ＴＲ－ＳＬＴ－００１３

Multilingual Application using Java:

Solution and Example

Nicolas Auclerc　Yves Lepage

June 2002

This report describes solutions to create multilingual applications using Java. These solutions aims at Natural Language Processing, and more precisely for the data preparation. A Java Virtual Machine (JVM) is needed to run any Java program. A JVM is embedded in the Java Runtime Environment (JRE) which defines the basic environment of the JVM (default locale, default fonts, default security policy, etc...). A default JRE installation does not contain enough settings and resources to handle different languages to be input or displayed. Therefore, this report will describe how to setup a chosen JRE to run correctly internationalized and localized multilingual applications.

（株）国際電気通信基礎技術研究所
音声言語コミュニケーション研究所
〒619－0288「けいはんな学研都市」光台二丁目２番地２　TEL：0774－95－1301

Advanced Telecommunication Research Institute International
Spoken Language Translation Research Laboratories
2-2-2 Hikaridai" Keihanna Science City" 619-0288,Japan
Telephone:+81-774-95-1301
Fax　　:+81-774-95-1308

Contents

List of Tables

List of Figures

Introduction

This report describes solutions to create multilingual applications using Java. These solutions aim at Natural Language Processing, and more precisely for the data preparation. Nevertheless this proposal can be used for more general internationalized and localized applications.

- To internationalize an application is to support different languages, like Japanese or French, in displaying and editing.
- To localize an application is to be able to display menus and messages in many languages as different users may have different mother tongues.

Nowadays, there are general methods for handling different languages in the same application. The description of a particular language is done by locale. However, locales create problems when mixing languages. For example, an application in the Japanese locale can edit and display French characters like c-cedille ç, but the French locale cannot display and edit Japanese characters.

Nevertheless, the only problem of the locale lies in the particular set of character sets used to display its language. By using the ISO/IEC 10646 (unicode) character set, we simply find a solution for most of the cases. Now, a problem is to find a complete unicode font.

Of course in any multlingual application, we have to edit languages like Japanese which have more characters than can fit on a standard alphabetical keyboard. So the Japanese characters, like other Asian languages, are input with special methods called input methods.

A Java Virtual Machime (JVM) is needed to run any Java program. A JVM is embdded in the Java RunTime Environment (JRE) which defines the basic environment of the JVM (default locale, default fonts, default security policy, etc...). A default JRE installation does not contain enough settings

and ressources to handle different languages to be input or displayed. Therefore, this report will describe how to setup a chosen JRE to run correctly internationalized and localized multilingual applications.

Chapter 1

Background

1.1 Locales

A locale is determined by the application at runtime and describes the user's environment: the local conventions, culture, and language of the user's geographical region. A locale is made up of a unique combination of a language and a country. Two examples of locales are: French/Canadian and English/U.S.

1.2 Characters and Codesets

The 8-bit ISO 8859-1 codeset has special characters needed to handle the major European languages. However, in many cases, the ISO 8859-1 font is not adequate. The 16-bit JIS 0208-0 (1983) codeset is used for Japanese. Hence each locale will need to specify which codeset they need to use and will need to have the appropriate character handling routines to cope with the codeset. This part of the locale constitutes the main use for multlingual application, because one of the main function of multlingual application is to draw characters.

- The ANSI standard uses only a single byte to represent each character, so it is limited to a maximum of 256 character and punctuation codes. Although this is adequate for the French or German, it doesn't fully support other languages.

- The Double Byte Character Set (DBCS) is used in most parts of Asia. It provides support for many different East Asian language alphabets, such as Chinese, Japanese and Korean. DBCS uses the numbers 0 to 128 to represent the ASCII character set. Some numbers greater than 128 function as lead-byte characters, which are not really characters but simply indicators that the next value is a character from a non-Latin character set. In DBCS, ASCII characters are only 1 byte in length, whereas Japanese, Korean, and other East Asian characters are 2 bytes in length.

- Unicode is a character-encoding scheme that uses 2 bytes for every character[1]. The International Standards Organization (ISO) defines a number in the range of 0 to 65,535 for every character and symbol in every language. Although both Unicode and DBCS have double-byte characters, the encoding schemes are completely different.

