Global Software

Kevin Scannell

April 8, 2002

Internationalization: the process of designing and producing software which can be easily adapted for use by anyone in the world.

Localization: the process of performing the above adaptation to an internationalized piece of software (taking into account language, cultural conventions, etc., as we'll discuss)

Internationalization is often abbreviated "I18N".

Perfect internationalization allows localization without changing the "programming" at all (indeed without even recompiling). Cultural or language-specific material is stored in appropriate data files which the internationalized software uses to "deduce" the local situation – more on this later.

The ideal is one shrink-wrapped package which will run correctly anywhere in the world.

My primary reference: Tuthill and Smallberg, *Creating Worldwide Software*, 2nd ed., 1997.

Language survey.

The 148 languages surveyed are the living languages among those having two-letter identifiers under the ISO-639-1 standard. These identifiers are used in many ways to "flag" software or web content as being in a particular language. It's hard to be a computer user in a minority language and not know your code!

Naturally the standard includes the most widely spoken languages. But (by my rough calculations) only about 5,012,168,000 of the world's 6,235,000,000 people (80.4%) are native speakers of one of these languages. Indeed lots of biggies are left out: Awadhi (20,540,000), Bhojpuri (25,000,000), Maithili (24,260,000) in India, plus Hausa (24,200,000) and Yoruba (20,000,000) in Nigeria.

There are some 6800 languages in the world, so the 148 surveyed make up a measly 2.2%.

Indeed, as it is easy merely to place a language on a list and assign it a code, it comes as no surprise that many of the 148 have no useful software available at all. Only 31 have a complete "modern" operating system available according to my unscientific survey of Windows, SUN Solaris, Mac OS, and some varieties of Linux (the grassroots nature of the latter helps the total quite a bit).

Only half of the world's population speak one of these 31 languages, leaving more than 3 billion people without access.

Why internationalize?

1. The usual reason given in books on this subject is a market-based one which I'd like to transcend here.

2. "Universal access" is the primary mission of the WWW consortium: "One of W3C's primary goals is to make these [opportunities to share knowledge] available to all people . . ." See:
   http://www.w3.org/Consortium/Points/

3. The plight of the world's minority languages. Of the 6800 languages in the world, half are "moribund" (not being passed on to children) and up to 90% are expected to become extinct in the coming century. A major problem is the unavailability of services (including software) in smaller languages which leads to encroachment of global languages (mainly English).

Why should I care about minority languages?

1. Fundamental human right to communicate in the language of one's choosing; see http://www.linguistic-declaration.org/ "Universal Declaration of Linguistic Rights"

2. Intellectual and cultural evolution depends on diversity. "McWorld" . . . "pop monoculture that reduces everything to the level of the most stupendous boredom" – Irish language poet Nuala Ní Dhomhnaill in the *NY Times Book Review*, 1994.

3. Repositories of linguistic diversity coincide very well with the remaining repositories of biological diversity (the Amazon basin, Africa, New Guinea, etc.) Activities which support indigenous peoples reinforce the preservation of both kinds of diversity; see http://www.terralingua.org/

4. Loss of knowledge – each language is a unique and unrepeatable storehouse of human knowledge: historical, artistic, medical, ecological, etc.

5. Loss to science (linguistics). We understand the way language works by looking at examples. Only a small fraction of the world's languages have been documented to an acceptable extent. If a language becomes extinct without having been recorded, it is lost forever.

Basic internationalization of messages: naturally all text output that the user sees must be available in the desired language (plus any paper documentation or packaging). If this were the only issue, internationalization would be easy:

```
main()
{
printf("Hello, world!\n");
}
```

becomes:

```
main()
{
/* read two-letter locale from environment */
setlocale(LC_MESSAGES,"");
/* specify message database for this program */
textdomain("helloworld");
/* use gettext to read the message and print */
printf(gettext("Hello, world!\n"));
}
```

So what's the problem? Well, one major issue is that you have no control over the size of the translated strings. So when you organize your menus and windows and things, you can't assume that the label "File" will need just four characters in a local version.

Even worse, not all languages use the "Latin" alphabet! There are different alphabets for Russian (Cyrillic), Greek, Armenian, Georgian, Arabic, Hebrew, Chinese, Japanese, and for virtually every different major language spoken in India (Hindi, Bengali, Telugu, Tamil, Gujarati are all different) and so on. As with Latin, a certain amount of overlap occurs: Ukrainian, Bulgarian, etc. use Cyrillic, Urdu, Persian, Pashto, etc. use Arabic.

