L2/02-177

Asmus Freytag
Issues related to L2/02-143 Liaison report from ISO TC46/SC4/WG1
April 29, 2002

When the question comes to possibly encoding characters for compatibility, it's important to consider two main questions very carefully.

The first, also mentioned by Joan, is to ascertain that the characters in question do not have a non-compatibility use, and that no such use is expected. If careful investigation confirms their status, there are approximately five categories. A compatibility character can be

1. a unique entity, not usually considered a character
2. a variant, more or less duplicating an existing character
3. a unique format character of simple characteristics
   (including any other character with unique semantics)
4. a variant strongly at odds with Unicode's coding model
   (e.g. a positional form)
5. a format character violating the plain text model
   (for example by setting up a scope [via a start-end]).

Going down the list, the cost of encoding such a character increases dramatically. All of the additions accepted for compatibility with JIS X0213 were for the first two categories. [BTW: All compatibility characters in the fifth category that are in Unicode today have either become discouraged or deprecated. And the remaining ones are discouraged in many contexts (HTML, etc), so that there appears limited use in trying to add any new ones]. The greater the cost the more urgent is it to make sure a compatibility character is in actual use and that encoding it will be of actual value to existing or intended implementations, and, at the same time, that in the context of these applications there are no reasonable alternatives - which leads to the second question.

The second is to consider the need of actual (or reasonably projected) compatibility applications for such characters. In some cases, mapping characters from existing sets to PUA or control characters, as Joan suggests, may be a perfectly appropriate answer - but the decision cannot be made entirely in the abstract; it must consider the types of applications that need to implement these characters.

An example of a large set of compatibility characters that was recently added one could consider the case of the Graphics for Terminal Emulation characters. In that case, the requirement was to be able to receive an old-style terminal data stream, transcoded it into Unicode, present a view of a terminal screen on a Unicode-based platform, and apply the reverse process to the user input or response. The decisions which character codes to unify with existing characters, and which to add as compatibility character was arrived at after considering the effect on an application in the above scenario.

With library information systems, similar terminal based access protocols apply, and it is conceivable that a similar transcoding of a transient data stream, and emulation of old-style data terminals on Unicode-based systems is desired/ desirable, [unless the emulation itself is web-hosted, in which case, it may be possible to attempt transient translation to XML.] In a pure context of emulating data access, "private" interpretations of control codes are very appropriate, and in fact that's the intended mechanism for implementing non-6429 terminals.

However, in addition to implementing data access protocols, library catalogs represent databases, which may need to be permanently transferred onto Unicode-based platforms. In that second case, the existing data records will soon co-exist with more recent records that are Unicode-based, and are accessed with modern protocols. In such case, if they are needed to represent features in the data, replacing control codes with XML like constructs makes (a lot more) sense, than for a simple emulation situation, but unification of other characters needs to be approached much more cautiously. That is because, after existing data in old character sets have been transcoded, their character set of origin is lost, and the compatibility and current use of the unified character must be very compatible or else risk the problem of overloading.

While this argues against 'blanket' coverage of the set requested it does provide a yardstick by which we could re-evaluate the set and move forward if actual impact to existing or reasonably projected implementations is shown.

A./