1.3 Input method editing styles

Each platform (Unix/X, Macintosh, Windows) supports the input of several Asian languages (e.g., Japanese, Chinese, Korean) through a special system service called an Input Method. An input method is a software component that converts keystrokes into text input which cannot be typed directly. Input methods are normally used to input text for languages which have more characters than can fit on a standard keyboard. Input methods are commonly used for Japanese, Chinese and Korean, but also show up in other languages, like Thai and Hindi. There are four basic styles of input method editing: on-the-spot, over-the-spot, off-the-spot, and root-window[2].

1.4 The Unicode Standard and ISO/IEC 10646

The Unicode Standard is fully compatible with the international standard ISO/IEC 10646-1:2000, Information Technology–Universal Multiple-Octet Coded Character Set (UCS)–Part 1: Architecture and Basic Multilingual Plane, which is also known as the Universal Character Set (UCS). The Unicode Standard also specifies a numeric value and a name for each of its

[1] To be exact, 21 bits.

[2] see Paper [Auclerc & Lepage 96]

characters. In addition to character codes and names, some other information is proposed: a character's case, directionality, and alphabetic properties. The Unicode Standard also provides case mapping tables and mappings to the repertoires of international, national, and industry character sets.

The Unicode Standard, Version 3.0, contains 49,194 characters. Scripts include the European alphabetic scripts, Middle Eastern right-to-left scripts, and scripts of Asia. The unified Han subset contains 27,484 ideographic characters defined by national and industry standards of China, Japan, Korea, Taiwan, Vietnam, and Singapore. In addition, the Unicode Standard includes punctuation marks, mathematical symbols, technical symbols, geometric shapes, and dingbats. The Unicode Standard reserves 6,400 code values in the basic 16-bit encoding for the Private Use Area, which may be used to assign codes to characters not included in the repertoire of the Unicode Standard.

Unicode provides for two encoding forms: a default 16-bit form and a byte-oriented form called UTF-8 (Unicode transformation format) that has been designed for ease of use with existing ASCII-based programming systems. Using a 16-bit encoding means that code values are available for more than 65,000 characters. While this number is sufficient for coding the characters used in languages, the Unicode Standard and ISO/IEC 10646 provide the UTF-16 extension mechanism (called surrogates), which allows for the encoding of as many as 1 million additional characters without any use of escape codes.

Chapter 2

Java Platform: the Java Virtual Machine

2.1 Choice

2.1.1 Java 2 version 1.3 for IME support

The Java 2 platform, Standard Edition (J2SE) provides cross-platform compatibility and safe network delivery. In addtion, Java introduced an Input Method Engine Service Provider Interface that enables the development of input method engines in the Java programming language and the use of Below-the-spot text input, which uses a separate composition window that is positioned automatically to be near the point where the text is to be inserted after being committed.

Java also provides the Remote Method Invocation (RMI) interface that makes it possible to create distributed Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. In addition, Java proposed the Java Native Interface (JNI), which is the native programming interface for Java. JNI allows Java code that runs within a Java Virtual Machine to operate with applications and libraries written in other languages, such as C, C++, and assembly. The invocation API also allows you to embed the Java Virtual Machine into your native applications.

