ADF Code Corner

86. Reading boilerplate images and icons from a JAR

twitter.com/adfcodecorner

Abstract:

Images that are a part of an application UI are usually queried from a folder within the public_html folder of the ADF Faces web project. The Fusion Order Demo (FOD) sample has an "images" directory defined that contains images and icons. For best performance the images are referenced like /images/img_name.png, avoiding to serve them through the JSF servlet.

However, what if you want to keep images separate in a different Oracle JDeveloper project or workspace for better management and sharing across applications? In this article we show how the ADF Faces resource loader can be used to read images from a Java Archive (JAR) library, which also can be configured as a shared library on WebLogic Server.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
08-JUL-2011

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

Resources in ADF Faces can be loaded by a resource loader, which is accessed from a resource servlet configured in the web.xml file of the web project. An example for resources loaded by the resource loader are skins as documented in chapter 20 of the Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework 11g Release 1

http://download.oracle.com/docs/cd/E21764_01/web.1111/b31973/af_skin.htm#CHDBEDHI

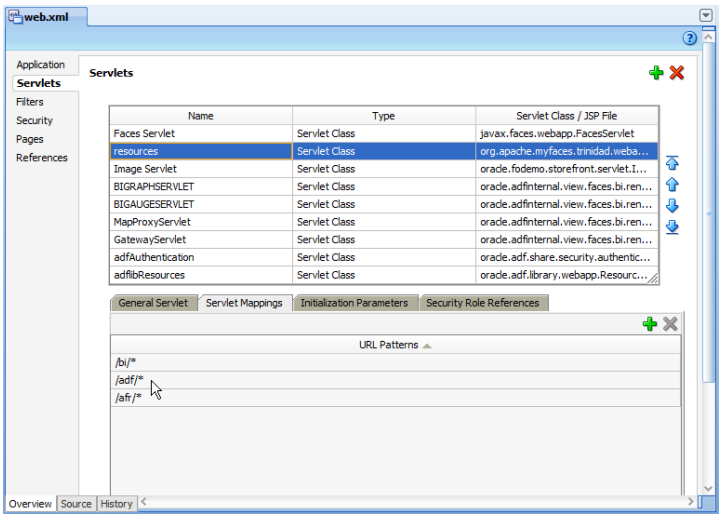
This approach is also documented on page 15 of the November 2010 OTN Harvest montly bulletin, which explains how to deploy skins as shared libraries on WLS

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/nov2010-otn-harvest-190744.pdf

Since the resource loader not only loads skin definitions (CSS) but images and JavaScript sources, too, its worth looking at this feature again in this article.

ADF Faces Resource Loading

ADF Faces loads external web resources used by components or templates, for example CSS, images and JavaScript files through a resource servlet that consults a distinct resource loader implementation.



The image above shows the servlet configuration of the Apache Trinidad `ResourceServlet`, which is mapped to the /adf/* URL path.

```
<servlet>
      <servlet-name>resources</servlet-name>
      <servlet-class>org.apache.myfaces.trinidad.webapp.ResourceServlet</servlet-class>
  </servlet>
…

<servlet-mapping>
  <servlet-name>resources</servlet-name>
   <url-pattern>/bi/*</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
    <servlet-name>resources</servlet-name>
    <url-pattern>/adf/*</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
    <servlet-name>resources</servlet-name>
    <url-pattern>/afr/*</url-pattern>
</servlet-mapping>
```

Unlike loading files from the **public_html** directory, or a sub directory within, the resource loader also knows how to load resources from JAR files located in the application classpath. The resource loader that is used by the resource servlet is determined by the servlet mapping.

For example, the servlet mapping **/adf/*** looks for its resource loader implementation in a configuration file named **adf.resources**. The entry of this file points to the distinct resource loader implementation class to be used to serve the resources referenced by this servlet path. In the case of the **adf.resources** file, the resource loader class is

```
org.apache.myfaces.trinidadinternal.resource.CoreRenderKitResourceLoader
```

The **/afr/*** servlet mapping pattern, which is used internally by the ADF Faces framework to server JavaScript and CSS files looks at a file named **afr.resources** for the resource loader to use, which is

```
oracle.adfinternal.view.resource.rich.RenderKitResourceLoader
```

