UTC #163 ucd-dev ad hoc feedback & recommendations

Markus Scherer & ucd-dev ad hoc, 2020-apr-21
F5 revised 2020-apr-27

F1: Lack of precomposed capital Greek letters complicates lowercasing, uppercasing, and normalizing Greek text

Date/Time: Thu Jan 9 21:21:51 CST 2020

Name: Alex Henrie

Report Type: Other Question, Problem, or Feedback

Unicode defines precombined characters for various lowercase Greek letters with diacritics, but not their uppercase forms.[1] However, this can cause Greek texts encoded in NFC to no longer be NFC-normalized after changing case: For example, if "Ῥᾶρος" (the name of an ancient Greek hero) is converted to lowercase, its first character changes from 03A1 0313 to 03C1 0313 and must be normalized again to get to 1FE5. The lack of capital characters creates other complications as well, such as breaking any uppercasing or lowercasing algorithm that does not allow changing the length of the string.

Would you please reconsider including these characters in the standard so that Greek NFC text does not need to be renormalized after lowercasing? Or at least add a note about this problem to the Greek Language FAQ?[2]

[1] https://www.opoudjis.net/unicode/unicode_gaps.html#gaps
[2] https://www.unicode.org/faq/greek.html

Recommended UTC actions

1. AI for Rick: Respond to Alex Henrie, informing them that the UTC declines to add composite uppercase Greek letters, including the rationale in L2/20-108 or a link to it.
2. AI for someone: Add text to the Greek FAQ and/or the casing FAQ reflecting a summary of the rationale in L2/20-108 for not adding composite uppercase Greek letters.

Rationale

1. It is Unicode policy to not add additional composite characters.

2. As for Greek uppercase letters: Adding a composite where all parts of the Decomposition_Mapping are already in Unicode would require to decompose the new composite under NFC; that is, the composites would never appear in NFC text. (Normalization stability: https://www.unicode.org/policies/stability_policy.html#Normalization)
3. Moreover, lowercase forms of uppercase Greek letters — regardless of composites or sequences — would normalize to the existing lowercase composites.
4. Thus there would be no appreciable gain; just additional complication.
5. Any proper Unicode string-uppercase or string-lowercase implementation must handle changes in the string length as specified. This is not limited to Greek characters.
6. An addition to the Greek FAQ is a good idea. Maybe also elsewhere as a gotcha for case mapping + normalization.

Background information

https://www.unicode.org/Public/UCD/latest/ucd/SpecialCasing.txt
<code>; <lower>; <title>; <upper>; (<condition_list>;)? # <comment>
...
No corresponding uppercase precomposed character
…
1FE4; 1FE4; 03A1 0313; 03A1 0313; # GREEK SMALL LETTER RHO WITH PSILI

https://unicode.org/policies/stability_policy.html
Case pair stability allows adding uppercase forms.
Normalization stability forbids adding composites without Composition_Exclusion.

F2: Request uniform version syntax

Date/Time: Wed Feb 19 16:07:42 CST 2020

Name: Karl Williamson

Report Type: Error Report

This isn't an error, but it is an annoyance that the data files you furnish have at least three different syntaxes for specifying the versions they apply to:

Files in the UCD have the version embedded in the first line of the file

Files in the security subdirectory have a separate line like 'Version: 13.0.0'

And EmojiData.txt has a line 'Version: 13.0'.

There really is no need to have disparate syntaxes, and it means code reading them has to have extra intelligence.

Recommended UTC actions

1. AI for Markus and the ucd-dev ad hoc: Add a machine-readable version number to the UCD file PropertyAliases.txt.
2. AI for Markus and the ed committee: Add text to UAX #44 documenting the new machine-readable version number, and otherwise recommend not parsing version numbers from comments in Unicode data files.

Possible alternative

1. (If adding a machine-readable version number to PropertyAliases.txt is not approved)
   AI for someone: Add a release check to make sure that at least PropertyAliases.txt contains a comment with a stable format and the correct version number.