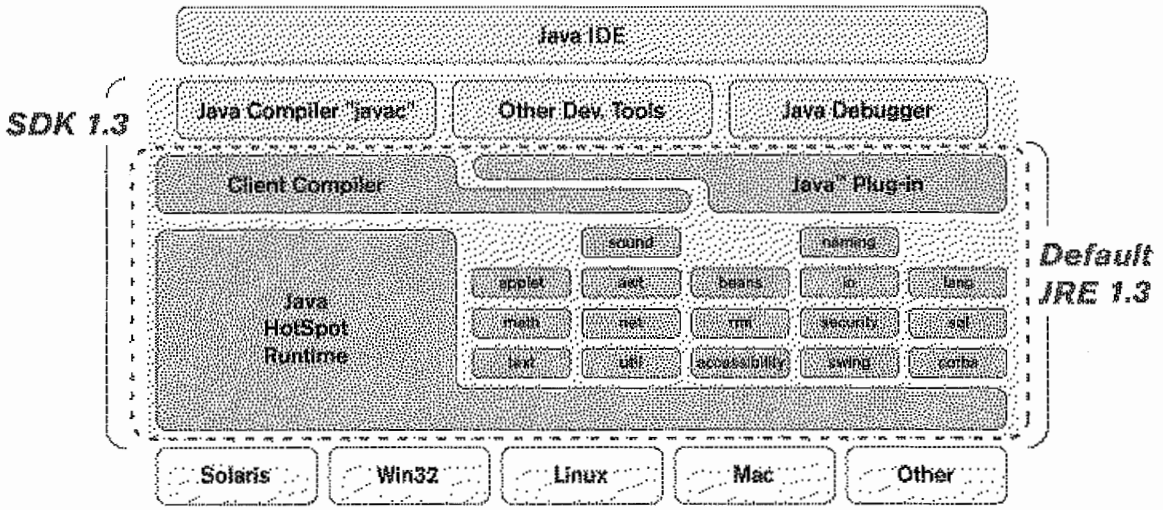

Figure 2.1: Java 2 SDK, Standard Edition v. 1.3

2.1.2 IBM Java 1.3 for fonts

The IBM Developer Kit for Linux, Java 2 Technology Edition, Version 1.3 is a development kit and runtime environment that contains IBM's just-in-time compiler and Java 2 virtual machine. The IBM Developer Kit for Linux passes Sun's Java compatibility test.

The IBM Developer Kit for Linux is provided in the following packages:

- Software Developer Kit package (SDK): The Developer Kit package is used to develop and run Java applications and applets.
- Java Runtime Environment package (JRE): The Runtime Environment package is used to run Java applications and applets or to include a Java runtime environment with your application.

In addition, with this release of the Developer Kit for Linux, IBM offers several font packages. The font packages provide additional double-byte character set (DBCS) support. The font packages are provided because Java is able to display all Unicode characters, but most versions of Linux install only the fonts needed to display the country-specific font. The font packages do not replace the system fonts.

There are two types of font packages available: Times New Roman WorldType fonts (Table 2.1) and Monotype Sans Duospace WorldType fonts (Table 2.2). There is a country-specific font for each font type.

Font	File Name	Country
Times New Roman WT J	tnrwt_j.ttf	Japan and other countries
Times New Roman WT K	tnrwt_k.ttf	Korea
Times New Roman WT SC	tnrwt_s.ttf	China (Simplified Chinese)
Times New Roman WT TC	tnrwt_t.ttf	Taiwan (Traditional Chinese)

Table 2.1: Times New Roman WorldType Fonts

Font	File Name	Country
Monotype WT J	mtsansdj.ttf	Japan and other countries
Monotype WT TK	mtsansdk.ttf	Korea
Monotype WT SC	mtsansds.ttf	China (Simplified Chinese)
Monotype WT TC	mtsansdt.ttf	Taiwan (Traditional Chinese)

Table 2.2: Monotype Sans Duospace WorldType Fonts

2.1.3 IM from Slangsoft

Slangsoft offers Spirus, a package of Input Methods and Virtual Keyboards that allow text input of non-English characters into all Java text components according to the Input Method Framework specifications of the Java 2 Standard edition version 1.3. Spirus jar files are 100% pure Java Input Methods and Virtual Keyboards for 42 National Languages, allowing text input into all Java text components of Java programs running in Java 1.3 Run Time Environment (JRE), regardless of the Operating System being used.

Input Method : This consists of Java classes that map single keys or combinations of keys to Unicode characters. Input Method classes correctly interpret the input on the QWERTY keyboard for each language as well as providing a Virtual Keyboard for the convenience of the user. For languages where a character takes on different forms depending on its position in a word,

the Input Method classes provide the logic required to convert the characters into their correct form, as well as language and conversation dictionaries.

Virtual Keyboard : This is a window in which a keyboard with fully-functional keys is displayed. Virtual Keyboards allow users to type freely in any language, even if their local keyboards do not support it.

Spirus contains a set of Java classes that are organized into language groups, housed in jar files. These jar files (Table 2.3) include the functionality of Input Methods and Virtual Keyboards.

