References

Books


*Usability and internationalization of information technology.* N. Aykin editor, LEA 2005. Chapters 1 overview, 2 what to localize, 6 icons.


References

Tutorials, slides, web pages:

- **Software localization.** Edutech wiki entry at http://edutechwiki.unige.ch/
- **Internationalization in Java.** Oracle tutorial at http://docs.oracle.com/javase/tutorial/i18n/.
- **Slides on localization in Android.** http://www.coreservlets.com/android-tutorial/. See also slides on handling screen rotations.
- **Slides and comments of MIT course 6.813** User interface design and evaluation, lecture 21.
Internationalization, \textit{i18n}

Process of designing and implementing a software application so that it can be very \textit{easily adapted} to various languages and regions without engineering changes to the programming logic.

\textit{Easily}: No need to recompile. Only one set of source code, binary, help, installer, data... is produced to support all the places in which software will be installed.

Localization, \textit{l10n}

Once a piece of software has been internationalized, is the process of adapting it for a specific language and region, that is, to a locale.

Are they different?

- Internationalization is the adaptation of products for potential use virtually everywhere.
- Localization is the addition of special features for use in a specific locale.
- Internationalization is done once per product.
- Localization is done once for each combination of product and locale.
Internationalization and localization

Locale

Collection of features of the user's environment that is dependent on language, country and region, and cultural conventions.

What kind of features? Obviously language, plus

- Language related: regional version, script (not only alphabet), writing direction, text sorting, plurals . . .
- Cultural conventions: format of date and time, calendar, numbers (decimal separator), addresses, names, person titles, currency, physical units (weight, length, size), meaning of colors and gestures, icons . . .
- Legal issues: country borders, names of seas and places . . .

<table>
<thead>
<tr>
<th>Country</th>
<th>YMD</th>
<th>DMY</th>
<th>MDY</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albania</td>
<td>YMD</td>
<td>DMY</td>
<td>MDY</td>
<td>dd-mm-yyyy</td>
</tr>
<tr>
<td>Algeria</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Argentina</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Armenia</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Austria</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Austria</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Azerbaijan</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Bahrain</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Bangladesh</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Barbados</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Basque Country</td>
<td>YMD</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Belarus</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Belgium</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Belize</td>
<td>MDY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Bolivia</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Bosnia and Herzegovina</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
<tr>
<td>Brazil</td>
<td>DMY</td>
<td></td>
<td></td>
<td>dd-mm-yyyyyyyy</td>
</tr>
</tbody>
</table>

Wikipedia entry “Date format by country”
Localization example


Localization example: MS-Word
Why

- **People from other countries are used to the [American | Western] way and they like to be [American | Western]ized.** This may seem offensive at some places. Cultural differences **do exist** and must be taken into account.

- **The world speaks English.** Only 8% to 10% of the world’s population speaks English as their primary language.

- **My users can read English, so why should I localize my software?** To avoid confusion and increase usability. Today, more and more web sites are providing content in multiple languages, and more and more software products are being designed with cultural differences in mind.

- **Target users are my country’s nationals**

  *Google Play* is available in more than 190 countries. While the exact breakdown varies by application category, in most cases more than 50 percent of application installations are downloaded from countries outside the United States on devices whose language is set to non-English.

  Japan and South Korea represent the two largest consumers of applications outside the United States.

  (...) Anecdotally, it has been found by many developers that bad translations are considered worse than no translation.

Why

- Users will ask for localized software if they need it. They expect that the software works correctly in their language using their local conventions. “Internationalization is not a feature”: it is an essential part of the software design.

Why i18n is difficult?

Can not rely on automatic translation. Meaning depends on context.

Can’t necessarily rely on bilingual members of your design team, either. They may be reasonably fluent in the other language, but not sufficiently immersed in the culture or national standards.

Actually, there is an industry of software localization.

“You are not the user” is especially true in internationalization.
Why i18n is difficult?

I18n is not only about translating (possibly large amounts of) text within a context. Also

- text format, size and reading direction
- images and icons
- the UI layout itself . . .

