

ADF Code Corner

Oracle JDeveloper OTN Harvest 06 / 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-June-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>*

June2011 Issue – Table of Contents

ADF Rule of Thumb.....	4
Reserved port number impact on ADF Faces.....	4
Naming conflict when using declarative components in JSF 2.0.....	4
Five hours of Task Flow Overview Recordings Available	4
Downloading SRDemo	5
ADF Enterprise Application Development – Made Simple	6
How-to execute an ADF method action from Java.....	7
JDeveloper and WLS certification matrix	8
JDeveloper 11.1.2 browser certification.....	9
Where to find advanced ADF documentation	9
How-to notify the server when user tabs out of input field.....	9
Initializing queries when entering page fragments in a BTF	11
Downgrading from Flash in JDeveloper 11g R2	19
What if Oracle ADF Skin Editor doesn't start on Windows	20
JDeveloper 11.1.2 deployment to WLS	21
How-to learn more about JDeveloper 11.1.2	21
How-to change the WS connection for a WS DC.....	21
Getting selected row in inputListOfValues returnPopupListener	22
JDeveloper 11.1.2: Link in Table Column Work Around	25

Solving the problem of inactive pages after file download	26
Building master-detail tables with af:panelCollections	27
Declarative Components with ADF bindings?	28
Performing Logging in Oracle ADF	28

ADF Rule of Thumb

To avoid trouble when working with ADF, follow this simple rule of thumb:

Always work through the binding layer!

Reserved port number impact on ADF Faces

Interesting what impact the use of reserved port number can have to running ADF Faces pages. The ADF Faces release notes for Oracle JDeveloper 11.1.2 state the following

"Ensure that you avoid reserved port numbers when deploying your application. Some application servers will fail to prevent you from deploying your application on a reserved server port (e.g. 1, 7, 9, 11, 13, 15, 17, 19, 20, 21, 22, 23, 25, 37, 42, 43, 53, 77, 79, 87, 95, 101, 102, 103, 104, 109, 110, 111, 113, 115, 117, 119, 123, 135, 139, 143, 179, 389, 465, 512, 513, 514, 515, 526, 530, 531, 532, 540, 556, 563, 587, 601, 636, 993, 995, 2049, 3659, 4045, 6000, 6665, 6666, 6667, 6668, 6669).

If you use one of these reserved ports, some web browsers may honor it but at least WebKit-based browsers (e.g. Safari and Google Chrome) will not. You may see some page content but all resources (images and styles) will fail to load."

Wondering how many "ADF Faces doesn't run on my laptop" reports on OTN had their source in this use of reserved port numbers.

Naming conflict when using declarative components in JSF 2.0

As of JavaServer Faces 2.0 **component** is a reserved word and is used as an implicit expression to reference the current UI component. The **component** word also is used as the default value of the **componentVar** variable in ADF Faces declarative components. To use declarative components and templates that have the **componentVar** property defined as **component**, you need to refactor your templates and declarative components and change the value of the **componentVar** property from **component** to something else. Sorry for the inconvenience, but who knew some years ago that JSF 2.0 was going to claim this string as a reserved word.

Five hours of Task Flow Overview Recordings Available

Do you **want to become an ADF Controller Expert?** Check this out! In addition to the ADF Controller documentation in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework 11g Release 1...*

http://download.oracle.com/docs/cd/E21764_01/web.1111/b31974/partpage3.htm#BABHIIAI

... the **ADF Insider** website ...

<http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adfinsider-093342.html>

... hosts five online videos that explain how to build and work with ADF Controller task flows in Oracle ADF.

ADF Task Flow - Overview (Part 1)

This 90 minute recording introduces the concept of ADF unbounded and bounded task flows, as well as other ADF Controller features. The session starts with an overview of unbounded task flows, bounded task flows and the different activities that exist for developers to build complex application flows.

Exception handling and the Train navigation model is also covered in this first part of a two part series. By example of developing a sample application, the recording guides viewers through building unbounded and bounded task flows. This session is continued in a second part.

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/taskflow-overview-p1/taskflow-overview-p1.html

ADF Task Flow - Overview (Part 2)

This 75 minute session continues where part 1 ended and completes the sample application that guides viewers through different aspects of unbounded and bounded task flow development. In this recording, memory scopes, save for later, task flow opening in dialogs and remote task flow calls are explained and demonstrated. If you are new to ADF Task Flow, then it is recommended to first watch part 1 of this series to be able to follow the explanation guided by the sample application.

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/taskflow-overview-p2/taskflow-overview-p2.html

ADF Region Interaction - An Overview

This session covers most of the options that exist for communicating between regions. It briefly discusses what it takes to build regions from bounded task flows before going into details using slides and samples. The following interaction is explained: contextual events, queue action in region, input parameters and PPR, drag and drop, shared Data Controls, parent action and region navigation listener.

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/adf-region-interaction/adf-region-interaction.html

ADF Region Interaction - Contextual Events

Contextual event is used as a communication channel between a parent view and its contained regions, as well as between regions. By example, this session explains how to set up contextual events, how to define producers and event listeners and how to define the payload message.

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/AdfInsiderContextualEvents/AdfInsiderContextualEvents.html

Downloading SRDemo

SRDemo is the reference sample shipped for ADF in Oracle JDeveloper 10.1.3.x. Use the Help | Check for Update menu option in Oracle JDeveloper to download and setup SRDemo versions for EJB and ADF BC models.

ADF Enterprise Application Development – Made Simple

Sten E. Vesterli wrote the "Oracle ADF Enterprise Application Development – Made Simple" book published by Packt Publishing in 2011

<http://www.packtpub.com/oracle-adf-enterprise-application-development/book>

A common question on OTN, but also when talking to clients or customers is about where and how to start your ADF application development. Especially when the current programming background is not in Java, but 4 GL or PLSQL, developers often look for answers to the following questions:

- How long does it take to learn Oracle ADF ?
- How long does it take to replace a Forms application with ADF ?
- How many developers do I need?
- Do I need to know Java to use ADF and if yes, how good do I need to know this?
- How do I structure my programming files, organizing them in JDeveloper work spaces, projects and libraries?
- What is best practices for naming Java packages and how to void naming conflicts in ADF in general?
- How many Application Modules do I need or should I create?
- How to test applications?

Sten Vesterli answers all of the above questions and more in his book, which makes a great addition to the three existing Oracle ADF books.

In order of complexity (which also is the order in which reading the available Oracle ADF books makes sense from my perspective) Sten's book should come second (though it also is useful to those that are already more advanced with Oracle ADF).

So if you are absolutely new to Oracle ADF, then the order of books to read to get you to an expert level is:

1. Grant Ronald; **"Quick Start Guide to Oracle Fusion Development: Oracle JDeveloper and Oracle ADF"** (McGraw Hill 2010)
2. Sten Vesterli; **"Oracle ADF Enterprise Application Development – Made Simple"** (Packt Publishing 2011)
3. Duncan Mills, Peter Koletzke; **" Oracle JDeveloper 11g Handbook: A Guide to Fusion Web Development"** (McGraw Hill 2009)
4. Frank Nimphius, Lynn Munsinger; **" Oracle Fusion Developer Guide: Building Rich Internet Applications with Oracle ADF Business Components and Oracle ADF Faces"** (McGraw Hill 2010)

If you are not new to Oracle ADF and Oracle JDeveloper, then buy Sten Vesterli's book anyway. It is worth it and you want to have it on your book shelf. See below the table of content to get a better idea of what this book covers:

- Chapter 1: The ADF Proof of Concept
- Chapter 2: Estimating the Effort
- Chapter 3: Getting Organized
- Chapter 4: Productive Teamwork
- Chapter 5: Prepare to Build
- Chapter 6: Building the Enterprise Application
- Chapter 7: Testing your Application
- Chapter 8: Look and Feel
- Chapter 9: Customizing the Functionality
- Chapter 10: Securing your ADF Application
- Chapter 11: Package and Deliver
- Appendix: Internationalization

The book is written with a lot of good humor, which makes the read very joyful (from a geek perspective, of course). My favorite quote – just in case you are interested - is from page 97, when Sten talks about getting organized:

" Stop sending e-mails to your team. Just stop it. E-mail is so last century...."

