ADF Code Corner

Oracle JDeveloper OTN Harvest 01 / 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-JAN-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

Table of Content

How-to logout from ADF Security

ADF Security configures an authentication servlet, `AuthenticationServlet`, in the web.xml file that also provides a logout functionality. Developers can invoke the logout by a redirect performed from an action method in a managed bean as shown next

```
public String onLogout() {
  FacesContext fctx = FacesContext.getCurrentInstance();
  ExternalContext ectx = fctx.getExternalContext();
  String url = ectx.getRequestContextPath() +
               "/adfAuthentication?logout=true&end_url=/faces/Home.jspx";
  try {
    ectx.redirect(url);
  } catch (IOException e) {
    e.printStackTrace();
  }
  fctx.responseComplete();
  return null;
}
```

To use this functionality in your application, change the *Home.jspx* reference to a public page of yours that the user is redirected to after successful logout.

Note that for a successful logout, authentication should be through form based authentication. Basic authentication is known as browser sign-on and re-authenticates users after the logout redirect. Basic authentication is confusing to many developers for this reason.

How-to create custom component models

The ADF binding layer provides the implementation of ADF Faces component models at runtime. If however the binding does not meet the user requirement, you may want to build your own component model – e.g. for the af:query component. The ADF Faces component demo is built with a POJO model that does not leverage ADF bindings. Thus looking at the component demo source code gives you a head start in building your own component models. You can download the ADF Faces component demo and its sources from here: http://www.oracle.com/technetwork/testcontent/adf-faces-rc-demo-083799.html

How-to read context parameters in web.xml from ADF?

Context parameters in `web.xml` can be used to provide application specific configurations to ADF. To access context parameters in ADF, you use the `FacesContext` static context class as shown below.

```
FacesContext fctx = FacesContext.getCurrentInstance();
ExternalContext ectx = fctx.getExternalContext();
ServletContext servletContext = (ServletContext) ectx.getContext();
String myContextParam = null;
myContextParam = (String)servletCtx.getInitParameter("mycontextparam");
```

The `web.xml` context parameter entry looked up by the code above is shown below.

```
<context-param>
  <param-name>mycontextparam</param-name>
  <param-value>myvalue</param-value>
</context-param>
```

To access context parameters from EL, you use the following expression

```
${initParam.mycontextparam} or #{initParam.mycontextparam}
```

What's the difference between View Criteria and Where clause?

A **View Criteria** is a filter that you apply programmatically or by definition to a View Object instance. It augments the WHERE clause in a View Object query. Named **View Criteria** are defined in the Query panel of the View Object and are used

- In combination with the **af:query** component to build search forms. To do this, you drag and drop the View Criteria from the **Named View Criteria** node of the View Object in the Data Controls Panel. In the context menu, you then select the Query component – optionally with a result table
- To restrict a View Object instance in the Application Module model. For this, select a View object instance in the right hand list of the ADF Business Component Data Model panel. Use the **Edit** button to add a View Criteria to the View Object instance. This ensures that the View Object instance also runs with a query filter applied.

View Criteria use bind variables for query conditions that you want to pass in dynamically at runtime. Beside of the ability to apply View Criteria declaratively, you can apply them programmatically in Java.

A WHERE clause, if added to a View Object query by design restricts all instances of this View Object, which usually is not what developers want. Because of the benefits – and the configuration options not explained above but in the product documentation referenced below – the recommendation is to use View Criteria.

