' :' '}">
      <af:inputText value="…" … >
        …
      </af:inputText>
  </af:column
  <af:column headerText="#{bindings.DepartmentsView11.hints.DepartmentName.label}"
             inlineStyle="#{row.row.entities[0].entityState == 0?'background-color:yellow;' :
                            row.row.entities[0].entityState == 2? 'background-color:orange' :' '}">
      <af:inputText value="…" … >
        …
      </af:inputText>
```

```
  </af:column
 …
</table>
```

For the table to repaint upon row creation or data update, make sure you use **partialSubmit=true** on the command component that created the new row, or submitted the data change, and that you set the table **PartialTriggers** property to point to the command component Id value. If you use **autoSubmit="true"** to submit data changes, then set the PartialTriggers property of the table to point to this component's id.

How to use skinning instead of the InlineStyle property? If you use skinning for coloring your application, which also is what we recommend, you would need to change the code as shown next:

```
<af:column headerText="#{bindings.DepartmentsView11.hints.DepartmentName.label}"
                  sytleClass="#{row.row.entities[0].entityState == 0?'backgrYellow' :
                                row.row.entities[0].entityState == 2? 'backgrOrange' :''}">
        <af:inputText value="…" … >
            …
        </af:inputText>
 </af:column
```

The skin definition file (CSS) would then have an entry like this

```
---- START CSS file content ----
 .backgrYellow{background-color:yellow}
 .backgrOrange{background-color:orange}
---- END  CSS file content ----
```

Or, if, for better readability, you prefer keeping color settings with the components in the ADF skin file, you could use

```
---- START CSS file content ----
 af|column::data-cell.backgrYellow  {background-color:yellow}
 af|column::data-cell.backgrOrange{background-color:orange}
---- END  CSS file content ----
```

Reading UI component settings from a properties file

I never question a use case if it allows me to show new tricks in Oracle ADF – so for this one only focus on the solution to the problem reported on OTN.

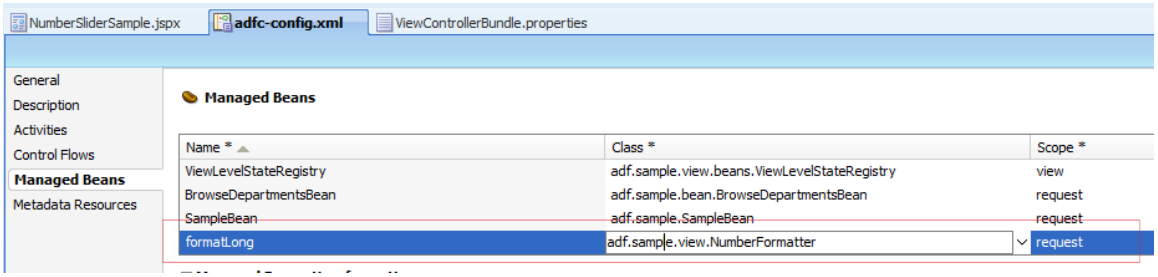
The question on OTN was if it is possible to define the **minimum** and **maximum** attribute values of an `af:inputNumberSlider` from a properties file or resource bundle. Properties files return strings,

while the input slider settings expect a numeric value. A solution to the problem is to write a managed bean that exposes setter/getter methods for the minimum and maximum attribute to reference using EL. The getter methods then could be used to lookup the resource bundle and read and convert the values for the slider to use. This however seems to be more code than needed and instead a generic managed bean could be used that just does the transformation for the two.

Managed beans can be created from `java.util.HashMap` or classes that extend `HashMap`. In the latter case, overriding the `HashMap get(key)` method allows you to take the key as an input parameter but to treat it as a string value that should be transformed into a numeric value.