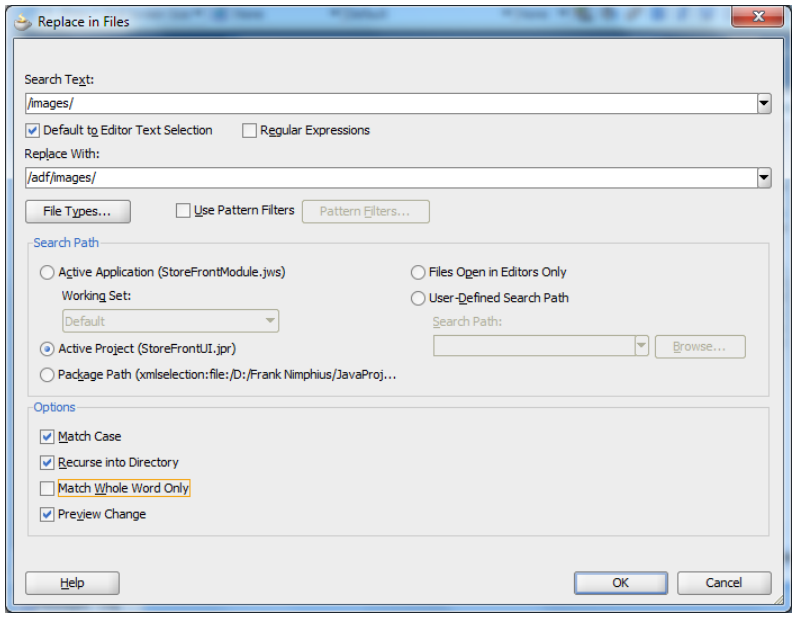
Not to mess with the ADF Faces framework internal resource loading (which would be a pity if you broke it), we suggest to use the **/adf/*** mapping for application specific image loading.

Preparing the FOD project

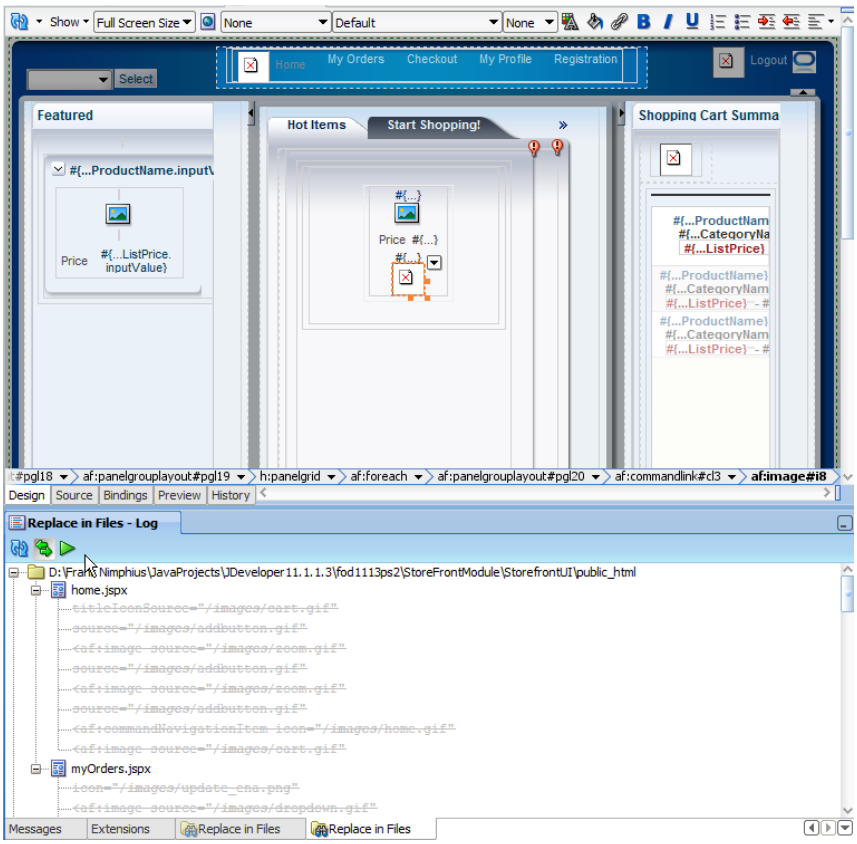
Fusion Order Demo (FOD) is an end-to-end application sample developed by Fusion Middleware Product Management.

http://www.oracle.com/technetwork/developer-tools/jdev/index-095536.html

The purpose of the demo is to demonstrate common use cases in Fusion Middleware applications. In this article we use the **StoreFrontModule** workspace that uses images as icons in its **StorefrontUI** project. To read images from JAR files instead of the file system, we need to change image references from **/images/** to **/adf/images/**, for which the search and replace function of the Oracle JDeveloper **Search | Replace in Files** menu option can be used.



