- DRAFT - For UTC Consideration Only -

L2/98-055
February 22, 1998

Support for Implementing Inline and Interlinear Annotations

Submitted by: Asmus Freytag,
Version 0.02

Introduction

This paper is in response to UTC action item 73-22 It is presented for discussion at UTC # 75.

Background

Japanese uses several forms of inline and interlinear annotations. (Some of these are also used in other languages)

Ruby

(Typically) 1/2 sized text above one or more base characters. Most commonly used to indicate pronunciation. In Japanese the most common form is hiragana characters on top of ideographs, but this is not the only way Ruby are used. In rich text, certain formatting information applies only to the Ruby as an 'object' while other formatting information applies to the annotation or base text. Example formatting information includes the separation of base and annotation as well as the horizontal alignment mode.

There is no way to algorithmically determine or reconstitute ruby texts.

あじさい
庭に紫陽花の

Group ruby

A string of annotation characters is placed relative to a string of base characters. If a

Horizontal in Vertical

昭和44年2月19日に彼は生まれ

Short (usually two character long) runs of digits or alpha- betics laid out horizontally in an otherwise vertically positioned line of ideographic text. The H.I.V. tends to take up the space of a single character in the line, but carries twice the infor- mation. Some formatting applies to the 'object' as a whole, but in more ways than Ruby, H.I.V can be considered part of the regular line.

花が咲いている

group ruby annotation string has (invisible) separator characters in it (akin to optional hyphens) the choice of group ruby vs. mono ruby becomes a formatting choice, and no longer a content difference.

た。

It is conceivable to have an algorithm detect H.I.V opportunities, but such an algorithm would almost certainly need overrides (similar to hyphenation overrides).

Mono ruby

WATANABE KO JI
渡 邉 幸治

Each part of the annotation is placed relative to the single base character it relates to.

Warichu

Longer annotations inline. Usually of 1/2 or less the regular font size and laid out like a short paragraph inline (typically two lines above each other for horizontal text, with large brackets around the Warichu. Warichu imply complicated line breaking logic and may need rich text formatting to select specific line breaking options.

Kumimoji

These are on the fly equivalents to the squared Katakana words in the Unicode standard.

Proposal

Characters

The following is the smallest number of characters that would satisfy the implementation and interchange concerns below.

+XX01 ANNOTATION ANCHOR

+XX02 BASE-ANNOTATION SEPARATOR

+XX03 ANNOTATION SEPARATOR

+XX04 ANNOTATION TERMINATOR

These are proposed for the specials block, failing that the CJK punctuation block.

Usage

XX01 is intended to be used as an anchor character, preceding the annotation object.

XX02 is intended to separate the base characters in the text stream from the annotation characters that follow.

XX03 is used to segment the annotation for line breaking or splitting into mono ruby.

XX04 is used to terminate the object (and returns to the regular text stream)

Example: The mono ruby example from above would be encoded as:

XX01 渡 邊 幸治 XX02 WATA XX03 NABE XX03 KO XX03 JI XX04

Example: The first HIV example form above would be encoded as:

XX01 44 XX04

Discussion

The association of annotation segments to base characters is strictly on a per (base) character basis. It might make sense to allow the separator to segment the base character stream as well, but that would likely be overkill.

It might be possible to overload existing Unicode characters for some of these purposes. However, a robust implementation needs to avoid restrictions in the allowable characters for base or annotation.

Implementation

The entities in question are 'objects' from the perspective of the line layout algorithm. For the same reason that FFFC was added as an 'object replacement' character, implementation for these objects can be regularized by the presence of a dummy character. Unlike image or audio objects, these inline or interlinear objects carry textual information. Therefore the line layout algorithm, to make a complicated description simple, is applied recursively to these sublines. Implementations are, again, simplified with the presence of separator and terminator characters. These character codes need not all be distinct from each other, but for implementations that purport to support any Unicode character and (since they are Far East implementations) also need to retain the Private Use Area for use as EUDC, it is important to have reserved character codes to support these implementations.

Recall that the FFFC OBJECT REPLACEMENT CHARACTER was intended to act as an anchor point for non-textual formatting information. The same would be true for the proposed characters.

Interchange

Like the use of FFFC the use of these characters does not by themselves make it possible to interchange final form documents. That is, all the object and text specific formatting are gone. Unlike non-text objects, the legible information in the annotation is retained and the content relation between characters (base and annotation) and line breaking information (Warichu) can be maintained.

Fallback

Filtering the characters on input gives the same behavior as typical current plain text fallbacks. Especially for Ruby, these fallbacks are not at all 'readable'. Retaining the characters, but not being able to display them yields a text stream that is more human-editable than existing minimal fallbacks. Therefore, adding these characters to the standard does not cause harm to existing implementations.

To create 'clean' plain text in the current situation, an application needs to treat ruby different from the other three annotations. Since there is no fallback support, but the base characters are definitely part of the plain text content, an implementation must actively extract the base characters from the ruby objects.

For Ruby, rendering the BASE-ANNOTATION SEPARATOR and ANNOTATION TERMINATOR as open and close parens and the other two characters as zero width, will yield a standard fallback mechanism that is used in newspapers or wherever ruby annotations are needed, but interlinear space is at a premium.

Experience

Several implementations are known that use similar, but slightly different sets of characters for this purpose.