**Programmatically Call Iterator and Refresh Iterator from UI (Managed Bean)**

The code snippet is as follows:  
  
 DCBindingContainer bindings = (DCBindingContainer)BindingContext.getCurrent().getCurrentBindingsEntry();  
 DCIteratorBinding iter = bindings.findIteratorBinding("ReceiptsIterator");  
 iter.executeQuery();

BindingContext bctx = BindingContext.getCurrent();

BindingContainer bindings = bctx.getCurrentBindingsEntry();

DCIteratorBinding iter = (DCIteratorBinding)

bindings.get("listRequestsIterator");

iter.clearForRecreate();

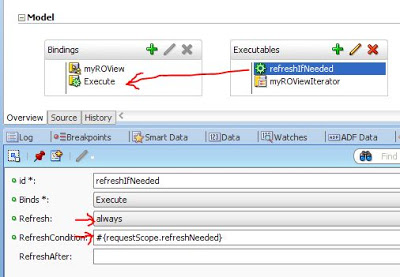
**ADF 11G : Catch the exception at runtime and Display it on page**

**ADF 11G : Catch the exception at runtime and Display it on page**  
  
  
catch(Exception e) {   
  
FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY\_ERROR, e.getMessage(), "");   
  
FacesContext.getCurrentInstance().addMessage(null, msg);   
  
}   
FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY\_WARN, msgHead , msgDetail);   
  
facesContext.addMessage(uiComponent.getClientId(facesContext), msg);

**How to force a table to refresh itself**

Sometimes you will need to refresh a table after some event takes place in your backing bean.  You can refresh the table component using a partial trigger or refresh it from the backing bean using   getAdfFacesContext().addPartialTarget(uicomponent);  but that pulls the current values from the model through the page binding and doesn't necessarily pull fresh data from the database when the data has not been inserted from the ADF application.  I was having a problem in that when data was modified externally to our application (for example by an external web service call), that refreshing the table itself was not re-executing the query to bring in the new data.  
  
This technique is simple and works across region boundaries instead of using[contextual events](http://biemond.blogspot.com/2009/01/passing-adf-events-between-task-flow.html).  Contextual events are Oracles' recommended practice for inter-region communication and I use them in many places but I have found them to carry a performance penalty, at least at the time of this writing and they are more difficult to configure.  
  
When you have a table somewhere on your page (any [region](http://jdevadf.oracle.com/adf-richclient-demo/docs/apidocs/oracle/adf/view/rich/component/rich/fragment/RichRegion.html)) that needs refreshing, perform the following steps

* In a backing bean method, set a Boolean.TRUE into a requestScope variable. (see code in red below).  
  ADFContext.getCurrent.getRequestMap().put("refreshNeeded", Boolean.TRUE);   
  This can be done from the backing bean of any region (task flow) running on the page.
* In the executable bindings for your table, add a invokeAction executable (named refreshIfNeeded) and bind it to the Execute or ExecuteWithParams action binding for your table’s view (on the left hand side).
* Set the refreshIfNeeded binding to refresh “Always”
* Add a refresh condition expression that references your request scope variable.

[](http://4.bp.blogspot.com/_yWM57wdqk5o/SgDeLMgdv6I/AAAAAAAAAAk/10sg0mJFIOU/s1600-h/REFRESH.JPG)  
  
  
In your backing bean you could  retrieve the "Execute" Action from the bindings and executed it yourself but that only works if your backing bean is in the same region as the table you want to refresh.  Also, this technique assures the query is only executed once during the ADF lifecycle.   If the variable is set on requestScope more than once by different regions, no harm done.  You will still need to refresh the table component itself either through a partial trigger or from the backing bean as mentioned but this technique will force the query to execute as well.

 inlineStyle="#{row.isDirty ? 'background-color:#FFF9B3;' : ''}"

# OFMW Guide: 18 Working programmatically with ADF

 April 19, 2017 [ADF APIs](http://amrsalah.tech/mwcwp/tag/adf-apis/), [ADF BC](http://amrsalah.tech/mwcwp/tag/adf-bc/), [ADF UI](http://amrsalah.tech/mwcwp/tag/adf-ui/), [App Module](http://amrsalah.tech/mwcwp/tag/app-module/), [Service Methods](http://amrsalah.tech/mwcwp/tag/service-methods/)



In this post I am going to show you 18 Working programmatically with ADF to overpass ADF limitations and fix bugs.