Note that you need to accept the replacement for the found occurrences by pressing the green run icon on the **Replace in Files – Log** window. Once you've done so, the images on the user interface appear as broken like shown below. No worries, we are going to fix this soon.

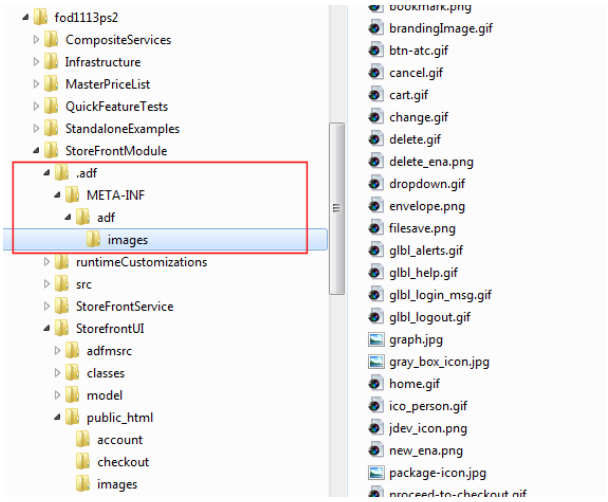


Note: FOD serves the product images from the database. These images are not to be added to the JAR file. Only boilerplate images used on command items and for branding are targeted by this approach.

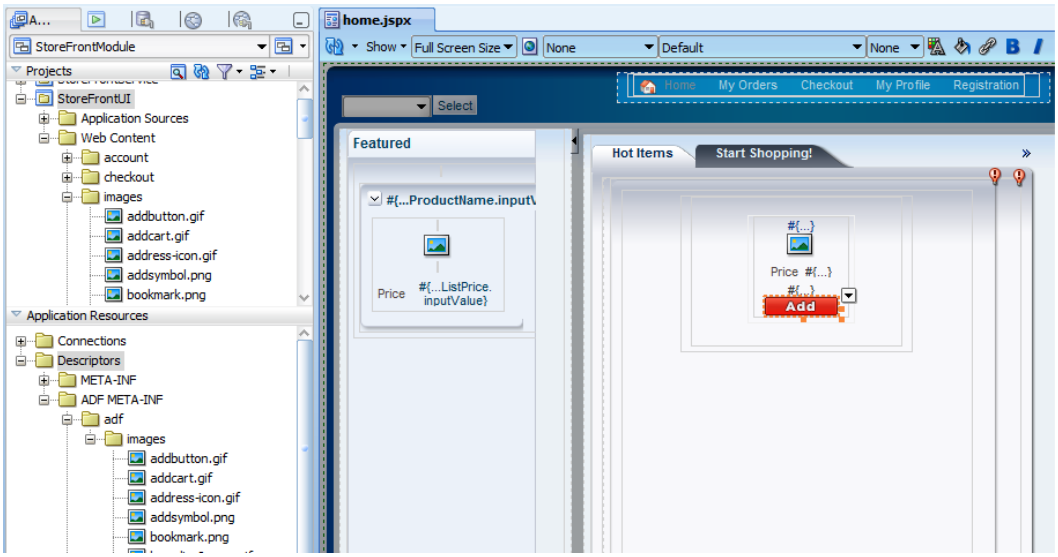
Configuration Option #1: ADF META-INF directory

Every UI project in Oracle ADF has an .adf/META-INF folder which you can use to locally store the images used in an application. The images will show in the IDE at design time and are deployed with the application within the application EAR file.

Note that while this approach is a good interim step to prepare for image sharing, which also allows version controlling the image files, it makes sense only for projects that don't yet have a final version of the images to share across applications. If you want to configure and deploy images in a shared WLS library, you need to provide them in a stand-alone JAR file.



As shown in the image above, the application boilerplate images are stored within the **adf/images** sub directory under the **.adf/META-INF** directory of the FOD StoreFrontModule application.



