L2/24-073

Toward Clarifying the Rules of the Character−Name Namespace

To: Unicode Technical Committee
From: Erik Carvalhal Miller
Subject: Problems in the rules of the character−name namespace
Date: Monday, October 2, 2023

This document discusses inconsistencies, ambiguities, and loopholes in the Unicode Standard's specifications and policies regarding the character−name namespace — as defined in the Character Encoding Stability Policies, the core specification, and some of its annexes — and suggests remedies for greater consistency and clarity.

I. General Rules to Determine Uniqueness Within the Namespace

Unicode's Name Uniqueness policy guarantees that character names, character−sequence names, and character aliases are unique within a single shared namespace, in accordance with loose−matching rules which the policy statement does not delineate but rather points to in UAX #44, *Unicode Character Database*. These rules, found more specifically in that annex's §5.9.2 ("Matching Character Names") and designated UAX44-LM2, state that uniqueness is determined by ignoring letter case and disregarding any whitespace, underscores, and medial hyphens, with the exception of the hyphen (more precisely, a HYPHEN-MINUS) in HANGUL JUNGSEONG O-E (U+1180). (Underscores, whitespace other than U+0020 SPACE, and non−uppercase letters are not permitted in the names and aliases in question, but the scope of §5.9.2 is broader to accommodate implementations' identifiers based on names and aliases.) These loose−matching/uniqueness rules — positioned by the Name Uniqueness policy as definitive — are contradicted by stricter rules articulated elsewhere within the Standard, as we are about to see.

Another stability policy, Named Character Sequence Stability, points to UAX #34, *Unicode Named Character Sequences*, for more information about said sequences. That annex, in its §4 ("Names"), contains a rule UAX34-R3 which, in discussing not just character−sequence names but all names and aliases in the shared namespace, in effect imposes the same loose−matching strictures (without addressing the identifier variations covered by UAX44-LM2) but also spells out the *additional* stricture that the words "CHARACTER", "DIGIT", and "LETTER" be ignored in comparisons to determine uniqueness; the most recent (2020) publicly available version of ISO/IEC 10646, in its §27.5.4 ("Determining uniqueness", pg. 45), also states this additional requirement. UAX34-R3 further declares an exception for the existing control aliases CANCEL CHARACTER (U+0094) and CANCEL (U+0018).

In the core specification, chapter 4 ("Character Properties"), §4.8 ("Name") contains a paragraph headed "Character Name Matching" (pg. 181), which gives loose−matching rules consistent with UAX #44's less restrictive loose−matching rules and points to them.

> ***Suggestion 1:*** Update UAX44-LM2 to incorporate the additional "CHARACTER"/"DIGIT"/"LETTER" stricture of UAX34-R3, including its exception for CANCEL CHARACTER (U+0094) and CANCEL (U+0018). Consider revising UAX #34, §4 so that it points to the revised UAX44-LM2 for the loose−matching rules. Locate other restatements of UAX44-LM2 in the Standard (a further example being in UAX #24, *Unicode Script Property*, under §2.4 ["Script Designators in Character and Block Names"]) and revise them as needed.

ISO/IEC 10646, in including the "CHARACTER"/"DIGIT"/"LETTER" stricture in §27.5.4 while failing to make an exception for the existing CANCEL and CANCEL CHARACTER aliases, places its own character repertoire in violation of said standard's very rules, thereby nominally complicating Unicode's required conformance with a self−violating ISO/IEC 10646.

***Suggestion 2**:* Recommend the relevant ISO/IEC group(s) incorporate the CANCEL CHARACTER/CANCEL exception into ISO/IEC 10646's own statement of the matching rules, for internal consistency, if such repair has not already been made.

The foregoing suggestions assume that the additional rules of UAX34-R3 are to be retained, but that is not the only possible tack:

***Alternative to Suggestions 1 & 2**:* Update both the Unicode Standard and ISO/IEC 10646 to remove any statements of the "CHARACTER"/"DIGIT"/"LETTER" uniqueness stricture and its exception.

II. Code−Point Labels and the Namespace

In the core specification's aforementioned chapter 4, §4.8, under "Code Point Labels" (pg. 186), rules for the construction of code−point labels such as <control-000D> or <reserved-03A2> are given, followed by this paragraph:

> Unicode code point labels are included in the unique namespace for Unicode character names. This ensures that there will never be a naming conflict between a code point label and an actual, assigned Unicode character name.

Code−point labels receive inconsistent treatment within the Standard. UAX #34 and UAX #44 affirm them as part of the shared namespace; but on the other hand, chapter 3 ("Conformance"), §3.3 ("Semantics"), definition D6 (for *namespace*) on pg. 87 omits them in describing the character−name namespace (indeed, they are not mentioned anywhere in chapter 3). Furthermore, they are not mentioned in the Name Uniqueness policy (which nevertheless does point to UAX #44, whose UAX44-LM2 does explicitly include code−point labels in the character−name namespace).

***Suggestion 3**:* For clarity, revise the Standard to consistently include code−point labels in the character−name namespace, particularly by updating the Name Uniqueness policy's stated scope and by updating chapter 3's definition entry D6.

Code−point labels were formally added to the Standard as of version 5.2 to serve essentially as names for the unnamed. Of the five applicable code−point types, four (control, noncharacter, private−use, and surrogate) are immutably assigned; the fifth, reserved, is unique among identifier types in the namespace in that its repertoire actually *shrinks* with successive versions of the Standard. Or does it? On the one hand, the label−construction rules are explicitly stated to apply to code points without character names, therefore indicating that the repertoire of "reserved" labels does indeed shrink as code points are assigned character names; on the other hand, the paragraph quoted above explicitly states that labels' inclusion in the namespace "ensures that there will *never* [emphasis mine] be a naming conflict" between labels and character names (and, presumably, aliases and character−sequence names). Therein lies contradiction: Implementations confronted with character data aligned with different versions of the character database have no guarantee that a "reserved" label valid as of one release will not conflict with a character name, character−sequence name, or character alias assigned in a later release — for example, should RESERVED E-FEED ever turn out to be an excellent name for an accepted pictograph but unfortunately collide with today's valid label <reserved-EFEED>, even if code point U+EFEED itself is already assigned a character with some other name by that time. Even without actual collision, there is uncertainty as to the continuing validity of implementations' use of "reserved" labels as the character database grows.

***Suggestion 4**:* Form a policy clarifying that character names, character aliases, and character−sequence names may not conflict with "reserved" code−point labels from as far back as Unicode version 5.2.

***Suggestion 5**:* Clarify that, even if the Standard discourages continued use of obsoleted "reserved" code−point labels for code points that are currently assigned character names, it is recommended that implementations handle those obsoleted labels gracefully.

III. Nomenclature Nomenclature

Some of the inconsistency in the Standard's treatment of character names, character aliases, names of named character sequences, and code‑point labels is attributable to the history of the Standard's evolution; but it is also likely that the cumbersome task of enumerating those various identifier types has also contributed to inconsistency and poses a continued danger to consistency as the Standard evolves.

> ***Suggestion 6****:* Formally devise and define a cover term for character names, character aliases, names of named character sequences, and code‑point labels (e.g., *character‑level lexical identifiers* [*CLLIs*]) and implement it as applicable throughout the text of the Standard, for the sake of clarity and of ease of updating (including in the implementation of some of the previous suggestions). Also consider a standard name for those identifiers' namespace (even if just based on the cover term [e.g., *CLLI namespace*] or as simple as *character‑name namespace*).

✦✦ END ✦✦