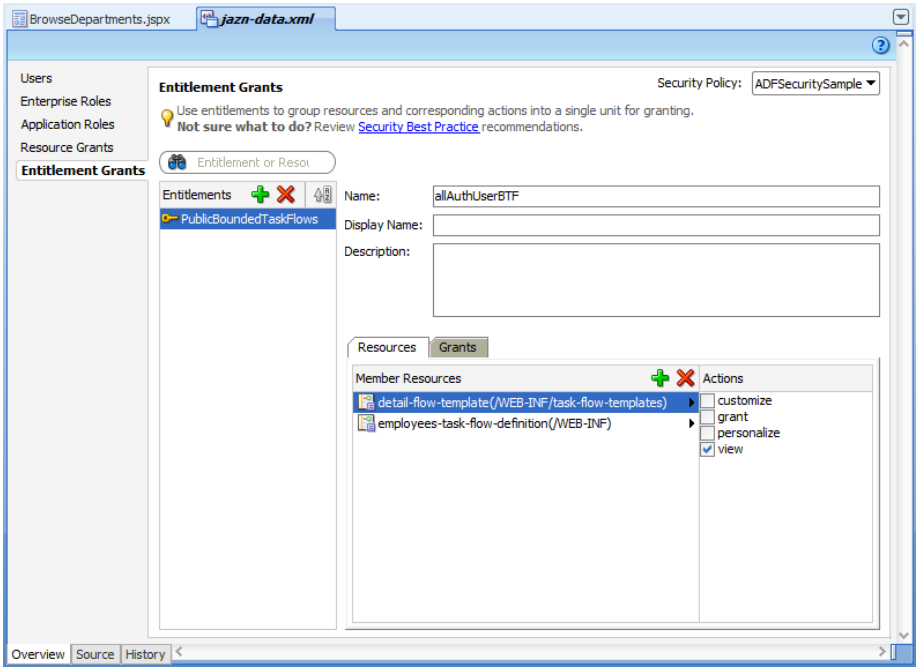
The product documentation explains View Criteria in chapter 5 of the Developer Guide:

http://download.oracle.com/docs/cd/E15523_01/web.1111/b31974/bcquerying.htm#BCGIFHHF

About Entitlement Grants in ADF Security of JDeveloper 11.1.1.4

Oracle JDeveloper 11.1.1.4 comes with a new ADF Security feature called "entitlement grants". This has nothing to do with Oracle Entitlement Server (OES) but is the ability to group resources into permission sets so they can be granted with a single grant statement. For example, as good practices when organizing your projects, you may have grouped your bounded task flows by functionality and responsibility in sub folders under the WEB-INF directory. If one of the folders holds bounded task flows that are accessible to all authenticated users, you may create an entitlement grant **allAuthUserBTF** and select all bounded task flows that are accessible for authenticated users as resources. You can then grant **allAuthUserBTF**

to the **authenticated-role** so that with only a single grant statement all selected bounded task flows are protected.



The jazn-data.xml metadata for entitlement grants is shown below

```
<permission-sets>
    <permission-set>
      <name>PublicBoundedTaskFlows</name>
      <member-resources>
       <member-resource>
        <resource-name>
            /WEB-INF/public/home-btf.xml#home-btf
        </resource-name>
        <type-name-ref>TaskFlowResourceType</type-name-ref>
        <display-name> … </display-name>
        <actions>view</actions>
       </member-resource>
       <member-resource>
        <resource-name>
              /WEB-INF/public/preferences-btf.xml#preferences-btf
       </resource-name>
        <type-name-ref>TaskFlowResourceType</type-name-ref>
        <display-name>…</display-name>
        <actions>view</actions>
       </member-resource>
      </member-resources>
```

```
      </permission-set>
  </permission-sets>
```

The grant statement for this permission set is added as shown below

```
<grant>
  <grantee>
    <principals>
      <principal>
        <name>authenticated-role</name>
        <class>oracle.security.jps.internal.core.principals.JpsAuthenticatedRoleImpl</class>
      </principal>
    </principals>
  </grantee>
  <permission-set-refs>
    <permission-set-ref>
      <name>PublicBoundedTaskFlows</name>
    </permission-set-ref>
  </permission-set-refs>
</grant>
```

Do View Object instances in shared AM share queried data?

View Object instances that belong to a shared Application Module perform a query each and don't share the data queried by one of them. Is this the correct behavior, or is a shared Application Module the wrong approach if list data should be queried only once? The truth is that the shared Application Module is not the problem because the data caching is in the domain of the View Object instances. Instances of a View Object have nothing in common except the definition they base on. The data caches are owned by the View Object usage – the instance - so that each View Object usage executes a separate query. To create a list with a single query, you need to base the list items on the same View Object instance.

How-to enable user session time out warning (JDev 11.1.1.4)

Oracle JDeveloper 11.1.1.4 contains a new session time-out warning functionality.

Quoting the *Oracle® Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework11g Release 1 (11.1.1.4.0) documentatiom*

http://download.oracle.com/docs/cd/E17904_01/web.1111/b31973/ap_config.htm#BABFIGBA

"When a request is sent to the server, a session timeout value is written to the page and the session timeout warning interval is defined by the context parameter

`oracle.adf.view.rich.sessionHandling.WARNING_BEFORE_TIMEOUT.`

The user is given the opportunity to extend the session in a warning dialog, and a notification is sent when the session has expired and the page is refreshed. Depending on the application security configuration, the user may be redirected to the log in page when the session expires.