Background information

Version numbers are in comments, and often as part of a version-specific file name that does not match the actual name of the released files. They are not documented as part of the file formats.

It should not usually be necessary to parse the version number out of the files. Relying on the version number from a comment risks missing it when the format changes, and risks getting the wrong version number if there is an editorial mistake.

FYI ICU preparseucd.py does parse the version number from one file, PropertyAliases.txt, in order to populate a non-comment version data line in its ICU-specific output ppucd.txt (see http://site.icu-project.org/design/props/ppucd)

Note: PropertyAliases.txt is (at least logically) the first UCD file that has to be parsed.

https://www.unicode.org/Public/13.0.0/ucd/Blocks.txt # Blocks-13.0.0.txt
https://www.unicode.org/Public/13.0.0/ucd/DerivedAge.txt # DerivedAge-13.0.0.txt
https://www.unicode.org/Public/UCA/13.0.0/allkeys.txt # allkeys-13.0.0.txt
https://www.unicode.org/Public/emoji/13.0/emoji-sequences.txt # Version: 13.0
https://www.unicode.org/Public/idna/13.0.0/IdnaMappingTable.txt # Version: 13.0.0
https://www.unicode.org/Public/math/revision-15/MathClass-15.txt # Revision: 15
https://www.unicode.org/Public/security/13.0.0/IdentifierStatus.txt # Version: 13.0.0

F3: UAX #14 for 13.0.0: LB27 first's line is obsolete

Date/Time: Tue Mar 3 16:17:10 CST 2020

Name: Daniel Bünzli

Report Type: Error Report

Hello,

I think (more precisely my compiler thinks [1]) the first line of LB27 is already handled by the new LB22 rule and can be removed.

Best,

Daniel

[1]
File "uuseg_line_break.ml", line 206, characters 38-40:

```
206 |   | (* LB27 *)  _, (JL|JV|JT|H2|H3), (IN|PO) -> no_boundary s
                                    ^^
```

Warning 12: this sub-pattern is unused.

[Filed by Rick on behalf of user, per KW. We can delete this if original poster submits it.]

Recommended UTC action

1. AI for Chris: In UAX #14: Remove the redundant rule (JL | JV | JT | H2 | H3) × IN from LB27.

Background information

https://www.unicode.org/reports/tr14/#LB22

> ***LB22*** *Do not break before ellipses.*
>
> × IN
>
> *Examples:* ‘9...’, ‘a...’, ‘H...’

https://www.unicode.org/reports/tr14/#LB27

> ***LB27*** *Treat a Korean Syllable Block the same as* ***ID****.*
>
> (JL | JV | JT | H2 | H3) × IN
>
> (JL | JV | JT | H2 | H3) × PO
>
> PR × (JL | JV | JT | H2 | H3)
>
> When Korean uses SPACE for line breaking, the classes in rule **LB26**, as well as characters of class **ID**, are often tailored to **AL**; see *Section 8, Customization*.

Andy Heninger replied on the same day on the unicode list:

> I agree. The LB27 first part rule
>
> (JL | JV | JT | H2 | H3) × IN
>
> appears to be redundant.

Good catch.

F4: Mistake in section 6.2 of UAX #29

Date/Time: Sun Mar 8 10:50:59 CDT 2020

Name: Zack Newman

Report Type: Error Report

> I'm unsure if this is a mistake in sections 3.1.1 and 4.1.1 or section 6.2, but 6.2 incorrectly states "ignoring Extend is sufficient to disallow breaking within a grapheme cluster". The sequence of Unicode scalar values (U+0600, U+0020) is considered a single grapheme cluster due to rule GB9, but the sequence is parsed into two words according to 4.1.1. While it would be ideal to not have sequences of Unicode scalar values that can be parsed into more words than grapheme clusters, I think it's OK for that property to not hold as long as there are no incorrect claims that it does hold like there currently is in section 6.2.

