L2/03-294

Handling CJK compatibility characters with variation sequences

Ken Lunde, Adobe Systems Inc.
Eric Muller, Adobe Systems Inc.
August 24, 2003

1. Introduction

In L2/02-389, Ken Lunde shows that the canonical decomposition of CJK compatibility ideographs is problematic, as it “erases” distinctions in source standards which need to be preserved for proper rendering. This situation goes beyond the need for locale specific forms, as those distinctions are needed within a single locale.

When the UTC reviewed L2/02-389, there was a general agreement on this problem, and the UTC decided to investigate tailored normalizations as a solution. The investigation revealed that this approach was problematic, and the UTC rejected that approach. The UTC also asked us to investigate the use of variation sequences as a solution. This document presents various options.

Throughout this paper, the term “compatibility ideograph” refers to the canonical decomposable characters encoded in the CJK Compatibility Ideographs block and the CJK Compatibility Ideographs Supplement block. The twelve characters in those blocks which are not canonical decomposable are covered by the term “unified ideographs”, along with the characters in the CJK Unified Ideographs block, the CJK Unified Ideographs Extension A block and the CJK Unified Ideographs Extension B block.

2. Some examples

Here are a couple of examples that will be used to illustrate the various situations.

Example 1: U+8612, and the compatibility ideograph that normalizes to it, U+FA42.

code point	Unicode 4.0 glyph	Source	ISO 10646-1:2000 or -2:2001 glyph
U+65E2		G0-3C48	
		J0-347B	

既

T1-514D 既

U+FA42  J3-752B

This is the simplest case: the same glyph shape is present in three of the source standards that contribute to the URO, they are unified as U+65E2. Later, JIS 0213 is added; the glyph shape for J3-752B is unified with the original one, but to provide round-tripping to JIS 0213, U+FA42 is encoded to maintain the distinction between J3-752B and J0-347B.

Note that the fact that the two shapes are unified mean that in the absence of other information. either shape is an appropriate rendering for either character,

Example 2: U+4FAE, and the two compatibility ideographs that normalize to it, U+FA30 and U+2F805:

code point	Unicode 4.0 glyph	Source	ISO 10646-1:2000 or -2:2001 glyph
U+4FAE		G0-4E6A	侮
		J0-496E	侮
		KP0-5932	
		K0-5932	侮
		T1-4F78	侮
U+FA30		J3-2E38	

U+2F805  KP1-3534

T4-253D 

What we have here are two glyph shapes: let's call them the G shape (oblique strokes) and the J0 shape (vertical stroke). According to the unification rules, those two shapes represent the same abstract character.

Thus, when the URO was built, the source characters G0-4E6A, J0-496E, K0-5932 and T1-4F78 were encoded as a single coded character, U+4FAE.

When Extension B was built, the T4 source was added. Because (T1, T4) is not subject to the source separation rule, T4-253D did not result in a separately encoded character. But to maintain round-tripping with the TCA-CNS 11643-1992 standard (the various planes of it are the T sources), the compatibility ideograph U+2F805 was encoded. (Note that round-tripping with CNS 11643-1992 is actually not provided in general; presumably, this is done one a character by character basis, and this case was deemed worthy of a compatibility ideograph.)

Finally, with Unicode 3.2, the J3 and J4 sources were added. Again (J0, J3) is not subject to the source separation rule, but round-tripping with the combined JIS 0208/0213 standard (the J0, J3 and J4 sources), the compatibility ideograph U+FA30 was encoded.

It is not known to the authors when the KP sources were added.

Note that it may have been nice to invert the J sources, that is to give J3-2E38 as the source for U+4FAE, and J0-496E as the source for U+FA30. This would have made the preferred glyph shape for U+4FAE the same in all locales. However, this was probably prevented by compatibility considerations.

3. The problem

Consider what happens for an implementation that consumes and produces JIS 0208/0213 characters, and uses Unicode to represent characters internally. The input J0-347B is represented internally by U+65E2; the input J3-752B is represented internally by U+FA42. Using a font appropriate for Japanese, the correct glyphs can be displayed. On output, the appropriate JIS characters can be generated. So far so good.

