

Tutorial: Internationalization Techniques

This tutorial describes how OTN developers used Java internationalization and localization features to implement support for foreign languages in the Virtual Shopping Mall (VSM) sample application.

OTN members can [download](#) complete VSM source code and installation instructions.

Contents

1. [Concepts](#)
2. [Design](#)
3. [Required Software](#)
4. [Setup](#)
5. [Implementation](#)
6. [Resources](#)
7. [Feedback](#)





Required Software

To internationalize the VSM, OTN developers used standard Java packages provided in the J2EE SDK and the Struts SDK (both are included with Oracle9i JDeveloper). For information about downloading these packages, see the [Required Software](#) section of the *About the Virtual Shopping Mall* tutorial.





Concepts

The Virtual Shopping Mall (VSM) sample application uses functionality provided by the J2EE SDK to display messages and UI elements in various languages. Here is some background information from [Sun's Java Tutorial](#):

Internationalization is the process of designing an application so that it can be adapted to various languages and regions without engineering changes. Sometimes the term internationalization is abbreviated as i18n, because there are 18 letters between the first "i" and the last "n."

An internationalized program has the following characteristics:

- With the addition of localized data, the same executable can run worldwide.
- Textual elements, such as status messages and the GUI component labels, are not hardcoded in the program. Instead they are stored outside the source code and retrieved dynamically.
- Support for new languages does not require recompilation.
- Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.
- It can be localized quickly.

Localization is the process of adapting software for a specific region or language by adding locale-specific components and translating text. The term localization is often abbreviated as l10n, because there are 10 letters between the "l" and the "n." Usually, the most time-consuming portion of the localization phase is the translation of text. Other types of data, such as sounds and images, may require localization if they are culturally sensitive. Localizers also verify that the formatting of dates, numbers, and currencies conforms to local requirements.

The J2EE SDK provides several classes to help developers. Two of the most important classes are `java.util.Locale` and `java.util.ResourceBundle`.

The J2EE documentation says, "A Locale object represents a specific geographical, political, or cultural region. An operation that requires a Locale to perform its task is called locale-sensitive and uses the Locale to tailor information for the user. For example, displaying a number is a locale-sensitive operation--the number should be formatted according to the customs/conventions of the user's native country, region, or culture."

Java applications are typically configured using Properties files. Properties files are the basis for the ResourceBundles that Struts uses to provide message resources to an application. A ResourceBundle enables programmatic access to locale-specific objects. Typically, these objects are represented by key-value pairs stored in text files deployed with the application. For example, the VSM includes the files `OrderMessages_en.properties` and `OrderMessages_fr.properties` which store Strings in English and French, respectively.

OrderMessages_en.properties (English)

```
error.order.usernull=Username not specified
error.order.retrieval=Unable to retrieve orders
error.order.shopnull=Shop not specified
```

OrderMessages_fr.properties (French)

```
error.order.usernull=Username non indiqué
error.order.retrieval=Incapable de récupérer des ordres
error.order.shopnull=Magasin non indiqué
```

The [Implementation](#) section of this tutorial describes how OTN developers used Locale objects, ResourceBundles, and properties files to internationalize and localize the VSM.





Design

Internationalization is considered at each layer of the application:

- The VSM uses UTF-8 encoding so the application can handle a large set of characters.
- Database data is also UTF-8 encoded.
- The structure of the database tables takes in to account multilingual data. For example, the ShopDetails table can have descriptions in different languages for the same shop.

In terms of its Java implementation, the VSM uses two classes to handle the bulk of the internationalization and localization chores:

- `oracle.otnsamples.util.MessageCache` provides a set of methods to retrieve messages from resource bundles. The class reads from the file `Misc.properties` to load the set of resource bundles and creates a table of resource bundles indexed by language. A set of `getMessage` methods can access these cached resource bundles. The cache can be refreshed at run time by invoking the `refresh` method.
- `oracle.otnsamples.util.Utilities` implements methods used by other modules of the VSM. Among these utility methods is `loadParams(String file, String language)`, which creates a `ResourceBundle` by reading key-value pairs from the properties file for a specified language.

This approach enabled OTN developers to centralize the functionality, making the application modular and easier to maintain.

