Internationalizing and Localizing Web Applications

Internationalization is the process of preparing an application to support more than one language and data format.
Localization is the process of adapting an internationalized application to support a specific region or locale.
Examples of locale-dependent information :
messages
user interface labels
character sets and encoding
date and currency formats.
Internationalization & localization : especially important to web applications (as server side community can be totally different than client side , in the global setup)

Providing Localized Messages and Labels

Messages and labels should be tailored according to the conventions of a user's language and region.
There are two approaches to providing localized messages and labels in a web application:

1. Provide a version of the JSP page in each of the target locales and have a

controller servlet dispatch the request to the appropriate page depending on the requested locale.

Use this approach when -

large amounts of data on a page or an entire web application need to be internationalized.

2. Isolate any locale-sensitive data on a page into resource bundles, and access the data so that the corresponding translated message is fetched automatically and inserted into the page.

Thus, instead of creating strings directly in your code, you create a resource bundle that contains translations and read the translations from that bundle using the corresponding key.

Challenges in terms of internationalization arise due to differences from one country to another regarding :

spoken languages

colloquiality of the language.

different currencies

different tax laws

different forms of payment

different business rules governing the application.

There are two parts to the internationalization of a Web application.

The first part is internationalization of the application code. This involves preparing the code so that it can adapt itself to new languages and regions. In practice, this preparation involves the separation of text, labels, display messages, and any other data that is sensitive to language and region of the world. This type of adaptation of code enables generalization of the product in such a way that it can handle new languages and countries without any re-design.

The second part is localization of the application. This involves actual adaptation of the internationalized code to a specific language or region (aka locale). In practice, localization involves creation of translated text, labels, and messages, and the addition of any other application data that is specific to a certain locale.

Handling Localized Content
The important step in internationalization is the isolation of data elements that are candidates for localization. The following are some of the displayable data elements that can either be translated or formatted based on user's locale:

1. Elements that can be translated:

- Application messages
- Labels and headers on the form fields
- Online help content
- Images and icons
- Personal titles and greetings

2. Elements that can be formatted:

- Date and time
- Numbers
- Measurements (miles vs. km, etc.)
- Currencies
- Phone numbers
- Mailing address

Java provides the ResourceBundle class in the java.util package to handle translatable data elements. A resource bundle in its simplest form is a collection of key-value pairs (a key is an index to a value that can be localized or translated). Property files are typically used to specify resource bundles.

For example, a properties resource bundle "AppResources" for the default locale will have a name "AppResources.properties." However, if the property values are translated to Chinese and French locales, then the new files will have names "AppResources_zh_CN.properties" and "AppResources_fr_FR.properties," respectively. If the application uses a custom locale with some variant such as the one used in the example application "fr_US_core," then the locale suffix should also contain the variant name, for example, "AppResources_fr_US_core.properties."

Resource bundles are loaded within the application by calling the "getBundle" static method on the java.util.ResourceBundle class and passing the base name of the bundle and the current locale.

Typical Bookstore applications follow the second approach. Here are a few lines from the default resource bundle messages.BookstoreMessages.java:

```
    {"TitleCashier", "Cashier"},
    {"TitleBookDescription", "Book
    Description"},
    {"Visitor", "You are visitor number "},
    {"What", "What We're Reading"},
    {"Talk", " talks about how Web
    components can transform the way
    you develop applications for the Web.
    This is a must read for
    any self respecting Web developer!"},
    {"Start", "Start Shopping"},
```

JSTL defines tags for setting the locale for a page, creating locale-sensitive messages, and formatting and parsing data elements such as numbers, currencies, dates, and times in a locale-sensitive or customized manner.

Table 14-7 Internationalization Tags

Area	Function	Tags	Prefix

I18n	Setting Locale	setLocale requestEncoding	fmt
	Messaging	bundle message param setBundle	
	Number and Date Formatting	formatNumber formatDate parseDate parseNumber setTimeZone timeZone	

JSTL i18n tags use a localization context to localize their data. A *localization context* contains a locale and a resource bundle instance.

To specify the localization context at deployment time, you define the context parameter javax.servlet.jsp.jstl.fmt.localizationContext, whose value can be a javax.servlet.jsp.jstl.fmt.LocalizationContext or a String. A String context parameter is interpreted as a resource bundle base name.

For the Bookstore application, the context parameter is the String messages.Messages. When a request is received, JSTL automatically sets the locale based on the value retrieved from the request header and chooses the correct resource bundle using the base name specified in the context parameter.

Setting the Locale

The setLocale tag is used to override the client-specified locale for a page.

Messaging Tags

By default, the capability to sense the browser locale setting is enabled in JSTL. This means that the client determines (via its browser setting) which locale to use, and allows page authors to cater to the language preferences of their clients.

The setBundle and bundle Tags

You can set the resource bundle at runtime with the JSTL fmt:setBundle and fmt:bundle tags. fmt:setBundle is used to set the localization context in a variable or configuration variable for a specified scope. fmt:bundle is used to set the resource bundle for a given tag body.

The message Tag

The message tag is used to output localized strings. Eg. Following code is used to output a string inviting customers to choose a book from the catalog.

```
<h3><fmt:message
key="Choose"/></h3>
```

The param subtag provides a single argument (for parametric replacement) to the compound
message or pattern in its parent message tag.

Formatting Tags

JSTL provides a set of tags for parsing and formatting locale-sensitive numbers and dates.
The formatNumber tag is used to output localized numbers.

The following tag from example is used to display a localized price for a book.

```
<fmt:formatNumber
value="${book.price}" type="currency"/>
```

Note that because the price is maintained in the database in dollars, the localization is somewhat simplistic, because the formatNumber tag is unaware of exchange rates. The tag formats currencies but does not convert them.

Analogous tags for formatting dates (formatDate) and for parsing numbers and dates (parseNumber, parseDate) are also available. The timeZone tag establishes the time zone (specified via the value attribute) to be used by any nested formatDate tags.

Eg : "pretend" ship date is created and then formatted with the formatDate tag:

```
<jsp:useBean id="now"
class="java.util.Date" />
<jsp:setProperty name="now"
property="time"
  value="${now.time + 432000000}" />
<fmt:message key="ShipDate"/>
<fmt:formatDate value="${now}"
type="date"
  dateStyle="full"/>.
```