

The Secret Life of Unicode

A Peek at Unicode's Soft Underbelly

Suzanne Topping

Unicode. Most of us in the technology world view it as a basic necessity for dealing with multiple character sets. No one really talks about its strengths or weaknesses; it simply is. But like any technical solution, Unicode has advantages and disadvantages. It even has (gasp) competitors. So rather than ignoring the fact that downsides exist, this article takes a look at Unicode's weak points.

Not long ago, a request appeared on the Unicode mailing list for help in compiling a list of Unicode benefits. The request prompted an interesting dialog, which got even more interesting when the inevitable counter-question arose: What about the downsides? Opinions rumbled in, along with historical explanations and proposals for fixes. This article summarizes the problem areas.

While some complaints are limitations of the character set, others are really implementation challenges rather than problems with Unicode itself.

Interestingly enough, there seems to be a fundamental difference in approach for objections based on the type of script users deal with. Complaints about CJKV (Chinese/Japanese/Korean/Vietnamese) scripts tend to be general and broad, and tend not to focus on change. In contrast, objections about alphabetic scripts tend to be specific, and are often focused on getting changes made (for example, requesting that characters be added or that rules be modified).

The need for compatibility with existing standards has led to many of the complaints voiced about Unicode today. The ability to convert back and forth between Unicode and other character sets was a necessity. Many design compromises had to be made during Unicode's evolution so that it could play well with others. But enough of the excuses, on to the list!

East Asian Issues

Given the large number of users in Japan and China, and the complexity of East Asian character sets, it's not surprising that complaints about CJKV would be voluble. The plain truth is that many Japanese and Chinese users do not trust Unicode. It's hard to believe when watching all the new operating systems, development tools, and technologies move to Unicode as the default character set, but resistance remains high in Japan, particularly among those conducting multilingual software research. There is a sentiment that decisions about Japanese issues are being made by people who do not have a native familiarity with them, despite the fact that the consortium includes a variety of Japanese and other East Asian members.

This section discusses the largest areas of concern for CJKV characters and languages.

Han Unification

In order to support Chinese, Japanese, Korean, and Vietnamese, encoding for many thousands of ideographs must be provided. A subset of all these ideographs occurs in the four writing systems. This set of characters is referred to as Han, because it originated in China during the Han dynasty. Han Unification is the process of assigning single code points to the Han characters. The resulting set of coded ideographs is called UniHan.

Han unification issues are probably the most frequent objections raised concerning CJKV character

The Secret Life of Unicode

A Peek at Unicode's Soft Underbelly

Suzanne Topping

support. But if the Han ideographs are the same, why all the complaints?

The problem stems from the fact that Unicode encodes characters rather than "glyphs," which are the visual representations of the characters. There are four basic traditions for East Asian character shapes: traditional Chinese, simplified Chinese, Japanese, and Korean. While the Han root character may be the same for CJK languages, the glyphs in common use for the same characters may not be.

For example, the traditional Chinese glyph for "grass" uses four strokes for the "grass" radical, whereas the simplified Chinese, Japanese, and Korean glyphs use three. But there is only one Unicode point for the grass character (U+8349) regardless of writing system. Another example is the ideograph for "one," which is different in Chinese, Japanese, and Korean. Many people think that the three versions should be encoded differently.

Fonts are the delivery mechanism for glyph display. This means that while Han characters have been unified, fonts can't be. In order to provide the glyph variations required to meet the needs of writing system differences, fonts must be specific to the system in use. Trying to use only one font guarantees that some characters won't look right to everyone.

Part of the objection to Han Unification is cultural. There is a perception that languages have been merged rather than just characters. This is an understandable confusion given that previous character sets and encoding methods have been language-specific. If the old character sets were merged and unified in Unicode, there is a feeling that the languages are somehow unified as well. East Asian cultures are unique and diverse, and would have valid objections if there were attempts made to somehow merge languages. While this is not the case, there is obviously some confusion about how unification works.