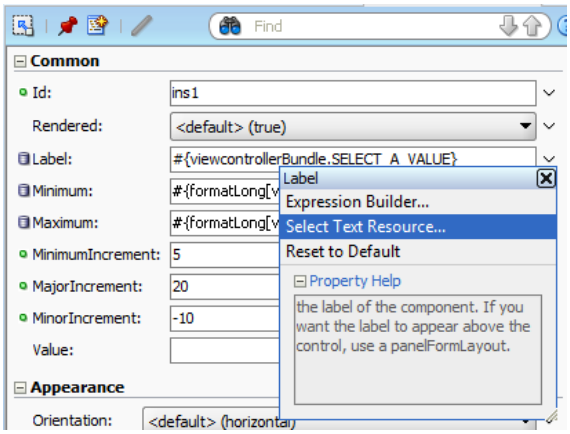
```
public class NumberFormatter extends HashMap {
 @SuppressWarnings("compatibility:7833863967783853944")
  private static final long serialVersionUID = 1L;
  @Override
  public Object get(Object key) {
   String strValue = (String) key;
    try {
     Long value = Long.parseLong(strValue);
      return value;
     } catch (NumberFormatException e) {
        e.printStackTrace();
     }
     return new Long(0);
   }
}
```

This class needs to be configured as a managed bean in a scope no longer than request (as it doesn't keep state and also is not used on all pages, which would justify a long scope to be used). You configure the bean in the task flow definition file that contains the page or page fragment referencing it.

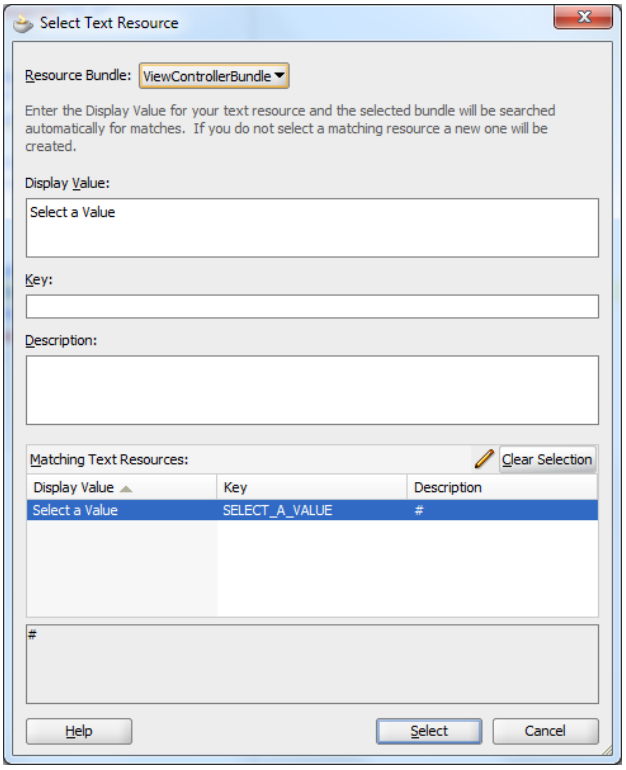


In this example, the managed bean name is chosen to be **formatLong**.

You create the properties file the easiest by selecting the **label** property of the input number slider component, unless you already have one created. Just select **Select Text Resource** from the context menu that opens when clicking the arrow icon next to the label attribute.



Create a key entry for the properties file as shown below:



This will automatically create a properties file if it doesn't exist yet. Open the properties file, which is located in the default package structure of the ViewController project, in JDeveloper and, for example, add the following keys:

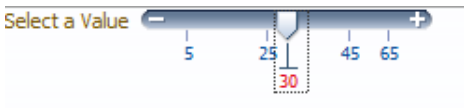
NumberSlider_MAX = 55

NumberSlider_MIN = 5

Save the properties file and reference these keys from the `af:inputNumberSlider` configuration on the page, as shown below:

```
<af:inputNumberSlider label="#{viewcontrollerBundle.SELECT_A_VALUE}" id="ins1"
      maximum="#{formatLong[viewcontrollerBundle.NumberSlider_MAX]}"
      minimum="#{formatLong[viewcontrollerBundle.NumberSlider_MIN]}"
      minimumIncrement="5" majorIncrement="20"
      minorIncrement="-10"/>
</af:form>
```

At runtime the slider shows the values specified in the resource bundle as shown in the image below.