Use the `oracle.adf.view.rich.sessionHandling.WARNING_BEFORE_TIMEOUT` context parameter to set the number of seconds prior to the session time out when a warning dialog is displayed. If the value of WARNING_BEFORE_TIMEOUT is less than 120 seconds, if client state saving is used for the page, or if the session has been invalidated, the feature is disabled. The session time-out value it taken directly from the session.

Example A-3 shows configuration of the warning dialog to display at 120 seconds before the time-out of the session.

Example A-3 Configuration of Session Time-out Warning

```
<context-param>
  <param-name>
     oracle.adf.view.rich.sessionHandling.WARNING_BEFORE_TIMEOUT
  </param-name>
  <param-value>120</param-value>
</context-param>
```

The default value of this parameter is 120 seconds. To prevent notification of the user too frequently when the session time-out is set too short, the actual value of WARNING_BEFORE_TIMEOUT is determined dynamically, where the session time-out must be more than 2 minutes or the feature is disabled.

How-to remove the close icon from task flows opened in dialogs

ADF bounded task flows can be opened in an external dialog and return values to the calling application as documented in chapter 19 of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework11g*:
http://download.oracle.com/docs/cd/E17904_01/web.1111/b31974/taskflows_dialogs.htm#BABBAFJB

Setting the task flow call activity property **Run as Dialog** to **true** and the **Display Type** property to **inline-popup** opens the bounded task flow in an inline popup. To launch the dialog, a command item is used that references the control flow case to the task flow call activity

```
<af:commandButton text="Lookup" id="cb6"
    windowEmbedStyle="inlineDocument" useWindow="true"
    windowHeight="300" windowWidth="300"
    action="lookup" partialSubmit="true"/>
```

By default, the dialog opens with a close icon in its header that does not raise a task flow return event when used for dismissing the dialog. In previous releases, the close icon could only be hidden using CSS in a custom skin definition, as explained in a previous OTN Harvest publishing (12/2010)

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/dec2010-otn-harvest-199274.pdf

As a new feature, Oracle JDeveloper 11g (11.1.1.4) provides an option to globally remove the close icon from inline dialogs without using CSS. For this, the following managed bean definition needs to be added to the adfc-config.xml file.

```
<managed-bean>
  <managed-bean-name>
    oracle$adfinternal$view$rich$dailogInlineDocument
  </managed-bean-name>
  <managed-bean-class>java.util.TreeMap</managed-bean-class>
```

```
<managed-bean-scope>application</managed-bean-scope>
  <map-entries>
    <key-class>java.lang.String</key-class>
    <value-class>java.lang.String</value-class>
    <map-entry>
      <key>MODE</key>
      <value>withoutCancel</value>
    </map-entry>
  </map-entries>
</managed-bean>
```

Note the setting of the managed bean scope to be **application** which applies this setting to all sessions of an application.

How-to detect the browser type and version

The Trinidad `RequestContext` object provides developers access to the user browser type and browser version. This information is available from Java and Expression Language

```
#{requestContext.agent.agentName}
```

```
#{requestContext.agent.agentVersion}
```

In Java, you call `RequestContext.getCurrentInstance()` to get a hold onto the request context instance before calling `getAgentName()` or `getAgentVersion()`.

The Trinidad `RequestContext` class is exposed through the `AdfFacesContext` class, which means that you can also use

```
#{adfFacesContext.agent.agentName}
```

```
#{adfFacesContext.agent.agentVersion}
```

How-to horizontally stretch UI components

Using percentages (%) in defining component width sizes in ADF Faces is a **no-go** because it fights the framework internal geometry management. However, if using *100%* for defining the component *width* is not an option, how do you tell a component to take up all the available space? The ADF Faces framework offers a pre-defined style class **AFStretchWidth** that you can use as a value of the component **styleClass** property.

```
styleClass="AFStretchWidth"
```

How-to use dependent LOV in an af:query component

Oracle ADF Business Components allows you to build dependent list of values on the attribute level, aka model driven lists. However, dragging a View Object that has dependent list of value attributes defined from the Data Controls panel and dropping it as a an af:query component to a page does not show the dependent list of values behavior in the query form it creates. For this to work, you need to set the **auto**

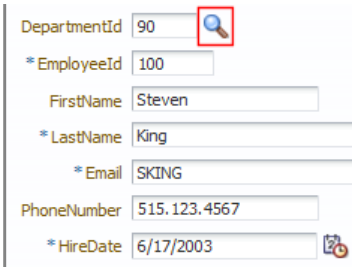
submit property on the View Object attribute that represents the master list. In his blog, Shay Shmeltzer from Oracle Product Management demonstrates this in a short video

http://blogs.oracle.com/shay/2011/01/dependent_lovs_in_an_afquery_c.html

How-to control the keyboard tab behavior in a LOV field

The keyboard tabbing behavior of list of value fields in ADF Faces is such that the focus is set to the LOV icon when navigating out of the input field.