Many potential locales: just main Western languages plus CJK (Chinese, Japanese, Korean) may account for 10 versions of the UI.
Graphics

Generic rules for images and icons:

- Acceptable and understandable worldwide: less will have to be localized for. Draw on imagery of medicine, maths, transportation, sports, travel and traffic signs, consumer goods, business office.
- Remove text from graphics or it will need to be translated.
- Avoid political, religious and gender stereotype references.
- Avoid referring to culture-specific heroes, symbols, puns, and idioms.

- Take into account different color meanings in different cultures.
- Avoid images of people, body parts and gestures, they are subject to different interpretations even sexual or offensive.
- Show the best know version of an object.
- If there is no universal symbol for a concept, mix several versions.
- Make reading direction explicit or unimportant.
Scripts

**Script**
A collection of characters for displaying written text. One script can be used for several different languages.

Several languages may share a common script: Spanish, French, English are written in Latin script.

Some languages require multiple scripts: Japanese requires at least hiragana, katakana syllabaries and the kanji ideographs imported from China.

Japanese

Japanese writings are a mix of

- 2000 *Kanji* ideographic characters inherited from Chinese or newly created
- *Hiragana* and *Katakana* syllabaries, each with 46 symbols for the same sounds. Katakana is used to write foreign loan words, incorporated into Japanese.
- Latin numerals and characters
Japanese

Based on the Hangeul alphabet. Left to right, with spaces and same punctuation symbols. However, syllables can be made of 2, 3 or 4 characters.
The former scripts were described just to give an idea of their complexity and variety. But there are many others: Arabic, Hebrew, Indic (Devanagari, Tamil, Malayalam), Thai, Cyrillic, Greek . . .

Main features:

- Types of script: alphabetic, syllabic, ideographic plus mixtures
- Context-dependent Glyph shaping: position within a word, ligatures with other characters
- Text direction: left to right, right to left, vertical, bidirectional
- Punctuation: words can be separated or not, different punctuation symbols
Introduction Graphics Encodings Internationalization in Java Internationalization in Android

Scripts

Fonts

Character sets are ordered collections of abstract references to characters and do not specify their exact visual appearance. Fonts are collections of glyphs of a specific style and appearance for a certain script or character set.

Examples: Helvetica, Times Roman, Courier . . .

Face

A certain style determining shape details and relative character sizes for rendering several fonts.

Examples for Latin fonts: italics, bold, sans serif, small caps . . .

But also for other scripts

Brush Ḟṳţꅓ 궁체
Serif 바탕
Gothic 돈움
Encoding

Encoding or Character set

A method or system of assigning numeric values to characters.

For example, ASCII, Latin-1 or ISO 8859-1, Unicode, UTF-8, Windows 1252.

One same script can be encoded differently. Some encodings are restricted to a unique or few scripts.

ASCII

The code set that most programmers know is the American Standard Code for Information Interchange (ASCII).

7-bit code set containing:

- control characters (NUL, CR, etc.)
- upper and lowercase English letters (A-Z and a-z)
- European digits 0 . . . 9
- punctuation and other symbols # $ % & \ { } ( ) [ ] + - * / ? @ : ; , . ‘ ’ etc.

Not even sufficient for English (no £). No characters with diacritics (accents) to support Spanish, French, Catalan etc.
The International Standards Organization (ISO) created a series of 8-bit code sets to handle a much larger selection of languages by extending ASCII filling in the free slots 1 byte offers.

The ISO 8859 Character Set includes language-specific groupings:

- **ISO 8859-1** or Latin 1, Western Europe (French, German, Spanish, . . . )
  default for the Web
- **ISO 8859-2** or Latin 2, Central/Eastern Europe (Hungarian, Polish, Romanian, Croatian, . . . )
- **ISO 8859-5** Cyrillic
- **ISO 8859-6** Arabic
- **ISO 8859-7** Greek
- **ISO 8859-8** Hebrew
- **ISO 8859-9** or Latin 5, Turkish
- **ISO 8859-11** Thai
- **ISO 8859-13** or Latin 7, Baltic Rim (Estonian, Latvian, Lithuanian)
- **ISO 8859-14** or Latin 8, Celtic (Gaelic and Welsh)

Unicode

Fixed-width, 16-bit character encoding scheme.

Characters from all the world’s major scripts are uniformly supported. It is possible to combine Arabic, French, Japanese, and Russian characters all in the same string.

