ADF Code Corner

95. How-to Navigate to Printable Pages

twitter.com/adfcodecorner

Abstract:

Commonly, you build printable pages in ADF Faces using the `af:showPrintablePageBehavior` tag added to a command component. Pressing the command component will show a printable version of the page in an external browser dialog. It's the recommended way of doing this and a purely declarative implementation. What however if you don't want an browser dialog to show the printable page and instead prefer navigating to it in the same browser window? This is what this blog article covers.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
26-JAN-2012

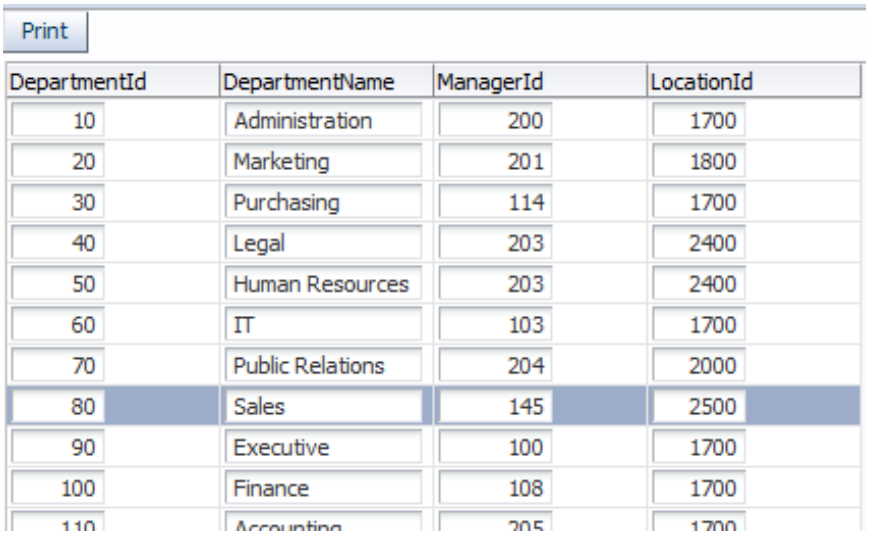
Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

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

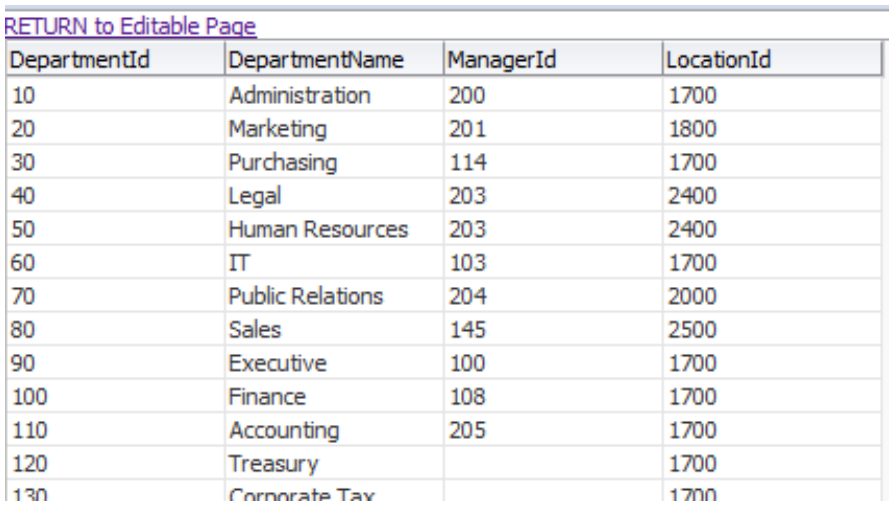
Introduction

The difference between editable pages rendered in ADF Faces default mode and pages rendered in printable mode is that printable pages lack all navigational components and scrolling so that all contents can be printed. To explain the solution, I use a simple use case of an ADF bound table.



The page shows a Print button that when pressed navigates to the same page in a printable page layout. Navigation however doesn't need to be to the same page for printing. It can also be to a specifically designed ADF Faces page you use e.g to show an invoice at the end of a shopping process. Showing pages in printable mode just ensures that the page is optimized by not showing command components or input fields. In addition, you may want to fine tune pages for printing, for example, showing a company logo.

The printable page shown below is optimized so that it shows a HTML link to navigate back to the editable page, restoring any state on this page, and so the selected table row highlighting is removed.


