L2/10-151

Comments and Proposal for fixing Loose Matching Rules for Character Names

Karl Williamson
2010-04-29

I am the person who called Unicode's attention to the problems with loose matching of character names, and I am not fully satisfied with the changes that are being proposed in response to this, so am submitting my own thoughts and proposal for consideration.

First, in regard to L2/10-091. I have one quibble with its item 2. I agree that TR18 should refer to UAX44-LM2 instead of having its own pocket definition. However, I would rather it not list the existing three characters anomalous under that pocket definition. I think it should just refer to UAX44, whose definition has whittled the number of exceptions to 1, and if it needs to list exceptions, list just that one.

The background of this is that TR18's pocket definition used to be the correct one, but the definition changed as a result of L2/04-012, and this occurrence of the old definition was overlooked, and has remained so for over 6 years. The new definition was to avoid having to add exceptions. It is true that so far these 3 characters are still the only exceptions, but that may not always be true, as L2/04-012 says that WG2 is considering adding more characters that would be exceptions under the old (still in TR18) rule. As an implementor of the Standard, I want as few exceptions as possible, and I want the documentation to be up-to-date, and I don't want the possibility of new exceptions being added later that would force me to rewrite code, and force the people who use my code to rewrite theirs. (Background: I have implemented a number of fixes to the Perl programming language in the latest version (5.12) that bring it more in line with the Standard, and I'm planning to do more, which is how I discovered these anomalies.)

Second, and finally, the proposed changes to UAX44-LM2. The current official wording is the following:

> ***UAX44-LM2.*** Ignore case, whitespace, underscore ('_'), and all medial hyphens except the hyphen in U+1180 HANGUL JUNGSEONG O-E.
> - "zero-width space" is equivalent to "ZERO WIDTH SPACE" or "zerowidthspace"
> - "character -a" is *not* equivalent to "character a"

I have several problems with the Unicode 6.0.0 draft 9 wording. I'll do it section by section. The first line of it is unchanged from the current. I think it should be more like:

> ***UAX44-LM2.*** Ignore case, underscores ('_'), white spaces that are not adjacent to a hyphen, and all medial hyphens except the hyphen in U+1180 HANGUL JUNGSEONG O-E.

There is a contradiction in both the original and draft wording, in that they effectively require the implementor of this rule to simultaneously ignore and not ignore whitespace. This is because you can't decide if a hyphen is medial without considering whitespace, but it says that whitespace is to be ignored. I know of developers who have been confused by this in the past, and have come to erroneous conclusions as a result. My other changes above are for somewhat added clarity.

The next part that I have a problem with is this:

> In this rule "medial hyphen" is to be construed as a hyphen occurring immediately between two letters in the normative Unicode character name, as published in the Unicode names list, and not to any hyphen that may transiently occur medially as a result of removing whitespace before removing hyphens in a particular implementation of matching.

There are several issues here. I suspect this statement was inserted to avoid having to change the previous one to say that whitespace can't be ignored everywhere. But this statement comes somewhat later, after a bullet list of examples. It is far better to not introduce a contradiction in the first place than to clarify it later. Some people will read only the first sentence and not get this far. Second, it doesn't actually clarify anything for me, but introduces more problems. It defines medial hyphens in terms of the character's normative name, but additionally it has a clause saying they are not transient medial hyphens. It doesn't therefore address medial hyphens in the input that aren't in any normative name. For example, if the programmer wrote “FULL-STOP”, it isn't clear what to do about that. Is that an error or should it parse to “FULL STOP”? (The answer has to be that it parses to “FULL STOP”. Otherwise loose matching is mostly useless.) And third, it ignores the point of view of the audience this document is intended for. I believe I represent that audience: someone who is implementing the standard, and someone who wants to use Unicode to do useful things. As an implementor, I want to know how to parse input I'm given to find the programmer's intent. As a programmer, I want to know the rules of how to express my intent unambiguously. As an implementor, I'm given input that may contain names that appear to have medial hyphens. I want to get from point A, that input, to point B, the real character. Telling me that medial hyphens are only applicable in the normative name is putting the cart before the horse; I don't know the normative name yet; the statement is not helpful in bridging the gap to get to point B. I tried to point this out on the unicode list, but got no response; this is the main reason I'm writing this proposal. I propose simply omitting this sentence, given that something like the first change I suggested for LM2 is taken, which stops the contradiction before it's started.

The last section of the 6.0.0 draft 9 wording that I have problems with is:

> An implementation of this loose matching rule can obtain the correct results when comparing two strings by doing the following three operations, in order:
>
> 1. remove all medial hyphens (except the medial hyphen in the name for U+1180)
> 2. remove all whitespace and underscore characters
> 3. apply toLowercase() to both strings

I believe that “medial” in Operation 1 must refer to hyphens that are medial in the input, but again the sentence I want to get rid of defined “medial” differently, so it's unclear what is meant, as long as that sentence remains. Second, Operation 1 lists the exception for U+1180. But I don't know, as an implementor, at this stage of parsing, what the character I'm trying to parse is, so it's asking me to do the impossible. Later, the text says that these steps are merely a logical statement of what an algorithm would do. Maybe that is good enough to allay my concern. But I think it would be better to state the recipe in a way that is actually feasible. The way I would implement it is to remove all medial hyphens without exceptions, and then after everything is done, see if the result is HANGUL JUNGSEONG OE. If so, I would look at the original input to see if there was a hyphen there, in order to do the disambiguation between the two characters.