File name	Content
European.jar	contains the IME for the European language set
HebrewArabic.jar	contains the IME for the Hebrew and Arabic language set
Chinese.jar	contains the IME for the Chinese language set
Japanese.jar	contains the IME for the Japanese language set
Korean.jar	contains the IME for the Korean language set

Table 2.3: Slangsoft Input Methods

2.2 The Java Virtual Machine: Download & Installation

The IBM Developer Kit package and Runtime Environment package for Linux can be downloaded for free from the IBM website (`http://www.ibm.com/java/jdk`). Each package is available in an installable RedHat Package Manager (RPM) file and in a compressed tape archive (TAR) file. All of the files mentioned are also available on the Java Basic Software CDROM, proposed and distributed by the Aleph Group of Department 3 of SLT.

2.2.1 Java Virtual Machine Installation

Package file name

Package	Installable RPM file name	Compressed TAR file name
Developer Kit	IBMJava2-SDK-1.3-1.0.i386.rpm	IBMJava2-SDK-13.tgz
Runtime Environment	IBMJava2-JRE-1.3-1.0.i386.rpm	IBMJava2-JRE-13.tgz

Table 2.4: Developer Kit and Runtime Environment packages

All of the files mentioned in table 2.4 are available on the Java Basic Software CDROM, proposed and distributed by the Aleph Group of Department 3 of SLT, in the directory linux/java/ibm..

Installable RPM packages

To install an installable RPM package:

- Download the installable RPM package to any directory.
- Use the rpm command to install the package, as follows:

```
rpm -i filename.rpm

where filename is the file name of the package.
```

The package is installed, by default, in the /opt/usr/IBMJava2-13 directory. If you want to install it in a different directory, use the –prefix option on the rpm command, as follows:

```
rpm -i --prefix dirname filename.rpm

where filename is the file name of the package and dirname is
the directory where you want to install the package.
```

If you use the –prefix option, the package is installed in the directory IBMJava2-13, which is created in the directory you specified.

Compressed TAR packages

To install a compressed TAR package:

- Download the compressed TAR package to the directory where you want to install it.
- Use the tar command to unpack the file, as follows:

```
tar -zxvf filename.tgz

where filename is the file name of the package.
```

The package is installed in the directory IBMJava2-13, which is created in the directory where you unpack the package.

Package directory structure

IBMJava2-13	
↪ `bin`	SDK only
↪ `lib`	SDK only
↪ `include`	SDK only
↪ `demo`	SDK only
↪ `docs`	SDK & JRE
↪ `jre`	SDK & JRE
↪ `bin`	SDK & JRE
↪ `lib`	SDK & JRE
↪ `ext`	SDK & JRE
↪ `images`	SDK & JRE
↪ `audio`	SDK & JRE
↪ `cmm`	SDK & JRE
↪ `fonts`	SDK & JRE
↪ `security`	SDK & JRE

2.2.2 IBM Font Installation

To install a font, put the font file (Tables 2.1 and 2.2) in the /opt/IBMJava2-13/jre/lib/fonts directory. If you have installed the Java Virtual Machine in a different directory, then put the file in the IBMJava2-13/jre/lib/fonts of this directory. Please refer to Tables 2.1 and 2.2, to know which file to install.

All of the files mentioned in Tables 2.1 and 2.2 are on the Java Basic Software CDROM, proposed and distributed by the Aleph Group of Departement 3 of SLT, in the directory linux/java/ibm/font.

2.2.3 SlangSoft IME Installation

To install SlangSoft IME:

- uncompress the Spirus.zip file, using unzip.
- put all of the .jar files into the IBMJava2-13/jre/lib/ext directory, as follows:

```
unzip Spirus.zip

where Spirus.zip is the real file name of the Slangsoft IME.
```

The IME jar files are created in the same directory where you unpack the Spirus.zip archive. To install an IME, put the IME file (Table 2.3) in the /opt/IBMJava2-13/jre/lib/ext directory. If you have installed the Java Virtual Machine in a different directory, then put the file in the IBMJava2-13/jre/lib/ext of this directory. Please refer to Table 2.3, to know which file to install.

All of the files mentioned in Table 2.3 are available on the Java Basic Software CDROM, proposed and distributed by the Aleph Group of Departement 3 of SLT, in the directory java/SlangSoft.

Chapter 3

Test

In this chapter, we propose a simple example. Our aim here is to introduce you to multilingual programming. We are going to step through the construction of the application piece by piece, and then, once it is all together, make it a little more interesting.