So true, so true! This quote's runner up was the "boss key" on page 128 where Sten talks about productivity and how Oracle Team Productivity Center (TPC) can help you with this. Quotes like these stick in your brains and make sure you never forget. So – pay for the book once, remember it always and anywhere.

How-to execute an ADF method action from Java

ADF operations or method bindings are accessed through Expression Language from a command component in ADF Faces.

For example, dragging the Commit operation from the ADF Data Controls panel onto a page, and rendering it as a command button, configures the button's **ActionListener** property as shown below

```
#{bindings.Commit.execute}
```

To override the **ActionListener** configuration with a managed bean reference that preserves the default call to the ADF binding method or operation, you use the `OperationBinding` class as shown below.

```
public BindingContainer getBindings() {  
    //access the ADF binding container  
    return BindingContext.getCurrent().getCurrentBindingsEntry();  
}  
  
public String commitAction() {  
    BindingContainer bindings = getBindings();
```

```
//Action and method bindings are accessed through the
//OperationBinding class
OperationBinding operationBinding =
    bindings.getOperationBinding("Commit");

//Execute the operation binding and check for errors to show a user
//message if required
Object result = operationBinding.execute();
if (!operationBinding.getErrors().isEmpty()) {
    return null;
}
return null;
}
```

The good news to this is that you don't need to write any code yourself. Just double click an ADF bound command item (like button), which brings up a dialog to create or select a managed bean and to create an action method to be invoked by the command item.

A checkbox, which is selected by default, ensures that if the command item is ADF bound, the required ADF binding code to preserve the default functionality – as shown above – is generated. Note that the above code is referenced from the button's **Action** property and not from the **ActionListener** property, which is the way Oracle JDeveloper implements it. To use the **ActionListener** property, you change the method signature to

```
public void commitAction(ActionEvent actionEvent){
    //same method code as above goes here without the "return null"
    //line's. Use "return" only
}
```

JDeveloper and WLS certification matrix

Migrating ADF applications from one release to a next always is a source for confusion when it comes to which server the upgraded application needs to be deployed to. A general rule of thumb is that migrated applications need to be deployed to the server that matches the JDeveloper patch set. For stand-alone WLS server domains that are not extended for ADF you need to install the ADF runtime libraries (JRF), which again needs to be chosen according to the WLS version and the JDeveloper Patch Set. The table below lists the different versions for the current production systems.

JDev/ADF Version	WLS Version	JRF Version
11.1.1.2	10.3.2	11.1.1.2
11.1.1.3	10.3.3	11.1.1.3
11.1.1.4	10.3.4	11.1.1.4
11.1.1.5	10.3.5	11.1.1.5
11.1.2	10.3.5	11.1.1.5 + 11.1.2 patch (number 12611176)

JDeveloper 11.1.2 browser certification

The Oracle ADF Faces rich client components are certified or supported with the following browsers. For the ADF Faces components that render in Flash, only Flash 10 is certified.

Browser	ADF (ADF Faces)
Firefox 3.0, 3.5+, 4.0	Certified
Internet Explorer 7, 8, 9	Certified
Safari 3.2, 4.0, 5.0	Certified
Safari 3.0 (mobile)	Supported
Chrome 1.0+	Certified

Where to find advanced ADF documentation

How good can documentation be? Well, it doesn't matter because it will **never be good enough** to meet the requirements of all developers working with a tool or product. I think that the documentation that is available for Oracle ADF – including the various books written – is extraordinarily good, but also agree that we can do better (and there is no day we go to work without wanting to do better).

Doing better however takes time (sit down yourself and measure the time it takes to write a page that is in print quality) and probably will still not meet all documentation requirements. But here is where the ADF blogger community comes to the rescue in that they document solutions to not-so-generic problems or challenges.

The Oracle JDeveloper team (thanks to Juan Ruiz) aggregates these blogs on the Cannotea website: <http://www.connotea.org/user/jdeveloper>.

A nice entry point to search this page and other collateral is exposed on the JDeveloper website on OTN: <http://www.oracle.com/technetwork/developer-tools/jdev/index-087798.html>

So if you need advanced documentation that goes beyond of what is in the product's ADF developer guides (approx. 4000 pages all in all) then this is a starting point.

For whatever you don't find, post it to: <http://forums.oracle.com/forums/forum.jspa?forumID=83> and you find a note about it later in a blog or here on the OTN Harvest site.

How-to notify the server when user tabs out of input field

Though the use case covered in this section is more of an individual problem than of generic interest, the code involved to implement the solution may be useful. First the use case: For an input text component, a notification should be sent to the server when the user steps out of the field using mouse or the keyboard navigation. The notification should be sent no matter if users changed the field content (in which case a value change listener would have done) or not (in which case a value change listener is not enough). The solution to this problem is to use an `af:clientListener` tag on the input tag component to invoke a JavaScript function when the user leaves the field (The component event for this is "blur").

Using an `af:serverListener` tag on the input text component then allows to notify the server from the JavaScript function.

```
<f:view>
  <af:document id="d1">
    <af:resource type="javascript">
      function onBlurTxtField(evt) {
        var source = evt.getSource();
        var custEvent = new AdfCustomEvent(source, "onBlurNotifyServer",
          {submittedValue : source.getSubmittedValue(),
            localValue : source.getValue()}
          , true);
        custEvent.queue();
      }
    </af:resource>
    <af:form id="f1">
      <af:inputText label="Label 1" id="it1">
        <af:clientListener method="onBlurTxtField" type="blur"/>
        <af:serverListener type="onBlurNotifyServer"
          method="#{TxtHelper.onBlurNotify}"/>
      </af:inputText>
      <af:inputText label="Label 2" id="it2"/>
    </af:form>
  </af:document>
</f:view>
```

The page code above sends a custom event to the server whenever a user leaves the input text field. The payload sent to the server contains the submitted and the local component value. Note that in the above sample – as we will see next – the payload is not really required because the same information can be accessed from the component instance in the referenced managed bean method.

```
<af:serverListener type="onBlurNotifyServer"
  method="#{TxtHelper.onBlurNotify}"/>
```

The managed bean method receives a single argument of type `ClientEvent`:

```
public void onBlurNotify(ClientEvent clientEvent) {
  // get a hold of the input text component
  RichInputText inputTxt = (RichInputText) clientEvent.getComponent();
  //do some work on it here (e.g. manipulating its readOnly state)
  //...
  //Get access to the payload

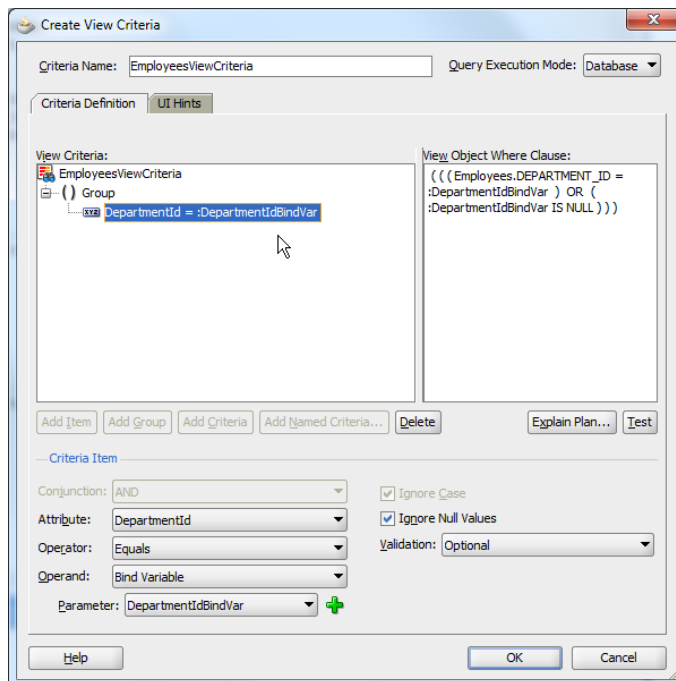
  Map parameters = clientEvent.getParameters();
  System.out.println("SubmittedValue =
                    "+parameters.get("submittedValue"));
  System.out.println("LocalValue = "+parameters.get("localValue"));
}
```

As mentioned, the use case may not be of generic interest. However, the way you work with the client and server listener for sure made sense to publish as an example.