Note the use of the **formatLong** managed bean reference, which, as an argument takes the resource properties access to the **MIN** and **MAX** keys defined in the resource bundle. The nice thing with this solution is that it is generic for numeric value conversion of `String` to `Long`.

Disabling the browser form auto-complete

As often, somebody's heaven is another one's hell. The browser auto-complete functionality is one example for this.

In Oracle ADF Faces, there is no property that switches auto complete-off for input field components or the `af:form` component. Thanks to the ADF Faces client side architecture switching off this browser functionality is easy to achieve:

```
<af:form>
    …
    <af:clientListener type="mouseOver"
                       method="suppressAutoComplete"/>
</af:form>
```

The mouse over event is issued one time when you enter a form. Given that you can only have a single form on a page, this means it fires one time for the page.

The JavaScript function referenced by the `af:clientListener` element is shown below

```
function suppressAutoComplete(evt){
   var domElement =
       AdfRichUIPeer.getDomContentElementForComponent(evt.getSource());
   domElement.setAttribute( "autocomplete", "off" );
 }
```

If you put this into a JS library that you reference from the `af:resource` tag then all you need to remember is add the `af:clientListener` tag to the `af:form` tag.

Optimized Groovy data access to view objects

In a recent ADF Insider Essentials recording about model driven LOV switching in Oracle ADF Business Components (see ADF BC Model Driven LOV Switcher recording) I used a Groovy string to populate a transient attribute with content from another view object. For this I exposed a public method on the lookup view object, which, though it works, may not be the only and probably not the best way to do it.

Grant Ronald from the PM team felt inspired to investigate in better practice for this part of the sample and came up with a good one for dependent view objects. Grant blogged about his findings under the title of "Using Groovy to read values from a different view object" at:

http://blogs.oracle.com/grantronald/entry/using_groovy_to_read_values

In the ADF Insider example, a view object for orders, *S_ORD*, accessed another view object holding values for payment types, *S_PAYMENT_TYPES*. Instead of using a public method, which may be okay if the view objects aren't related, Grant created and used an *Accessor* between the two view object, as in fact the *ID* column of the *S_PAYMENT_TYPES* table is referenced as a foreign key in the *S_ORD* table.

Grant explains his finding by the *EMPLOYEES* and *JOBS* table of the HR schema. Here's what Grant wrote and explained about his approach:

"Lets assume, that when you create a new employee their default role is AC_MGR (account manager) and their default salary should be the minimun salary for an accout manager. In this case, we'll actually read the minimum salary into the EmployeesVO.Salary field from the JobsVO.Min_Salary field.

1. *First create default ADF BC based on Employees and Jobs*
2. *In the EmployeesVO add a view accessor to JobsVO (and call it JobsView). This "links up" the EmployeesVO to the JobsVO*
3. *Set the default EmployeesVO.JobId to AC_MGR*
4. *Set the default EmployeesVO.Salary to JobsView.findByKey(key(JobId),1)[0].MinSalary*

So what does this Groovy expression do? It "walks" to the JobsVO using the JobsView accessor and then calls findByKey using the JobId. This returns an array and since we know JobId is unique we can just get the first entry's MinSalary."

In summary: to err' is human, not to learn from it is worse practice. We are not going to re-record the sample published on ADF Insider, but like you to consider Grant's solution for dependent view objects. Another solution could be to create a View Object that combines data from the *S_ORD* table and the *S_PAYMENT_TYPES* table, in which case no Groovy is required at all. However, it is interesting to see what Groovy can do for you in a single line of code.

ADF Faces web crawler support

Web crawler support, also known as search engine optimization (SEO), has been added to ADF Faces in Oracle JDeveloper 11.1.1.5 (PS4) and Oracle JDeveloper 11g R2 (11.1.2).

"Search engine optimization (SEO) is the process of improving the visibility of a website or a web page in search engines via the "natural" or un-paid ("organic" or "algorithmic") search results."

