L2/14-035

Namespace Limitations and Loose Matching

Mark Davis
24 Jan 2014
Live doc

I had the following action:

136	A003	Mark Davis	In reference to L2/13-142, look at disentangling the name space limitations from loose matching. (retargeted to UTC #138)

I reviewed L2/13-142, and the conclusion that I came to was that we should use the same algorithm for limiting the name space as for the process of loose matching.

I was looking in particular at what was needed to clarify the definition LMR-B, which was raised in L2/13-142:

> "Another potentially serious disadvantage is that the scope of ignoring "CHARACTER", "LETTER", and "DIGIT" isn't exactly clear, so the rule might need further elaboration and examples added to make it clear.*"

Here is the definition that Ken gave.

> LMR-A: Ignore (i.e., fold away) any casing distinctions, spaces, and any medial hyphen-minus characters in names. Compare the resulting strings. If the folded strings are binary equal, then the names match. There is one grandfathered exception: U+1180 HANGUL JUNGSEONG O-E does not match U+116C HANGUL JUNGSEONG OE.
>
> LMR-B: Identical to LMR-A, except that one also ignores (i.e., folds away) any substring "CHARACTER", "LETTER", or "DIGIT".

After reviewing this, I believe that we can supply an explicit, well-formed definition of loose matching that we can also use for both the name space limitations. (Of course, we have other constraints on the names, A-Z, etc.) The loose matching cannot be more lenient than the name space, otherwise we get collisions. The name space limitations more narrow, but there's no real need for that; it just complicates the model.

Here is my more explicit statement of LMR-B. This is not the only possible such statement.

> *Two names collide if and only if their skeletons collide.*
>
> *define RemovalString to be any of the following:*
> [:PatternWhitespace:], <U+002D>, "CHARACTER", "LETTER", or "DIGIT"
>
> *define the method matchesAt(name, i, result) to be:*
> A. If the substring of s starting at i is not one of the RemovalStrings, return 0.
> B. If the matching RemovalString is <U+002D>
> a. If the character before (ie, s[i-1]) is not in [a-zA-Z0-9] or the character after (ie, s[i+1]) is not in [a-zA-Z0-9], return 0.

b. If the characters in result are “HANGULJUNGSEONGO” and the character after is “E”, return 0.

C. If the matching RemoveString is “CHARACTER”, and the characters in result are “CANCEL”, return 0.

D. Otherwise, return the length of the matching RemovalString.

(Clauses B and C are grandfathering clauses.)

define the method getSkeleton(name) to be:

1. Lowercase the name.
2. Start with i = 0, and initially set result = “”.
    a. If i == length(name), return result
    b. Set n = matchesAt(name, i, result)
    c. If ≠ 0, set i = i + n
    d. If not, append the character at i to result, and set i = i + 1

The only collisions in the 7.0 names, name aliases, and name sequences are the following, which need grandfathering.

Skeleton	**Code Points**
HANGULJUNGSEONGOE	U+116C (ᅬ) HANGUL JUNGSEONG OE U+1180 (ᆀ) HANGUL JUNGSEONG O-E
CANCEL	U+0018 (␘) <CANCEL> U+0094 (␔) <*CANCEL CHARACTER*>
TIBETANSUBJOINEDA	U+0FB0 (ྰ) TIBETAN SUBJOINED LETTER -A U+0FB8 (ྸ) TIBETAN SUBJOINED LETTER A
TIBETANA	U+0F60 (ཀ) TIBETAN LETTER -A U+0F68 (ཨ) TIBETAN LETTER A

One consequence of LMR-B is that we could not define a character with the following names (as noted in L2/13-142).

- MAHJONG TILE ONE OF S
- TIE

Personally, I don’t feel that is a real problem, since in such rare cases, we can always append some other term (we already use contrived terms in many cases just to avoid collisions, such as the many ARROW names or FACE names).