Number of Ideographs

The characters needed for everyday use by Japanese, Chinese, and Korean users is relatively small compared to the huge set of ideographs that exists. At the other end of the spectrum, specialized users such as linguists, historians, government record keepers, and scholars of archaic texts might not be able to easily access some of the ideographs they need.

A significant challenge is that new ideographs are difficult to create on the fly. One workaround is to describe the new character in graphic form. An Ideographic Description Sequence (IDS) is created by combining a graphic character with an existing coded ideograph or with other descriptive components to generate a graphic description. The user then looks at the description and tries to visualize the intended result. The rendering engine does not create a true glyph from the IDS. This would be somewhat like trying to visualize the "a" character by reading the description "u with an umlaut." Obviously, this is not an optimal solution. More about this subject is included in the section titled "Difficulty of adding new characters."

Radical Handling

Radicals are the structural components of Han characters that are used for indexing. Dictionaries and indices are ordered by radical. Ideally, Han characters could be encoded by their radicals rather than as entire ideographs; this would make it easier to do sorts and to compose characters on the fly. If

The Secret Life of Unicode

A Peek at Unicode's Soft Underbelly

Suzanne Topping

radicals were encoded, there would be hundreds of code points rather than the tens of thousands of ideographs currently needed to support CJKV.

The problem with this approach is that there are currently no rendering methods that are efficient enough to deal with composing ideographs from radicals. Unicode has added commonly used radicals and positioning characters, but we are still a long way from being able to render characters from radicals on the fly.

Japanese Personal Name Characters

Gaiji are rare Japanese personal name characters that frequently need to be entered into computer systems but are not part of the Unicode set. The current mechanism of support is to allow individual implementers to add gaiji to the Private Use Area in Unicode. (The private use concept also existed in legacy CJKV character sets.) While this method provides support, the characters entered in the Private Use Area are not assigned official code points. In order to communicate between systems, all parties must understand what items are included in the Private Use Area, and how they are assigned. This obviously creates limitations for widespread use. More about this subject is included in the section titled "Difficulty of adding new characters."

Thai Ordering

The primary objection for Thai support is that characters are not stored in logical (spoken) order. This makes use of the Unicode collation algorithm complicated. The ordering came about because Unicode inherited the Thai industrial standard, which was based on typewriters using visual ordering of the reordrant vowel signs, and no viramas. This flaw was recognized, and so a small set of logically ordered vowel signs was added to the Thai block. Unfortunately, when Unicode aligned with ISO 10646, a compromise was reached, and these logically ordered Thai vowels were removed in order to go back to matching the old standard.

Competitor Character Sets

Because of all the problems related to support for East Asian languages, several alternative multilingual character sets have been developed, primarily in Japan. These character sets are described below, however, none of them are serious competitors to the Unicode standard. Because Unicode is used by most software producers, these character sets are very minor players in comparison. (For example, Unicode is used internally in current Microsoft Office products.)

TRON

TRON's claim to fame is that it doesn't unify Han characters; instead it attempts to distinguish all the glyph variants. One commercial implementation called Cho Kanji 3 (meaning "Super Kanji") claims that 171,500 characters are supported. TRON is not as widely used as Unicode in Japan, but it is popular in some anti-Unicode technical communities.

UTF-2000

The UTF-2000 character set initiative has a goal of providing a flexible abstract character data type. The idea is to offer a framework that allows characters to be displayed using glyphs defined by users.

The Secret Life of Unicode

A Peek at Unicode's Soft Underbelly

Suzanne Topping

(Note: It is difficult for an English-speaking person to learn about UTF-2000, as most of the information available online is in Japanese.)

Giga Character Set (GCS)

GCS is a display code scheme created by Coventive Technologies, which claims to overcome Unicode's perceived CJKV flaws. Instead of assigning binary codes to characters, GCS is a set of encryption algorithms (one per language) that are used to transition between natural language characters and computer bits. GCS also claims to be faster and require less memory than Unicode because it derives characters rather than looking them up.