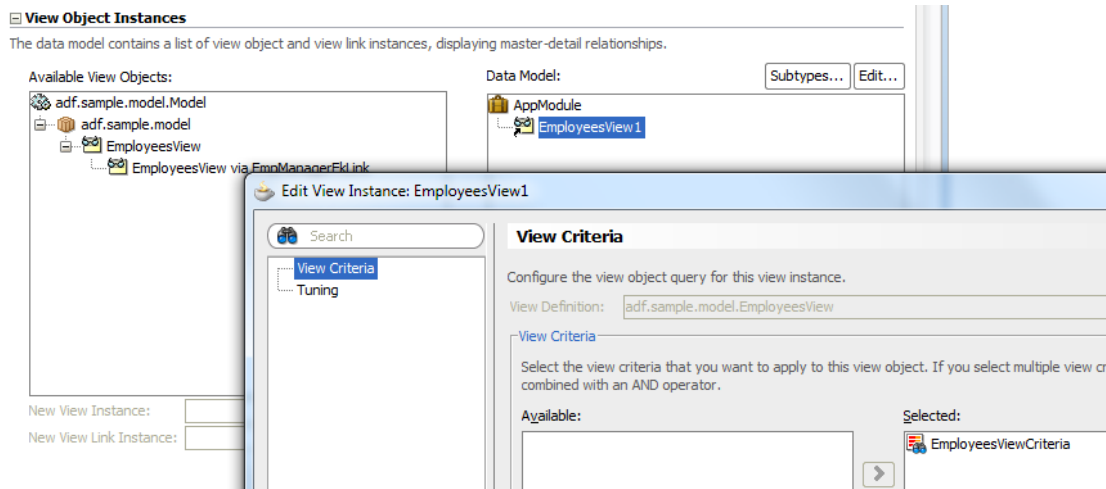
Initializing queries when entering page fragments in a BTF

A frequent question on the Oracle JDeveloper forum on OTN is about executing logic on load of a page fragment.

A typical use case is to filter a query with data read from a parameter or memory attribute. Using ADF Business Components the implementation of this use case can be completely declarative, as shown in the following.



Starting from an ADF Business Component model that consist of a View Object for the Employees table in the Oracle HR sample schema, a View Criteria is defined with a bind variable to filter employees by the id of the department they are in. The View Criteria shown above queries all employees if the bind variable value is null, but filters the result set if a value us provided. Bind variables can be created by pressing the green plus icon next to the **Parameter** field in the **Create View Criteria** dialog.

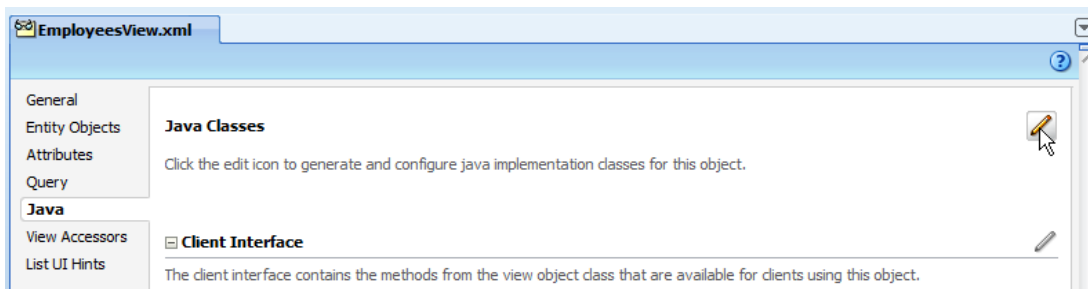


The View Criteria is then applied to a View Object instance either statically from the Application Module editor, or dynamically from Java at runtime.

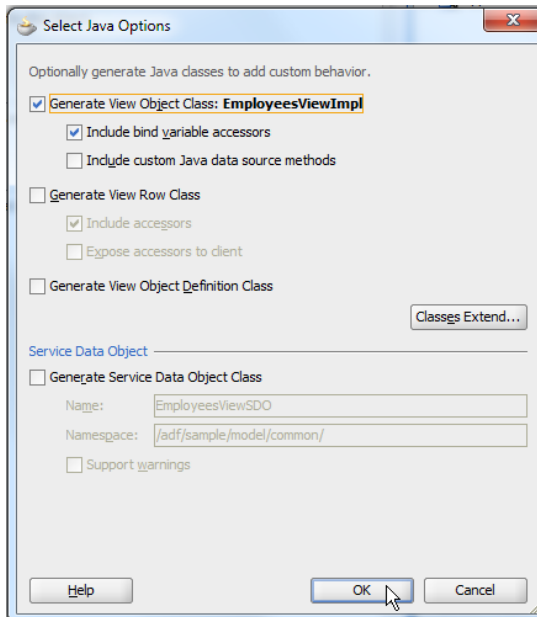
To statically apply the View Criteria, the Application Module in the Application Navigator needs to be clicked twice for the edit dialog to open. The View Object instances exposed in the Application Module are shown in the **Data Model** category.

In this example, the **EmployeesView1** instance is selected and the **Edit** option chosen from the context menu to assign the View Criteria. Selecting the View Criteria in the **Available** list and shuffling it to the **Selected** list then attaches it to the View Object instance. All queries issued by the **EmployeesView1** View Object instance now include the View Criteria.

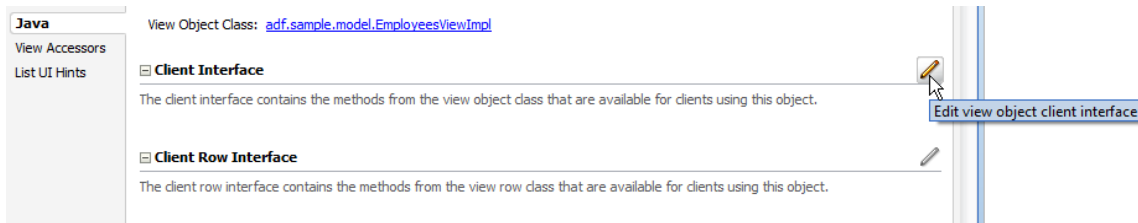
The View Object editor also has a **Java** category that is used to generate an implementation class for a View Object, which extends the generic ADF Business Component View Object base class. Creating a custom View Object implementation class allows developers to add custom methods and business logic to a View Object. In this example it is used to expose setter and getter methods for the bind variable defined for the View Criteria.



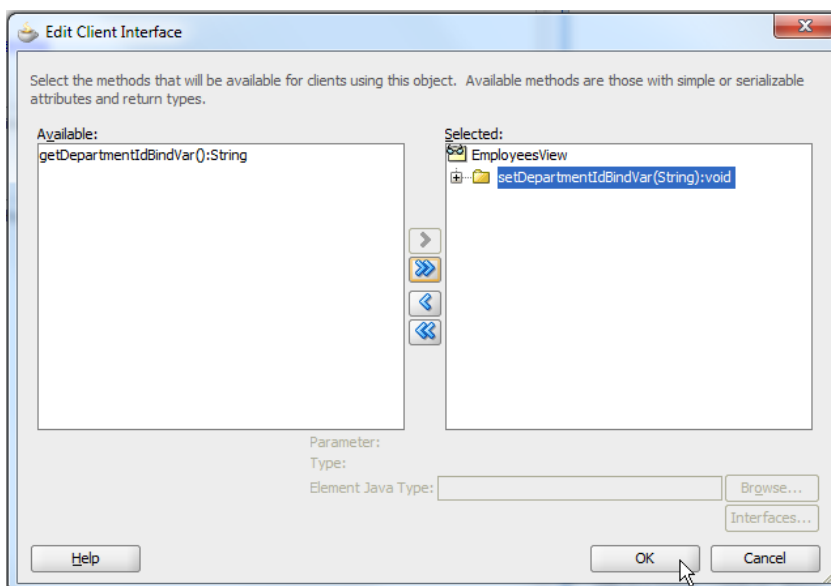
As shown in the image below, getter and setter methods for bind variables can be automatically generated when creating the View Object implementation class. The setter method of the bind variable defined for the Employees View Criteria then is exposed on the Data Control panel by configuring it on the ADF BC client interface.