Unlike most of Java frameworks, ADF is not an open source and despite it is providing a developer with unlimited features it’s also providing us with a limited version of bugs and some limitations might reflect part a customer requirement.

Still ADF is very strong framework and I believe it’s not an optional anymore to use or leave it because simply it’s now a part from Oracle fusion stack (Application and Middleware); therefore nowhere to go if you are developing BPM application or working with Oracle cloud.

The strength of other frameworks like angular JS, the ability to integrate it with ADF not to replace it and your dexterity with ADF comes from your solid knowledge about the framework and the way you play with its APIs.

For more Information about the framework you can check the [ADF Basics topic – Oracle](https://docs.oracle.com/cd/E24382_01/web.1112/e16182/intro_tour.htm#ADFFD121) Documentation

I will use the following Model Example in my code:

We have a University consists of Professors, Students and Classes.

(University)

ــــ one to many ـــ (Professors)

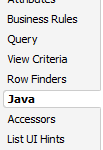
ـــ one to many ـــ (Class)

ـــ one to many ـــ (Student)

#### ****Working programmatically with Application Module (Service Methods)****

First generate the Java classes including accessors for all views

* Open View Object
* Under Java Tab



* click (Edit icon) https://i0.wp.com/amrsalah.tech/mwcwp/wp-content/uploads/2017/04/save.png?resize=27%2C26
* Check “Generate View Object Class” and “Include Bind Variable Accessors”
* Check “Generate View Row Class” and “Include Accessors”

**Working with View Criteria**

Using view criteria you can set the current row in view object by applying view criteria

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public void setUniversityByName(String univName){      ViewCriteria univCriteria = this.getUniversityView().getViewCriteria("findUniversityByName");      univCriteria.ensureVariableManager().setVariableValue("universityName", univName);      this.getUniversityView().applyViewCriteria(univCriteria);      this.getUniversityView().executeQuery();  } |

**Work with bind variable in view object**

Now you have current row in university so we will get one of its professor by Id

Also you can verify how many returned rows in RowSetIterator.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public void setProfessorById(int professorId){      UniversityViewImpl universiryRow = this.getUniversityView();      UniversityViewRowImpl currentUnivID = universiryRow.getCurrentRow().getAttribute("UniversityID")+"";        ProfessorViewImpl professorRow = this.getProfessorView();      professorRow.setNamedWhereClauseParam("UNIVERSITY\_ID", currentUnivID);      professorRow.setNamedWhereClauseParam("PROFESSOR\_ID", professorId);      professorRow.executeQuery();      if (this.getProfessorView().getAllRowsInRange().length &amp;amp;amp;amp;amp;amp;amp;amp;gt; 0) {              //Do your actions      }    } |

**Get the first, Last Row in View object**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public ClassViewRowImpl getFirstClassRow(int professorId){      ProfessorViewRowImpl professorRow = (ProfessorViewRowImpl)((UniversityViewRowImpl)this.getUniversityView().getCurrentRow()).getProfessorView().getCurrentRow();        ClassViewRowImpl firstClass = professorRow.getClassView().first();      return firstClassRow;        /\*      \* or get the last      \* ClassViewRowImpl firstClass = professorRow.getClassView().first();      \* return firstClass;      \*/  } |

**Find Row by Key and Set attribute with particular value**

Update an existing row by find the row then update an attribute

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public void updateStudentName(String studentName){       ClassViewRowImpl classViewRow = this.getFirstClassRow();       StudentView studentView = this.getStudentView();         Key key = new Key(new Object[] { this.getFirstClassRow.getAttribute("studentFK")});         // Get the RowSetIterator Object       RowSetIterator studentViewRSI = studentView().createRowSetIterator(null);       StudentViewRowImpl studentRow = (StudentViewRowImpl) studentViewRSI.findByKey(key, 1)[0];       studentRow.setAttribute("StudentName", studentName)  } |

**Create new row in RowSetIterator**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public StudentViewRowImpl createNewStudent(classId){      //Refresh the view object      this.getStudentView().executeQuery();      RowSetIterator studentIterator = this.getStudentView().createRowSetIterator(null);      StudentViewRowImpl newStudent = studentIterator.createRow();      newStudent.setAttribute("classIdInput", classId);      studentIterator.insertRowAtRangeIndex(studentIterator.last()+1, newStudent);      return (StudentViewRowImpl)studentIterator.last();    } |

