Unicode for Under-Resourced Languages

Daniel Yacob

The Ge'ez Frontier Foundation
7802 Solomon Seal Dr., Springfield, VA 22152, USA
yacob@geez.org

Abstract

Best known as a standard for character encoding, Unicode should now be understood as a collection of resources for textual computing that can aid solution of NLP problems. When an under-resourced language (U-RL) makes use of symbols or an entire system of writing that is not supported by electronic standards, Unicode is by design the best option for the temporary encoding the written symbols before a standard can be formalised. Leveraging Unicode for such U-RLs involves more than just symbol encoding and the NLP researcher must also anticipate developing the most basic of computer resources such as fonts, keyboard and transliteration systems. The paper describes in an overview the work entailed and goes further to cover raising the work products of a personal NLP project to the level of a national and international ICT standard where the native speaker community at large will also benefit directly.

1. What is Unicode?

Very often software engineers not familiar with text representation techniques will perceive "Unicode" as something used to support non-English fonts. Many early Unicode fonts did in fact append the term "Unicode" to their base name[1] and fonts are likely the most frequently encountered utilization of Unicode that a person will be made consciously aware of. However, Unicode is now as pervasive in applications and computer systems as text itself –aware of it or not, you are already using it and have been for some time.

"Unicode" is a very broadly used term, having many contexts, and will mean different things to different people. Historically, "Unicode" refers to the Unicode Standard which defines character encodings for the writing systems of the world. The character encoding standard is synchronised with a parallel standard known as "ISO/IEC 10646" that also addresses character encoding. "Unicode" is also used interchangeably with "ISO/IEC 10646" by all but experts.

"Unicode" most commonly refers to only the Basic Multilingual Plane (BMP) of the original 16 bit standard for encoding modern writing systems. In the right context "Unicode" may also include the 16 additional supplementary planes now a part of the Unicode Standard where historical scripts and scripts used by smaller communities are encoded.

"Unicode" may refer to the Unicode Consortium[2] the governing body that defines Unicode Standard. The consortium membership is comprised of members of the computer industry, academic institutions, and private citizens with an interest in the activities of the consortium. The International Standards Organization (ISO) and the International Electrotechnical Commission (IEC) jointly maintain the ISO/IEC 10646 standard and participants are almost entirely appointed by a government recognised national standards body. The Unicode consortium works so closely with the ISO/IEC that it has become the primary driving force behind revisions of the standard. To further distinguish between these standards bodies and their unified standards is not important unless one intends to work very deeply on defining the standards themselves.

"Unicode" may also refer to the Unicode Character Database (UCD) which defines the names and properties of characters. The database is critical for text processing tools such as regular expressions libraries and layout engines. Similarly, "Unicode" may refer to a collection of annexes, technical reports and technical standards that further define textual properties and behaviours such as text boundaries, collation and equivalence classes.

"Unicode" may refer to the projects under the Unicode Consortium such as the Common Locale Data Repository (CLDR) that defines basic vocabulary and cultural conventions used in operating systems. Likewise "Unicode" can refer to the International Components for Unicode (ICU), a project involving many of the same people, which implements most of the specifications of the Unicode Consortium and data such as CLDR.

Finally, "Unicode" may refer to the Unicode home page portal, mail lists, community, technical committee (UTC), twice annual International Unicode Conference (IUC), or the process by which these aspects work together to further develop the Unicode Standard and its family of specifications.

For the purposes of this paper "Unicode" will refer then to the family of standards and technologies associated with the Unicode Consortium that can be utilised for working with a written language in a computer environment. For the NLP researcher, "Unicode" need be no more than a means to some other end. How Unicode can be applied to help solve problems in NLP, and in particular for Under-Resourced Languages (U-RLs) that may not yet be supported in Unicode, will be the focus of this paper.

2. Working *With* Unicode

More than anything else, what Unicode offers NLP is a resource for representing written languages. To a lesser degree, Unicode can also be an aid for tokenizing spoken language with phonetic symbology and for text corpora processing from comprehensive properties provided for all written symbols.

[1] e.g. in Microsoft Windows the "Lucida Sans Unicode" font is the companion to "Lucida Sans" and "Arial Unicode MS" the companion to "Arial".

[2] Unicode Consortium Homepage: http://unicode.org/

A review of software under the Natural Language Software Registry[3] reveals that many applications have already migrated to Unicode which makes them accessible to U-RLs already. Typically, Unicode based tools are designed to be language neutral and will be able to process language specific information through modules (classes, xml data, etc) containing settings and rules applicable for that language. So adapting a tool for a specific language is often a matter of developing the needed module.