Because the images are accessible to the resource loader at design time as well, the visual appearance of the IDE is corrected to show boilerplate images instead of the broken image links.

Configuration Option #2: Imported JAR file

This option is what you want to do for sharing images between applications and for deploying them in WLS shared libraries. To create the JAR file, you create a temporary folder **META-INF/adf** on the file system and copy the whole **images** folder of the FOD application into it. If you later do this for your application then the folder containing your images will be the one to copy into this directory.

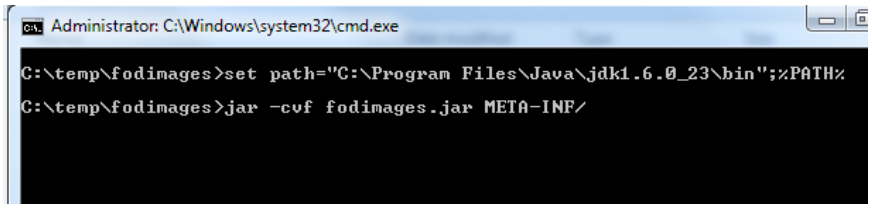
Open the command window and ensure the Java JDK (you can also reference the one used by Oracle JDeveloper) is in the path.

```
Set path="<JDK install drive>\Program Files\Java\jdk<version>\bin";%PATH%
```

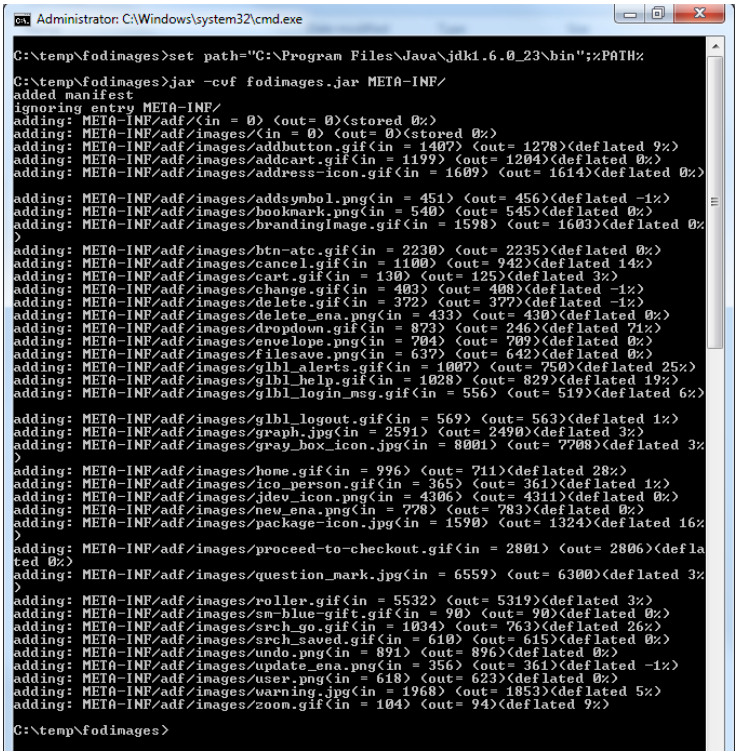
You then create the JAR file by issuing

```
jar -cvf fodimages.jar META-INF
```

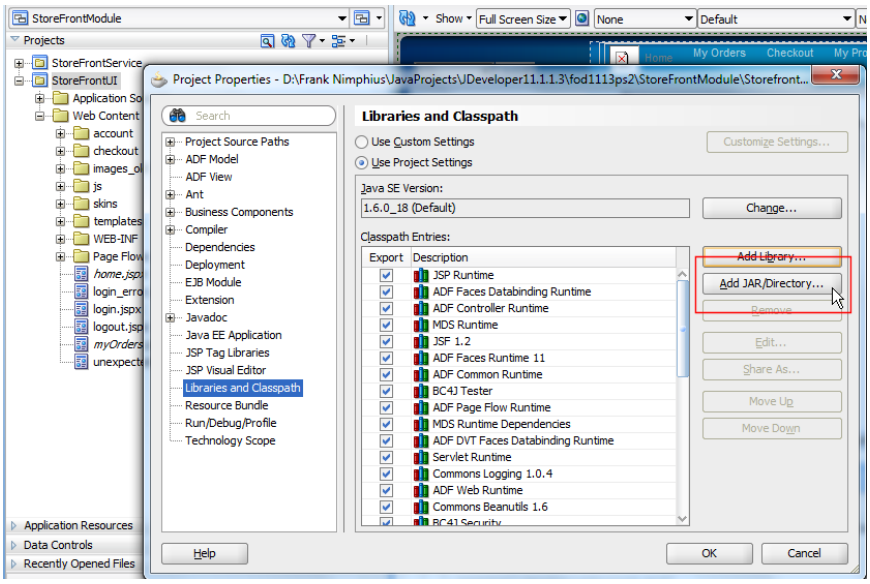
For this to work, the cursor must be in the temporary directory that holds the META-INF directory you created. In this example this directory is c:\temp