- Wikipedia, http://en.wikipedia.org/wiki/Search_engine_optimization

The difference between ADF Faces pages queried by a user and a web crawler is that the agent is different in that browsers know how to render and display dynamic JavaScript, whereas a crawler cannot.

To index a page, crawlers need static links pointing to wherever site navigation goes to next. In addition, the ADF Faces window and controller token prevented pages from being indexed successfully. This has been fixed for Oracle JDeveloper 11.1.1.5 and 11.1.2 The documentation for this new feature is in the Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework for the two releases:

http://st-doc.us.oracle.com/review/rsb/html/B31973_09/ad_output.htm#CHDEIGJB **(11.1.1.5)**

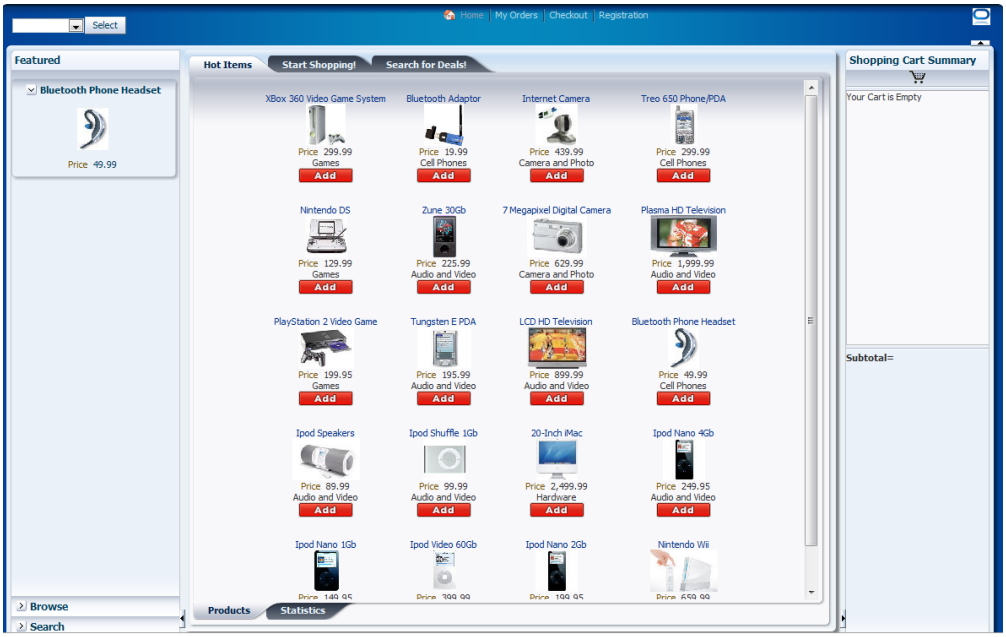
http://download.oracle.com/docs/cd/E16162_01/web.1112/e16181/ad_output.htm#CHDEIGJB **(11.1.2)**

As documented, you can also use EL to "tune" pages for web indexing. A documentated sample shows how to enrich an ADF Faces page with a "goLink" to indicate navigation to another site.