While this behavior is convenient for those who frequently use List of Values, it's a bit annoying to users who prefer to type ahead and either click onto the LOV icon or use a keyboard shortcut to launch the select dialog if needed.



In the following, I explain how to change the default LOV behavior in ADF Faces with a little JavaScript. The modified behavior will set the focus to the next input field when the tab keyboard key is pressed. To launch the list of values dialog, users press the F2 key.

The JavaScript is added to the page using the **af:resource** tag. Note that while the JavaScript code in the sample is added in the tag body, in real life it should be referenced from an external file.

```
<af:document id="d1">
 <af:resource type="javascript">

   function tabOutOfLOVField(evt){
      keyCode = evt.getKeyCode();
      if (keyCode == AdfKeyStroke.TAB_KEY){
         var lovField = evt.getSource();
         var nextField = lovField.findComponent("it2");
         nextField.focus();
         evt.cancel();
      }
      else if(keyCode == AdfKeyStroke.F2_KEY){
        var lovField = evt.getSource();
        AdfLaunchPopupEvent.queue(lovField,true);
       evt.cancel();
     }
   }
   </af:resource>
```

```
    ...
</af:document>
```

The JavaScript code first determines the keyboard key a user pressed. If the key is the **TAB key** then the list of value field is referenced from the event object. The next navigation target, the input field to add focus to next, is found by a relative search using the input text ID property value. If the user pressed key is **F2** then the list of value field is queued for its popup launch event so the LOV dialog opens. In both cases, as there is no more to do for ADF Faces on the server side, the event is cancelled, to suppress server propagation of the keyboard action.

To execute the JavaScript code, an **af:clientListener** tag is added to the af:inputListOfValues component as shown below

```
<af:inputListOfValues id="departmentIdId" ... >
  <f:validator binding="#{bindings.DepartmentId.validator}"/>
  <af:convertNumber groupingUsed="false"
                    pattern="#{bindings.DepartmentId.format}"/>
  <af:clientListener method="tabOutOfLOVField" type="keyDown"/>
</af:inputListOfValues>
```

The client listener tag added to the input component listens for the keyboard down event and calls the JavaScript function explained earlier. For the next field, the employee Id field in this sample, to be found from JavaScript, it must have its **clientComponent** property set to true.

```
<af:inputText value="..."  clientComponent="true">
        ...
 </af:inputText>
```

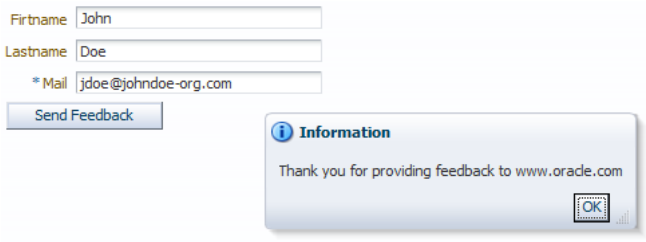
ADF Faces uses a hybrid approach in rendering its rich user interface. For best performance – unlike other Ajax frameworks – only components with a behavior render client JavaScript objects. To ensure a component is represented by a JavaScript object, an ADF Faces client side component, the component **clientComponent** attribute must be set to true or a **clientListener** tag need to be added. If you forget to set one of the two then any attempts to access the component from JavaScript fails

Note: Just as a reminder: Don't use JavaScript DOM access when referencing ADF Faces components. The browser DOM API operates on the generated markup and not the ADF Faces client component. When using JavaScript in ADF Faces, always use the ADF Faces client component API documented here