3.1 MyTestApp

The example that we are going to build will have two text components: a label and a text edit component. In this first try, the application does not have any menu or toolbox. The label component aims to show a static word.

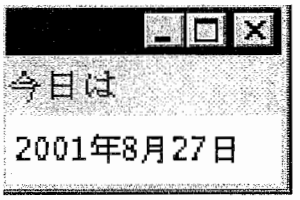

Figure 3.1: MyTestApp: a screenshot

3.2 Unicode String

In Java, the char type denotes characters in the Unicode encoding scheme. The familiar ASCII/ANSI code that is used in C programming is a subset of

Unicode. More precisely, it is the first 255 characters in the Unicode coding scheme. Thus, character codes like 'a', '1', and ']'[1] are valid Unicode characters. Unicode characters are most often expressed in terms of a hexadecimal encoding scheme that runs from '\u0000' to '\uFFFF' (with '\u0000' and '\u00FF' being the ordinary ASCII/ANSI characters). The \u prefix indicates a Unicode value, and the four hexadecimal digits provide the Unicode character.

In Java we can instantiate a label component and set its caption like this:

```
// create the label component
jLabel1 = new javax.swing.JLabel();
// set the caption of the label component with Unicode
// character
jLabel1.setText("\u4ECA\u65E5\u306F");
```

The caption "\u4ECA\u65E5\u306F" represent the word "今日は".

3.3 Locale in Java

We have already explained that Locale (see section 1.1) embraces a specific language in combination with a given cultural, geographical, and political region. In Java, in addition to the language and formatting information such as date and currency (, FFR, EUR), *localeincludes* :

- Names of the months
- Days of the week
- The first day of the week
- Collation sequences (sort order)
- Time zone information

In Java, locale information is maintained in the `java.util.Locale` object and represents:

[1]Note that 'H' is a character and "H" is a string containing a single character.

- A language
- A country or a region

Java objects and methods that modify thier behavior based on the locale are considered to be locale-sensitive. For example, DateFormat is a class for date/time formatting that formats and parses dates or time in a language-independent manner, e.g., depending on the locale given as an argument.

```
// retrieve a DateFormat object for the default locale
DateFormat df;
Locale locale=Locale.getDefault();
df= DateFormat.getDateInstance(DateFormat.FULL,locale);
// get a new instance of a text edit component
jTextPane1 = new javax.swing.JTextPane();
// set the text of the text edit component with a date using
// the DateFormat object
jTextPane1.setText(df.format(new Date()));
```

3.4 Compiling and Running MyTestApp

There are two programs for compiling and launching a Java program:

- `javac` program is the java compiler. It compiles the file MyTestApp.java into the file MyTestApp.class.
- The `java` program is the Java interpreter. It interprets the bytecodes that the compiler placed in the class file.

To compile our example, please execute these two command lines:

```
javac MyTestApp.java
java MyTestApp
```

To run a java program using the `java` command, you only need to specify the class name. For example, in this case it would be MyTestApp and not the class file name MyTestApp.class. Be sure that the `javac` and the `java`

commands are in your path. To test it:

```
which javac
which java
```

Be sure that the `javac` and the `java` commands which are in your path are the ones you have just installed. To check it, you can ask for the version of the java interpreter, as follows:

```
java -version
```

And the answer might be:

```
java version "1.3.0"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.0)
Classic VM (build 1.3.0, J2RE 1.3.0 IBM (JIT enabled:  jitc))
```

3.5 MyTestApp.java

```
import java.text.DateFormat;
import java.util.Locale;
import java.util.Date;
import java.awt.BorderLayout;

public class MyTestApp extends javax.swing.JFrame {

  public MyTestApp() {
    // create the label component
    jLabel1 = new javax.swing.JLabel();
    // set the caption of the label component with Unicode
    // character
    jLabel1.setText("\u4ECA\u65E5\u306F");

    // retrieve a DateFormat object for the default locale
    DateFormat df;
    Locale locale=Locale.getDefault();
    df= DateFormat.getDateInstance(DateFormat.FULL,locale);
    // get a new instance of a text edit component
    jTextPane1 = new javax.swing.JTextPane();
    // set the text of the text edit component with a date
    // using the DateFormat object
    jTextPane1.setText(df.format(new Date()));

    getContentPane().add(jLabel1,BorderLayout.NORTH);
    getContentPane().add(jTextPane1,BorderLayout.CENTER);
    pack ();
  }
```

```
    public static void main (String args[]) {
      new MyTestApp ().show ();
    }

    private javax.swing.JLabel jLabel1;
    private javax.swing.JTextPane jTextPane1;
}
```

In this example, the date will always appear in the language defined by the Java Virtual Machine (JVM), and so by the operating system. In the next section, we will change that.