**Create new row in view object**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public StudentViewRowImpl createNewStudent(classId){      //Refresh the view object      StudentViewImpl studentView = this.getStudentView();      StudentViewRowImpl newStudent = (StudentViewRowImpl) studentView.createRow();      newStudent.setAttribute("classIdInput", classId);      studentView.insertRow(newStudent);      studentView.setCurrentRow(newStudent);        return this.getStudentView().getCurrentRow();  } |

**List result from RowSetIterator**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public void listAllStudents(){      StudentView studentView = this.getStudentView().executeQuery();      RowSetIterator studentIterator = studentView.createRowSetIterator(null);      while(studentIterator.hasNext()){          StudentViewRowImpl student = (StudentViewRowImpl) studentIterator.next();          //do you actions      }  } |

**Refresh View Object after applying View Criteria (Remove View Criteria)**

|  |  |
| --- | --- |
| 1  2  3  4 | public void refreshUniversityObject(String univName){      this.getUniversityView().removeViewCriteria("findUniversityByName");      this.getUniversityView().executeQuery();  } |

**Commit/Rollback transaction**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public void commitTransactionToDB() {      this.getDBTransaction().commit();  }  //Rollback  public void commitTransactionToDB() {      this.getDBTransaction().rollback();  } |

#### ****Work programmatically with ADF on View Controller project****

In the following section I will use ADF APIs to access service methods, Data controls from UI as well as working with UI iterators.

**Call Service method from UI**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public void excuteServiceMethodFromUI(String opName) {      OperationBinding opCaller = ADFUtils.findOperation(opName);      //set parameters if exist or skip next line      opCaller.getParamsMap().put("param1", "value1");      opCaller.execute();      //Check if its executed successfully      if (opCaller.getErrors().size() &gt; 0) {           throw new ModerationException("Error in executing opCaller");           //Report the error or show error message      }  } |

**Work with UI Iterator**  
This iterator is different than RowSetIterator. The UI Iterator has range size only 25 rows by default and sometimes you have to extend its range to list all values by configuring its range size value, but you have to rollback that change again to keep application performance in good shape.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public String getCurrentStudentNameFromUIIterator(){       DCIteratorBinding studentIterator =       ADFUtils.getDCBindingContainer().findIteratorBinding("StudentViewIterator");       ViewObject studentViewObject = studentIterator.getViewObject();       Row currentStudent = studentIterator.getCurrentRow();       return currentStudent.getAttributeName("studentName") + "";  }  //or list all student names  public ArrayList&lt;String&gt; getStudentNamesFromUIIterator(){          ArrayList&lt;String&gt; studentlist = new ArrayList&lt;String&gt;();       DCIteratorBinding studentIterator =       ADFUtils.getDCBindingContainer().findIteratorBinding("StudentViewIterator");       while(studentIterator.hasNext()){           StudentViewRowImpl currentStudent = (StudentViewRowImpl)studentIterator.next();           studentlist.add(currentStudent.getAttributeName("studentName")+"");       }         return studentlist;  } |

**Add Object to Session**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public void addObjectToADFSession(String name, Object object) {          FacesContext context = FacesContext.getCurrentInstance();          ExternalContext exContext = context.getExternalContext();          HttpSession sessionRef = (HttpSession) exContext.getSession(true);          //Add Object to session          sessionRef.setAttribute(name, object);          } |

**Get Object from Session**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public Object getObjectFromADFSession(String name) {         FacesContext context = FacesContext.getCurrentInstance();         ExternalContext exContext = context.getExternalContext();         HttpSession sessionRef = (HttpSession) exContext.getSession(true);         return (Object) sessionRef.getAttribute(name);       } |

**Remove Object from session**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public Object getObjectFromADFSession(String name) {         FacesContext context = FacesContext.getCurrentInstance();         ExternalContext exContext = context.getExternalContext();         HttpSession sessionRef = (HttpSession) exContext.getSession(true);         return (Object) sessionRef.removeAttribute(name);       } |