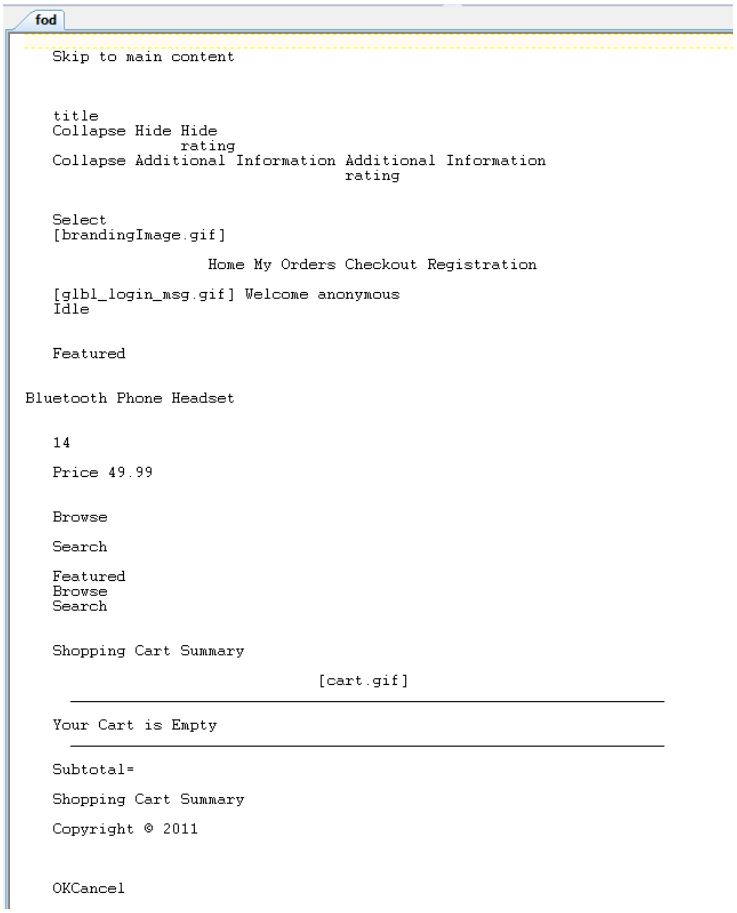
```
<c:if test="#{requestContext.agent.type ==  'webcrawler'}">
  <af:goLink text="This Link is rendered only for web crawlers"
                    destination="http://www.newPage.com"/>
</c:if>
```

Note that there is nothing developers need to configure or do for the search engine optimization to work.

The image below shows what browsers display when running the Oracle Fusion Order Demo in Oracle JDeveloper 11.1.2.



The image next is created from a screen print produced with the Lynx tool which is part of the Cygwin software you can download from http://cygwin.com/ . Configuring the Lynx user agent header property as "Mozilla/5.0 (compatible; Googlebot/2.1) " makes the tool simulating access through the Google web crawler (http://www.google.com/bot.html).



The content shown in the image above is not the only information web crawlers have available as they also have access to the response headers and the page markup, so they understand what links are. What they don't see is the JavaScript used and the CSS.

Note Though the Cygwin tool is good for testing, it does not replace a real test. If there is content shown in Cygwin Lynx that you don't expect or expect differently for your application, please verify otherwise first before reporting bugs. Keep in mind that ADF Faces supports search engines, not their simulation tools.

How-to hide or show components in printable pages

Not all web pages are suitable for printing, which means they contain components that either should not print, print different or should be replaces by other content. ADF Faces allows you to print pages using the `af:showPrintableBehavior` tag, which you add as a child to a command component.

http://download.oracle.com/docs/cd/E21764_01/apirefs.1111/e12419/tagdoc/af_showPrintablePageBehavior.html

For example, the image below shows a table with 10 columns of the Employees table of the Oracle HR schema.

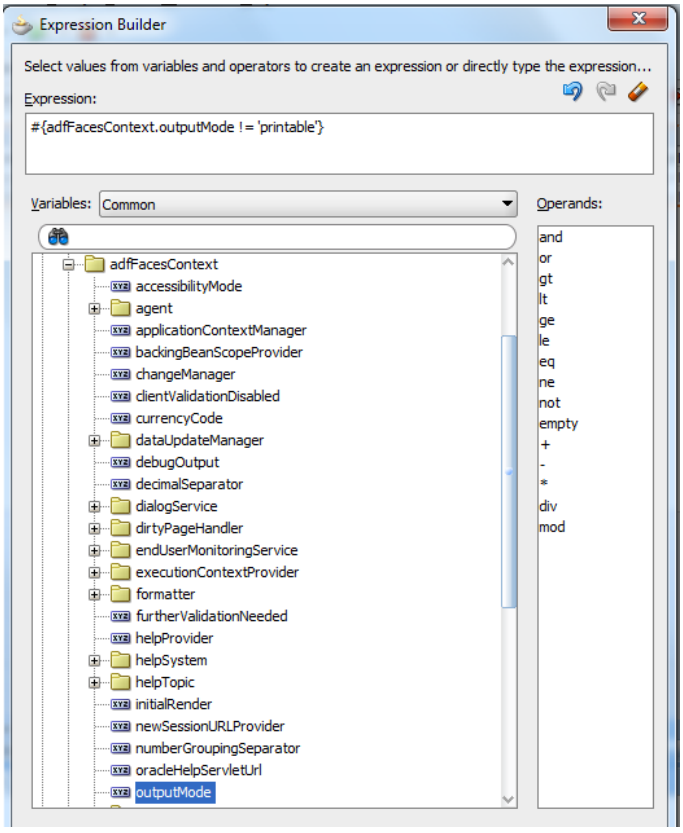
View ▾ Detach

DepartmentId	EmployeeId	FirstName	LastName	Email	PhoneNumber	HireDate	JobId	Salary	CommissionPct	ManagerId
90	100	Steven	King	SKING	515.123.4567	6/17/2003	AD_PRES	24000		
90	101	Neena	Kochhar	NKOCHHAR	515.123.4568	9/21/2005	AD_VP	17000		100
90	102	Lex	De Haan	LDEHAAN	515.123.4569	1/13/2001	AD_VP	17000		100
60	103	Alexander	Hunold	AHUNOLD	590.423.4567	1/3/2006	IT_PROG	10000		102