RETURN to Editable Page

DepartmentId	DepartmentName	ManagerId	LocationId
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Legal	203	2400
50	Human Resources	203	2400
60	IT	103	1700
70	Public Relations	204	2000
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700

The basic implementation

The `trinidad-config.xml` file is in in the application WEB-INF directory of every ADF Faces application and can be configured with an **outcome** element. The outcome element if missing in the `trinidad-config.xml` file, which is the default, by default is set to editable pages ("default"). Other value options for the **outcome** element are **printable** and **email**. As the value of the outcome element can be set using Expression Language it can be set dynamically.

```
<output-mode>#{PrintableBehavior.outcomeMode}</output-mode>
```

In the code sample above, the **outcome** parameter values in the `trinidad-config.xml` file is read from a managed bean in request scope. The nice thing about using a managed bean in request scope is that after a print request where the outcome is set to **printable** it immediately is set back to **default** so that upon navigating back to the calling page or any other ADF Faces page, the mode is back to edit.

In the sample I built for this article, I used two managed beans in request scope. One managed bean is referenced from the page to set the outcome mode to printable and to hide or show the return link you see on the printable page. Also, the table selected row indicator is removed from the printable page. The second managed bean only holds the **outcome** state referenced from the `trinidad-config.xml` file.

The managed beans are configured in the `adfc-config.xml` file shown below

```
<managed-bean id="__5">
    <managed-bean-name>PrintPageBean</managed-bean-name>
    <managed-bean-class>
          adf.sample.view.PrintPageBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property id="__7">
      <property-name>printableBehavior</property-name>
      <property-class>
          adf.sample.view.PrintableBehaviorBean
      </property-class>
      <value>#{PrintableBehavior}</value>
```

```
    </managed-property>
  </managed-bean>
  <managed-bean id="__6">
    <managed-bean-name>PrintableBehavior</managed-bean-name>
    <managed-bean-class>
       adf.sample.view.PrintableBehaviorBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Note the managed property that is configured on the `PrintPageBean` managed bean. The managed property makes the `PrintableBehavior` bean available to the `PrintPageBean` so that on pressing the print button, the **outcome** mode can be changed to **printable**. The `PrintableBehaviorBean` managed bean code is shown below. It is referenced from the **outcome** element in the `trinidad-config.xml` file:

```
<output-mode>#{PrintableBehavior.outcomeMode}</output-mode>
```

PrintableBehaviorBean

```
public class PrintableBehaviorBean {
      String outcomeMode = "default";
      public PrintableBehaviorBean() {
        super();
    }
    public void setOutcomeMode(String outcomeMode) {
        this.outcomeMode = outcomeMode;
    }
    public String getOutcomeMode() {
        return outcomeMode;
    }
}
```

The other bean is referenced from the ADF Faces page (though it is not a backing bean) and has a bit more code in it.

PrintPageBean

```
public class PrintPageBean {
    //managed property to hold the PrintableBehaviorBean reference
    PrintableBehaviorBean printableBehavior = null;
    String printablePageReturnURL = null;

    public PrintPageBean() {
    }


//referenced by the print button
 public String onPrint() {
   printableBehavior.setOutcomeMode("printable");
```

```
  //"print" is the navigation case I created to the
  //printable page
   return "print";
 }
  public void setPrintableBehavior(
                 PrintableBehaviorBean printableBehavior) {
        this.printableBehavior = printableBehavior;
    }

    public PrintableBehaviorBean getPrintableBehavior() {
        return printableBehavior;
    }


    public void setPrintablePageReturnURL(
                              String printablePagereturnURL) {
        this.printablePageReturnURL = printablePagereturnURL;
    }


    //generates the HTML link on the printable page
    //to return to the calling page
    public String getPrintablePageReturnURL() {
        String viewId = "/printablePage";
        ControllerContext controllerCtx = null;
        controllerCtx = ControllerContext.getInstance();
        String activityURL =
              controllerCtx.getGlobalViewActivityURL(viewId);
    return "<a href=\""+activityURL+"\+
           ">RETURN to Editable Page</a>";
   }
}
```

Some more

As mentioned, the table row selection is not shown in the printable page, for which I needed to set the table **rowSelection** property from **single** to **none** for printable pages. The following EL does the trick

```
<af:table …
         rowSelection="#{adfFacesContext.outputMode == 'printable' ?
                        'none' : 'single'}">
```

Similar, the return link is supposed to only show on the printable page:

```
<af:outputText
    escape="false"
    value="#{PrintPageBean.printablePageReturnURL}"
```

```
rendered="#{adfFacesContext.outputMode == 'printable'}"
id="gl1"/>
```

Setting the **escape** property to **false** ensures the HTML markup returned by the `PrintPageBean` is rendered on the page. The **rendered** property too uses EL to ensure it only renders when the page is displayed in printable mode.

Hope you get the idea how you can add/hide images from printable output this way. The output mode is also Java accessible using `AdfFacesContext.getCurrentInstance().getOutputMode()`.

Sample Download

You can download the simple testcase as sample 95 from the ADF Code Corner website. The sample is built with JDeveloper 11g R2 (11.1.2.1). However, the code showed and explained in this document will work in JDeveloper 11g R1 too. The sample requires you to configure the database connection to a HR schema of a local database.

RELATED DOCOMENTATION

⊠	
⊠	
⊠	