*Unicode provides 65,536 unique slots. Actually, the Standard has a back door that cannibalizes 2048 slots to make room for an additional 1,048,576 characters. Together ( . . . ) allows a total of 1,112,064 characters.*

*The total number of individual markings that humans have made since the beginning of time is estimated at roughly 500,000, so we should not need to upgrade to a new encoding scheme anytime soon.*

To send and receive Unicode data from systems (web browsers, email clients) that can only deal with 7 or 8-bit data, you need to use encoding scheme different from the default 2 bytes/character (Universal Character Set, UCS-2).

The Unicode Standard supports several encoding schemes known as Unicode transformation formats (UTF). UTF-8 has become the most common encoding scheme for Unicode data.

UTF-8

Unicode encoding scheme of 8-bit variable length encoding: one, two, or three bytes could represent a Unicode code point.

<table>
<thead>
<tr>
<th>UCS-2 Encoding Range</th>
<th>UTF-8 Bit Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>\u0000 to \u007F</td>
<td>0xxxxxxx</td>
</tr>
<tr>
<td>\u0080 to \u07FF</td>
<td>110xxxxx 10xxxxxx</td>
</tr>
<tr>
<td>\u0800 to \uFFFF</td>
<td>1110xxxx 10xxxxxx 10xxxxxx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text String</th>
<th>UCS-2 Encoding</th>
<th>UTF-8 Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>I speak 日本語</td>
<td>0049 0020 0073</td>
<td>49 20 73 70 65</td>
</tr>
<tr>
<td></td>
<td>0070 0065 0061</td>
<td>61 6B 20 E6 97</td>
</tr>
<tr>
<td></td>
<td>006B 0020 65E5</td>
<td>A5 E6 9C AC E8</td>
</tr>
<tr>
<td></td>
<td>672C 8A9E</td>
<td>AA 9E</td>
</tr>
</tbody>
</table>
Resource bundles

Common solution: separate what varies from what stays the same. In this case,

- the text in the labels is moved to an special text file, the **resource bundle**
- each locale supported by the application will have its own resource bundle, named after language and region, like `MessagesBundle_fr_CA.properties`
- the specific locale to use will be selected at run-time
Design the complete interface with the actual labels in your language. Here with SWT + www.cloudgarden.com’s Jigloo plugin.

The label texts go to the source code.
Resource bundles

1. Create a file with pairs (key, text) MessagesBundle.properties. It will be the default labels text.

   Here key is the dialog class name.

   Note: non Ascii characters are represented by their hexadecimal Unicode.

   ```
   NewDialog.0 = Nom
   NewDialog.1 = Cognoms
   NewDialog.2 = TeIl\u00E8fon fix
   NewDialog.3 =
   NewDialog.4 = M\u00E0bil
   NewDialog.5 = Data naixement
   NewDialog.6 = dia
   NewDialog.7 = mes
   NewDialog.8 = /
   NewDialog.9 = any
   NewDialog.10 = /
   NewDialog.11 = Adre\u00E7a
   NewDialog.12 = D'acord
   NewDialog.13 = Cancelar
   ```

2. Add variable

   ```java
   ResourceBundle Messages = ResourceBundle.getBundle("iu.MessagesBundle");
   ```
Replace constant strings of labels by key in the bundle.

Now the text of the dialog has been internationalized.

Eclipse can do this for you: Source → Externalize strings.

```
label1 = new Label(composite1, SWT.NONE);
label1.setText(Messages.getString("NewDialog.0").
label2 = new Label(composite1, SWT.NONE);
label2.setText(Messages.getString("NewDialog.1").
label3 = new Label(composite1, SWT.NONE);
label3.setText(Messages.getString("NewDialog.2").
label4 = new Label(composite1, SWT.NONE);
label4.setText(Messages.getString("NewDialog.4").
label5 = new Label(composite1, SWT.NONE);
label5.setText(Messages.getString("NewDialog.4").
label6 = new Label(composite1, SWT.NONE);
label6.setText(Messages.getString("NewDialog.5").
label7 = new Label(composite1, SWT.NONE);
label7.setText(Messages.getString("NewDialog.6").
label8 = new Label(composite1, SWT.NONE);
label8.setText(Messages.getString("NewDialog.8").
button1 = new Button(composite1, SWT.PUSH | SWT.CENTER);
button1.setText(Messages.getString("NewDialog.12").
button2 = new Button(composite1, SWT.PUSH | SWT.CENTER);
button2.setText(Messages.getString("NewDialog.13").
```
Resource bundles