NLP tools and applications are most often engineered to work on a very specific problem set so it will not be the purpose here to recommend one NLP resource over another. Rather the objective is to make an overview of the most likely tasks entailed to customise an existing application to support a given language via Unicode as well to leverage Unicode in applications that you author yourself.

2.1. Operating Systems

Applying Unicode, the first thing that you will need is a recent computer operating system (OS) with native support for the encoding standard. Fortunately, this side of the 21st century it is harder and harder to find an operating system that does *not* support Unicode. The newer the operating system the better but you will want to consider upgrading if your computer runs with Apple MacOS earlier than version 9.2, Microsoft Windows other than CE, NT, XP, Vista or 2000, Solaris earlier than 2.8, or GNU/Linux with glibc earlier than 2.2.2.

Linux systems are provided by many different distributors with as many different configurations and so deserve a little more attention. To determine if the version of glibc is of version 2.2.2 or later enter the following at the command line:

```
% ls –l /lib/libc.so.* /lib/libgtk*
```

which should return a response containing libraries similar to:

```
lrwxr-xr-x  1 root root  13 Nov 29
2004 /lib/libc.so.6 -> libc-2.3.2.so*
```

libc.so.6 is a symbolic link to the current version of glibc on the system which in the above example is version 2.3.2. If the operating system is not at the version level indicated here this does not mean that you will be unable to work with Unicode. The consequence will be that you are less assured that you will be able to view or enter Unicode text in terminals or use it with some applications. You may in fact still apply the encoding for your needs but there are fewer guarantees and you will likely have to try several editor applications before finding one that can display Unicode text as expected (consider Yudit discussed later).

The window desktop environment in a GNU/Linux system is entirely separate from the OS. If your system is recent enough to have glibc 2.2.2 or later, then it very likely also has GNOME 2.0 or KDE 2.0 or later which are Unicode safe window environments.

[3] NL Software Registry: http://registry.dfki.de/

2.2. The IPA

When working with phonology, Unicode offers the symbols of the International Phonetic Alphabet. (IPA) The IPA is maintained by the International Phonetic Association and defines a unique symbol for every phoneme used in spoken languages. The IPA is itself a standard for the linguistics community to apply for phonetic transcription that will assure mutual comprehension within the field in the present and future. The IPA should be applied in the NLP community wherever spoken language must be tokenised and avoids creating your own system where data can only be understood by your own applications. Using the IPA under Unicode will allow you to take advantage of Unicode based software and reduces the new resources that you would otherwise have to invest time to develop.

The SIL Doulos font presents all the IPA symbols with excelling quality using typeface resembling the Times Roman[4].

2.3. The PUA

When developing a new orthography, extending an existing one, or have found that Unicode does not have the symbol that you need –you can still use Unicode. Unicode has a built in way to support additions to its own character repertoire via the Private Use Area (PUA). In this region of the character encoding standard you may define your own symbols. This is helpful when working with an experimental orthography or an as yet unsupported writing system, two frequent occurrences with U-RLs.

You may define additional letters in this region of the standard and use them safely with your own applications and others. If you want to visualise the symbols that you have encoded in the PUA you will need to modify a font to contain these symbols. Since you have made your own "private encoding" within Unicode you risk losing some portability. Sending a document to a colleague that contains your additions, you will also need to send along your modified font. If your PUA additions are processed in a multilingual system where another researcher has also made PUA extensions, there is the possibility of encoding collisions. This is an exceedingly rare occurrence but becomes possible when you go from the private use case to one that is public.

The PUA may also be used to encode other useful tokens, for example tags and other markers that you would like to have in a text stream that you would processes yourself. You may not need to view tokens of this type but if you chose to do so it is again a matter of modifying a font.

2.4. Fonts

Fonts provide us with an instance of Unicode encoded characters. Fonts can have many styles (typefaces) and the Unicode standard does not tell us how letters should appear (glyphs) or how they should be typed (hardware dependent). The standard simply assigns numeric addresses (in computer memory these are byte sequences) to the abstraction notion of a

[4] SIL Doulos Font: http://scripts.sil.org/DoulosSILfont

"letter". A "Unicode font" is one that applies these assignments (encoding) to the letters that it contains. Not every "Unicode font" will contain all letters of Unicode, in fact few do. The reasons here are primarily the labour required to produce a complete coverage of Unicode's 51,980 graphical characters as well as the undesirably file sizes that result (fonts are files) which can be on the order of 20 Mb. Few people really need all characters of all alphabets and so most Unicode fonts will only support a few writing systems needed by a target community. This approach works fine so long as you are aware of the supported character range within the font, when letters are not available an application may instead substitute a blank space, a dot, a question mark or commonly a rectangle (□).

When working with a U-RL there is a greater likelihood that the orthography of the language is not yet supported in Unicode. In this case you will need to create your own fonts in order to view text in the native script. This need not be an obstacle because there are freely available tools to create and edit fonts, but it will require an investment of your time that will vary depending on number of letters that need be created.

Fonts come in two types they are either bitmaps – a matrix of on or off dots like a tile design, or they are outlines of the shapes of the letters. The outlined versions ("TrueType" is the most common) are more portable and scale to different sizes with better quality than do the bitmap fonts, but they can take more time to produce. If your intent is not to market the fonts commercially than you do not need to be an artist to be able create letters you need only be able to use a mouse.

A number of commercial and free tools are available for creating and manipulating fonts. Working with bitmap fonts "gbdfed"[5] is a very easy to use tool that runs natively under Linux but can also be used on other operating systems where the GTK library has been ported to. "FontForge"[6] is an open source outline font editor that can run on most every major operating system (some additions may be required) that also supports some bitmap formats. It will always be a less intensive effort to start with an open source font and to modify it for your needs than to start completely from scratch. Fonts that do not also contain the Latin letters can later be problematic to work with, some systems will refuse to use them, so including the ASCII range of letters is always recommend. Seemingly countless free and open source fonts can be found readily with an internet search.

2.5. International Components for Unicode

By far the most extensive, complete and Unicode specific resource is the International Components for Unicode (ICU)[7]. ICU comes in the form of both a C/C++ library and a Java JAR. The resource was initially a project of IBM before going Open Source, it has the participation of many of the key Unicode personnel and offers reference implementations of the Unicode family of standards.

The ICU homepage described the resource as "…a mature, widely used set of C/C++ and Java libraries for Unicode support, software internationalization and globalization (i18n/g11n). It grew out of the JDK 1.1 internationalization APIs, which the ICU team contributed, and the project continues to be developed for the most advanced Unicode/i18n support. ICU is widely portable and gives applications the same results on all platforms and between C/C++ and Java software." (ICU)

ICU should be considered as a resource for Unicode text processing, some of its services will be touched on in following sections.

2.6. Transliteration

Transliteration is the systematic conversion of one system of writing onto another. It is most often used in NLP for converting text to and from some encoding system into a Romanised form that legacy resources can then understand. As more resource become Unicode enabled there is proportionally less reliance on the conversion technique. Transliteration will however always be useful when facing text that is not in a script that you are familiar with, working with toponymic lexicons, and making rough conversions to and from a writing system and the IPA.

```
<icu:transform type="Latin">
  # variables
  $gammaLike = [ΓΚΞΧγκξχϰ] ;
  ...
  # convert all to decomposed
  ::NFD (NFC) ;
  ...
  α ↔ a ;  Α ↔ A ;
  β ↔ v ;  Β ↔ V ;
  # contextual transforms
  γ } $gammaLike ↔ n } $egammaLike ;
  Γ } $gammaLike ↔ N } $egammaLike ;
  γ ↔ g ;  Γ ↔ G ;
  δ ↔ d ;  Δ ↔ D ;
  ε ↔ e ;  Ε ↔ E ;
  ζ ↔ z ;  Ζ ↔ Z ;
  # contextual transform
  Θ } $beforeLower ↔ Th ;
  θ ↔ th ; Θ ↔ TH ;
  ι ↔ i ;  Ι ↔ I ;
  κ ↔ k ;  Κ ↔ K ;
  λ ↔ l ;  Λ ↔ L ;
  μ ↔ m ;  Μ ↔ M ;
  # contextual transforms
  ν } $gammaLike → n\' ;
  N } $gammaLike ↔ N\' ;
  ν ↔ n ;  Ν ↔ N ;
  ...
  # convert back to composed
  ::NFC (NFD) ;
 </icu:transform>
```

Figure 1: Latin ↔ Greek Transliteration Sample in ICU

As provided, the ICU transliteration capability is promoted as supporting "50+" systems. New

[5] gbdfed Homepage: http://crl.nmsu.edu/~mleisher/gbdfed.html
[6] FontForge Homepage: http://fontforge.sf.net/
[7] ICU Homepage: http://icu-project.org/

transliteration systems can be added without requiring source code recompilation. Figure one presents a familiar example with the Latin and Greek alphabets:

A very valuable feature of transliteration in ICU is the extension to "compound transforms". Under the concept of compound transformations defined transliteration systems may be chained together for special conversions. For example:

```
[:Lu:] Latin-Katakana; Latin-Hiragana;
```

Defines a compound transformation where uppercase Latin letters (identified with the Unicode character class [:Lu:]) are converted into Katakana. The output of the first transformation, terminated with the semicolon ";" symbol, becomes the input for the next. The remaining lowercase Latin characters will be converted into Hiragana. For example:

Input ⇒ [:Lu:] Latin-Katakana; ⇒ Latin-Hiragana;
Washington ⇒ ウashington ⇒ ウあしんぐとん

Reversible transliteration systems require strict adherence. The most common deviations from stringent transliteration come from the application of transcription rules. There are legitimate reasons for applying transliteration and transcription together, for example when a transcription is well established and the transliterated rendering would cause confusion. The meta-language for transcription should be capable of contextual processing to make some conversion of transcription and other phonological phenomena that manifest in an orthography possible. This capability will also be useful in validation and conformance work where conversion from transcription to stringent transliteration is the objective.

For example, a regular expression based rule for the elision of gemination characters:

```
([^aeiou]){2} ⇒ $1;
```

which replaces two occurrences of a non-vowel (i.e. a consonant) with a single occurrence. Contextual transliteration of Greek gamma:

```
{γ} [ ΓΚΧΞγκχξ ] > n;
γ > g;
```

which converts gamma into *n* if gamma is followed by any of: Γ, Κ, Χ, Ξ, γ, κ, χ or ξ. To enhance phonetic transcription it is advantageous to precondition an English string to better represent the spoken value of the word in the target language. For example "progeny" in American English spoken form is "pro-ğə-ni" (IPA). Rules can be applied here, within the domain of American English orthography:

```
([^aeiou])y$ > $1i;
oge > oje;
```

thus the Cyrillic rendering, for example becomes "пройени" and not naïvely as "прогены". These special rules should generally be applied prior to regular transliteration. At the end of a transliteration process the capability may also help further simplify symbol clusters that have occurred from the application of earlier rules.

2.7. Keyboards

When a U-RL requires a writing system which has little or no computer legacy it can be useful to develop a typing system for entering new in-language text. Adding an Input Method (a typing system, or "IM") is very platform and API specific exercise. Unlike character encoding and fonts formats, there are no recognised standards for a keyboard implementation that is portable across operating systems or window environments. Unless you have hired typist to enter or compose text for your project, developing a keyboard system will likely not be worth the investment of time to learn the APIs and develop the IM software. When time is an issue it will be more efficient to develop a transliteration system, enter the text samples you need in regular Latin script, and convert them into the target script (still under Unicode encoding) with a transliterator.

If it is essential to have an IM for your project the best options on a Window system will be to work with the very robust Tavultesoft Keyman Developer[8] which has a small licensing fee. On Linux systems the trend has been to move towards the Smart Common Input Method (SCIM)[9] where an IM can be defined in configuration files without having to develop new source code. The SCIM based Keyboard Mappings For Linux (KMFL) also offers some Linux compatibility for Keyman IMs. Keyman and SCIM attempt to provide keyboard support for all applications within a window environment. Some applications do provide their own keyboard interpreters as a means to obtain greater independence from the window system and thus offer some portability between window and operating systems. Yudit[10] and Emacs are two such examples of editor applications that provide their own IM infrastructure. Adding an IM to Emacs[11] will take a little knowledge of the Lisp programming language and in Yudit it is a matter of writing a text based mapping file.

As a last resort, a platform portable IM can be developed in either Java or JavaScript but under the restricted contexts of where these languages can be used.

2.8. Text Processing

Unicode and ICU do not address analytical linguistic issues directly but do provide many of the language neutral facilities that you would build upon to address language specific problems. For instance ICU does not have support for stemming but will apply Unicode definitions for character properties to offer text segmentation, normalization, and highly advanced pattern matching.

A number of writing systems have numerous legacy encoding systems; for example Vietnamese had 43 systems (Erard) and Ethiopic over 70. Unicode offers a

[8] Tavultesoft Homepage: http://tavultesoft.com/
[9] SCIM Homepage: http://www.scim-im.org/
[10] Yudit Homepage: http://yudit.org/
[11] Emacs Home: http://www.gnu.org/software/emacs/

vendor neutral encoding that legacy systems can be converted into. Algorithms may then be developed to understand only a single system. ICU offers over 700 encoding system conversions and developers may add to it as needed. (Davis and Scherer)

ICU also provides character normalization services. A character in Unicode may still have for than one form, this is common with "composed characters". For example 'ä' may be the singe character with address U+00E4 or may also be comprised of the two characters 'a' + '¨' with the addresses U+0061 and U+0308. There are seemingly endless numbers of possible composed characters and ICU will know how to map components into a single character if available or possibly a Unicode defined named sequence; thus making text comparisons more successful.

Similar to normalization the notion of equivalence classes is supported in regular expressions languages. ICU extends the Java regular expressions support to account for all character classes in the "Unicode Character Database"[12]. Here every property of a character is defined, such as case, letter type, punctuation type, numeric type, and so on, there are a great number of these classes. In a very simple example of how they are applied in a regular expression, the following statement would match only letters with "uppercase" property but excluding ('-') those in the Latin alphabet:

```
/[\p{Lu}-\p{Latin}]/
```

Both the C/C++ and Java APIs support the Unicode style regular expressions extensions as defined in the Unicode Technical Report #18[13]. The Perl and C# programming languages also support Unicode style regular expressions without depending on ICU. While very powerful for complex pattern matching the Unicode regular expressions syntax does not always go far enough to support lesser understood properties of U-RL scripts. The Perl regular expressions support is very simple to extend and some modules have been developed to support pattern matching in syllabic scripts such as Cherokee and Ethiopic. For example with the Regexp::Ethiopic[14] Perl module the expression:

```
/[ሙ-ቀ]{#4,6#}/
```

matches any character in the range of ሙ through ቀ but only in the 4th or 6th orders, ie the set:

[ማምሚሥራርሳስሻሽቃቅ]

The module demonstrates overloading of the Perl regular expressions engine. The same can be done with the ICU classes, it is often faster however to prototype and experiment in Perl first and then follow with a Java or C/C++ implementation as needed.

3. Working *for* Unicode

As an NLP researcher working with an U-RL your objective will be to solve some very specific problem within a limited period of time and within the fiscal constraints of a budget. The native speaker community will benefit from the body of knowledge of their language having been expanded by your work. You will have made it easier for future researchers to enter into and further explore the language. You may solve the problem that lies before you and move on to others, you have no further obligation to work with the language beyond your original mandate. If you're lucky, however, your experience with the U-RL may become a labour of love that you continue to seek funding to work on or take up in personal time.

If you are so able and wish to do so, working beyond the problem that brought you to the U-RL, you can expect to be drawn deeper and deeper into the community and engage in every broader computing resource problems. Indeed, you may be one of very few, or even the only, person actively developing software for the community for quite some time. As the resources that you develop and provide become utilised by the research and native-speaker community, you inherit some responsibility to maintain them and provide support services.

Gradually you will also become a technical bridge between the community and the greater software industry. At this level you have become a subject expert on the language and its computing resources. The best service that you can do for the language, and yourself, is to build upon your experience and community contacts to advance your work to the level of standards for the language. With standards available for the language software companies have in a sense the "legal basis" that they need to begin support for the requirements of the language. This will be a great benefit to the native speakers and with the software industry picking up support for a language you will be relieved of the burden of having to maintain and support your earlier work.

3.1. Character Encoding and The Script Encoding Initiative

If in the course of your work you have developed a font or experimental encoding system for the script, you will want to consider permanent encoding in the Unicode Standard. Doing so will ensure that the letters will live on for the life of the standard itself and give software companies the technical foundation needed to support the script and language.

The challenge of obtaining the standard however is not unlike trying to be your own trial lawyer in courtroom in a foreign environment where you do not know the laws. It is not for the faint of heart and will most definitely become a bigger undertaking than you had in mind.

It will be much better to have an experienced standardisation expert on your side that knows the system and can navigate the process for you while you provide the subject matter expertise. Launched in April 2002 at University of Berkeley, the Script Encoding Initiative (SEI)[15] is just such an expert.