3.6 Going Further

In this section we are going to modify our example MyTestApp. The aim is to choose directly on the command line in which language we want the date to appear. In the previous example, the DateFormat object takes the locale from the method `getDefault()` of the `Locale` class. By changing the default locale of the JVM, the MyTestApp class is not updated. The main method is just modified.

```
public static void main (String args[]) {
  if (args.length == 2)
    Locale.setDefault(new Locale(args[0],args[1]));
  new MyTestApp ().show ();
}
```

3.6.1 Locale Constructors

Two constructors for creating `Locale` objects are provided. They are:

- Locale(String language, String country)
- Locale(String language, String country, String variant)

In this new version, we work with the first constructor, in which you specify the ISO-639 language code (in lowercase) and the ISO-3166 country code (in uppercase). The Unicode website [http://www.unicode.org/unicode/onlinedat/] provides a complete listing for each of these codes.

Language	Language code	Country	Country code
Japanese	ja	Japan	JP
French	fr	France	FR
French	fr	Canada	CA
Korean	ko	Korea	KR
chinese	zn	China	CH
English	en	USA	US
English	en	United Kingdom	GB

Table 3.1: Examples of correct Locale.

3.6.2 Arguments

We grab the first two arguments from the command line and use them as language and country code, respectively, to create a `Locale` object. We use this new object to set the JVM Default Locale. If there are more or fewer than two arguments, we do not change the JVM Default Locale. Here are some examples of how you should use MyTestApp.

```
java MyTestApp
java MyTestApp fr FR
java MyTestApp ja JP
java MyTestApp ko KR
java MyTestApp en US
```

Screenshots

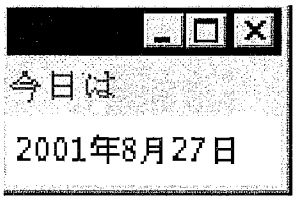

Figure 3.2: Command Line: java MyTestApp

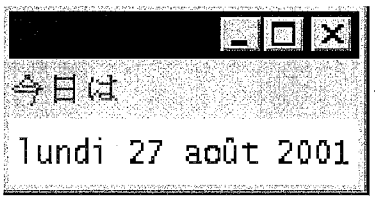

Figure 3.3: Command Line: java MyTestApp fr FR

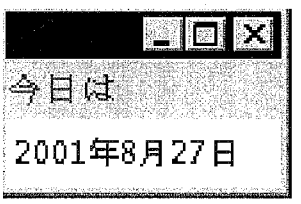

Figure 3.4: Command Line: java MyTestApp ja JP

Figure 3.5: Command Line: java MyTestApp ko KR

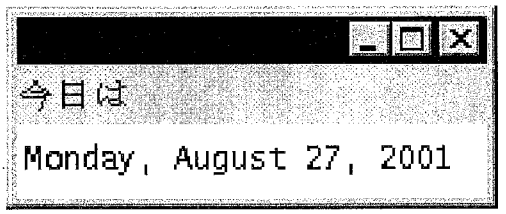

Figure 3.6: Command Line: java MyTestApp en US

Summary

In this report, we have installed, setup and tested the Java Virtual Machine. We have also explained and manipulated the `Locale`, which is a fundamental Java object for localizing a program. To know more about internationalization and localization of Java program, we recommend to read the Java Internationalization book from O'Reilly. This book also convers topics like fonts and text rendering for internationalized applications and input methods (and the Java Input Method Framework).

Bibliography

[Auclerc & Lepage 96] Nicolas Auclerc & Yves Lepage
`Boardedit`*: a Multilingual Board Editor*
From Interactive Tree Editing To Analysis by Analogy
ATR report TR-IT-0306, Kyoto, September 1999.