Issues for Other Scripts

Bi-directional text behaviors are challenges for Arabic, Hebrew, and other scripts, where most text runs from right to left, but some text (such as numbers, Western names, etc.) run from left to right. Positional forms for many scripts introduce additional difficulty for users because the text "dances" (changes form) on the screen as new characters are typed in.

Because of all the user entry problems described below, many users rely on old word processors that use visual markup languages rather than newer WYSIWYG applications. With the older systems, users feel like they have more control over text.

In order to address the issues of bi-directional and conjoining text, text input methods need to become much more sophisticated. One proposed solution is to store directionality information in embedded levels with text, along with "state" information so that user actions can be handled more intelligently.

Because of all these difficulties, problems with bidi scripts are primarily implementation challenges rather than actual Unicode limitations.

Inserted Characters

When dealing with bidi scripts, it can be difficult to go back and insert text into the middle of a sentence. A user can position the cursor visually on the screen, but the text that is typed might appear somewhere else based on its logical position. The result is that users might need to make a few attempts before getting the text where they want it. This is because logical rather than visual ordering is used. Implementers use a logical buffer and cursor, and a position translation table that maps from visual to logical position. The translation table causes the jumps to the logical positions.

Dancing Positional Characters

For a number of writing systems, the form a character takes is dependent on its position within the sentence, and the presence of other characters before and after it. This changing state of a character is called its positional form. With current Unicode implementations, users are faced with characters that dance on the screen as new characters are added. The changes can be interesting to those of us watching demos in order to understand the changing positional nature of characters, but it can be quite distracting to people who need to type a simple sentence.

The Secret Life of Unicode

A Peek at Unicode's Soft Underbelly

Suzanne Topping

Zero-width Characters

Some scripts require zero-width characters for performing certain functions, but they create confusion for users. When a zero-width character (such as the ZWNJ in Persian) occurs in a sentence, a user might try to arrow over it but the cursor would not appear to move. Worse, when deleting, a user might delete the wrong character because the cursor position is not accurately displayed due to the presence of a zero-width character.

Bidi Implementation Issues

Locale information is needed for bidi implementations to deal with collation algorithms and numeric handling. For example, European numerals are dealt with as normal numerals in Hebrew, but as foreign numerals in Persian. Sophisticated bidirectional handling is particularly important for Web authoring tools. These applications need to have the intelligence to deal effectively with tags that must be surrounded by < and >, which point in the correct direction.

Cross-language Issues

Some of Unicode's criticisms are general rather than applying to specific script types. These generic issues are described below.

Standards, Standards, Standards

Standards are supposed to make things easier, right? Well believe it or not, one of the major problems with Unicode is pre-existing standards. Unicode was developed in a world filled with existing standards, and the consortium therefore had to make design compromises in order to work with them. Compatibility with legacy code pages and other standards has made the standard complex.

Limited Number of Code Points

Some critics worry that Unicode will eventually run out of code points. Competing character sets like TRON claim to be "limitlessly extensible." GCS cites character limits as Unicode's primary weakness, and addresses it by using encryption algorithms instead of code points. The reality of this concern is debatable. By the time Unicode runs out of room for new characters, technology will most likely have evolved beyond Unicode's useful life.

Inconsistencies in Handling

Another common complaint about Unicode is that there are numerous inconsistencies in handling from script to script. There is more than one solution to a single problem. Many of the inconsistencies are the result of efforts to be compatible with legacy character sets. The biggest problem with the inconsistencies is that they make it difficult to create systems that support multiple scripts. When working on single-script implementations, inconsistencies are much less of an issue. Some examples are listed below.

