JavaServer Faces – Message Resources

This tutorial explains the internationalization of a web application using JSF message resource bundle.

General

Author:
Sascha Wolski
Sebastian Hennebrueder
http://www.laliluna.de/tutorials.html – Tutorials for Struts, EJB, xdoclet and eclipse.

Date:
March, 21 2005

Software:
Eclipse 3.x
MyEclipse 3.8.x is recommended

Source code:
The sources do not contain any project files of eclipse or libraries. Create a new project following the tutorial, add the libraries as explained in the tutorial and then you can copy the sources in your new project.
http://www.laliluna.de/assets/tutorials/jsf-message-resource-source-light.zip
PDF Version
http://www.laliluna.de/assets/tutorials/jsf-message-resources-en.pdf

What are message resources

The message resources allows the developer to internationalize his web application easily. He can put labels and descriptions texts in a central file and can access them later in the JSP file or Java class. The advantage is that you can reuse the specified labels or descriptions in multiple JSP files and provide multiple language support for your web application.

Locales

A local represents a language and country combination, like *de_DE* or *en_US*. The language codes are lowercase, two-letter strings defined by the International Organization for Standardization (ISO) and the country codes are uppercase, two-letter strings, also defined by ISO. The country code is optional, so *en* alone is a valid locale string for the English language, regardless of country. If you want to have different message resource files for English in the U.K. and English in the USA, you have to add the country code (en_GB, en_US).

Configure locales

First you have to tell your JSF application which language it should support. You specifies the supported locales with the *<local-config>* element in the Faces configuration file, inside the *<application>* element.

Here's a little example how to specify the locales:

```
<application>
      <locale-config>
            <default-locale>en</default-locale>
            <supported-locale>en</supported-locale>
            <supported-locale>en_US</supported-locale>
            <supported-locale>de</supported-locale>
      </locale-config>
      <message-bundle>
            de.laliluna.tutorial.messageresource.MessageResources
      </message-bundle>
</application>
```

The element *<default-locale>* specifies the the default locale which is used if the current locale is not found. With the *<supported-locale>* element you define the locales that are supported by your web application. The *<message-bundle>* tells the application where the message resource files are located.

Message resource bundle

Message resource bundles are simply property files with key/value pairs. You specify for each string a key, which is the same for all locales. The following listing show three message resource files which supports only the English language, the English language for the USA and the German language. All files have the extension *property* and after the name the language and country code.

If you specify a message resource bundle in the Faces configuration file or in a JSP, you only specify the name of the resource file, without the language and country code and the extension *property*.

```
MessageResource_en.property
MessageResource_en_US.property
MessageResource_de.property
```

Here's a little example which shows the content of a message resource file:

```
welcome=Welcome on this site.
login={0} have been logged in on {1}.
```

The key welcome stands for a string that represent a welcome message. The second line is parameterized, and can accept two parameters that can either be hardcoded or retrieved form an object at runtime. If the two parameter were “Peter” and “10/10/2005”, the user would see “Peter have been logged in on 10/10/2005”.

Using message resource bundles

First you have to load the message resource bundle with the core tag *<f:loadBundle>*. That loads the bundle and stores it in the request scope. Here's a little example:

```
<f:loadBundle basename="myPackage.MessageResources" var="msg"/>
```

The attribute *basename* specifes the name of the resource bundle, prefixed by its location in your classpath. The variable's name is specified by the *var* attribute of the *<f:loadBundle>* element.

Now let's show how you can display a localized string of our message resource bundle with a JSF component.

```
<h:outputText value="#{msg.welcome}" />
```

It's the same way you associate a property of a backing bean with the component. This displays the string “Welcome on this site.” for the key *welcome* under the variable name *bundle*.

The *HtmlOutputFormat* component allows you to have parameterized strings that work in different locales. Usually the parameterized strings comes from a model object and may or may not be localized themselves. The first parameterized string is hardcoded, the second comes from an object.

```
<h:outputFormat value="#{msg.login}">
      <f:param value="Peter" />
      <f:param value="#{myBean.date}" />
</h:outputFormat>
```

You can also access message resource keys within a Java class. This is shown in the example below.

The example application

Create a new Java project and add the JavaServer Faces capabilities.

Utils class

We provide a class which has a static method to get a message string of a message resource bundle for a specified locale.

Create a new Java class named *Utils* in the package *de.laliluna.tutorial.messageresource.utils*.

The code starts with a utility method, *getCurrentClassLoader(..)*, that returns either the class loader for the current thread or the class loader of a specifed default object.

The method getMessageResourceString(..) returns the localized message key.

```
public class Utils {

	protected static ClassLoader getCurrentClassLoader(Object defaultObject){

		ClassLoader loader = Thread.currentThread().getContextClassLoader();

		if(loader == null){
			loader = defaultObject.getClass().getClassLoader();
		}

		return loader;
	}

	public static String getMessageResourceString(
					String bundleName,
					String key,
					Object params[],
					Locale locale){

		String text = null;

		ResourceBundle bundle =
				ResourceBundle.getBundle(bundleName, locale,
							getCurrentClassLoader
(params));

		try{
			text = bundle.getString(key);
		} catch(MissingResourceException e){
			text = "?? key " + key + " not found ??";
		}

		if(params != null){
			MessageFormat mf = new MessageFormat(text, locale);
			text = mf.format(params, new StringBuffer(), null).toString();
		}

		return text;
	}
}
```

Bean class

Create a new Java class named MyBean in the package *de.laliluna.tutorial.messageresource.bean*.

We provide getter method *getWelcomeMessage()* which returns a message string, to show you how you can use the method *getMessageResourceString(..)* of the *Utils* class within a backing bean class.

The following example shows the class *MyBean*:

```
public class MyBean {

	public String getWelcomeMessage() {

		FacesContext context = FacesContext.getCurrentInstance();
```

```
            String text = Utils.getMessageResourceString(context.getApplication
()
                        .getMessageBundle(), "welcome", null,
context.getViewRoot()
                        .getLocale());

            return text;
      }

}
```

Message Resource Bundle

Create three simple text files in the package *de.laliluna.tutorial.messageresource*:

MessageResources_de.properties

MessageResources_en.properties

MessageResources_en_US.properties

Add the following key/value pairs to each of the message resource file.

MessageResources_de.properties:

```
welcome=Willkommen auf der Seite
login={0} hat sich eingeloggt am {1}.
imagePath=/images/image_de.gif
```

MessageResources_en.properties:

```
welcome=Welcome on this site.
login={0} have be logged in on {1}.
imagePath=/images/image_en.gif
```

MessageResources_en_US.properties:

```
welcome=Welcome on this site.
login={0} have be logged in on {1}.
imagePath=/images/image_en_US.gif
```

Configure the Faces configuration file

Open the *faces-config.xml* and first add the supported locales for your application. Than add the mapping for the backing bean class MyBean.

The following source code shows the content of the *faces-config.xml*:

```
<faces-config>
      <application>
            <locale-config>
                  <default-locale>en</default-locale>
                  <supported-locale>en</supported-locale>
                  <supported-locale>en_US</supported-locale>
                  <supported-locale>de</supported-locale>
            </locale-config>
            <message-bundle>
                  de.laliluna.tutorial.messageresource.MessageResources
            </message-bundle>
      </application>

      <managed-bean>
            <managed-bean-name>myBean</managed-bean-name>
            <managed-bean-
class>de.laliluna.tutorial.messageresource.bean.MyBean</managed-bean-class>
            <managed-bean-scope>request</managed-bean-scope>
      </managed-bean>
</faces-config>
```

Output with JSP

Create a JSP file named example.jsp in the folder */WebRoot* of your project.

The following code shows the JSP file example.jsp:

```
<%@ page language="java" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
      <title>JSF message resource</title>
</head>

<body>
      <f:view>
            <f:loadBundle
basename="de.laliluna.tutorial.messageresource.MessageResources"
                  var="msg"/>

                  <h:outputText value="#{msg.welcome}" />

                  <br><br>

                  <h:outputFormat value="#{msg.login}">
                        <f:param value="Peter" />
                        <f:param value="10/03/2005" />
                  </h:outputFormat>

                  <br><br>

                  <h:graphicImage value="#{msg.imagePath}" />

                  <br><br>

                  <h:outputText value="#{myBean.welcomeMessage}" />
      </f:view>
</body>
</html>
```

First we load the message resource bundle with the *<f:loadBundle>* element.

The *<h:outputText>* element displays the message string for the key *welcome* under the variable name *msg*.

The second example shows the usage of the *<h:outputFormat>* element with parameterized strings.

You can also use a graphic path instead of a display text to display different graphic for a each locales your application supports.

The last *<h:outputText>* element associates with the method *getWelcomeMessage()* of our backing bean class and display the welcome message string. This message string comes dynamically from a method in the Java class.

Now that's all, you can deploy and test the example. We recommend a Jboss or Tomcat installation. Call the project with the following link:

http://localhost:8080/JSFMessageResource/example.faces