The **Client Interfaces** panel too is accessible from the **Java** category in the View Object editor as shown in the image below.

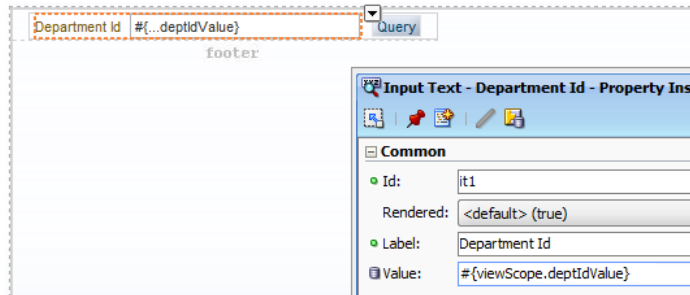


All public methods that are contained in the View Object implementation class are listed on the left side of the opened dialog. Selecting the setter method of the bind variable and OK'ing the dialog exposes the method in the Data Control panel for declarative invocation.



The sample page uses an `af:inputText` field to simulate dynamic data used to filter the employee table in the page fragment. A command button is added to submit the parent form. The command button **partialSubmit** property is set to "true" to avoid a full page refresh

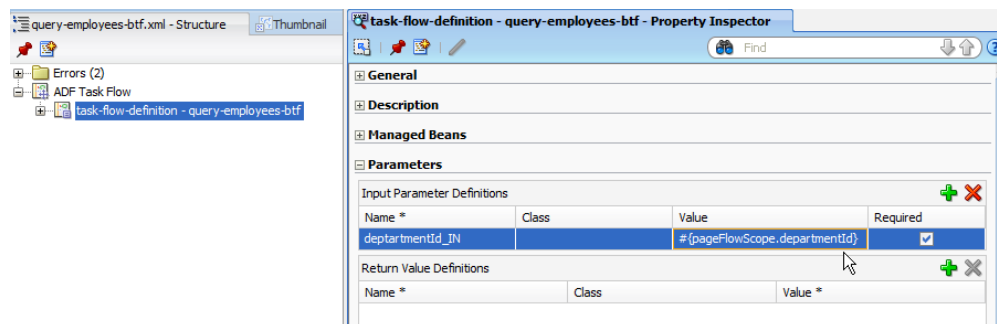
(Yes, well spotted. It's a search form in which the search form resides in a parent page whereas the result set displays in a bounded task flow exposed as a Region. SO this sample could also hold as an example for parent to region communication)



Note that the input text field **value** property points to a memory attribute in **viewScope**. This is because the input value needs to survive the duration of a single request. If the search field was ADF bound then no memory scope is needed.

The Employees View Object is added to a bounded task flow (note that the use case is to initialize the query within a bounded task flow using page fragments). The bounded task flow has an input parameter defined, **departmentId_IN**, that writes the department Id value that is passed from the parent page into an memory attribute within the task flow's own **pageFlowScope**.

Note: For demos it is good to write parameter values into a memory scope attribute. In praxis, using a managed bean is a better choice because it allows documentation (JavaDocs), type save input, and the discovery of properties (Expression Builder). So if striving for best practices when working with ADF, using a managed bean to hold input parameter data is one of them.

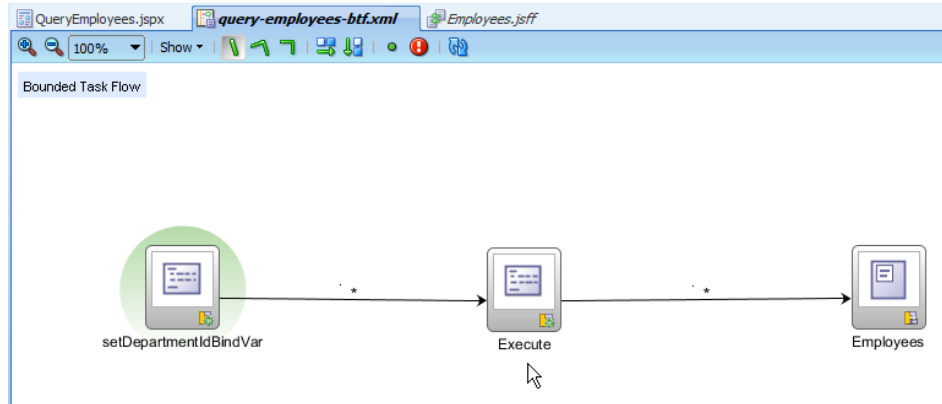


To implement the use case, two things need to happen next next

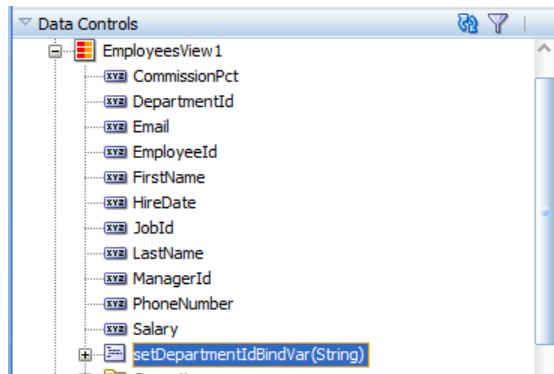
1. A page fragment needs to be created that shows the **EmployeesView1** collection in a read only table. This is a pure drag on drop action: first by dragging the view activity from the Component Palette, and then dragging the View Object onto the opened page fragment. The page fragment is created by JDeveloper in response to a double click onto the view activity.
2. A method activity needs to be created that reads the department Id input parameter from the task flow memory attribute to set it on the ADF Business Component model. Another method

activity then executes the query before navigating to the page fragment displaying the employees view

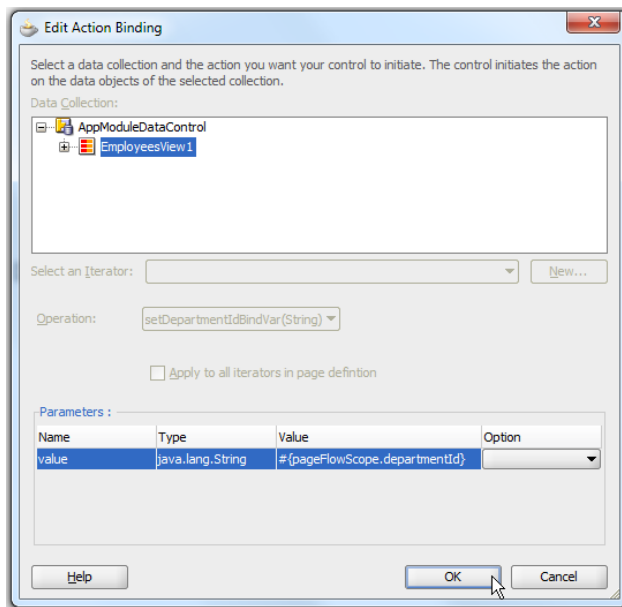
The bounded task flow diagram is shown below. Note the default activity being a method activity.



The **setDepartmentIdBindVar** method was dragged from the Data Controls Panel and added as the first (which then automatically becomes the default activity) to the bounded task flow diagram. Its responsibility is to set the bind variable in the business service.

