

ADF Code Corner

022. How-to extend the default ADF Faces Component Message Bundle

ORACLE
CODE CORNER



twitter.com/adfcodecorner

Abstract:

When working with ADF Faces RC, many component uses default labels, tool tips and validation messages that are not customizable through their properties. To customize the default labels, developers need to implement a custom skin, which for this usecase does not have to define a custom look and feel. To change the default messages, like the initial "Loading..." message shown with the splash screen, you need to know about the message keys used by the components. The keys, as well as skinning guides, are available online at otn.oracle.com/products/jdev, but for the special usecase of changing the default messages, this how-to is all you need.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
21-Sep-2008

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 first experience users get with a new web application is the user interface. No matter how good the application's code quality is, how many Java EE design patterns and object oriented principles developers were able to built in, if the user interface doesn't pass the first impression, users won't like the application as a whole. While this statement greatly expresses the primary responsibility of skinning in ADF Faces RC, there is a second functionality in skinning: customizing the default component strings.

Configuring Custom Skins

To apply a custom skin to an ADF Faces RC application, you

- create a styles sheet file (CSS) containing ADF Faces RC component selectors
- create a file trinidad-skins.xml located in the WEB-INF directory of the view layer
- change the configuration in trinidad-config.xml to points to the new skin

The trinidad-skins.xml file is a registry file of all custom skins available to an application. The file doesn't exist by default as it is not needed when using the Oracle look and feel. To create this file, choose **New** from the project's context menu and create a new XML document in the view layer WEB-INF directory.

In the first releases of JDeveloper 11 there doesn't exist a dialog to help creating this document, which is why a copy and paste approach from an existing trinidad-skins.xml file is best practice to do.

The following code snippet shows a custom skin entry in the trinidad-skins.xml file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
  <skin>
    <id>coffee_sample.desktop</id>
    <family>coffee_sample</family>
    <render-kit-id>
      org.apache.myfaces.trinidad.desktop
    </render-kit-id>
    <style-sheet-name>skins/coffee_sample.css</style-sheet-name>
    <bundle-name>
      fnimphiu.sample.skin.StringBundleOverride
    </bundle-name>
  </skin>
</skins>
```

```

    <extends>blafplus-rich.desktop</extends>
  </skin>
</skins>

```

The custom skin is defined by the style sheet referenced through the **style-sheet-name** element. The CSS file must be located relative to the view layer project's `public_html` directory. In the above example, the file is in the `public_html\skins` directory. If only the message bundle should be changed then the CSS file is empty, otherwise it contains the ADF FAcEs RC component selectors with the custom style definitions. The skinning framework doesn't work with the file directly but accesses it through its **id** and **family** element. The **id** element is used when referencing existing skins that the custom skin is supposed to extend. For the usecase to only customize some or all default messages, the look and feel should be kept to the default. The **extends** element in `trinidad-skins.xml` thus references **blafplus-rich.desktop**, the id element value of the Oracle default skin. The **family** name is used to configure the custom skin so it gets applied to the running application. This can be done as a static configuration or dynamically using Expression Language.

The **bundle-name** element is key to the usecase explained in this how-to. The bundle-name element points to a custom Java class that extends `ListBundle` to provide the desired customization of the default strings.

The custom skin is configured in the `trinidad-config.xml` file as

```

<?xml version="1.0" encoding="windows-1252"?>
<trinidad-config xmlns="http://myfaces.apache.org/trinidad/config">
  <skin-family>cofee_sample</skin-family>
</trinidad-config>

```

Lets assume and showcase an online coffee shop that wants to have their tag line *"Got time? Get goffee"* shown with the splash screen while the ADF Faces RC application loads. The splash screen is a functionality of the **af|document** component, which has the following message keys defined:

<code>af_document.LABEL_SKIP_LINK_TEXT</code>	Text written out as part of link in screenreader mode to skip to the content on the page.
<code>af_document.LABEL_SPLASH_SCREEN</code>	The label for the splash screen that is displayed the first time a page is shown.
<code>af_document.MSG_FAILED_CONNECTION</code>	The error text brought up in an alert box when a connection to the server fails.

The message key highlighted in bold is the key to reference in the custom message *bundle*, `fnimphiu.sample.skin.StringBundleOverride` used above. The String bundle used with this example looks as follows

```

package fnimphiu.sample.skin;
import java.util.ListResourceBundle;
public class StringBundleOverride extends ListResourceBundle{

    public StringBundleOverride() {
    }

    @Override
    public Object[][] getContents() {
    return _CONTENTS;
    }
}

```

```
private static final Object[][] _CONTENTS =
{
  {"af_document.LABEL_SPLASH_SCREEN", "Got time? Get coffee!"}
};
}
```

As you can see, only a single component key is defined in the resource bundle, in opposite to all the 200. The other strings are taken from the default definition. This brings up the splash screen as



Extending a List bundle that extends the default bundle

No a common usecase, but doable: There already exists a customization of the messages contained in the Oracle component default list. To extend this bundle to customize the customized strings, you need to read the custom messages first before applying the changes. Using this technique comes with a bit of an overhead that plays in minimal when starting the application. The String bundle code in this case looks as follows

```
package com.oracle.adcs;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;
import oracle.adfinternal.view.faces.bi.renderkit.resource.RichBundle;

public class PartialOverrideBundle extends SingleStringBundle {
  public PartialOverrideBundle() {
    super();
  }

  @Override
  public Object[][] getContents() {
    return getCustomKeys();
  }
}
```

```
private Object[][] getCustomKeys() {

    Object[][] messages = super.getContents();
    // my overrides
    HashMap hm = new HashMap();
    for (int i = 0; i < messages.length;i++){
        hm.put(messages[i][0],messages[i][1]);
    }
    // add custom strings
    hm.put("af_document.MSG_FAILED_CONNECTION", "Crash Boom Bang");

    //renew
    messages = new Object[hm.size()][2];
    Set keySet = hm.keySet();
    Iterator keyIter = keySet.iterator();

    for (int i = 0; i < hm.size(); i++) {
        String keyStr = (String) keyIter.next();
        messages[i] = new Object[]{keyStr,hm.get(keyStr)};
    }
    return messages;
}
}
```

The above code keeps the custom Splash screen message, but adds "Crash Boom Bang" as the message shown for a failed server connection during the initial loading time. Of course, this message too could have gone into the first example, but keep in mind that this is a simplified example and the extended message bundle may not be available in source code.

RELATED DOCUMENTATION

	
	
	