UTC #189 properties feedback & recommendations

Markus Scherer & Josh Hadley / Unicode properties & algorithms group, 2026-oct-19

Participants **2**
1. Core Spec **3**
1.1 Code snippet in Core Spec for UTF-8 in UTF-16 Order is wrong [#624] 3
2. UCD **4**
2.1 Feedback: Georgian_Extended has incorrect titlecase mapping [#578] 4
2.2 Modifier stacked arrowheads should be wb=ALetter like the modifier arrowheads [#581] 6
2.3 change Vertical_Orientation of U+1F201 🈁 from Tu to U (PRI-556) [#593] 7
2.4 Initial teaching alphabet: Soft_Dotted ligatures? (PRI-556) [#642] 8
2.5 UAX #31: toFoldedcase(S) is not defined anywhere [#647] 9
2.6 UAX #31: Table 3a and its UnicodeSet disagree [#646] 9
2.7 UAX #24 rules for scx values are incomplete [#650] 10
2.8 Define what Age=Unassigned means [#655] 11
2.9 L2/26-255 Some Changes to Indic_Syllabic_Category: A Report on UTC 158-A106 [#687] 12
2.10 L2/26-256 Categorizing Some Uncategorized Indic Characters: A Report on UTC 164-A63 [#688] 13
3. Proposed new scripts & characters **13**
4. Bidi **17**
4.1 Fast path in UAX #9 needs more restrictions [#607] 17
4.2 Five bugs in bidi reference implementations [#675] 17
5. Normalization **22**
5.1 UAX #15 buffering guidance is incorrect [#628] 22
5.2 UAX #15 and UAX #41 refer to removed normalization charts [#681] 23
6. Line Break **23**
6.1 Dashes aren't spaces [#582] 23
6.2 Change line break property of Tulu-Tigalari independent vowels [#689] 24
7. Segmentation **25**
7.1 L2/26-135 Finite automata for line breaking and segmentation [#567] 25
7.2 Add Malayalam example to GraphemeBreakTest.txt [#611] 27
7.3 More ALetters: A Report on UTC 162-A2 [#665] 27
7.4 Reviewing Word_Break Open Issues: A Report on UTC 162-A3 [#670] 28
7.5 U+21AE breaks segmentation consistency with canonical equivalence [#682] 30
7.6 more GraphemeBreakTest.txt test cases (PRI-556) [#685] 32
7.7 Feedback L2/26-217 on Finite automata for line breaking and segmentation (PRI-555) [#679] 33
8. Collation **35**
8.1 Tertiary level collation of characters secondary equal to ß [#663] 35
8.2 Notes on the normalization of collation folding [#680] 36
9. Security **37**
9.1 Bad Java code in UTS #39 [#594] 37
9.2 Fast path in UTS #39 skeleton calculation is incorrect [#598] 38
9.3 Wrong claim about idempotency of skeleton() in UTS #39 [#595] 38
9.4 Discrepancy between UTS #39 and RFC 6532 about local part of email addresses [#601] 39

9.5 UAX #31: consider withdrawing the Excluded Scripts table [#645].....40
10. Confusables.....41
10.1 UTS #39: Better description of non-goals for scope of confusables [#478].....41
11. Emoji.....43
11.1 Standardize East_Asian_Width and Vertical_Orientation for new emoji [#347].....43
12. Math.....44
12.1 Approve UTR #25 [#691].....44
13. Other.....45
13.1 Draft UTS #61 editorial changes [#583].....45
13.2 CSSWG shares feedback on the handling of Hangul for text-autospace (PRI-510) [#589].....45
13.3 Joiners in UTS #58: A report on UTC 183-A76 [#657].....47
13.4 Approve UTS #61 [#683].....48
14 Feedback not addressed in this document.....48

Participants

The following people have contributed to this document:

Markus Scherer (chair), Josh Hadley (vice chair), Robin Leroy (vice chair), Anushah Hossain, Asmus Freytag, Charlotte Buff, John Wilcock, Ken Whistler, Ned Holbrook, Peter Constable, Robert Bastian, Roozbeh Pournader

1. Core Spec

1.1 Code snippet in Core Spec for UTF-8 in UTF-16 Order is wrong [#624]

Recommended UTC actions

1. **Action Item for** Roozbeh Pournader, EDC: Correct the C code snippet for "UTF-8 in UTF-16 Order" in Section 5.7.1 of the Core Specification. For Unicode Version 19.0. See L2/26-235 item 1.1.