**Access UI Component**

|  |  |
| --- | --- |
| 1  2  3  4  5 | public UIComponent getComponent(String componentId) {         UIViewRoot viewRoot = FacesContext.getCurrentInstance().getViewRoot();         UIComponent component = viewRoot.findComponent(componentId);         return component;     } |

**Set/Get attribute value (Using Regular Expression)**

|  |  |
| --- | --- |
| 1  2  3  4  5 | ADFUtils.setEL("#{bindings.studentName.inputValue}", "");      ADFUtils.evaluateEL("#{bindings.studentName.inputValue}");  //or you can use      ADFUtils.setBoundAttributeValue("studentBirthDate", new oracle.jbo.domain.Timestamp(new Date().getTime()));      ADFUtils.getBoundAttributeValue("studentName"); |

**Skip validation programmatically**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public void skipValidationOnChange(ValueChangeEvent valueChangeEvent) {  valueChangeEvent.getComponent().processUpdate();  //Implement all required scenarios  //now jump to renderResponse  FacesContext.getCurrentInstance().  } |

**AdfFacesContext common uses**

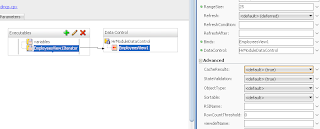
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | //Set value in pageFlowScope  AdfFacesContext.getPageFlowScope().put("objectName", object);  //get value from pageFlowScope  AdfFacesContext.getPageFlowScope().get("objectName");  //Add partial trigger on binding component  AdfFacesContext.getCurrentInstance().addPartialTarget(this.getStudentTable()); |

I did my best to cover most of points in one post hoping to get what are you looking for…

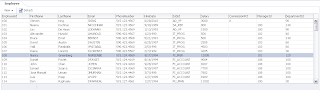
Later I will add posts to work grammatically with ADF Tables, LOV and different common bugs in ADF, See you in another post

[**Cache Results for ADF Iterator Property**](http://andrejusb.blogspot.com/2013/06/cache-results-for-adf-iterator-property.html)

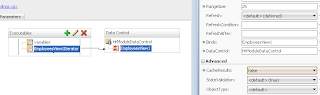
There are various properties in ADF developers tend to click around. Based on my experience from various ADF projects and what strikes me the most - often some property change is done without actually understanding what it means for application performance, at first comes desire to find solution with any cost. I will give you one example related to CacheResults property for ADF bindings iterator.  
  
By default CacheResults property is set to True:

[](http://3.bp.blogspot.com/-rogI_WHFaJ0/UbB7hoj0_0I/AAAAAAAAJnQ/61eA5mYJaeQ/s1600/1.png)

This means when requests are submitted from ADF UI, current rowset is not re-executed again if rows were fetched already. For example when table is loaded and users selects different rows, during new row selection there is no SQL query executed again:

[](http://2.bp.blogspot.com/-Y_21J3gTBfk/UbB8IzvkFdI/AAAAAAAAJnY/0x-9VT6NbJo/s1600/2.png)

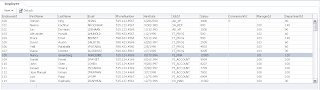
On opposite, when CacheResults is set to False:

[](http://4.bp.blogspot.com/-0WwkiYIewr8/UbB8fIbclXI/AAAAAAAAJng/Pl78fa-xzqA/s1600/3.png)

Before every request ADF will re-execute SQL query and fetch previously already fetched rows again and again. Obviously this is not what you want in most of the cases, so be careful with CacheResults property. SQL is executed and rowset is fetched:

[](http://3.bp.blogspot.com/-jUrShGs5cdY/UbB87XaXNII/AAAAAAAAJno/uuWiZkBcZ6c/s1600/4.png)

In this example - [CacheResultsApp.zip](https://jdevsamples.googlecode.com/files/CacheResultsApp.zip), such behavior happens on every row select. I just wanted to show what effect you can get by setting CacheResults property to False.

[](http://4.bp.blogspot.com/-fjaO5GL1y2Y/UbB-vOPkSKI/AAAAAAAAJn4/bZ3AeSodimA/s1600/5.png)

For my specific case, developer wanted to make sure data is refreshed everytime when there is change in DB. But later this requirement was dropped, however no one remembered to set CacheResults back to default, form complexity was increasing until performance became bad. Ok, but now it is fixed !