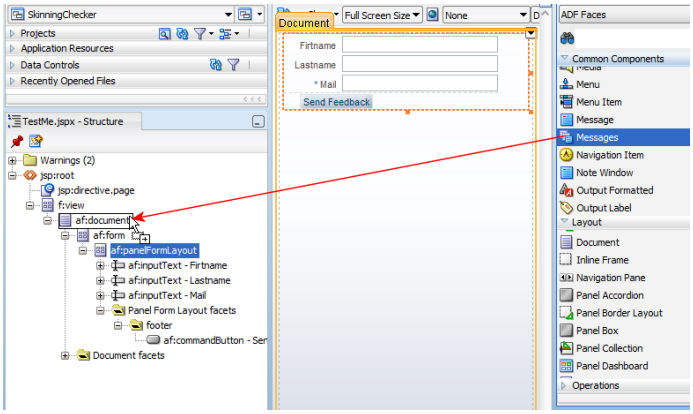
http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12046/toc.htm

How-to suppress error popup for inline messages

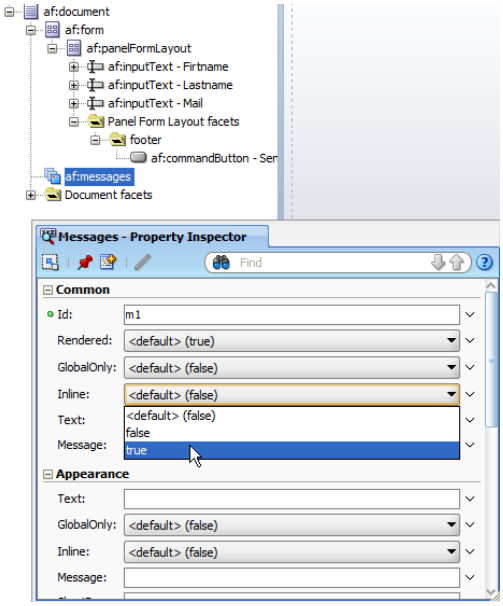
Error messages in ADF Faces are displayed in a DHTML popup dialog.



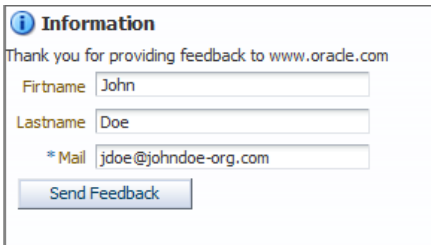
To change this behavior so messages are displayed within the page, you add the **af:messages** tag to your ADF Faces page and set its **inline** attribute to true.



After dragging the **af:messages** tag as a child to the **af:document** element, make sure you move it to become the first child. Otherwise the message doesn't show on top of the page but somewhere below.



Set the **Inline** property to **true** to change the display behavior.



With the **af:messages** tag, the message display as part of the page, as shown in the image above.

To write messages in JavaServer Faces, you use code similar the lines shown below

```
FacesContext fctx = FacesContext.getCurrentInstance();
fctx.addMessage(null, new FacesMessage(
   FacesMessage.SEVERITY_INFO, "Good Job", "Thank you for providing"+
                                           "feedback to www.oracle.com"


));
```

If the message you display is in response to an error caught during one of the early JSF request phases, you should consider adding …

fctx.renderResponse();

… after setting the message to return to the page without updating the model.

For more information see the tag documentation for the **af:messages** tag

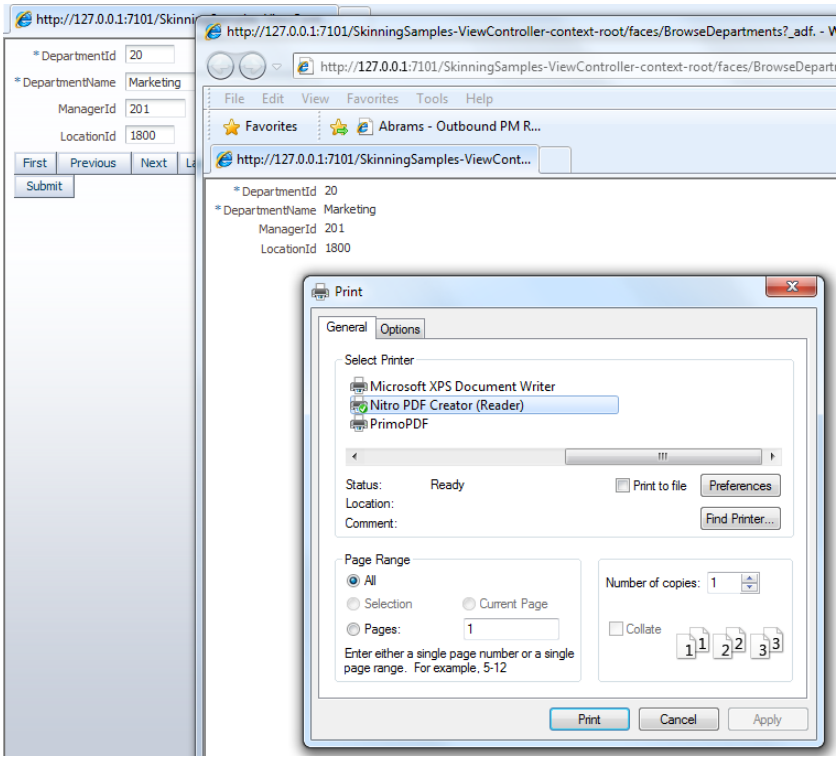
http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_messages.html