PAG input

From Roozbeh Pournader, PAG:
Section 5.7.1 "UTF-8 in UTF-16 Order" of the Core Spec contains the following code snippet:

```
int strcmp8like16(unsigned char* a, unsigned char* b) {
  while (true) {
    int ac = *a++;
    int bc = *b++;
    if (ac != bc) return rotate[ac] - rotate[bc];
    if (ac == 0) return 0;
  }
}

static char rotate[256] =
          {0x00,  ...,                    0x0F,
           0x10,  ...,                    0x1F,
           .                          .
           .                          .
           .                          .
           0xD0,  ...,                     0xDF,
           0xE0,  ...,              0xED, 0xF3, 0xF4,
           0xEE, 0xEF, 0xF0, 0xF1, 0xF2, 0xF5,  ..., 0xFF};
```

It appears to me that the code won't work in environments where char is signed. Compare UTF-8 for U+007F and U+0080:

```
a = "\x7F";      // U+007F
b = "\xC2\x80";  // U+0080
```

With 8-bit signed char:

```
rotate[0x7F] == 127
rotate[0xC2] == -62
```

So the function returns 127 - (-62) > 0 (both char operands undergo int promotion), incorrectly claiming U+007F sorts after U+0080.

The table should use unsigned char explicitly.

2. UCD

2.1 Feedback: Georgian_Extended has incorrect titlecase mapping [#578]

Recommended UTC actions

1. **Consensus**: Change the Simple_Titlecase_Mapping and Titlecase_Mapping assignments of the Georgian mtavruli letters in the Georgian_Extended block U+1C90..U+1CBF to map these letters to their mkhedruli counterparts, which are given by the Lowercase_Mapping property. For Unicode Version 19.0. See L2/26-235 item 2.1.
2. **Action Item** for Robin Leroy, PAG: In UCD file UnicodeData.txt and derived files, change the Simple_Titlecase_Mapping and Titlecase_Mapping assignments of the Georgian mtavruli letters in the Georgian_Extended block U+1C90..U+1CBF to map these letters to their mkhedruli counterparts, which are given by the Lowercase_Mapping property. For Unicode Version 19.0. See L2/26-235 item 2.1.
3. **Action Item** for Asmus Freytag, PAG: In the core spec, note that default titlecasing maps Georgian all-caps-like mtavruli to lowercase mkhedruli, which is a fallback for what is a nonsensical operation for the Georgian script. For Unicode Version 19.0. See L2/26-235 item 2.1.

Feedback (verbatim)

PRI-544

Date/Time: Fri April 17 17:26:18 PT 2026
ReportID: ID20260417172618
Name: Jules Bertholet
Report Type: Report Error in Publication/Data
Opt Subject: Georgian_Extended has incorrect titlecase mappings

Letters in the Georgian script have two forms: the standard, lowercase-like "mkhedruli" form, and the variant uppercase-like "mtavruli" form. The former are encoded in the Georgian block (U+10A0..U+10FF),
while the latter are in the Georgian Extended block (U+1C90..U+1CBF).

Unlike uppercase in other languages, however, mtavruli is only used in ALL CAPS contexts. It is not Used In Titlecase or. To start sentences. More information can be found in the original proposal for adding mtavruli to Unicode. Because of this behavior, that proposal very deliberately specified, and Unicode adopted, that lowercase/mkhedruli Georgian letters should titlecase to themselves, and not to the uppercase/mtavruli form. This ensures that applying the toTitlecase() transformation to a mkhedruli string does not incorrectly give a mixed-case result:

Uppercasing converts mkhedruli to mtavruli: toUppercase("ქართული ენა") = "ᲥᲐᲠᲗᲣᲚᲘ ᲔᲜᲐ"
Titlecasing leaves mkhedruli unchanged: toTitlecase("ქართული ენა") = "ქართული ენა"

