ADF Code Corner

63. How-to save DVT graphs to a file

twitter.com/adfcodecorner

Abstract:

ADF Faces Data Visualization components render in different formats – Flash, SVG and PNG – dependent on the format setting defined by the application developer. To download graphs, or to merge them into documents, it is also possible to export graphs to an image file on the server. This how-to article provides a sample of how to export a bar graph to a PNG image file format.

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

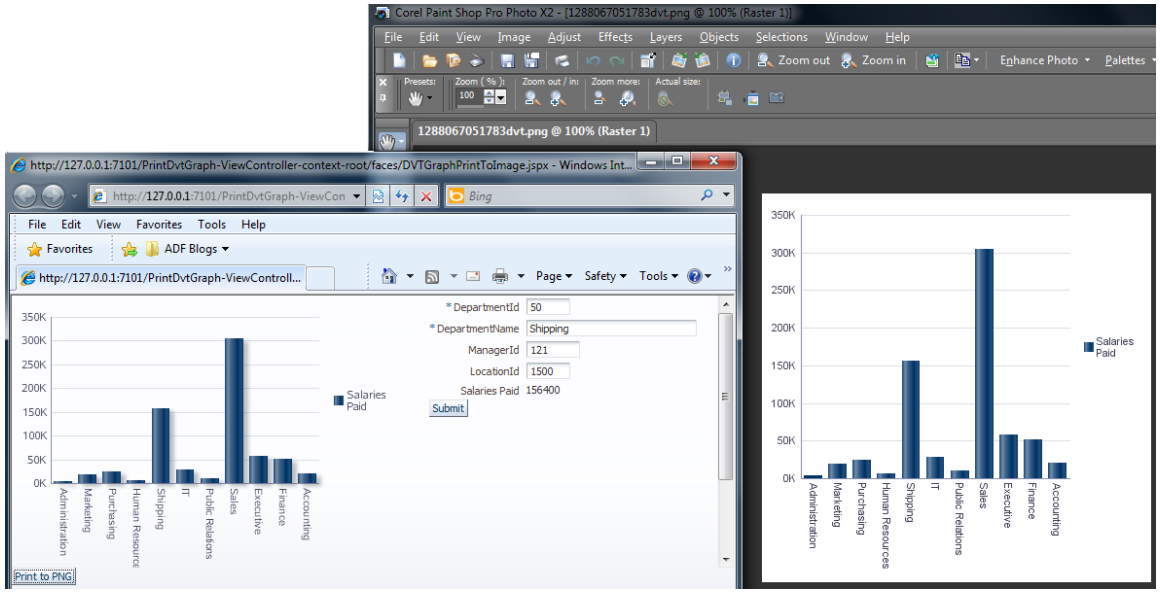
Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

The Oracle JDeveloper workspace that you can download from ADF Code Corrner for this sample exports a bar graph showing department names and salaries paid of the Oracle HR schema. Pressing a button exports the graph to an image file from where it can be further processed.