All hard-coded strings in the application code that contribute to the user interface are

stored in text files. These files can be maintained independently, allowing updates to the application without having to recompile, and translators don't have to touch the Java code. To support multiple languages, the Java locale name is required to be appended to the name of the corresponding text file. For example, the VSM includes the files `OrderMessages_en.properties` and `OrderMessages_fr.properties` which store Strings in English and French, respectively.

To present the user interface in the user's desired language, applications need to detect the desired locale and construct HTML content in the desired language and use the correct cultural conventions. You can determine a user's desired locale in three ways:

1. **User's Profile Information.** If there is a user profile table in a database or a directory server (LDAP) with the locale preference information, use it as the desired locale for the user.
2. **User's Input.** Allow users to specify a locale. The default locale should be English ("en").
3. **Locale of the Browser.** Get the ISO locale setting from the browser. The ISO locale of the browser is sent via the Accept-Language HTTP header. A ISO locale is composed of a ISO language and ISO country code. For example, fr-CA is the locale for French-speaking Canadian. If the Accept-Language header is null, the desired locale should default to English ("en").

The VSM uses a combination of options 2 and 3 -- the application's login page lets a user click a link to choose a language, and then the application parses the Accept-Language header to handle subsequent requests. This design offers flexibility -- a user can set and reset a language preference as often as desired during a session -- and the framework keeps track of the choice without the overhead of maintaining a profile.

The VSM also uses features provided by the Struts framework -- such as `<bean:message>` tags and `ApplicationResources.properties` files -- to display messages in the appropriate language.

Implementation details are presented in the [Implementation](#) section of this tutorial.

Other aspects of the VSM design are covered in various lessons in this [tutorial series](#).





Setup



Once you have installed and configured the required software as described in the [About the Virtual Shopping Mall](#) tutorial, no other special setup steps are required.





Implementation

The VSM user interface is defined by JSPs that combine HTML elements with other special tags that invoke custom Java code or invoke the Struts framework. When a user visits the VSM, the first page the application displays is `login.jsp`. This page gives the user a chance to click a link and specify a preferred language. (If the user doesn't click one of these links, the application uses the default language: English).



The user can return to the login page and choose a different language at any time, so how does the application keep track of the choices? HTTP is stateless: there is no way to know whether a request from a user is related to a previous request from that user. However, the servlet API supports a programmatic concept called a *session*, represented as an object that implements the `javax.servlet.http.HttpSession` interface. The servlet container ensures that requests from the same user include a session ID, so that state information saved in the session can be associated with multiple requests. This is how the VSM keeps track of a user's language preference.

The code for `login.jsp` includes the Struts tag `<html:html locale="true">`. This tag tells the container to start a session and add information about the user's language to the Accept-Language header that is sent with every subsequent HTTP request during that session.

Once a session is established and the Accept-Language header is set, application code can parse requests to get the language ID, as in this code from

`oracle.otnsamples.vsm.actions.mall.HomePageAction`:

```
// The data needed for the home page is a list of categories and
// a list of shops inside a category.
String langID = getLocale(request).getLanguage();
Category[] categories = navigation.getAllCategories(langID);
```

The VSM stores all user messages as resources in text files, not in the application code. By convention, the text files have an extension of `.properties`, and there is one file for each bundle of messages in each supported language. The VSM provides the resource files including `CartMessages_en.properties`, `EmailMessages_en.properties`, and `OrderMessages_en.properties` for messages in English, and `CartMessages_fr.properties`, `EmailMessages_fr.properties`, and `OrderMessages_fr.properties` for messages in French.

The resource files contain key-value pairs, where the keys are the same in each file, but the values are translated into the appropriate language. For example, the following listings show the contents of `CartMessages_en.properties` and `CartMessages_fr.properties`.