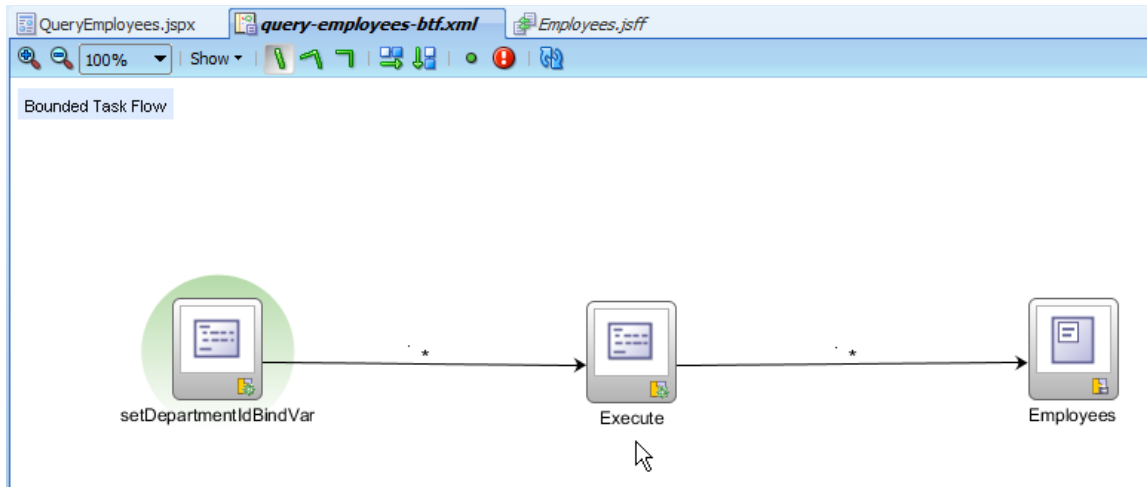


The dialog opens after dropping the method onto the diagram for the developer to reference the location of the filter data. In this example, the **#{pageFlowScope.departmentId}** task flow memory attribute is referenced that holds the value of the **departmentId_IN** task flow input parameter.

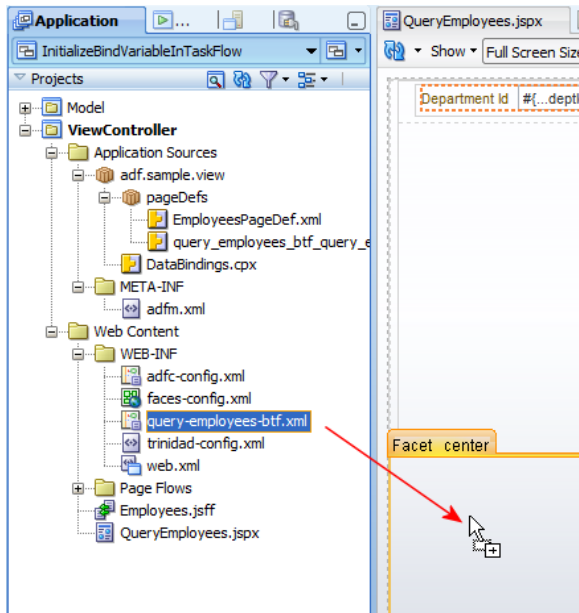


The next step is to execute the query, which actually then filters the employee result set. For this, the **EmployeesView1** instance has an **Execute** action listed in its **Operations** node in the Data Controls panel. Dragging this action to the bounded task flow diagram and drawing a control flow case between the two method call activities ensures the bind variable to be set and the query to be re-executed.

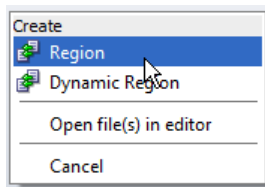
As shown below, the control flow cases from the **setDepartmentIdBindVar** call activity to the **Execute** activity and from here to the **Employees** page fragment use an asterisk as the control flow name. This is a decision done as a matter of taste. Control flow case names determine the navigation to perform when a command action is used for navigation. In the case of the method activities in this example, there is only one navigation from a method activity and thus it doesn't matter which command action outcome is used to perform the navigation. If a meaningful name is used for the control flow case, then this name must match the value of the **outcome** property of the method call activity. As said, it's a matter of taste, which option is chosen.



Once the bounded task flow is complete, it is dragged from the Data Controls panel onto the parent page containing the search form and there added as a **Region**.

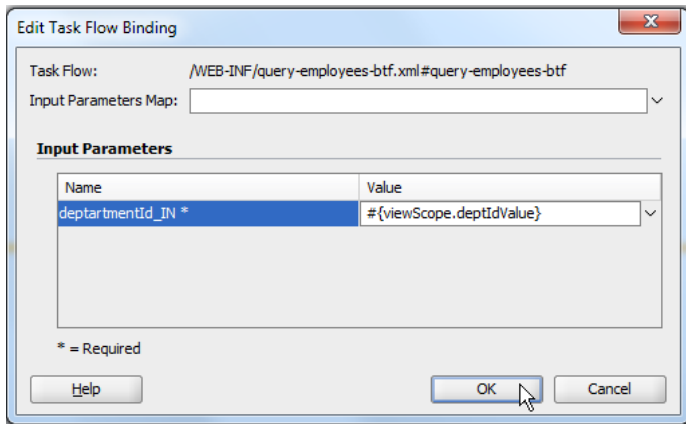


For the use case it does not matter if a static region or a dynamic region is used. In this fictive example, a static **Region** will do.



If the input parameter on the bounded task flow is defined as **Required**, then, after selecting the Region option from the context menu, a dialog shows for the developer to add a reference to where the task flow input parameter value should be read from.

The `af:inputText` field shown earlier writes this information to `#{viewScope.DeptIdValue}`, a memory attribute that is accessible from components (like the region) on this page.



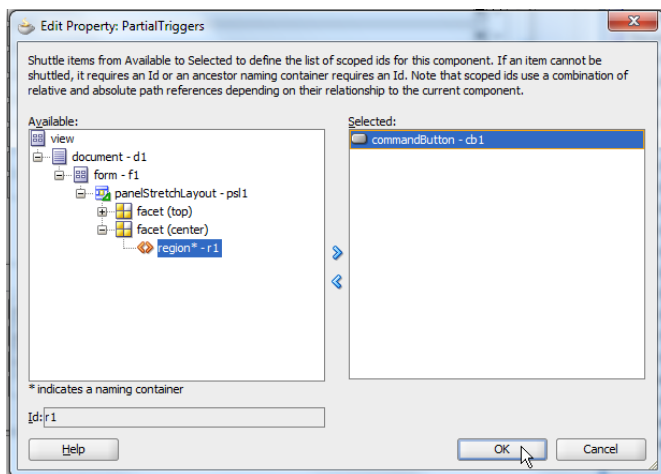
If the input text field was ADF bound, then the value could also be read from the binding layer using EL similar to

```
#{bindings.attributeName.inputValue}
```

The `af:region` **partialTrigger** property needs to reference the command button that submits the input text field to issue the search. This also can be configured declaratively using the **Edit** option in the context menu of the Property Inspector.

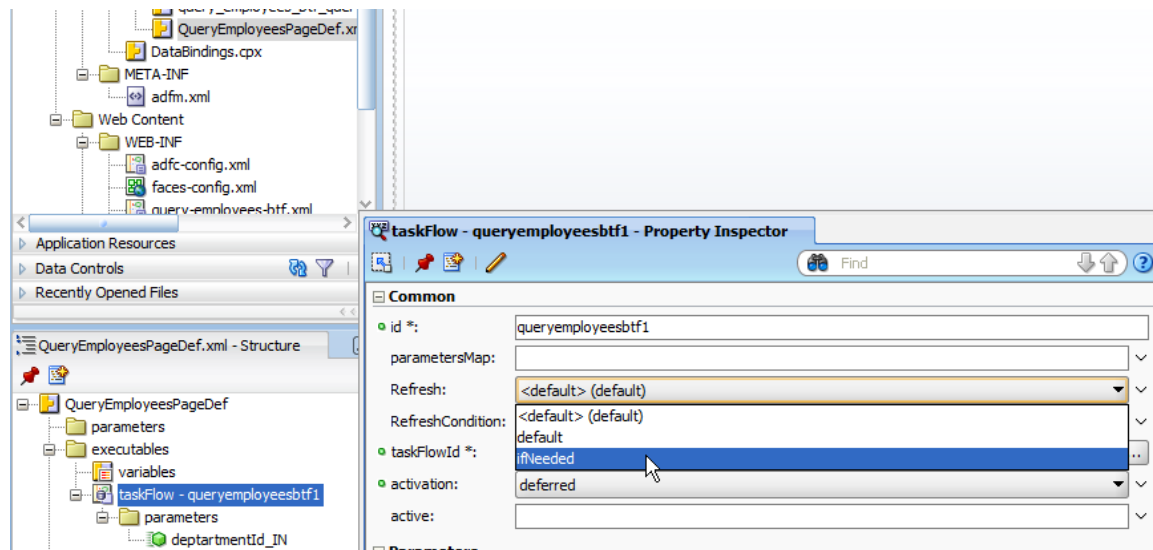


The command button shows in the component hierarchy to the left and can be shuffled to the list of **Selected** components the `af:region` tag then starts listening for.