Recommended UTC action

1. AI for Mark: In UAX #29 section 6.2 Replacing Ignore Rules: Modify the first paragraph as recommended in L2/20-108.

Recommended text change

Change the paragraph by removing that sentence, and making a few other wording changes:

> An important rule for the default word and sentence specifications ignores Extend and Format characters. The main purpose of this rule is to always treat a grapheme cluster as a single character—that is, not break a single grapheme cluster across two higher-level segments. For example, both word and sentence specifications do not distinguish between L, V, T, LV, and LVT: thus it does not matter whether there is a sequence of these or a single one. ~~In addition, there is a specific rule to disallow breaking within CRLF.~~ ~~Thus ignoring Extend is sufficient to disallow breaking within a grapheme cluster.~~ Format characters are also ignored by default, because these characters are normally irrelevant to such boundaries.

Note: Mark already has an action 160 A073 to “Investigate the best way to resolve inconsistencies in text segmentation and linebreak algorithms, and report back to the UTC. See feedback in PRI #396 from Charlotte Buff [Sat Jul 6 16:57:48 CDT 2019].” which is in progress.

Background information

https://www.unicode.org/reports/tr29/#Grapheme_Cluster_and_Format_Rules

> An important rule for the default word and sentence specifications ignores Extend and Format characters. The main purpose of this rule is to always treat a grapheme cluster as a single

character—that is, as if it were simply the first character of the cluster. Both word and sentence specifications do not distinguish between L, V, T, LV, and LVT: thus it does not matter whether there is a sequence of these or a single one. In addition, there is a specific rule to disallow breaking within CRLF. Thus ignoring Extend is sufficient to disallow breaking within a grapheme cluster. Format characters are also ignored by default, because these characters are normally irrelevant to such boundaries.

F5: Zero Width Space vs Arabic shaping: non-interop

Date/Time: Wed Apr 1 17:29:56 CDT 2020

Name: Elika J. Etemad

Report Type: Error Report

There was some discussion in the W3C, triggered by some new test cases, about whether ZWSP should break Arabic shaping, given spaces generally break shaping. We found that Unicode clearly defines it as not breaking shaping, but also found that Unicode's behavior does not seem to be widely implemented, see [1].

The question to the UTC is, therefore, should ZWSP continue to be defined as transparent wrt shaping, or should its definition be adjusted to match what appears to be the current implementation reality?

[1] https://github.com/w3c/csswg-drafts/issues/3861#issuecomment-529348086

[Fwiw, a number of participants in the discussion initially expected that ZWSP would break shaping, just like all the other "space" characters. So given that expectation plus the state of implementations, it might actually make sense to spec this behavior and introduce a new character, if needed, for an explicit break opportunity that does not break shaping.]

Recommended UTC action

1. AI for someone: Respond to Elika regarding April 1 feedback about “Zero Width Space vs Arabic shaping”: Ask for confirmation that the request is to change the Joining_Type property of U+200B Zero Width Space from T to U, which would cause it to break Arabic Shaping; and request a document discussing pros and cons of T vs U behavior.

Background information

Despite the character name, U+200B Zero Width Space does not function as a space, it is not White_Space, and since Unicode 4.0.1 it has General_Category=Cf (Format).

UCD ArabicShaping.txt specifies that code points “that are not explicitly listed and that are of General Category Mn, Me, or Cf have joining type T.”

There are some gc=Cf characters with jt≠T. For example:

200C; ZERO WIDTH NON-JOINER; U; No_Joining_Group
200D; ZERO WIDTH JOINER; C; No_Joining_Group
2066; LEFT-TO-RIGHT ISOLATE; U; No_Joining_Group
2067; RIGHT-TO-LEFT ISOLATE; U; No_Joining_Group
2068; FIRST STRONG ISOLATE; U; No_Joining_Group
2069; POP DIRECTIONAL ISOLATE; U; No_Joining_Group