Title: In-process Private Use Character

Source: Microsoft (expert contribution)
Date: August 8, 2000

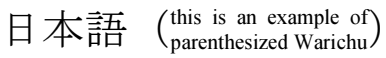
For a long time we have had a need in the Unicode space of transient code positions used as anchors for higher-level protocols. Some of them have been officially encoded: the Object Replacement Character (U+FFFC) and the Interlinear Annotation Characters (U+FFF9, U+FFFA and U+FFFB). Typically these characters are only used temporarily by internal processing and should not be exported in transmitted data. Although the intent of the submitters was made clear, the adoption of these characters created a lot of discussion, as it was feared that the semantic of these characters could permeate into exchanged documents and create confusion in other character formats by creating alternate notation to their own formatting information. A good example is the perceived intersection of the Interlinear Annotation Characters that may be used for internal processing of Ruby annotations and the proposed RUBY element into XHTML.

Some examples of higher-level constructs that are usefully anchored in text ranges are following:

- Inline Horizontal text layout in Vertical text layout, and Inline Vertical text layout in Horizontal text layout.

Example of Inline Horizontal text layout in Vertical text layout.

- Inline annotation (called Warichu in Japanese typography)



Example of Warichu

In these examples, it is convenient to anchor in the text ranges the delimiters of these special text constructs, and as each of these various text effects may be embedded into another text effect, it is necessary to have more than few anchors.

Typically those anchor code values would be part of the text stream and exchanged by internal processing through Application Programming Interface (API). For example, text editors would typically use these anchor code values.

In most cases, these anchors are better represented by markup in SGML derived data, so it is desirable to avoid creating ‘characters’ in plain text that could conflict with these markups. The solution is to assign code values that are not characters.

Unicode already contains two such characters: U+FFFE and U+FFFF, although the first is already used in byte order mark processing. These characters are qualified as <not a character> in the Unicode Standard. In the future an additional 32 characters will be available (all code values ending by FFFE and FFFF in planes 1-10). However it is strongly desired to have additional anchor code values in the Plane 0 (BMP). All these values would be <not a character> and therefore would be subject to the same restrictions as the ones applied to U+FFFE and U+FFFF, specifically (Unicode Standard 3.0, page 28):

Programming receiving this code are not required to interpret it in any way. It is good practice, however, to recognize this code as a non character value and to take action, such as by indicating possible corruption of the text.

The author proposes the following ranges for these additional <not a character>:

U+FE00-FE1F and U+FFF0-FFF8.

Again, by being <not a character> these code values can be safely ignored by unaware programs.

Michel Suignard
Microsoft