If normalization occurs at some point during processing, U+FA42 is normalized to U+65E2. From then on, it is no longer possible to properly render or output in JIS. All the efforts by the Unicode standard and by the application to ensure round-tripping with JIS 0208/0213 are simply negated.

The normalization may be performed by a component of the application over which the author of the application has little or no control. In other words, “do not normalize” may not be a viable option. The alternative “normalize carefully” (aka tailored normalization) was not considered viable either.

4. Proposal

To solve the problems above, the idea is to transform each compatibility ideograph into something that is still distinct from the corresponding unified ideograph, yet is not reduced to it by normalization. A

possible choice would be a PUA code point. A better choice is a variation sequence involving the unified ideograph, as the (Unicode) identity of the character is retained and is accessible to any component or subprocess of the application, without the need for a private agreement.

Assuming that that transformation can be performed before any normalization occurs, may be as soon as inputs are converted from JIS to Unicode, then the distinction is preserved. The inverse transformation can be applied after any normalization may occur, may be as late as when the output is converted from Unicode to JIS.

The proposal is actually to define one variation sequence for each compatibility ideograph, as well as one for the corresponding unified ideographs, and to define the variation sequences by the source characters. Using our first example:

sequence	definition
U+65E2 VS1	representation of G0-3C48, J0-347B, T1-514D
U+65E2 VS2	representation of J3-752B

The transformation mentioned earlier is simply:

arbitrary input	“normalization-safe” equivalent
U+65E2	U+65E2 VS1
U+FA42	U+65E2 VS2

Note that we do define a variation sequence for the unified ideograph. The motivation is this: it is perfectly acceptable for a font to use the glyph shape of J3-752B to render U+65E2 alone; so when we want the glyph shape of J0-347B, we cannot use U+65E2 alone.

Similarly, for our second example, we have:

sequence	definition
U+4FAE VS1	representation of G0-4E6A, J0-496E, KP0-5932, K0-5932, T1-4F78
U+4FAE VS2	representation of J3-2E38
U+4FAE VS3	representation of KP1-3534, T4-253D

with the transformation:

arbitrary input	“normalization-safe” equivalent
U+4FAE	U+4FAE VS1
U+FA30	U+4FAE VS2
U+2F805	U+4FAE VS3

This is not entirely minimal; for example, there is no real need for the two variation sequences using VS2 and VS3; one would be enough. In fact, there was no real need for two compatibility ideographs in the first place (their sources are distinct), but it is not the goal of this solution to fix this situation.

5. A word of caution

The solution presented here seems reasonable, and it solves the problem at hand fairly nicely. In fact, one could imagine that had variation selectors been available from the start, they might have been used instead of compatibility ideographs to enable round-tripping.

However, as currently defined, at most one variation selector can be applied to a base character. Other uses of variation selectors on Han ideographs may be incompatible with the mechanism presented here. It is almost necessary to understand all the anticipated uses of variation selectors on a given base character before defining any variation sequence involving that character; otherwise, one could end up with something very difficult to manage.

For example, one potential use of variation selectors on Han ideographs is to preserve structural information about the visual representation, at a finer level than the unification rules allow. In the case of U+4FAE, one may wish to have:

sequence	definition
U+4FAE VS10	oblique strokes, as in 
U+4FAE VS11	vertical stroke, as in 

This would even work for our purpose, if we do the following transformations:

input	“normalization-safe” representation
G0-4E6A, KP0-5932, K0-5932, T1-4F78, J3-2E38	U+4FAE VS10
J0-496E, KP1-3534, T4-253D	U+4FAE VS11

So VS1, VS2, and VS3 on the one hand and VS10, VS11 on the other are somewhat equivalent, yet they are different. Dealing with both at the same time may prove overwhelming.

Document History

Authors: Ken Lunde, Eric Muller

Revision	Date	Comments
1	August 24, 2003	First version