Convert no Ascii characters to Unicode codes \uxxxx with JDK program native2Ascii: ų → \u00f1.

```
MessagesBundle_ko_KR.properties

NewDialog.8=\uc774\ub984
NewDialog.1=\
NewDialog.10=
NewDialog.11=\uc8fc\uc18c
NewDialog.12=\ud655\uc778
NewDialog.13=\ucde8\uc18c
NewDialog.2=\uc804\ud654 \ubc88\ud638
NewDialog.3=
NewDialog.4=\ud734\ub300 \uc804\ud654 \ubc88\ud638
NewDialog.5=\uc6dd\ub144\uc6d4\uc77c
NewDialog.6=\uc77c
NewDialog.7=\uc6d4
NewDialog.8=/
NewDialog.9=\ub144
```

Add Locale variables and select one.

```
Locale catalanLocale   = new Locale("ca","ES");
Locale espanyolLocale = new Locale("es","ES");
Locale coreaLocale    = new Locale("ko","KR");
Locale armeniLocale   = new Locale("hy","AM");

//Locale currentLocale = catalanLocale;
//Locale currentLocale = espanyolLocale;
Locale currentLocale = coreaLocale;
//Locale currentLocale = armeniLocale;

ResourceBundle Messages = ResourceBundle.getBundle("iu.MessagesBundle",
                                      currentLocale);
```
**Date and Time**

Date and time are formed by several **elements**: year, month, day of month, day of week, hour, minute, second . . . plus **delimiter** symbols (\, −, space, dot etc.) Depending on the locale the **format** and **values** of elements and separators change. In addition, each locale typically has several acceptable **several acceptable** formats for one same date or time.

The `DateFormat` class has several static factory methods that allow instantiation of a formatter object of time, date or time and date for the default locale or a given locale.

```java
DateFormat df1 = DateFormat getDateInstance(DateFormat.FULL, Locale.FRANCE);
GregorianCalendar cal = new GregorianCalendar(1965, Calendar.JUNE, 16);
System.out.println(df1.format(cal.getTime()));
Locale.setDefault(new Locale("de","DE"));
DateFormat df2 = DateFormat getTimeInstance(DateFormat.MEDIUM);
System.out.println(df2.format(new Date()));
```

**mercredi 16 juin 1965**
14:03 Uhr MEZ
Date versus Calendar

Date is a simpler class and is mainly there for backward compatibility reasons. If you need to set particular dates or do date arithmetic, use a Calendar. Calendars also handle localization.

When manipulating dates in Java, do not use methods of `java.util.Date`, such as `Date.getMonth()` or `Date.getYear()`, to build a `String`. These functions have been deprecated since JDK 1.1 a because require you to handle all date formats for different locales yourself.

Numbers

To display a number, it must be converted to a String and formatted according to local conventions. Locale-specific issues:

- **Group and decimal separators**
  - French: 123 456 fr
  - German: 123.456 de
  - English: 123,456 en_US

- **Numeric shapes**
  (Arabic, European, Indic, Thai, Chinese, etc.)
  - French: 345 987,246 fr
  - German: 345.987,246 de
  - English: 345,987.246 en_US

- **Negative numbers**
- **Currency symbols and placement**
- **Percentage symbols and placement**

<table>
<thead>
<tr>
<th>European</th>
<th>Arabic</th>
<th>Indic-Bengali</th>
<th>Thai</th>
<th>Ideographic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>٠</td>
<td>০</td>
<td>๐</td>
<td>零</td>
</tr>
<tr>
<td>1</td>
<td>١</td>
<td>১</td>
<td>๑</td>
<td>一</td>
</tr>
<tr>
<td>2</td>
<td>٢</td>
<td>২</td>
<td>๒</td>
<td>二</td>
</tr>
<tr>
<td>3</td>
<td>٣</td>
<td>৩</td>
<td>๓</td>
<td>三</td>
</tr>
<tr>
<td>4</td>
<td>٤</td>
<td>৪</td>
<td>๔</td>
<td>四</td>
</tr>
<tr>
<td>5</td>
<td>٥</td>
<td>৫</td>
<td>๕</td>
<td>五</td>
</tr>
<tr>
<td>6</td>
<td>٦</td>
<td>৬</td>
<td>๖</td>
<td>六</td>
</tr>
<tr>
<td>7</td>
<td>٧</td>
<td>৭</td>
<td>๗</td>
<td>七</td>
</tr>
<tr>
<td>8</td>
<td>٨</td>
<td>৮</td>
<td>๘</td>
<td>八</td>
</tr>
</tbody>
</table>