When creating the Jar file, the "v" flag produces the verbose output shown in the image below.



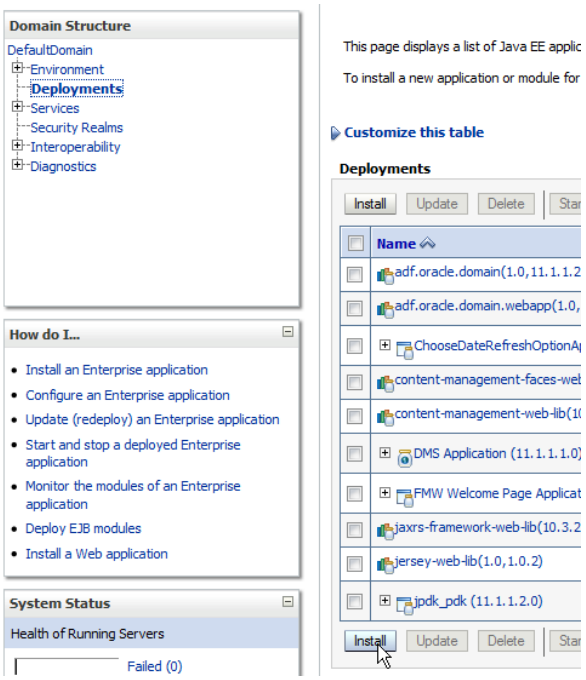
To show the images at design time, or to deploy them in the JAR file with the application, you open the view controller project properties (double click onto the StoreFrontUI project node in the FOD sample) and select **Libraries and Classpath**. Use the **Add JAR / Directory** button to find and select the JAR file to add to the project.



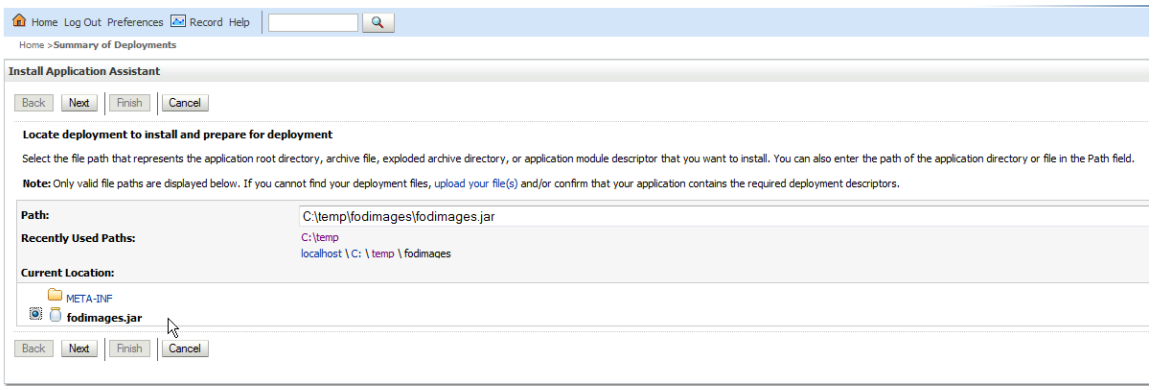
The IDE visual editor view refreshes and immediately shows the images referenced in the JAR file. While this approach provides you the option to deploy and share images in JAR files, it does not yet make it available in a shared instance.