Handling of Positional Forms

One of Unicode's inconsistencies is how it handles positional forms. For Arabic, Syriac, or Mongolian,

The Secret Life of Unicode

A Peek at Unicode's Soft Underbelly

Suzanne Topping

each positional form of a character is encoded with the same code point, and the rendering engine selects the proper form. For Greek and Hebrew, final and non-final forms are assigned their own code points.

Subjoined Letters

The Brahmi-derived scripts used in India, Tibet, and Southeast Asia use subjoined letters to form consonant clusters. For the scripts of India, Unicode encodes these subjoined letters with a sequence virama plus consonant. Other scripts like Tibetan have subjoined letters directly encoded as unique characters.

Logical Versus Visual Ordering

Some scripts use logical ordering (for example, Indic scripts), while others use visual ordering (Thai and Lao).

Handling of ASCII

ASCII appears as the first 256 characters of the Unicode character set. Unlike the rest of the standard, which is organized into blocks, ASCII code assignments are essentially random.

Difficulty of Adding New Characters

Language is an evolving entity, which means that new characters will continue to be formed and old ones changed. Getting new characters into the Unicode standard isn't fast or easy. While mechanisms exist for defining new characters (by using the Private Use Area) or describing ideographs (by using Ideograph Description Sequences), neither of these methods really adds the character to the set.

In addition, the Private Use Area is used in conflicting ways. Without various parties all agreeing to use it the same way, characters will appear as garbage. There is currently no standard mechanism for dynamically defining characters and broadcasting how to encode, decode, or translate them so that they become publicly known.

Equivalency Confusion

Unicode provides a variety of ways to generate a particular character. This means that when you look at a character on the screen, you won't know what approach was used to create it. For example, ü can be expressed as either u+00fc or as its equivalent u+0075 + U+0308. These matching characters are called equivalents. There are two types of equivalence; compatibility and canonical. Equivalence makes it difficult to implement search routines, and generally causes confusion.

Precomposed and Decomposed Forms

Composite characters can be represented in two forms -- either as precomposed characters or as decomposed characters. Precomposed characters are equivalent to a sequence of one or more other characters. Decomposed forms are broken into the character's basic component units. Decomposed characters provide greater flexibility and require less work to be added to the standard. In some cases, decomposed forms are the only option provided due to processing requirements.

The Secret Life of Unicode

A Peek at Unicode's Soft Underbelly

Suzanne Topping

Precomposed representations are considered preferable in Web protocols, and are supported better in existing software.

The result is a mix of primarily precomposed representations, with some decomposed representations thrown in when a precomposed form is not available. In order for software implementers to do things right, they need to support both forms. Unfortunately, it's more likely that implementations support one or the other, but not both.

Unicode does not Equal Internationalization

Somewhere along the line the idea was formed that Unicode is internationalization. Calling software the "Unicode version" implied that it was ready to be shipped around the world. The reality is that Unicode is a character set that makes internationalization easier. As a wise internationalization expert once summarized: Unicode doesn't eliminate problems with internationalization; it just makes the problems more interesting.

Conclusion

How important any of these criticisms really are depends very much on personal perspective. It's hard to say that one problem is critical while another is trivial, because if the issue affects your ability to deliver a product, it's important to you. For example, if you are trying to implement a system that allows entry of Japanese family names, the issue of gaiji support is critical. Bidirectional text entry challenges, however, are completely unimportant. Conversely, if you are trying to create a Hebrew word processing package, you aren't going to care much about whether gaiji are encoded or not. But dealing with zero-width characters might give you ulcers.

Although Unicode might not be a perfect solution to the challenge of handling all the world's characters, it has moved us a long way toward being able to create systems that can deal with a wide range of languages. During its evolution, the original design goals have had to evolve, and in some ways degrade, to meet real-life challenges.

As with any technology, Unicode will undoubtedly be replaced by something that works better. Eventually. But we aren't likely to see that happen any time soon. For now, it is a long-awaited and much-appreciated solution to the world's multilingual computing requirements.