Last, but not least, the **Region** binding in the parent page's **PageDef** file needs to be configured so that a change of the bounded task flow input parameter leads to a re-initialization (restart) of the bounded task flow when the `af:region` is refreshed through PPR.

The **Region** binding definition is in the **executables** section of the parent page **pageDef** file. To enforce the task flow to re-start, the region binding's **RefreshCondition** property needs to be set to **ifNeeded** so that a re-start of the bounded task flow is triggered whenever the input parameter (**departmentId_IN**) changes, which happens when the user enters a different department Id in the search form field.



The image below shows the runtime view of this example. The user data entry in the text field is written to a memory attribute in viewScope where it is referenced from the Region binding in the PageDef file.

The Region binding passes the input value to the task flow, which saves it in its own pageFlowScope. The **setDepartmentIdBindVar** method call activity is the first activity executed when the task flow initializes. It reads the input value from the pageFlowScope attribute and passes it on to the ADF BC model. The **Execute** method activity is called next and re-queries the model with the bind variable applied. When the page fragment is loaded, then the table, which is built based on the EmployeesView1 collection shows the result set of the query.

Department Id

EmployeeId	FirstName	LastName	Email	PhoneNumber	HireDate	JobId	Salary	CommissionPct	ManagerId	Departm
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	60

Conclusion: To initialize a query on page load using page fragments, you use method call activities to set query parameters and execute the query. There is no need to use Java in a Phase Listener for this.

Of course the solution may be different for other Data Controls that don't have an active model as ADF Business Components do. However this then may only require a slight variation of the above.

Downgrading from Flash in JDeveloper 11g R2

As Flash no longer appears to be a welcome technology on all platforms, Oracle JDeveloper 11g R2 DVT components respond to this with a downgrade option.

To quote the interesting lines of the Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework 11g Release 2 (11.1.2.0.0) appendix: **A.4.3**

----- Quote Start -----

"By default, the application uses the output format specified for each component. For example, ADF Data Visualization components specify a Flash output format to display animation and interactivity effects in a web browser. If the component output format is Flash, and the user's platform doesn't support the Flash Player, as in Apple's iOS operating system, the output format is automatically downgraded to the best available fallback.

You can configure the use of Flash content across the entire application by setting a flash-player-usage context parameter in adf-config.xml. The valid settings include:

downgrade: *Specify that if the output format is Flash, but the Flash Player isn't available, then downgrade to the best available fallback. The user will not be prompted to download the Flash Player.*

disable: *Specify to disable the use of Flash across the application. All components will be rendered in their non-Flash versions, regardless of whether or not the Flash Player is available on the client.*

Example A-7 shows the syntax for application-wide disabling of Flash in adf-config.xml.

```
<adf-config xmlns="http://xmlns.oracle.com/adf/config">
  <adf-faces-config xmlns="http://xmlns.oracle.com/adf/faces/config">
    <flash-player-usage>disabled</flash-player-usage>
  </adf-faces-config></adf-config>
```

The context parameter also supports an EL Expression value. This allows applications to selectively enable or disable Flash for different parts of the application, or for different users, based on their preferences.

Note:

Previously Data Visualization dvt:graph and dvt:gauge components used an imageFormat= "AUTO" value. The AUTO value has been deprecated and you should set use imageFormat="FLASH" and set flash-player-usage context parameter to downgrade to achieve the same effect application-wide."

----- Quote End -----

The most impressive message in my opinion, if you read between the lines, is how ADF Faces DVT components protect your investment. Probably by the time you started ADF application development Flash has been considered to be available everywhere. With Apple stepping out of this supporters reign, more and more companies and vendors rethink their attitude towards Flash. Using Oracle ADF and this ADF Faces as an abstraction layer handles this case for you and it does so gracefully. This is what makes the difference between a framework like ADF and manual coding.

What if Oracle ADF Skin Editor doesn't start on Windows

One of the new features in Oracle JDeveloper 11.1.2 is the ADF Faces Skin Editor for building pluggable application look and feels declaratively. However, the ADF Faces Skin Editor is also available in a stand-alone version for developers building skins for existing applications built with previous versions of Oracle JDeveloper 11g and for web designers that are tasked with building a look and feel but don't need the full Oracle JDeveloper 11g installed.

<http://www.oracle.com/technetwork/developer-tools/adf/downloads/index.html>

The ADF Faces Skin Editor comes in a ZIP file that you extract on your local machine. The parent folder the software unzips itself into is called **skineditor** and developers most likely feel attempted to change the folder name to include additional information like the version number (11.1.2.0.0) or the vendor (Oracle). However, a current issue with the skin editor is that it requires the parent folder name to be **skineditor**. It only allows the writing to be in mixed case, but no change to the name itself.

When changing the parent folder name, for example skineditor_11_1_2_00, then, when starting the skin editor, the Splash Screen is shown but the editor itself does not come up, though the process entry for the skineditor64W.exe is visible in the Windows Task Manager.

The reason for this behavior is that the **skineditor** folder is not considered to be the installation directory but being part of the software. To create a folder with the version number in it, or a name other than **skineditor**, you create a parent folder (installation folder) for the **skineditor** folder. A known issue is that the installation folder you create must not have blanks in the name.

JDeveloper 11.1.2 deployment to WLS

Oracle JDeveloper 11.1.2 (JDeveloper 11g R2) has been released early this month. The runtime environment for JDeveloper 11.1.2 applications is a patched WLS 10.1.3.5 server, according to the release notes:

The ADF Runtime will be installed into a standalone application server by applying an Opatch bundle on top of a PS4 11.1.1.5 version of the Application Developer shiphome. In the event that additional patches are needed or desired in addition to the Sherman 11.1.2 patch (for example, if an additional patch is needed for another component in the Application Development Runtime shiphome), those patches will need to be installed separately.

Important Note: *The patch is intended to be applied to an Application Developer shiphome only. Applying the patch to a different shiphome (for example SOA Suite or WebCenter) is not supported.*

See: <http://www.oracle.com/technetwork/developer-tools/jdev/shermanrelnotes-405777.html#deploy>

Oracle ADF 11.1.2.0 Application Development Runtime is provided as a patch for Oracle ADF Runtime 11.1.1.5 and is available through Oracle Support.

How-to learn more about JDeveloper 11.1.2

To learn more about Oracle JDeveloper 11.1.2, try the tutorials published for this release at

http://download.oracle.com/docs/cd/E18941_01/tutorials/toc.htm

How-to change the WS connection for a WS DC

Using the Web Service Data Control (WS DC) to access a remote service from ADF, stores the Web Service (WS) URL in a file called **Connections.xml** located in the application's **.adf\META-INF** directory. The connections.xml file contains information about the WS endpoint, the port number and the service itself. When moving an application from development to testing or production stage, you usually need to change the WS endpoint and port information. To do this you have 4 options:

- Select the DataControls.dcx file in the Application Navigator, open the Structure Window and choose the **Edit Web Service Connection** entry from the context menu of the Data Control entry.
- Change the end point upon deployment by, in JDeveloper, choosing deploy <name of app> | Application Server at the application level. You can then change the endpoint in the opened configuration dialog
- Using Oracle Enterprise Manager Fusion Middleware Control after deployment. It allows an administrator to declaratively change the endpoint for the deployed ADF application.