Images in Shared Libraries

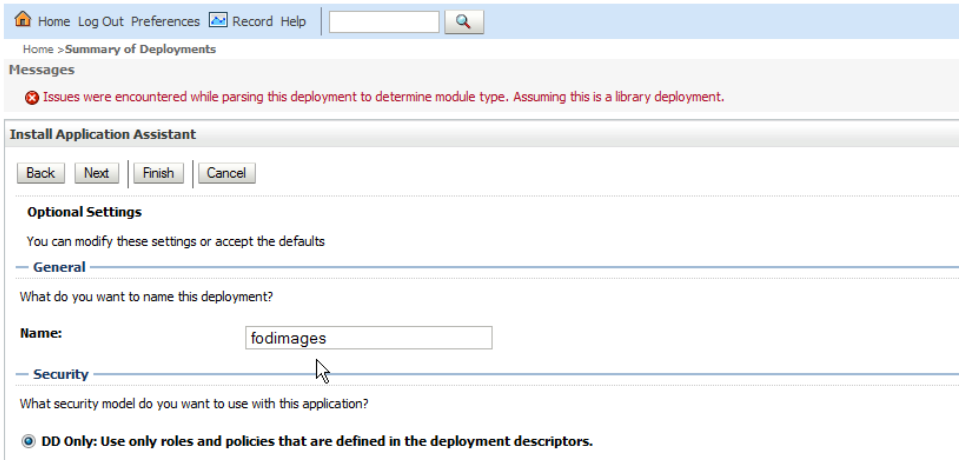
To create a shared library for the images, open the WLS console, issuing http://<host>:<port>/console in a browser URL field. Connect to the console using the **weblogic/weblogic1** username and password pair, or, in case of a changed username and password, you administrator account details.



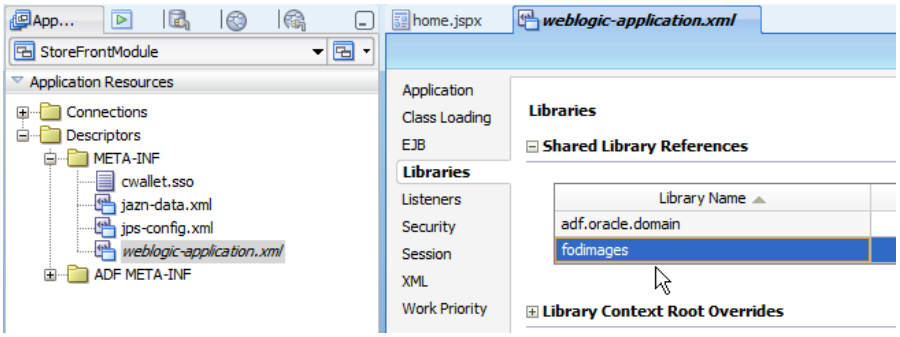
Select the **Deployments** node and press the **Install** button. Browse the file system for the JAR file that you produced following the steps in the previous section.



Copy the **Name** of the created library to the clipboard as you need it as a reference in the deployed application. The error message on top of the library selection screen can be ignored.



Once the JAR file is uploaded as a library to WLS, select the **weblogic-application.xml** file in the Oracle JDeveloper IDE for the application. The file is located in **Application Resources | Descriptors | META-INF**. Open the file with a double click and select the **Libraries** entry. Create a new line in the opened dialog and copy the name of the shared library deployment (**fodimages** in this example) as a name reference.



Note: Doing so will require this library to be installed on WLS to successfully deploy and run the application. So in case you experience deployment failures for this application on other servers, check if the library is installed.

You can now remove any direct image references in the JDeveloper project properties Library reference or the **.adf\META-INF** directory. At runtime the images are now served by the resource loader from the shared library.

Conclusion

In this article, we showed how boiler plate images can be read from a shared library to simplify resource management and enforce a consistent look and feel across applications.

We don't expect a performance boost by using this approach, given that the resource loader referenced by the **/adf/*** mapping is Apache Trinidad's `CoreRenderKitResourceLoader` which does not add extra caching or compression for images.

However, sometimes it's the knowing about what is possible that paves the bases for invention. So now you know what you can do and we leave it to your use cases to whether or not use it.

RELATED DOCOMENTATION

⊠	OTN Harvest Summary Nov 2010 http://www.oracle.com/technetwork/developer-tools/adf/learnmore/nov2010-otn-harvest-190744.pdf
⊠	
⊠	