How-to launch browser print dialog when showing printable page

The `af:showPrintablePageBehavior` tag , when added as a child tag to a command component, allows developers to show a print version of the current ADF Faces page to users.

http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_showPrintablePageBehavior.html

However, still users will have to actively launch the browser print dialog to print the page. This OTN harvest entry shows you how to automatically open the print dialog from a page phase listener.



Note: While the following solution does the trick, it is using an internal request parameter to identify the printable page state. In general it is not recommended to work with ADF Faces internal APIs, as they may change in the future. For this reason, I filed an enhancement request (ER 11687447) for this or a similar parameter to be exposed as public.

The solution outlined in this harvest note uses a page phase listener defined on the `f:view` tag of the document holding the printable area.

```
<f:view beforePhase="#{BrowseDepartmentsBean.beforePhaseMethod}">
  …
</f:view>
```

The **beforePhase** property points to a managed bean method, which you create declaratively using Oracle JDeveloper. To create the managed bean and the method therein, open the Property Inspector for the `f:view` tag and press the **arrow** icon next to the **beforePhase** property. Choose **Edit** from the context menu and provide the required information in the opened dialog.

In the managed bean method, you need to listen for the **RENDER_RESPONSE** phase, which is the right time to add a little JavaScript required for launching the browser print dialog. The before phase listener code is shown here:

```
import javax.faces.context.FacesContext;
import javax.faces.event.PhaseEvent;
import javax.faces.event.PhaseId;
import org.apache.myfaces.trinidad.render.ExtendedRenderKitService;
import org.apache.myfaces.trinidad.util.Service;
…
```

```
public void beforePhaseMethod(PhaseEvent phaseEvent) {
  //only perform action if RENDER_RESPONSE phase is reached
  if (phaseEvent.getPhaseId() == PhaseId.RENDER_RESPONSE){
    FacesContext fctx = FacesContext.getCurrentInstance();
    //check internal request parameter
    Map requestMap = fctx.getExternalContext().getRequestMap();
    Object showPrintableBehavior =
      requestMap.get("oracle.adfinternal.view.faces.el.PrintablePage");
    if(showPrintableBehavior != null){
      if (Boolean.TRUE == showPrintableBehavior){
         ExtendedRenderKitService erks = null;
         erks = Service.getRenderKitService(
                           fctx,ExtendedRenderKitService.class);
         //invoke JavaScript from the server
         erks.addScript(fctx, "window.print();");
      }
    }
  }
}
```

When the printable page renders, the code above is invoked before the RENDER_RESPONSE phase to check for the `oracle.adfinternal.view.faces.el.PrintablePage` request parameter. If the parameter is set, and if its value is set to **true** then the `window.print()` command is executed on the client using the Apache Trinidad `ExtendedRenderKitService` class.

How to get product support for Oracle JDeveloper

The OTN forum for Oracle JDeveloper (http://forums.oracle.com/forums/forum.jspa?forumID=83) and Fusion Middleware products (http://forums.oracle.com/forums/category.jspa?categoryID=13), though frequently monitored by Oracle employees, is not meant to be a product support site. Thy are product forums on which, for example, Oracle JDeveloper and ADF users help each other with technical assistance and programming hints. Many user problems can be solved in this forums with pointers or little code snippets. It's a great place to go and to be for every user of Oracle software.

As good as the OTN forum can, problems found in the product are tracked down and filed as bugs or enhancement requests. Public forums however cannot replace support analysts investigating individual problems. Especially for application deployed to production where the time it takes to resolve individual problems is critical, customer support services should be used. Customer support services are accessible from here:

https://support.oracle.com

A list of support sales contacts are listed on this page, linked from the support forum

http://www.oracle.com/us/support/contact-068555.html