The terminology in computer science is a *character set* – a mapping between internal representations (in terms of *bits*) and abstract "characters" (letters, numbers, punctuation, etc.) Different than a *font* which maps characters to graphical depictions.

From the early days of computing through the 1980's there was only one widely used character set in the West, called *ASCII: the American Standard Code for Information Interchange*. It was given its current form in 1968.

It represents 128 different characters: the 52 uppercase and lowercase Latin letters, 10 digits, 32 mathematical symbols and punctuation marks, and 34 other unprintable "control characters". Each is represented with 7 bits (on-off switches): 0000000, 0000001, . . . , 1111110, 1111111. This is reasonably satisfactory for English, but basically for NO other languages, even those using a Latin alphabet, because of the need for accented characters!

Now one *byte* is by definition *eight* bits, not seven. This is the basic unit of data in modern computers. This fact was exploited to extend the ASCII definition in various ways, taking into account the extra bit, and allowing 256 total characters. These different 8-bit extensions of ASCII form the ISO 8859 standard:

ISO 8859-1 (“Latin-1”): adds the accents needed for representing most Western European languages (á, à, å, â, ã, ä, æ, ø, ß, ñ, . . .)

-2: Eastern European Latin: Croatian, Czech, Hungarian, Polish, Romanian, Slovak, Slovenian (characters like ź, č, ā, ű, . . .)

-3: Maltese

-4: Estonian, Latvian, Lithuanian (ķ, ā, ė, . . .)

Four non-Latin alphabets:
-5: Cyrillic, -6: Arabic, -7: Greek, -8: Hebrew

-9: Turkish (replaces rare Icelandic characters from Latin-1 with ş, ı, and ğ)

The approach taken by ISO 8859 won't work for the ideographic alphabets of East Asia, which, in their unabridged forms, can have tens of thousands of different displayable characters. Obviously eight bits are not enough to map all of these. The most widely-used schemes currently employ a mixture of 8-bit and 16-bit characters, with escape sequences to switch between them (like TEX in a way).

The current goal is for all computers to use a single massive 16-bit character set (which means 65,536 different characters, fixed-length of two bytes each) representing all characters in all human writing systems. This project is called UNICODE.

One problem is that the data doesn't include any information about which language is being encoded, which affects things like sorting since the same character may sort differently in different languages ("ö" sorts with "o" in German but last in Swedish).

The real worry is that even 65,536 is not enough room! Indeed a high end estimate for Japanese alone may be near 60,000 characters. A certain amount of "squishing" of similar ideographs necessarily occurs. This has led to recriminations of "cultural imperialism" in East Asia and some fierce opposition to UNICODE.

As a fix, UNICODE has come up with a scheme which reserves a small collection (2048) of character codes as "surrogate pairs"; if you read a character in the first 1024 of these, then it is combined with the next character into a 32-bit character; giving a grand total of:

65536-2048 + 1024*1024 = 1,112,064

possible characters. Critics say this is no better than old variable-length schemes and wastes space with 32-bit representations, but if you use the surrogate pairs only for the rarest characters, this seems to be a viable solution.

Other language-specific difficulties:

(1) In some languages the way a character displays is context-sensitive. Arabic is like this. A simple example is the Greek σ, which, at the end of a word, looks like ς.

(2) Writing direction. Hebrew and Arabic are displayed right to left.

(3) Some languages (e.g. Thai) have no word delimiters (like spaces in Western languages). Makes writing an internationalized search function nearly impossible.

(4) Hyphenation rules vary by language. TEX has a scheme for specifying these rules in a kind of database as part of localization, but doesn't take into account unpleasantness like German: *necken* divides into *nek-* and *-ken*.

(5) Sorting. Issue noted above, but it gets worse. Can't count on sorting a character at a time (again, German "ß" is one of the culprits). Perhaps for this reason, Spanish recently changed from its historical system (where, e.g. "ch" collates after "c") to the English system. The C function `strcoll` is a version of `strcmp` which reads the locale and sorts appropriately.

(6) Dynamical messaging: inflections: "the file(s) you requested"; vocative: "You have one gold piece, Sean" vs. "A Sheáin, tá píosa óir ámhain agat"; word order:

A mish-mash of other issues:

(1) Translating single-letter mnemonics.

(2) Time and date formatting, calendars.

(3) Formatting numbers and currencies.

(4) “First” and “Last” names.

(5) Bad Images: bitmaps including text, Apple trash can, light bulb, mailbox, hand gestures, road signs.

(6) Color choices: