Unicode and Bible Translation — DRAFT of 5 January 2004

by Kahunapule (MichaelJohnson@EvangelBible.org)

Unicode is a world-wide standard for encoding text in (almost) any language. This standard offers many benefits for Bible translators.

A Very Brief History of Writing Systems

People have historically invented many effective ways of writing. These include carving, painting, using quill or pen and ink, smearing lead, charcoal, or graphite on rocks, tree bark, papyrus, paper, metal, wood, wax, or even plastic. Until the invention of movable type, printing presses, and typewriters, this was all done by hand. Hand writing systems include many different forms of pictograms, syllabaries, phonemic alphabets, and alphabets like the English one where the link between letters and sounds is kind of loose and historical (or should I say hysterical?). Probably the only reason that there aren't as many writing systems as languages is that many languages aren't written, yet, and languages that are newly written often tend to borrow from the writing system of another language.

Enter the computer. Computers are great at dealing with numbers, but it didn't take long for people to realize that by assigning each letter to a different number, you could do wonderful things with text on a computer, far beyond what you could do with a typewriter. After a while, an encoding standard called "ASCII" (American Standard Code for Information Interchange) won out over other standards. It worked well for the English alphabet, common punctuation, and some control characters. ASCII is a 7-bit code, and therefore had room for 127 code points. Since most computers were organized to store a character in 8 bits at a time, 256 code points were available.

Different people assigned different meanings to those upper 127 code points. This is where some of the European languages got included, with letters combined with accents (acute & grave), diereses, tildes, etc. This is also where line drawing characters so valuable in DOS days resided. With the coming of Microsoft Windows, there were better ways to draw lines, but more need for support for different writing systems. There wasn't room in these 256 code points to keep everyone happy in the USA, Europe, Russia, Japan, and certainly not in China. This is where the code page idea came in. Since people almost never needed to mix Cyrillic, Latin, Greek, Hebrew, and Arabic alphabets on one computer, we used code pages. These were standardized by the International Standards Organization. There could be a code page for each writing system, but sometimes there were more, because of special needs or the need to use certain combinations of characters together. MANY of these code pages were defined, and it still wasn't enough. There also had to be schemes to use more than one octet to encode Chinese characters and other writing systems with larger character sets. Then, of course, people wanted more symbols for mathematics, physics, chemistry, and just for fun. ☺

To manage these different character sets, and use them together, required switching code pages and using custom fonts for each code page. This, in turn, required more complex applications with higher level encodings to mark where the code page

and font boundaries were. The result was a mess not totally unlike the day construction stopped on the Tower of Babel.

The current best solution to this encoding mess is an international standard for assigning code points to characters is called Unicode™.

What is Unicode?

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. Using Unicode, you can more easily mix writing systems, even in one document, like this.

Unicode 給每個字元提供了一個唯一的數位

Η κωδικοσελίδα Unicode προτείνει έναν και μοναδικό αριθμό για κάθε χαρακτήρα,

יוניקוד מקצה מספר ייחודי לכל תו

यूनिकोड प्रत्येक अक्षर के लिए एक विशेष नम्बर प्रदान करता है

ユニコードは、すべての文字に固有の番号を付与します

Unicode - это уникальный код для любого символа,

The current release of the Unicode standard, version 4.0, has over 96,000 characters defined. Obviously, Unicode doesn't do this with the same octets used by the historical code pages. In today's computers, with lots of RAM & hard disk space, it doesn't really matter. It is OK to encode one character with 32 bits instead of 8. There are ways of using Unicode with a lower average bit per character count than 32, called UTF-16 and UTF-8, so it is not as inefficient as it might seem.

What Characters are in Unicode?

Unicode is a moving target, because characters are being added to it from time to time. However, the characters that have already been assigned to code points remain at those code points. Unicode includes every character required to write in every major living language in the world, plus some unique characters used in minority languages. It also includes some characters from historical (dead) languages. It does not include some historical languages like Egyptian hieroglyphics (although it might, some day, if enough people can agree on the proper extent of such a character set). It does not include characters from fantasy languages (like Tolkein's Elvish and Dwarvish or Paramount Studios' Romulan and Klingon). This is not a major concern for Bible translators, although there actually are some people working on Bible translations for Klingon and Romulan. (They are having real difficulty finding native speakers for field testing.)

In addition to the standard characters of Unicode, there are over 100,000 code points set aside as a "private use area" (PUA). This is where organizations or even individuals are free to assign code points for their own use. If you absolutely need a character that is not in the standard portion of Unicode, you can make your own, here. In the case of Bible translation, I strongly recommend avoiding doing so for practical and cost reasons, but if you do, please at least standardize it through SIL's PUA assignments.

Looks Can Deceive

My old mechanical typewriter (which was sold at a garage sale as an antique) had no key for the numeral “0”. Why should it? It looked just the same as the capital letter “O”. Of course, now with computers, we are learning to differentiate between the two. Likewise with numeral “1” and lower case “l”. Some fonts try to make them look different. Some don’t. Nevertheless, those characters clearly mean something different. With Unicode, the number of similar distinctions is multiplied greatly. There are, for example, Greek letter capital sigma (Σ) and mathematical n-ary summation (∑), which both mean different things. There are separate code points for typographic left and right single quotes (‘’), a straight single quote ('), and a modified letter apostrophe (’) that might be useful as a word-forming character (such as for a glottal stop). It is important to use the correct character for the intended meaning, in spite of the looks, so that computer programs can correctly process the information. Each Unicode code point is assigned attributes that indicate if the character is a word-forming character, punctuation, etc., so that programs can correctly compute where line breaks should go, among other things. If you are consistent in using the character with the correct meaning as well as the correct appearance, then there is hope that things like search & replace, database lookups, typesetting, and even machine-assisted translation will work properly. If not, then you can expect to do more manual labor.

Another case of looks deceiving in Unicode is where different fonts render a given code point two different ways. It is normal, and even desirable, for fonts to vary the way a letter looks, both for reasons of readability in different situations and for pure artistic considerations. Some fonts have serifs (the little wider parts of strokes at the ends, as the font used in this paragraph does), and some do not (like the font used in the section headings does). Moreover, sometimes the same character is rendered in a different way altogether, but considered equivalent. For example, the lower case “a” and “g” may be rendered more like hand printing in “literacy” fonts (i. e. “ɑ” and “ɡ”). Another example of a major shape difference is in the capital version of the lower case “ŋ”, which may be rendered as either “Ŋ” (same basic shape as the lower case version, but bigger) or “Ɲ” (like a capital “N” with a tail), depending on your choice of font.

Composed Characters and Normalization

There is more than one correct way to represent some characters. Some Unicode code points are assigned to combining diacritics or overlays that combine with whatever character went before it. Sometimes several of these diacritical marks can be stacked. (The results may or may not render in a pleasing or even acceptable fashion with current software, but at least that is the theory.) For example, “o” followed by diaeresis makes “ö”. There is also a single code point for “ö”. These two representations are treated by Unicode-compliant software as equivalent, and such software may freely convert between representations. There are two “normalized” forms of Unicode: maximally decomposed (NFD), and maximally composed (NFC). It may be easier to edit using decomposed forms, but it may be preferable to print using composed forms, as these often handle the spacing between diacritics or combining marks better. For searching, both the text to be searched for and the text to be searched need to be converted to the same form before comparing them, otherwise the different forms would

cause a mismatch on equivalent strings. Normally, this would not cause any problems, and it is all pretty much hidden from the user, except in one case. If you use non-Unicode character mappings with Unicode-compliant software, this may cause data to be corrupted. This is because the composed/decomposed equivalence in characters no longer is true in any other character set. The only way you won't have troubles with this switching back and forth is if (a) you don't use any code points with equivalent representations, (b) your software doesn't actually use its "right" to make such conversions, or (c) you can easily recover your data by normalizing it back the way it started.

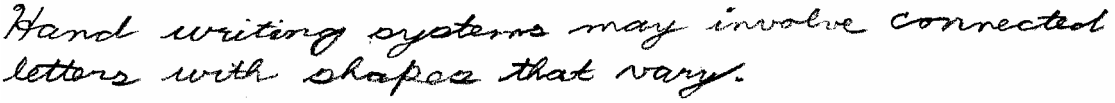
Composed Characters and Rendering Engines

Unlike some of the custom fonts we have used in the past, which have different versions of diacritical marks for different heights, Unicode font renderers are supposed to be smart enough to determine the height of the characters with which they are combining diacritical marks. This means that there is only one combining macron above, for example, in Unicode, but there are two in the code page implied by the SIL PNG Charis font. Font rendering engines like Microsoft Uniscribe and SIL Graphite actually work well for single diacritics. Your results may vary if you start stacking diacritical marks.

Because two combining diacritics in a custom code page often map to a single combining diacritic in Unicode, mapping characters from Unicode back to the custom code page requires a translation program that understands the context of the character and the heights of characters. Fortunately, SIL TECKit does such complex transformations (when the custom code page is properly described in a mapping file), so reliable bidirectional conversions can be done.

Challenging Scripts

There is more than just assigning numbers to glyphs to render a writing system properly. Each written language has rules about how words, sentences, and higher units of speech are to be laid down on paper (or other medium). Sometimes the form of a letter changes depending on its position in a word (like the Greek "σ" turning to "ς" whenever it is at the end of a word). This may mean that an alternate glyph (with its own code point) must be displayed, depending on the context. There are also certain combinations of characters that form ligatures, many of which are required in Arabic. Consider, for example, the programming challenge of properly rendering English cursive handwriting by computer. ***Some people have faked it with some fonts that come close to looking like handwriting, but which really don't convince anyone.*** *Some imitations are more believable than others.*



Writing complex scripts requires much more than having a number to assign to each letter, and much more than standard Unicode fonts. If you have to write with such a system, then you need much more than Unicode, but you would probably still want to use Unicode as a part of the solution.

Another class of challenging scripts to render involves very large character sets (like some of our Asian friends use), or cases where writing direction changes within a document (perhaps because of mixed languages). Unicode doesn't solve these rendering problems. Rather, it just provides the underlying correspondence of basic glyphs to code points that can be used by the fonts and rendering software. Dealing with glyphs that change shape based on their surrounding context is the job of rendering software like Graphite and Uniscribe, not Unicode. However, without the standardization provided by Unicode, the job of doing this rendering for any code page people might want to use would be much more complex (and therefore less likely to be done on time and under budget).

The Lure of Unicode

Unicode provides the following advantages to Bible translators:

- Any writing system in any written language in the world can be encoded with this standard.
- Commercial enterprises are demanding and funding applications that use and generate Unicode text.
- Unicode typesetting processes require less preprocessing of data and custom font work before the actual typesetting process begins, saving time.
- Unicode is stable in that once a glyph is accepted into the standard, it remains at a fixed code point.
- Multilingual software is easier to write using Unicode than with any competing method.
- Legacy encoded data can be converted to Unicode.

Unicode's Downside

To be fair, not all is rosy and cheery in the Unicode world. To wit:

- Not all applications can handle Unicode, yet. (Some never will.)
- Unicode fonts don't render ALL Unicode characters (or else they would be unreasonably huge—but several of them do cover many languages).
- Use the Private Use Area at your own risk, as your definitions will likely not be supported by anyone but you and anyone you can talk into making the same definitions.
- If all you write with are the characters defined in 7-bit ASCII (i. e. English letters, numerals 0-9, and common punctuation), then it makes little difference if you use Unicode or not, because the first 127 code points of Unicode are exactly the same as 7-bit ASCII.

- If some program that you need to use doesn't support Unicode, then you may have to convert your data back and forth between Unicode and a legacy encoding to use the old program.

- If any character you need to use in your language is not available in Unicode, or only available in the Private Use Area, then you lose much of the advantage of Unicode. (Orthography designers take note.)

Programs that Support Unicode

Some significant programs that support Unicode are:

- Microsoft Windows XP, 2000, and NT, along with their accessories (Notepad & Wordpad)

- Microsoft Word and much of Microsoft's other recent software.

- Paratext 6

- SIL Translation Editor (still under development)

- Lambda and Omega typesetting systems (based on LaTeX and TeX)

- TECKit (for converting between legacy encodings and Unicode and back)

- The Java programming language and things written in it.

- Current software development tools (so programmers can write more Unicode software).

Keyboarding Unicode

With over 96,000 characters in Unicode, you might wonder how to type all of these characters. Nobody really wants to build or learn to use a keyboard with that many keys! Fortunately, there is no need to do that. Under Microsoft Windows, there are four main ways of keyboarding Unicode: (1) entering the Unicode code point value in decimal on the numeric keypad, (2) pasting characters from the Character map applet (or the similar "insert symbol" function of Microsoft Word), (3) using an existing or custom Microsoft Windows keyboard layout, or (4) using a Tavultesoft Keyman keyboard layout.

Entering code point values or pasting from the character map applet can be really tedious, and is really only good for entering a character or two that you rarely use, like maybe the ☏ telephone dingbat for use in a letterhead. To enter any Unicode decimal codepoint, make sure the "Num Lock" is on, then hold down the left "Alt" key while pressing the numbers for the code point, then let up on the "Alt" key. For example, to ender the telephone dingbat symbol, you could hold down "Alt" and press 9743. (Make

sure you are using a Unicode font with this symbol in it, like Arial Unicode MS, to see your handiwork.)

If the characters you wish to type are all in a current major language, then chances are very good that Microsoft has already designed a keyboard for that language and shipped it with Microsoft Windows NT, 2000, and XP. If it is an Asian language, you may have to install additional support for those languages that normally aren't in the default installation. If none of these standard keyboard layouts meets your needs well, then a custom keyboard layout can be created. For example, if your language has an "ŋ", but no "q", and is otherwise the same as U. S. English, then you might want to use the U. S. English layout with just that one substitution made. Microsoft provides a free visual tool for creating such layouts, called "Microsoft Keyboard Layout Creator." You can download it at http://www.microsoft.com/downloads/details.aspx?FamilyId=FB7B3DCD-D4C1-4943-9C74-D8DF57EF19D7&displaylang=en (paste that whole URL into your browser or just go to http://microsoft.com and search for that download). If you are in Ukarumpa, you may prefer to save download charges and get it from ftp://ftp.sil.org.pg/Public/Software/Windows/KeyboardLayoutCreator/MSKLC.exe, instead. Instructions for using the program come with it, in the form of a help file.

Microsoft's free Visual Keyboard is a nice companion to your new keyboard layout. It displays the layout on screen, and even lets you enter characters into anothe application with the mouse if you want to. You can get your free download at http://www.microsoft.com/downloads/details.aspx?familyid=86a21cba-e9f6-41db-86eb-2adfe407e620&displaylang=en or, in Ukarumpa, at ftp://ftp.sil.org.pg/Public/Software/Windows/KeyboardLayoutCreator/vkeyinst.exe. This program is a relatively small download (235 KB), but requires the Microsoft .NET Framework (which you may already have if you have installed anything else that uses it, or you may download it for free from Microsoft).

One possible "Gotcha!" associated with Unicode keyboards is that not all applications read Unicode characters from the keyboard driver the same way. Windows makes several options available, such that raw key codes, ANSI code page character codes, and Unicode values are all available to the application. Because Microsoft standard keyboard drivers and Keyman drivers work slightly differently, some applications may not work well with one or the other when typing characters that don't appear in the current default code page. Some applications just don't deal with Unicode well at all. A well-written application will work well with Unicode from either type of Keyboard layout driver. Sometimes use of one or the other type of keyboard driver is required to take advantage of automatic keyboard switching to conform to the language properties of a given input field. I have found, however, that Microsoft did some things right, in that Unicode keyboard layouts created with MSKLC work well with some non-Unicode applications, as appropriate Unicode characters get converted to their code page equivalents when typing them into non-Unicode applications. This is convenient for me, in that I don't have to change keyboard layouts between Unicode and non-Unicode applications when I use my favorite Unicode keyboard layout.

Some sample Unicode keyboard layouts for characters found in PNG languages and for IPA characters (and in both Dvorak and Qwerty[1] versions) can be found at ftp://ftp.sil.org.pg/Public/Software/Windows/KeyboardLayouts/ or http://ebt.cx/translation/keyboards.zip for you to modify to taste. In the sample PNG baseline keyboard layouts, you can type every character found in the SIL PNG Branch standard fonts. However, it would be good to optimize the keyboard for your particular language(s) of interest, so that your fingers don't have to work so hard to type the characters that you use the most.

If you are using software that expects Keyman to be present to switch keyboard layouts, or if you want to use some of Keyman's other features, you may want to consider Tavultesoft's Keyman. It is available at http://www.tavultesoft.com/ (and in Ukarumpa at ftp://ftp.sil.org.pg/Departments/TSD/Software/Keyman/Keyman6/). Again, instructions come with the program.

Why Switch to Unicode?

The best reasons for a Bible translator to put their Bible translation data in Unicode instead of a legacy code page are:

- You want to use the full capabilities of Paratext 6, SIL Translation Editor, or Microsoft Word.
- You want your translation and language data to hold its value longer.
- You want to take advantage of some of the new Unicode fonts.
- You want your translation data to be more valuable when it is time to reprint and/or revise it.

How Do I Switch to Unicode?

If you are only using common English letters and punctuation, congratulations. You are already done. For orthographies where you need Unicode more, there are three steps:

1. Make sure you have a keyboard layout that works well for you. (This may be as simple as using an existing layout or slightly modifying one.)

---

[1] The Dvorak keyboard layout is designed for efficiency, speed, comfort, and minimum stress to your wrists while typing English text. The original Qwerty layout was designed to intentionally slow typists down so that they would not jam the strikebars on the first mechanical typewriters. The Qwerty layout requires more finger motion to type the same text as a Dvorak layout, and is therefore not recommended for people whose wrists might be stressed with lots of typing. Keyboard manufacturers make Qwerty keyboards, because that is what people learn to type on, and people learn to type on Qwerty keyboards because that is what keyboard manufacturers make. Now, in the age of computers, keyboard layout is software configurable, and you have a choice, at least on your own computer.

2. Convert your data with TECKit. You may ask your language software support person for help with this, or follow the instructions in the "Advanced Help" of Paratext 6. For the SIL PNG Branch, a working TECKit map is at ftp://ftp.sil.org.pg/ /Departments/TSD/Software/PNG To Unicode/, along with some helpful documentation.

3. Use Unicode-compliant software and fonts to work with your data. (To use legacy applications that can't handle Unicode, you may use TECKit to reverse the data conversion, provided that you haven't added new characters that are outside of the range that can be handled by the legacy encoding.)