60	104	Bruce	Ernst	BERNST	590.423.4568	5/21/2007	IT_PROG	6000		103
60	105	David	Austin	DAUSTIN	590.423.4569	6/25/2005	IT_PROG	4800		103
60	106	Valli	Pataballa	VPATABAL	590.423.4560	2/5/2006	IT_PROG	4800		103
60	107	Diana	Lorentz	DLORENTZ	590.423.5567	2/7/2007	IT_PROG	4200		103
100	108	Nancy	Greenberg	NGREENBE	515.124.4569	8/17/2002	FI_MGR	12008		101
100	109	Daniel	Faviet	DFAVIET	515.124.4169	8/16/2002	FI_ACCOUNT	9000		108
100	110	John	Chen	JCHEN	515.124.4269	9/28/2005	FI_ACCOUNT	8200		108

The printable page however should only have 5 columns in the print, as shown in the image below:

DepartmentId	EmployeeId	FirstName	LastName	Email
90	100	Steven	King	SKING
90	101	Neena	Kochhar	NKOCHHAR
90	102	Lex	De Haan	LDEHAAN
60	103	Alexander	Hunold	AHUNOLD
60	104	Bruce	Ernst	BERNST
60	105	David	Austin	DAUSTIN
60	106	Valli	Pataballa	VPATABAL
60	107	Diana	Lorentz	DLORENTZ
100	108	Nancy	Greenberg	NGREENBE
100	109	Daniel	Faviet	DFAVIET
100	110	John	Chen	JCHEN

The ADF Faces context object exposes an **outputMode** property that you can use to determine whether or not a page is rendered in printable mode.



In the example, the following EL is used on the **Rendered** properties of the columns that should be hidden in printable pages:

```
<af:column sortProperty="PhoneNumber" sortable="false"
           headerText="…" id="c5"
           rendered="#{adfFacesContext.outputMode != 'printable'}">
          <af:outputText value="…" id="ot11"/>
</af:column>
```

Similar, components can be added to the printable page. An `af:switcher` component could be used to change complete sections of a page for the printable output.

See also:

http://download.oracle.com/docs/cd/E16162_01/web.1112/e16181/ad_output.htm#CHDIDBAB

Drag-and-drop: Getting Started

Drag and drop is a framework feature in ADF Faces that allows developers to identify components as a drag source and other components as a drop target. The framework then notifies a drop handler, or in the case of an attribute drag and drop use case, updates the attribute

For JDeveloper 11g R2, a tutorial helps you to get started with drag and drop. The tutorial explains the use cases in which a drag and drop operation updates the component input value and where the drop operation invokes a drop event that developers then listen for to perform more complex operations in response to the drag and drop action.