`CartMessages_en.properties`

```
error.cart.emptycart=There are no items in the cart  
error.checkout.outofstock=The following item/s are out of stock
```

`CartMessages_fr.properties`

```
error.cart.emptycart=Il n'y a aucun itmes dans le chariot  
error.checkout.outofstock=Les articles suivants sont pas en réserve
```

Another file, `Misc.properties`, includes a line that identifies the VSM resource files:

```
resource.files=EmailMessages_en EmailMessages_fr CartMessages_en  
CartMessages_fr OrderMessages_en OrderMessages_fr ProfileMessages_en  
ProfileMessages_fr ShopMessages_en ShopMessages_fr
```

The J2EE SDK provides classes such as `java.util.Locale` for working with resource files, and the VSM uses them to display messages in the appropriate language, as in this code from `oracle.otnsamples.util.Utilities`.

```

...
public static Properties loadParams(String file, String language)
                                throws IOException {
    // Loads a ResourceBundle and creates Properties from it
    Properties prop          = new Properties();
    ResourceBundle bundle =
        ResourceBundle.getBundle(file, new Locale(language, ""));
    Enumeration enum = bundle.getKeys();
    String key       = null;
    while(enum.hasMoreElements()) {
        key = (String) enum.nextElement();
        prop.put(key, bundle.getObject(key));
    }
    return prop;
}
...

```

For efficiency, OTN developers centralized much of the VSM message-handling in one class, `oracle.otnsamples.util.MessageCache`. This class provides methods to retrieve messages from resource bundles, for example:

```

...
/**
 * This method returns a message for a given key and locale.
 *
 * @param <b>key</b> - The message key
 * @param <b>locale</b> - The locale
 *
 * @return <b>String</b> - The corresponding message
 */
public static String getMessage(String key, Locale locale) {
    return getMessage(key, locale.getLanguage());
}
/**
 * This method returns a list of messages for a given set of keys and
 * language id.
 *
 * @param <b>keys</b> - The set of message keys
 * @param <b>langID</b> - The language id
 *
 * @return <b>String[]</b> - The corresponding set of messages.
 */
public static String[] getMessages(String[] keys, String langID) {

```

```

int len          = keys.length;
String[] values = new String[ len ];
for(int i = 0; i < len; i++) {
    values [ i ] = getMessage(keys [ i ], langID);
}
return values;
}
...

```

The files mentioned in Misc.properties are primarily to internationalize messages for the services (model) layer. The VSM also uses features provided by the Struts framework to display messages in the appropriate language. For example, the file `helpPage.jsp` includes the following Struts tag.

```
<bean:message key="help.intro"/>
```

This tag tells the framework to load the message that corresponds to the key `help.intro`. The framework knows which resource file to read because the language ID is stored with the Locale information for the session. Locale handling is done in `HeaderFilter.java`. Here's the relevant code section:

```

// Set the character encoding
request.setCharacterEncoding("UTF8");
...
// Handle locales
Locale requestedLocale = null;

String lang = request.getParameter("lang");
System.out.println("Locale requested==" + lang);
// If the user has requested for a specific language
if(lang != null) {

    // check if the locale for the requested language is in the locale table
    requestedLocale = (Locale) localeTable.get(lang);

    // if not present, create a new locale and store it in the table
    if(requestedLocale == null) {
        // Locale is created with no country
        requestedLocale = new Locale(lang, "");
        localeTable.put(lang, requestedLocale);
    }
}

```

```
// change current locale
session.setAttribute(Action.LOCALE_KEY, requestedLocale);
} else {
    if(session.getAttribute(Action.LOCALE_KEY) == null) {
        session.setAttribute(Action.LOCALE_KEY, request.getLocale());
    }
}
```

The other tutorials in [this series](#) describe various application features and explain how they were implemented.





Resources

This tutorial is part of a series based on the Virtual Shopping Mall (VSM) sample application. Following are links to resources that can help you understand and apply the concepts and techniques presented in the tutorials. See the [Required Software](#) section to obtain the VSM source code and related files.

Resource	URL
Oracle9i JDeveloper	http://otn.oracle.com/products/jdev/
JDeveloper Online Help	http://otn.oracle.com/jdeveloper903/help/
Sun's Java Tutorial on Internationalization	http://java.sun.com/docs/books/tutorial/i18n/index.html
OTN Sample Code	http://otn.oracle.com/sample_code/
J2EE 1.3 SDK	http://java.sun.com/j2ee/sdk_1.3/





Feedback

If you have questions or comments about this tutorial, you can:

- Post a message in the [OTN Sample Code discussion forum](#). OTN developers and other experts monitor the forum.
- Send email to the author. <mailto:Robert.Hall@oracle.com>

If you have suggestions or ideas for future tutorials, you can

- Post a message in the [OTN Member Feedback forum](#).
- Send email to <mailto:Raghavan.Sarathy@oracle.com>.

