Localizing Output

In the Validating Input lesson, we used the validation framework to verify data submitted from a form. In the *Localizing Output* lesson, we move the validation messages to a message resource bundle.

When creating web applications, we often find ourselves using the same messages or field labels on multiple pages. We may also want to localize the messages if the application is going to be used by people of different languages.

Localizing Validation Messages and Fields

Let's add a message resource bundle and move into it the validation messages and field labels.

The Code

The framework associates message resources to classes. To add a message resource for the Logon action, we could just name the resource `Logon.properties` and set it on the classpath next to the Logon Action.

But, most people find it counter-productive to use separate message resource bundles for each class. Instead, many people prefer to add a bundle for an entire package of classes. To do this, simply add a `package.properties` file to the package. In our case, it would be the tutorial package.

tutorial/package.properties

```
requiredstring = ${getText(fieldName)} is required.
password = Password
username = User Name
```

We also need to make changes to the validator and Logon page. As you see a value in resource bundle can also be specified as an expression.

Logon-validation.xml

```
- <message>Username is required</message>
+ <message key="requiredstring"/>

- <message>Password is required</message>
+ <message key="requiredstring"/>
```

Logon.jsp

```
- <s:textfield label="User Name" name="username"/>
+ <s:textfield label="%{getText('username')}" name="username"/>

- <s:password label="Password" name="password" />
+ <s:password label="%{getText('password')}" name="password" />
```

How the Code Works

- The "key" attribute tells the validator to check for a message resource bundle.
- In the resource bundle, the expression

  ```
  ${getText(fieldName)}
  ```

  tells the framework to lookup the field name in the bundle too. This way we can use the same default message for all the `requiredstring` validators.
- Likewise, in the text filed, the expression

  ```
  %{getText('password')}
  ```

  tells the framework to lookup "password" in the message resources.

Localizing Other Messages

Other page elements can be localized too. For example, we could add the "Hello World" and the "Missing page" message to the bundle.

The Code

tutorial/package.properties

```
# ...
HelloWorld.message = Struts is up and running ...
Missing.message = This feature is under construction. Please try again in the next iteration.
```

This will work for `HelloWorld` since it is already in the tutorial package. But it won't work for the default Missing action, unless we add our own base class for the tutorial package.

TutorialSupport.java

```
package tutorial;
import com.opensymphony.xwork2.ActionSupport;
public class TutorialSupport extends ActionSupport {}
```

And update the default wildcard mapping.

struts.xml

```
<action name="*" class="tutorial.TutorialSupport">
  <result>/{1}.jsp</result>
</action>
```

Now, we can update `HelloWorld.jsp` and `Missing.jsp` to lookup the messages.

Missing.jsp

```
This feature is under construction. Please try again in the next iteration.
<s:text name="Missing.message"/>
```

In the case of HelloWorld, we set the message from the Action class. Let's update the class to use the message resource instead.

HelloWorld.java

```
package tutorial;

public class HelloWorld extends TutorialSupport {

    public static final String MESSAGE = "HelloWorld.message";

    public String execute() throws Exception {
        setMessage(getText(MESSAGE));
        return SUCCESS;
    }

  // ...
}
```

How the Code Works

- For `Missing.jsp`, we used the `text` tag to lookup the message from the resource bundle.
- For HelloWorld, we use the `getText` method in the Action class to lookup the message.
- The HelloWorld.jsp displayed the message set by the Action, so it didn't need to change at all.

What to Remember

The framework is internationalized. To localize an application, we add the resource bundles, and update some elements or tags to refer to the bundles instead of static text.

 For more, see Localization in the Core Developers Guide.

Next	Flying Solo
Prev	Validating Input