Developing drag and drop into ADF Faces UIs hasn't changed between R1 and R2 of Oracle JDeveloper 11*g*, so you can also try this with JDeveloper 11.1.1.x releases. If you use JDeveloper 11g R1 to walk through this tutorial, then you cannot use annotations in a POJO to mark it as a managed bean. In this case, and when using the ADF Controller task flow, you need to create the managed bean in faces-config.xml or the unbounded or bounded task flow definition.

http://download.oracle.com/docs/cd/E18941_01/tutorials/jdtut_11r2_41/jdtut_11r2_41.html

ADF Code Corner

OTN Harvest Spotlight

- Chris Muir

Chris Muir is an Oracle ACE Director, the founder of the ADF Enterprise Methodology Group (EMG)[1] and an ADF technical lead for SAGE Computing Services in Perth Western Australia.

"The ADF EMG is where ADF users get down and dirty in discussing their FMW experiences." – Chris Muir

ACC: **What is your current role?**

CM: ADF technical architect, consultant and trainer for SAGE Computing Services in Perth Western Australia

ACC: **What is your IT background?**

CM: Phew, let's try and summarize 16 years in computing: I started my career as a C++ coder so many years ago working on real-time SCADA based systems for trains.

Like a lot of IT development I learned a lot as a graduate out of university, but also like a lot of IT development the project was canned after 4 years for being too ambitious and complicated. A key lesson learned was the importance of getting developers to talk to the real people who will use the system, rather than working from some abstract and outdated requirements & design documents.

From there I moved into Oracle development having touched upon it as university. Over a number of years I used all of the traditional Oracle development tools, SQL, PLSQL, Oracle Forms & Reports, Oracle Discoverer, Oracle Designer and even 100% generation (and have the scars to prove it).

In the last decade I've been privileged enough to work for SAGE Computing Services, a small and passionate set of Oracle developers under Oracle ACE Penny Cookson. Over the last 5 or so years, I've had the chance to jump from traditional Oracle development to Java, JDeveloper and ADF, returning to my C++ inspired roots. I started using Oracle JDeveloper 9.0.4 and have used every version since.

[1] http://groups.google.com/group/adf-methodology/about

ACC:	**How do you currently use Oracle JDeveloper and ADF?**
CM:	My current role is split between 3 main tasks, that as an ADF technical lead for an 11g client, as a consultant to other organizations looking to adopt ADF, and as a trainer teaching week long ADF courses. It's a nice mix, I enjoy teaching and introducing beginners to the power of ADF, but I also get the chance to get my hands dirty in writing real solutions using the tool too. As is often said, the best teachers are also practitioners.
ACC:	**So far, what has been your biggest challenge in building Java EE application with Oracle ADF?**
CM:	I must admit when I started out with Oracle JDeveloper several years ago, the real struggle was trying to understand Struts and UIX, the technologies of choice in JDeveloper at the time (up to JDeveloper 10.1.2). Luckily Oracle had seen and contributed to JavaServer Faces, and the introduction of JSF in Oracle JDeveloper 10.1.3 was a watershed moment for me, because JEE development became hugely easier to understand and develop with. Since that time as Oracle has extended and augmented JSF with their own feature set, particularly with the introduction of ADF Faces RC and the extended controller with Task Flows in 11g, the product is just going from strength to strength in providing a rich and productive development environment.
ACC:	**Which feature of ADF was the greatest benefit to your project?**
CM:	For our current client's project it's a combination of two technologies together. The ADF UI Shell (aka. Dynamic Tabs Template) has allowed us, with the combination of Task Flows and their multi-transaction-per-session support, to build a sophisticated application for our current users. The web world is mostly simplistic in the features it supplies users, but for business applications and their educated and experienced users this isn't enough. They need a rich UI and the support for getting multiple things done at once, which the UI Shell & Task Flows have provided us.