However, Unicode unfortunately neglected to also make mtavruli letters titlecase to mkhedruli. Currently, these titlecase to themselves. When applying the titlecase tranformation to an entire mtavruli string, this results in an incorrect mixed-case result:

Titlecasing converts mtavruli into mixed case, which is not valid in modern Georgian usage:
toTitlecase
("ᲥᲐᲠᲗᲣᲚᲘ ᲔᲜᲐ") = "ᲥᲐᲠᲗᲣᲚᲘ ᲔᲜᲐ"

This error should be corrected by changing the Titlecase_Mapping of all assigned codepoints in the Georgian Extended block, setting it equal to the Lowercase_Mapping.

Background information / discussion

The Georgian script has several forms. See The Unicode Standard, section Georgian:

(Two old forms are today only used for liturgical purposes; for modern Georgian writing, see the following quotes about two newer forms.)

... an alphabet called *Mkhedruli* (military), which is the form used for nearly all modern Georgian writing. The Georgian Mkhedruli alphabet has been fundamentally caseless since its development.
... *Mtavruli* forms developed as a particular style of *Mkhedruli* in which the distinction between letters with ascenders and descenders was not maintained. All letters written in the *Mtavruli* style appear with an equal height standing on the baseline, similar to small caps in the Latin script.
... These *Mtavruli* letters have a casing relationship defined with *Mkhedruli* letters: the *Mtavruli* letters are the uppercase forms of the *Mkhedruli* letters, which now are considered lowercase forms.

The Unicode Standard also defines Default Case Conversion:

R3 toTitlecase(X): Find the word boundaries in X [...] For each word boundary, find the first cased character F following the word boundary. If F exists, map F to Titlecase_Mapping(F); then map all characters C between F and the following word boundary to Lowercase_Mapping(C).

Without adjustments (tailoring), this rarely yields desired results, even in English for words like “the”, “McGowan”, or “IBM”.

For an example of possible tailoring options, ICU allows a caller to provide custom word boundaries (which could omit the start of “the”) and provides an option to not lowercase text between word boundaries.

Similarly, CLDR display names are normally “intended to be the form used in the middle of running text”, but when used in other contexts, such as stand-alone or in a user-interface list or menu, many locales specify a transformation to `titlecase-firstword`:

raw CLDR text that is in middle-of-sentence form, typically lowercase, needs to have its first word titlecased

This is implemented by titlecasing only the start of the string and not lowercasing the rest.

The proposed change would have the following effects:

- In titlecasing according to TUS, all mtavruli text would map to mkhedruli. This might be surprising, but would be better than the current mixed-form result.
- In CLDR titlecase-firstword of what should be “lowercase” mkhedruli CLDR text, it would remain mkhedruli (no change).
- In CLDR titlecase-firstword of “uppercase” mtavruli, the result would now be words with an unintended mix of mkhedruli first letters followed by mtavruli letters. This appears to be an unusual / unexpected use case, and might be given low priority.

2.2 Modifier stacked arrowheads should be wb=ALetter like the modifier arrowheads [#581]

Recommended UTC actions

1. **Consensus**: Update the derivation of the Word_Break property to assign Word_Break=ALetter to the modifier stacked arrowheads U+1DFC9..U+1DFCC provisionally assigned by 184-C13, as well as to U+1DFC6 MODIFIER MIDDLE ACUTE ACCENT, U+1DFC7 MODIFIER LETTER SOUTH EAST ARROW, and U+1DFC8 MODIFIER LETTER LOW NORTH EAST ARROW provisionally assigned by 184-C15, when these characters are added to the standard. See L2/26-235 item 2.2.
2. **Action Item** for Josh Hadley, PAG: In Unicode Standard Annex #29, Unicode Text Segmentation, update the derivation to assign Word_Break=ALetter to the modifier stacked arrowheads U+1DFC9..U+1DFCC provisionally assigned by 184-C13, as well as to U+1DFC6 MODIFIER MIDDLE ACUTE ACCENT, U+1DFC7 MODIFIER LETTER SOUTH EAST ARROW, and U+1DFC8 MODIFIER LETTER LOW NORTH EAST ARROW provisionally assigned by 184-C15. For Unicode Version 19.0. See L2/26-235 item 2.2.

