Localization

Overview

The framework supports internationalization (i18n) in the following places:

1. the UI Tags
2. Messages and Errors from the ValidationAware interface (implemented by ActionSupport and ValidationAwareSupport)
3. Within action classes that extend ActionSupport through the getText() method

Resource Bundle Search Order

Resource bundles are searched in the following order:

1. `ActionClass`.properties
2. `Interface`.properties (every interface and sub-interface)
3. `BaseClass`.properties (all the way to Object.properties)
4. ModelDriven's model (if implements ModelDriven), for the model object repeat from 1
5. package.properties (of the directory where class is located and every parent directory all the way to the root directory)
6. search up the i18n message key hierarchy itself
7. global resource properties

This is how it is implemented in a default implementation of the `LocalizedTextProvider` interface. You can provide your own implementation using `TextProvider` and `TextProviderFactory` interfaces.

Package hierarchy
To clarify #5, while traversing the package hierarchy, Struts 2 will look for a file package.properties:

com/ acme/ package.properties actions/ package.properties FooAction.java FooAction.properties

If `FooAction`.properties does not exist, `com/acme/action/package.properties` will be searched for, if not found `com/acme/package.properties`, if not found `com/package.properties`, etc.

Default action's class

If you configure action as follow

xml<action name="index"> <result>/index.jsp</result> </action>

it will use a default class defined with `default-class-ref` in `struts-default.xml` which is `com.opensymphony.xwork2.ActionSupport`. It means you have two options here to get I18N working in that case:

- define `com/opensymphony/xwork2/ActionSupport.properties` and put messages there
- point `default-class-ref` to your base class and then defined appropriated `.properties` file (corresponding to class' name or package)

Examples

There are several ways to access the message resources, including `getText`, the `text` tag, and the `i18n` tag.

Using `getText` from a Tag

To display i18n text, use a call to `getText` in the property tag, or any other tag, such as the UI tags. (The `getText` technique is especially useful for labels of UI tags.){snippet:id=i18nExample|javadoc=true|lang=xml|url=org.apache.struts2.components.Property}

The default implementation of `TextProvider` which is used in `ActionSupport` perform evaluation of value read from bundle base on the provided key, see Localizing Output for an example.

Using the `text` tag

The text tag retrieves a message from the default resource bundle.{snippet:id=i18nExample|javadoc=true|lang=xml|url=org.apache.struts2.components.Text}

Using the `I18n` tag

The i18n tag pushes an arbitrary resource bundle on to the value stack. Other tags within the scope of the i18n tag can display messages from that resource bundle.{snippet:id=i18nExample|javadoc=true|lang=xml|url=org.apache.struts2.components.I18n}

i18n with SiteMesh
Internationalizing SiteMesh decorators is possible, but there are quirks. See SiteMesh Plugin for more.

Using the Key attribute of UI Tags

The key attribute of most UI tags can be used to retrieve a message from a resource bundle:

<s:textfield key="some.key" name="textfieldName"/>

I18n Interceptor

Essentially, the i18n Interceptor pushes a locale into the ActionContext map upon every request. The framework components that support localization all utilize the ActionContext locale. See I18n Interceptor for details.

Global Resources (struts.custom.i18n.resources) in `struts.properties`

A global resource bundle could be specified programmatically, as well as the locale.

Formatting Dates and Numbers

See Formatting Dates and Numbers for more details and examples.

Comparison with Struts 1

Struts 1 users should be familiar with the application.properties resource bundle, where you can put all the messages in the application that are going to be translated. Struts 2, though, splits the resource bundles per action or model class, and you may end up with duplicated messages in those resource bundles. A quick fix for that is to create a file called ActionSupport.properties in com/opensymphony/xwork2 and put it on your classpath. This will only work well if all your actions subclass XWork2's ActionSupport.

Using only global bundles

If you don't need to use the package-scan-functionality and only base on the global bundles (those provided by the framework and via `struts.custom.i18n.resources`) you can use existing `GlobalLocalizedTextProvider` implementation. To use this please define the following option in your `struts.xml`:

xml<constant name="struts.localizedTextProvider" value="global-only" />

Custom TextProvider and TextProviderFactory

If you want use a different logic to search for localized messages, or you want to use a database or just want to search default bundles, you must implement both those interfaces (or subclass the existing implementations). You can check a small example app how to use both. Please remember that the `TextProvider` interface is implemented by the `ActioSupport` class, that's why an extra layer - `TextProviderFactory` - is needed.

Next: Type Conversion