[12] http://www.unicode.org/Public/UNIDATA/UCD.html
[13] http://www.unicode.org/reports/tr18/
[14] http://search.cpan.org/~dyacob/Regexp-Ethiopic/
[15] SEI Homepage: http://www.linguistics.berkeley.edu/sei/

The SEI has "…the goal of organizing and orchestrating the completion of the Unicode Standard. SEI involves other institutions than software companies, reaching out to academia and the public sector, drawing upon scholars around the world as a major resource and on their research results, existing publications and script descriptions. SEI also involves a small group of experts in script encoding to work with scholars to make finished, workable proposals and to move those proposals through the standardization process as soon as possible." (Anderson)

Even if you never anticipate working on a standard for the script, do collect as much cultural information about the writing system as you can while you have the opportunity in the field. If not you, a native speaker or another researcher wishing to work with the SEI, can apply the information in a proposal process. There is almost always more to letters than just the sounds they make.

3.2. Script Name and Language Codes

Along with the encoding of the written symbols of the script, the script and language identities likewise need to be encoded in the respective standards. In this case the standard relevant to language encoding is the ISO 639 family (parts 1, 2 and 3). It is in this standard where "en" is defined as a code for "English" and "lol" for "Mongo". These ISO 639 language codes are used by applications and operating systems to configure a language setting.

Parts one and two cover only 506 of the 7,300 main languages tracked by the Ethnologue. (SIL) Part three attempts to cover all of the world's languages and has been a draft standard for a number of years and should become a final standard shortly.

If the U-RL that you are working with is not covered by ISO 639-3, you will want to request its inclusion from the registrar[16]. Requests can also be made for the encoding of a dialect. It is still possible to request a code assignment from ISO 639-2 which may help gain software support faster. However, ISO 639-2 does require evidence of 50 in-language documents from 5 institutes that most U-RLs are unlikely to have[17].

To encode a script identity, such as "tfng" for "Tifinagh", ISO 15924 is the applicable standard. As with language name encodings the registrar should be contacted if the script the U-RL uses has not been assigned a code[18].

3.3. National Standards

Most every country will have some government recognized body such as a ministry, industrial consortium, or professional organization entrusted to define standards for the nation as a whole. Working with a national standards body should always be attempted before approaching an international body.

It may be the case that the national standards body does not have the interest, expertise, funds or other resources to develop a national standard for technology focused topics such as character encoding, collation, keyboard mappings, or more deeply linguistic areas such as lexicons.

On the other hand, a standards body may be grateful for any help it can get and will welcome your initiative particularly in information technology areas. Potentially, a standards body may fund and even take over the "defacto standard" that you have intrinsically developed for your resources and evolve it into a formal standard. Having a national standards body endorse and legalize a defacto standard for a system puts the highest level of clout behind it. This will make it much easier to achieve international recognition for the standard, particularly if the standards body is also the nation's representative to the ISO. Local software companies may also begin supporting the standard without waiting for the international recognition.

4. Conclusion

In uncharted territory there will be more to explore academically, and more pure knowledge build, but at the cost of moving a little slower while you tread the first roads.

The entry cost into working on a U-RL will always be higher relative to working on a well explored language. This greater cost is incurred for the lack of informational, computational and the lowest level of resources for language representation on a computer. With each passing year the growing availability of Unicode based tools helps lower the entry cost in both time and money required to undertake U-RL research. Indeed, it has never been easier.

While there can be a greater burden upon an NLP researcher to develop basic level resources to engage in U-RL research, there is also the opportunity to realize a greater and more meaningful impact from the research activity. By developing those resources and filling the resource void, you are helping the language and its culture make the leap into the information era and have a better chance at surviving into the future.

5. References

Anderson, Deborah, September 2003. The Script Encoding Initiative, Multilingual Computing, Volume 14, Issue 6, 34.

Davis, Mark, and Markus Scherer, 2005. Globalizing Software, Retrieved April 14, 2006, from http://icu.sourceforge.net/docs/papers/globalizing_software.ppt

Erard, Michael, 2003, September. Computers Learn New ABCs, Technology Review, 28.

International Components for Unicode. (n.d.). Retrieved April 14, 2006, from http://icu.sourceforge.net/

SIL International, 2006. Ethnologue, Retrieved April 14, 2006 from http://www.ethnologue.com/

[16] ISO 639-3 Registrar: http://www.sil.org/iso639-3/
[17] ISO 639-2 Reg. : http://www.loc.gov/standards/iso639-2/
[18] ISO 15924 Registrar: http://www.unicode.org/iso15924/