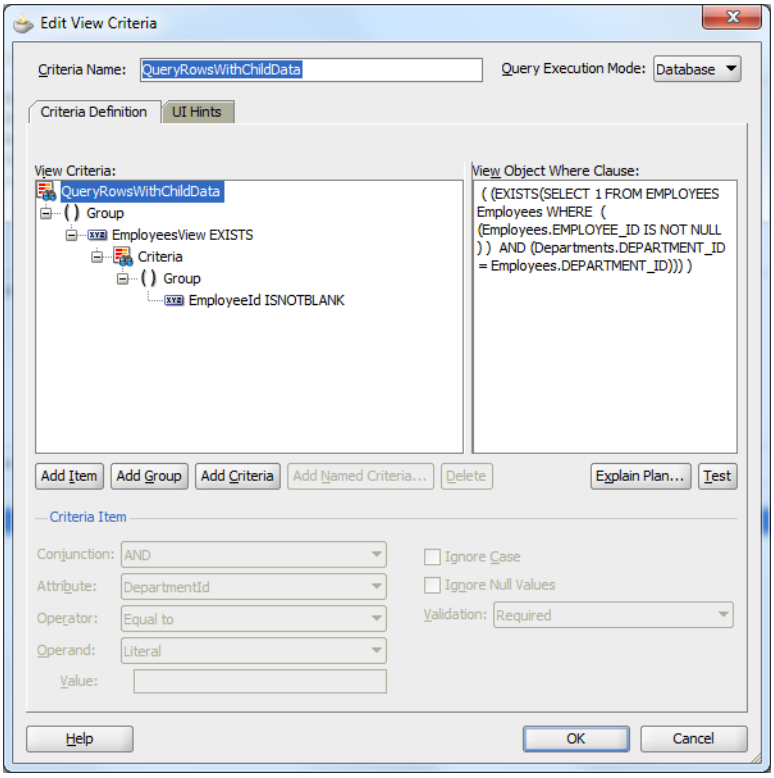


The image above shows a DVT graph at runtime in Internet Explorer and read from the file system in Corel PainShop Pro. Note that the image size of the image saved to the file system is different from the size of the image displayed on the web, a change configured upon exporting the image.

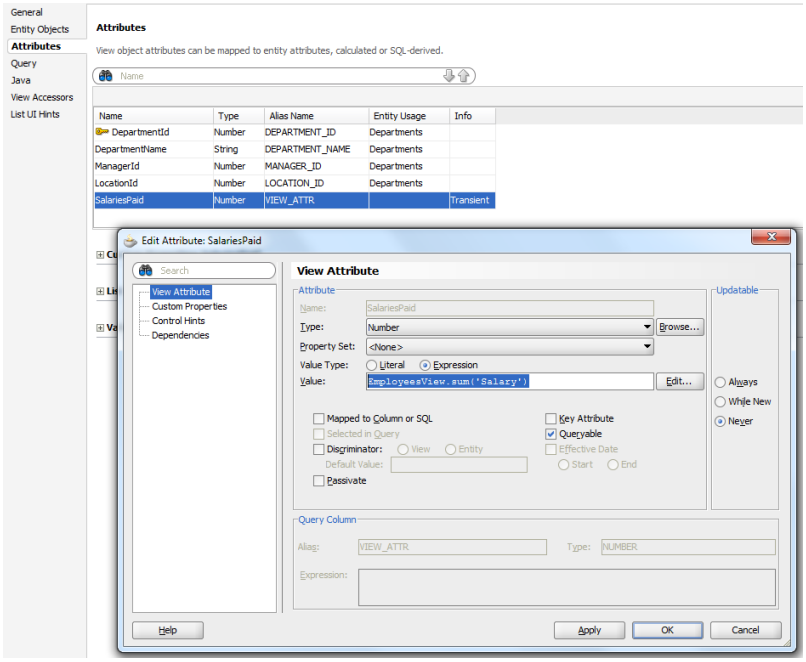
Building the ADF BC model

The sample data is queried from the ADF Business Component business service. The model uses a View Criteria applied to the DepartmentsView to query only those departments that have employees. A Groovy expression is used to compute the total salary value per department, which then is shown in the graph.

The View Criteria is declaratively build and applied to the View Object instance on the Application Module data model.

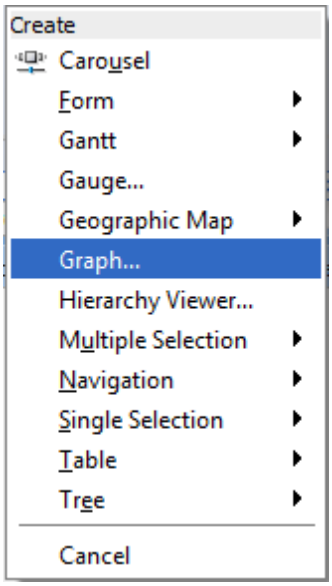


The Groovy expression that computes the total salary in a transient attribute references the view accessor that is automatically created by ADF BC for tables that have a PK/FK relation defined. Note that you need to set the transient attribute field type to Expression to have ADF BC executing the expression instead of displaying it.



Building the Graph

To create the graph, drag the Departments View collection from the Data Controls panel to a JSF page and choose the graph option.



A dialog will step you through the graph creation. If you check the "Set current row for master-detail" option, users can click into the graph to set the row currency of the underlying collection.

In the example that you can download from ADF Code Corner - http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html - this setting is enabled to synchronize the selected graph bar with an edit form. Note that this is not required to export the graph to a file and I did this only because I can ☺

Once the graph is created, a command button is added that looks up the graph from the JSF view root and exports its content to a file.

Note: Instead of looking the file up on the view root, you could create a JSF component binding. However, looking the component instance up at runtime allows writing generic, reusable code – which sounds like a desirable goal.

The managed bean logic to export the bean

The managed bean code that is invoked by the command button is little as the work of exporting the graph image is handled in a separate context callback class – DvtContextCallBack:

```
import javax.faces.component.UIViewRoot;
import javax.faces.context.FacesContext;

public class DvtPrintBean {

    public DvtPrintBean() { }

    //method called from the command button
    public String printAction() {
      //the String passed to the method is the graph component Id,
```

```
      //including the naming container Id if the graph is contained
    //in a naming container like dialog, table etc …
      printDVTComponent("barGraph1");
        return null;
    }

  private void printDVTComponent(String clientId) {

       //find starting component

        FacesContext fctx = FacesContext.getCurrentInstance();
        UIViewRoot root = fctx.getViewRoot();
        root.invokeOnComponent( fctx,clientId, new
DvtContextCallBack());
    }
}
```

The code above accesses the JSF view root and performs the DvtCallBack on it to find and export the graph component.

The real work is handled by the DvtCallBack class that searches the graph component in the current view and that exports its content to an image file:

```
import java.awt.Color;
import java.awt.Dimension;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.GregorianCalendar;
import javax.faces.component.ContextCallback;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import oracle.adf.view.faces.bi.component.graph.Background;
import oracle.adf.view.faces.bi.component.graph.UIGraph;
import oracle.dss.dataView.ImageView;

public class DvtContextCallBack implements ContextCallback {
    public DvtContextCallBack() {
        super();
 }

//method called on each component that matches the search criteria
public void invokeContextCallback(FacesContext facesContext,
   UIComponent uiComponent) {
   //PRINT
   //Only care for instances of UIGraph
   if (uiComponent instanceof UIGraph) {
       UIGraph dvtgraph = (UIGraph)uiComponent;
      //We can set a different background color. However,
```

```
 //this changes the graph instance and thus we need to
 //copy and set back the current values
Background orgBackground = dvtgraph.getBackground();
//create and set a new background
Background bg = new Background();
//backgrounds can not only be set to colors but also transparent
//with graduated fill
bg.setFillColor(Color.WHITE);
//explicitly set transparent fill to false for a white
//background
bg.setFillTransparent(false);
dvtgraph.setBackground(bg);
//this needs to be called to ensure the background color is set.
dvtgraph.transferProperties();

//image view is what we want to export. Also, this is where we
//set the exported image size
ImageView imgView = dvtgraph.getImageView();
//We can set a different image size. However, this changes the
//graph instance and thus best is to copy the current values to
//set them back after the image is processed
Dimension orgDimension = imgView.getImageSize();
                              //width, height
imgView.setImageSize(new Dimension(400,400));
//Get the OS specific file path separator
String slash = File.separator;
String dSlash = slash + slash;
//create a unique file name (you may want to change generating the
//file name using a real random so that concurrent access to the
//application don't conflict if they are processed just in the
//same fraction of a second
String filename =
      GregorianCalendar.getInstance().getTimeInMillis() + "dvt";
String drive = "c:";
String tmpFolder = "temp";
File file = null;
FileOutputStream fos;
try {
  file = new File(drive + dSlash + tmpFolder + slash + filename
         + ".png");
  fos = new FileOutputStream(file);
  imgView.exportToPNG(fos);
  fos.close();
}
catch (FileNotFoundException e) {
  //For sample - just show stack trace
```

```
        e.printStackTrace();
      }
      catch (IOException e) {
        e.printStackTrace();
      }
      finally{
        //reset the graph default - IMPORTANT ! - as otherwise the
        //web instance of the graph would change with the next refresh
        dvtgraph.setBackground(orgBackground);
        dvtgraph.transferProperties();
        imgView.setImageSize(orgDimension);
      }
    }
  }
}
```

Sample Download

The sample that you can download from ADF Code Corner - http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html - requires a local HR schema. Configure the database connection and – optionally – change the file directory hard coded in the DvtContextCallBack class. So if there is no c:\temp directory, then you need to change this Java class.

In a production environment the file directory should be dynamically passed into the application, for example using a web.xml context parameter.

RELATED DOCOMENTATION

☒	ADF Insider Video – Using Groovy in ADF Business Components - http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/ADF_Insider_Groovy/ADF_Insider_Groovy.html
☒	Java Example to create a temporary file that is cleaned up after its use http://www.exampledepot.com/egs/java.io/CreateTempFile.html
☒	