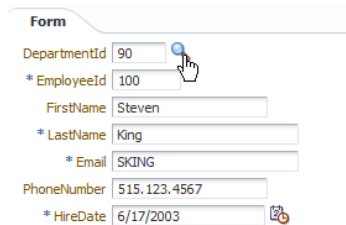
A more detailed description is posted on the **ADF Unleashed** blog run by the Oracle JDeveloper and ADF QA team

http://blogs.oracle.com/adf/entry/changing_endpoint_url_for_a_web_service_data_control

Getting selected row in inputListOfValues returnPopupListener

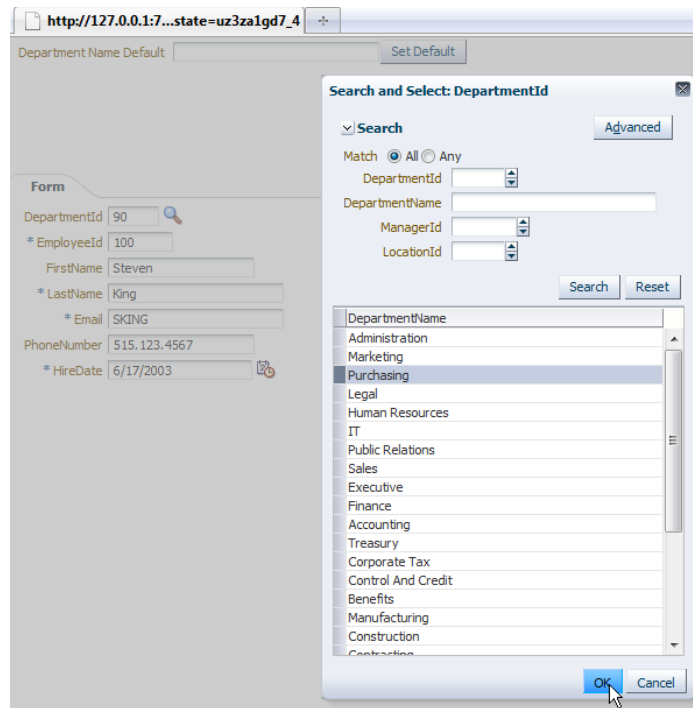
Model driven list-of-values in Oracle ADF are configured on the ADF Business component attribute which should be updated with the user value selection. The value lookup can be configured to be displayed as a select list, combo box, input list of values or combo box with list of values.

Displaying the list in an `af:inputListOfValues` component shows the attribute value in an input text field and with an icon attached to it for the user to launch the list-of-values dialog.



The screenshot shows a form titled "Form" with several input fields. The "DepartmentId" field contains the value "90" and has a small blue circular icon with a white arrow pointing to it, indicating a dropdown menu. Other fields include "EmployeeId" (value "100"), "FirstName" (value "Steven"), "LastName" (value "King"), "Email" (value "SKING"), "PhoneNumber" (value "515.123.4567"), and "HireDate" (value "6/17/2003").

The list-of-values dialog allows users to use a search form to filter the lookup data list and to select an entry, which return value then is added as the value of the `af:inputListOfValues` component.

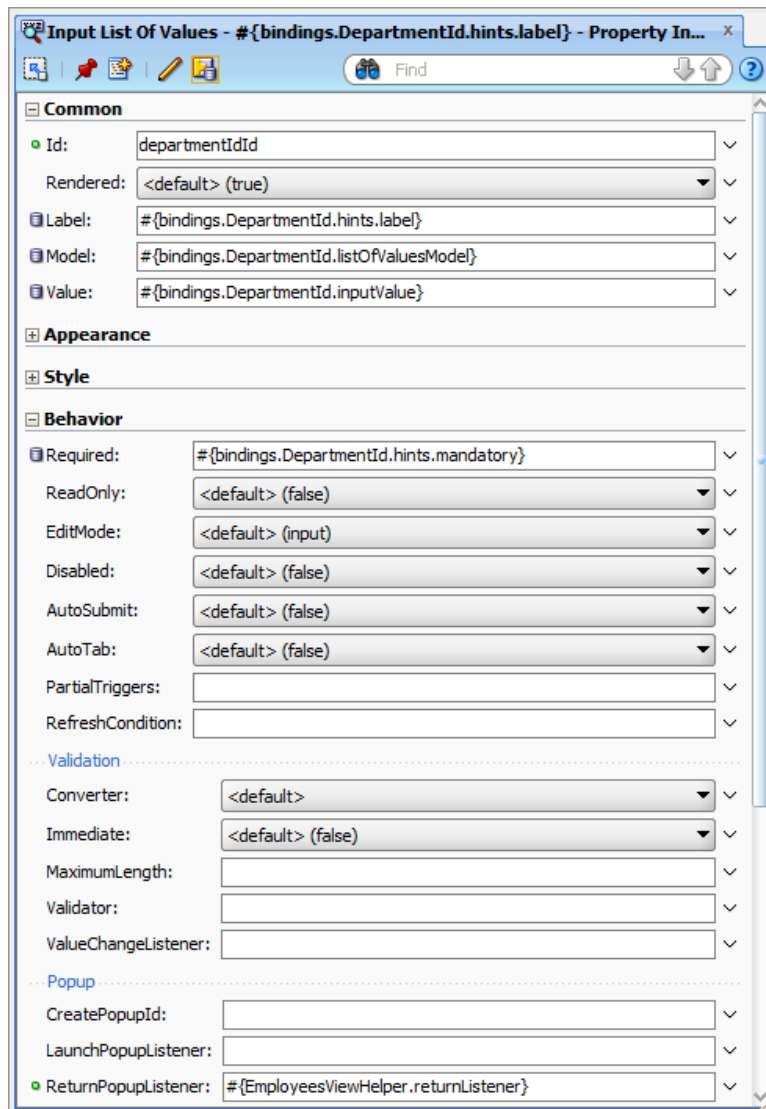


Note: The model driven LOV can be configured in ADF Business Components to update multiple attributes with the user selection, though the most common use case is to update the value of a single attribute.

A question on OTN was how to access the row of the selected return value on the ADF Faces front end. For this, you need to know that there is a **Model** property defined on the `af:inputListOfValues` that references the `ListOfValuesModel` implementation in the model. It is the value of this **Model** property that you need to get access to.

The `af:inputListOfValues` has a **ReturnPopupListener** property that you can use to configure a managed bean method to receive notification when the user closes the LOV popup dialog by selecting the **Ok** button. This listener is not triggered when the cancel button is pressed. The managed bean signature can be created declaratively in Oracle JDeveloper 11g using the **Edit** option in the context menu next to the `ReturnPopupListener` field in the PropertyInspector. The empty method signature looks as shown below

```
public void returnListener(ReturnPopupEvent returnPopupEvent) { }
```



The ReturnPopupEvent object gives you access the RichInputListOfValues component instance, which represents the `af:inputListOfValues` component at runtime. From here you access the **Model** property of the component to then get a handle to the CollectionModel.

The CollectionModel returns an instance of JUCtrlHierBinding in its `getWrappedData` method. Though there is no tree binding definition for the list of values dialog defined in the PageDef, it exists. Once you have access to this, you can read the row the user selected in the list of values dialog. See the following code:

```
public void returnListener(ReturnPopupEvent returnPopupEvent) {
    //access UI component instance from return event
    RichInputListOfValues lovField =
        (RichInputListOfValues) returnPopupEvent.getSource();

    //The LOVModel gives us access to the Collection Model and
    //ADF tree binding used to populate the lookup table
```



```
ListOfValuesModel lovModel = lovField.getModel();
CollectionModel collectionModel =
    lovModel.getTableModel().getCollectionModel();

//The collection model wraps an instance of the ADF
//FacesCtrlHierBinding, which is casted to JUCtrlHierBinding
JUCtrlHierBinding treeBinding =
    (JUCtrlHierBinding) collectionModel.getWrappedData();

//the selected rows are defined in a RowKeySet. As the LOV table only
//supports single selections, there is only one entry in the rks
RowKeySet rks = (RowKeySet) returnPopupEvent.getReturnValue();

//the ADF Faces table row key is a list. The list contains the
//oracle.jbo.Key
List tableRowKey = (List) rks.iterator().next();

//get the iterator binding for the LOV lookup table binding
DCIteratorBinding dciter = treeBinding.getDCIteratorBinding();

//get the selected row by its JBO key
Key key = (Key) tableRowKey.get(0);
Row rw = dciter.findRowByKeyString(key.toStringFormat(true));
//work with the row
// ...
}
```

JDeveloper 11.1.2: Link in Table Column Work Around

In Oracle JDeveloper 11.1.2, clicking on a command link in a table does not mark the table row as selected as it is the behavior in previous releases of Oracle JDeveloper. For the time being, the following work around can be used to achieve the "old" behavior:

To mark the table row as selected, you need to build and queue the table selection event in the code executed by the command link action listener. To queue a selection event, you need to know about the rowKey of the row that the command link that you clicked on is located in. To get to this information, you add an `f:attribute` tag to the command link as shown below

```
<af:column sortProperty="{bindings.DepartmentsView1.hints.DepartmentId.name}" sortable="false"
  headerText="{bindings.DepartmentsView1.hints.DepartmentId.label}" id="c1">
  <af:commandLink text="{row.DepartmentId}" id="cl1" partialSubmit="true"
    actionListener="{BrowseBean.onCommandItemSelected}">
    <f:attribute name="rowKey" value="{row.rowKey}"/>
  </af:commandLink>
  ...
</af:column>
```

The `f:attribute` tag references `#{row.rowKey}` which in ADF translates to `JUCtrlHierNodeBinding.getRowKey()`. This information can be used in the command link action listener to compose the `RowKeySet` you need to queue the selected row. For simplicity reasons, I created a table "binding" reference to the managed bean that executes the command link action. The managed bean code that is referenced from the `af:commandLink` action listener property is shown next:

```
public void onCommandItemSelected(ActionEvent actionEvent) {

    //Do here what the command link is supposed to do
    // ...

    //In the following, set the current row to the row that the command
    //link the user pressed is in

    //Get access to the clicked command link
    RichCommandLink comp = (RichCommandLink)actionEvent.getComponent();

    //read the added f:attribute value
    Key rowKey = (Key) comp.getAttributes().get("rowKey");

    //get the current selected RowKeySet from the table
    RowKeySet oldSelection = table.getSelectedRowKeys();

    //build an empty RowKeySet for the new selection
    RowKeySetImpl newSelection = new RowKeySetImpl();

    //RowKeySets contain List objects with key objects in them
    ArrayList list = new ArrayList();
    list.add(rowKey);
    newSelection.add(list);

    //create the selectionEvent and queue it
    SelectionEvent selectionEvent =
        new SelectionEvent(oldSelection, newSelection, table);
    selectionEvent.queue();

    //refresh the table
    AdfFacesContext.getCurrentInstance().addPartialTarget(table);
}
```

Solving the problem of inactive pages after file download

There are at least two options to download files in Oracle ADF faces. The recommended option is to use the file download listener explained in the component documentation

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

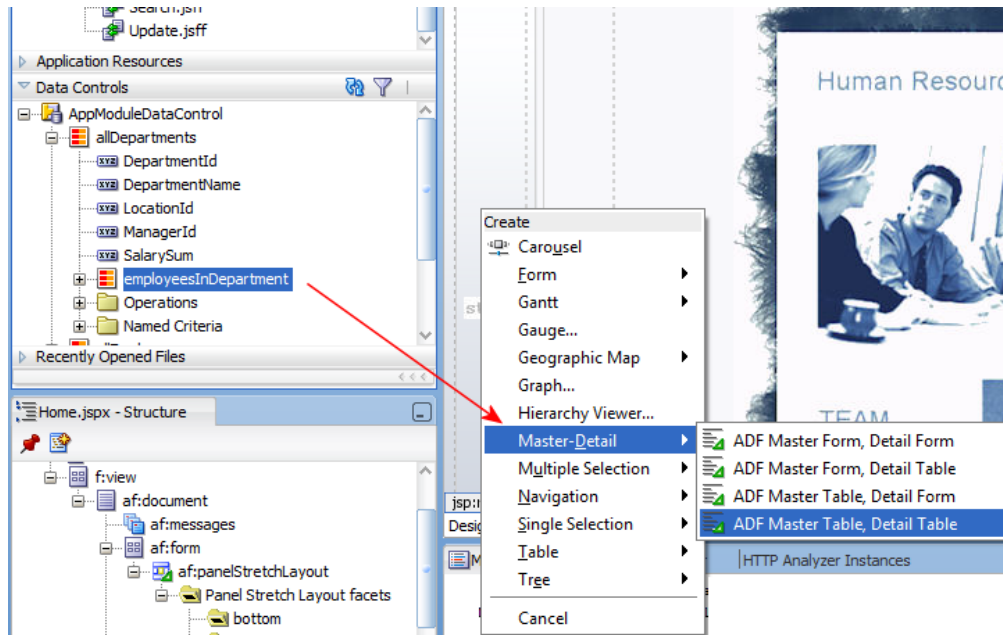
Another option is to use a request to a servlet that then handles the download. What could happen when using a servlet is that the ADF Faces page shows inactive after the download in that clicking a command

items does not produce an action. To avoid this, you can try the following JavaScript function that you call from an `af:clientListener` added to the command initiating the download.

```
function onServletDownload(evt) {
    evt.noResponseExpected();
}
```

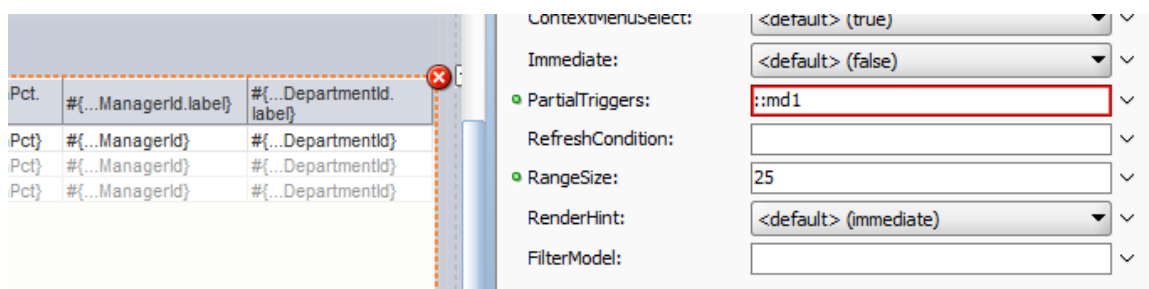
Building master-detail tables with `af:panelCollections`

Dragging a detail collection from the Data Controls panel to a page, as shown in the image below, produces the context menu option of **Master-Detail** forms and tables.



Using this option makes sure the detail component (form or table) references the master table or form in its **PartialTriggers** attribute.

When the master collection is rendered as a table and, in a later step, an `af:panelCollection` gets added to surround it, the master-detail behavior breaks because of the change of the master component reference.



Reason: The `af:panelCollection` component is a naming container, which means that at runtime it adds its component Id (e.g. `pc1`) as a prefix to the client Id of the contained master table. To preserve the master-detail refresh behavior of the detail table when the master table refreshes, you need to change

the **PartialTrigger** reference on the detail table to e.g. **::pc1:md1** (assuming pc1 is the panel collection Id and md1 the master table id).

Declarative Components with ADF bindings?

Declarative Components are regular JSF components except that they are built as a composite out of existing ADF Faces components. Declarative components cannot have a **PageDef** file associated with it and instead use value or method attribute to communicate with the consuming page. When creating a declarative component, the **attrs** implicit object is defined that allow you to reference the component's exposed value attribute from ADF Faces components within.

An input text field contained in an declarative component may reference a value attribute "compValue" defined on the declarative component by using the following EL

```
{attrs.compValue}
```

The declarative component **compValue** attribute then would us an EL reference like shown next to read/write values from or to the ADF binding layer:

```
{bindings.attribute_name.inputValue}
```

Performing Logging in Oracle ADF

There is quite some confusion among developers of what is the best way to provide logging functionality in Oracle ADF. The good news is that there is a built-in mechanism in ADF called **ADFLogger** that not only is easy to use but also makes it easy to analyze logs with analytic support added to Oracle JDeveloper and Oracle Enterprise Manager Fusion Middleware Control (usually referred to as Enterprise Manager).

Technically, **ADFLogger** is a logging mechanism that wraps the `java.util.Logging` API, just with some more bells and whistles added. Duncan Mills blogged a series on using the **ADFLogger** in ADF, which, given that the logging question comes up quite frequent on OTN, I reference here:

Part 1: http://blogs.oracle.com/groundside/entry/adventures_in_adf_logging_part

Part 2: http://blogs.oracle.com/groundside/entry/adventures_in_adf_logging_part1

Part 3: http://blogs.oracle.com/groundside/entry/adventures_in_adf_logging_part2

Part 4: http://blogs.oracle.com/groundside/entry/adventures_in_adf_logging_part3

Part 5: http://blogs.oracle.com/groundside/entry/adventures_in_adf_logging_part4

RELATED DOCUMENTATION