-1,234.

($1,234.50)
Numbers: groups and decimals

```
Integer quantity = new Integer(123456);
Double amount = new Double(345987.246);

Locale loc = new Locale("de","AT"); // Austria

NumberFormat numberFormatter;
numberFormatter = NumberFormat.getNumberInstance(loc);

quantityOut = numberFormatter.format(quantity);
amountOut = numberFormatter.format(amount);

System.out.println(quantityOut );
System.out.println(amountOut);
```

Numbers: currency

```
Double money = new Double(9876543.21);

// Locale loc = new Locale("fr","FR",
// "EURO");
// Locale loc = new Locale("de","DE",
// "EURO");
Locale loc = new Locale("en","US");

Currency cur =
    Currency.getInstance(loc);

NumberFormat cf =
NumberFormat.getCurrencyInstance(loc);

System.out.println(
    loc.getDisplayName() + ",\"" +
    cur.getDisplayName() + ":\"" +
    cf.format(money));
```

French (France), Euro:
9 876 543,21 €

German (Germany), Euro:
9.876.543,21 €

English (United States),
US Dollar:
$9,876,543.21

If all the interface text strings are static, simply externalizing them into a ResourceBundle is sufficient.

But there may be pieces of variable data that we need to insert into the middle of text before being shown:

*Flight number 4106 will begin boarding from gate 2 at 10:05 p.m.*
*Last login: Fri Sep 24 09:51:37 from calvin*
*You are visitor #9,238 to this web site since January 1, 1999*
*Volume Serial Number is 1F6E-16F1*

**Problem 1: order**

```java
int numMistakes;

Locale loc = new Locale("en","US");

ResourceBundle Messages = new ResourceBundle.getBundle(loc);

System.out.println(Messages.getString("Spelling.ThereWere")
  + numMistakes
  + Messages.getString("Spelling.Mistakes")
  + Messages.getString("Spelling.File")
  + filename);
```

There were 3 spelling mistakes in file `report.doc`. 
**Compound messages**

**Problem 1: order**

```java
int numMistakes;

Locale loc = new Locale("de","AT"); // Austria

ResourceBundle Messages = new ResourceBundle.getBundle(loc);

System.out.println(Messages.getString("Spelling.ThereWere")
    + numMistakes
    + Messages.getString("Spelling.Mistakes")
    + Messages.getString("Spelling.File")
    + filename);
```

*Datei report.doc enthält 3 Rechtschreibfehler.*

**Problem 2: concordance**

```java
Locale loc = new Locale("en","US");

ResourceBundle Messages = new ResourceBundle.getBundle(loc);

System.out.println(  
    Messages.getString("Device.What")
    + Messages.getString("Device.State"));
```

The printer is enabled. L’imprimante est activée.
The modem is enabled. Le modem est activé.
The network is enabled. Le réseau est activé.
Compound messages

The `MessageFormat` class provides a way of inserting arguments into a string, independent of the order in which the arguments appear to the user.

```java
Locale loc = new Locale("en","US");
ResourceBundle Messages =
    ResourceBundle.getBundle("MessageBundle", loc);

Object[] messageArguments = {
    Messages.getString("planet"),
    new Integer(7),
    new Date()
};
MessageFormat formatter = new MessageFormat("\n");
formatter.setLocale(currentLocale);
formatter.applyPattern(Messages.getString("template"));
System.out.println(formatter.format(messageArguments));
```

At 9:15 on January 10, 2013, we detected 7 spaceships on planet Mars.

Also, slides on Handling Screen Rotations and Other App Restarts at the same place.