ACC:	**Away from the on line help, what have been your most valuable sources of ADF knowledge?**
CM:	When I started out with JDeveloper, the OTN forum for Oracle JDeveloper was invaluable for finding solutions to my problems. Oracle Product Managers have shown great dedication to supporting and posting to the forum over the years. In the mid-term the detailed blogs of other ADF experts have been a great source of information and still to this day I carefully scan every ADF blog I know about for new and interesting information. In recent years one of the biggest problems is there's nearly too much information available now, blogs, books, tweets, Oracle manuals **and the ADF EMG**, it is becoming hard to filter it all. However I'm ever grateful to everyone out there who chooses to share their experiences and expertise on the internet for others to benefit, and through my blog I hope I contribute the same in return. (Yeah yeah, I didn't mention the ADF EMG.... more on that soon).
ACC:	**Are you in any way actively involved in the ADF Community?**

CM: My contributions to the ADF Community take three main forms:

Firstly I enjoy presenting at various user group events, and especially enjoy presenting on ADF to anyone whose ear I can bend.

Secondly I blog about ADF and other Oracle related issues, though this doesn't get as much time now as I like. I highly recommend anyone who has ambitions of a career in IT to present and blog as often as they can.

Thirdly I contribute and moderate the ADF Enterprise Methodology Group (ADF EMG), a group run and answered by some of the best ADF experts around the world in order to help the wider ADF community about concepts of ADF best practices, methodologies, deployment and other concepts beyond the how-do-I-get-this-to-work type questions on the OTN Forums (the ADF EMG doesn't compete with the OTN Forums but supplements it).

There are other activities I participate in as an Oracle ACE Director, helping organize user group conference streams, asking people to present and write papers, but that's less visible though not to be dismissed in the amount of volunteer time freely given.

ACC: **You are the founder of the ADF Enterprise Methodology (EMG) group. What has been the motivation for this and what is the long term vision you have for this group to become?**

CM: The idea for ADF EMG sprung up because of 1 customer. After introducing and teaching a skilled set of developers, I was surprised on returning 6 months later to see the organization had made little headway. With some questions I realized it wasn't the developers or ADF at issue, but rather the business was having a lot of trouble adjusting to the new way of doing things. As an Oracle Forms shop, everything, how they wrote requirements, how they implemented change control, their user expectations, their management expectations and more was still all based on their previous technologies.

This made me realize adopting new technologies is more than using the technology at hand, a cultural shift is required, the business of writing software needs to change, out with the old, in with the new so to speak. But how to do that, I'm certainly just 1 guy with some limited experience? So I thought it would be a valuable exercise to create an online group where many experienced ADF experts could share their views on how to get ADF adopted into an organization. From there some very kind and generous volunteers said they'd help out, we formed the online group, and the rest is history as they say. While I might be the founder, the key contributions are from the members and other volunteers today.

After 3-4 years the ADF EMG now sits on just under 700 members worldwide. We're holding a whole day of ADF EMG presentations at this year's Oracle Open World (http://one-size-doesnt-fit-all.blogspot.com/2011/07/year-of-adf-developer-at-oracle-open.html) and we even have several other EMG groups including OBIEE, SOA-BMP and WebCenter.

The group grows from strength to strength each year and we encourage new members to join and participate. Really, your career will benefit from it.

ACC:	**Are you attending OOW 2011 in San Francisco? If so, what are your planned activities there and how could people meet you just to say 'hello'?**
CM:	As an Oracle ACE Director I'm privileged to be invited to attend and present at Oracle Open World. Mind you as conferences go, each OOW year gets more and more hectic and this year will be no exception. What with organizing the ADF EMG presenters, presenting myself, rushing around to sessions I want to see, I'm sure I'll be sleeping hard and long on the flight home to Perth. Yet the best bit of OpenWorld has never been the presentations, but meeting and talking to like minded developers, so I highly encourage attendees to come up and say hello. You might have to put up with my horrible Aussie accent though.
ACC:	**ADF Genie grants you a wish, what would you ask for?**
CM:	Ha, that's one of our favorite questions from the ADF EMG new member survey. I guess my reply will have to be in line with a prominent Oracle Product Manager's answer when he signed up: "Don't tempt me".
ACC:	**Thanks Chris!**

RELATED DOCOMENTATION

☒	ADF EMG Group http://groups.google.com/group/adf-methodology/about
☒	ADF Insider Essentials http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adfinsider-093342.html#a3
☒	