PAG input

From Robin Leroy, PAG, while working out properties for the characters proposed in L2/25-157 and L2/25-156R, see elsewhere in this report.

2.3 change Vertical_Orientation of U+1F201 🈁 from Tu to U (PRI-556) [#593]

Recommended UTC actions

1. **Consensus**: Change the Vertical_Orientation of U+1F201 🈁 SQUARED KATAKANA KOKO from Tu to U. For Unicode Version 19.0. See L2/26-235 item 2.3.
2. **Action Item** for Robin Leroy, PAG: Change the Vertical_Orientation of U+1F201 🈁 SQUARED KATAKANA KOKO from Tu to U. For Unicode Version 19.0. See L2/26-235 item 2.3.

Feedback (verbatim)

PRI-556

Date/Time: Mon Aug 17 16:08:03 PT 2026
ReportID: ID20260817160803
Name: Ken Lunde
Report Type: Report Error in Publication/Data
Opt Subject: UAX #50 feedback (PRI #556)

I propose that the Vertical_Orientation property value of U+1F201 🈁 SQUARED KATAKANA KOKO be changed from Tu to U.
The character was originally added as an emoji per document L2/09-026R (see the "Square katakanas" table on page 2),
and there is no reason why an alternate form would be required for vertical writing. In terms of non-emoji
implementations, the broadly-deployed open source Source Han and Noto CJK fonts include a glyph for this symbol, yet
do not provide a separate vertical glyph.

2.4 Initial teaching alphabet: Soft_Dotted ligatures? (PRI-556) [#642]

Recommended UTC actions

1. **No Action**: PAG recommends no action.

Feedback (verbatim)

PRI-556

Date/Time: Sat Aug 22 10:47:38 PT 2026
ReportID: ID20260822104738
Name: Nug
Report Type: Report Error in Publication/Data
Opt Subject:

Suggestion: can the 18.0 ITA characters Ide and Oin have the soft-dotted property?

Background information / discussion

To clarify, "Ide" is U+1DF5D LATIN SMALL LETTER IE and "Oin" is U+1DF5E LATIN SMALL LETTER OI DIGRAPH, both new to Unicode 18.0.

We don't think either should be soft dotted, since any accent would not go over the i part (which is first in IE and second in OI), but would probably be centered on the whole "ligature".

We also have precedent in the following two- or three-part characters, none of which are soft dotted:

U+0133 ij LATIN SMALL LIGATURE IJ: i + j
U+01C8/U+01C9 Lj/lj LATIN LJ: L/l + j
U+01CB/U+01CC Nj/nj LATIN NJ: n + j
U+FB01 fi LATIN SMALL LIGATURE FI: f + i
U+FB03 ffi LATIN SMALL LIGATURE FFI: f + f + i

2.5 UAX #31: toFoldedcase(S) is not defined anywhere [#647]

Recommended UTC actions

1. **Action Item for** Robin Leroy, PAG: Change `toFoldedcase` to `toCasefold` in UAX #31 Figure 6. For Unicode Version 19.0. See L2/26-235 item 2.5.

PAG input

From Roozbeh Pournader, PAG:
UAX #31 in its Figure 6 mentions an operation titled `toFoldedcase(S)`. I could not find any definition for it either in UAX #31 or any other current document published by Unicode. It's probably a typo for `toCasefold`, defined in Chapter 3 of the Core Spec, R4 (https://www.unicode.org/versions/Unicode17.0.0/core-spec/chapter-3/#G63737).

Background

The offending text enters Draft UAX #31 in its Revision 3 (2004; Revision 5 was the first approved version as part of Unicode Version 4.1.0), but that is merely being moved from UAX #15, where it appeared in the Proposed Update Revision 20 (2001-03-09, targeting Version 3.1.0).

At the time, neither the Standard nor its annexes defined an operation toCasefold; the term appears in Unicode Version 3.2.0, with UAX #21, Revision 5, dated 2001-03-26. When that term was introduced, toFoldedcase in UAX #15 was not updated accordingly, and this undefined term remained for twenty-five years.

2.6 UAX #31: Table 3a and its UnicodeSet disagree [#646]

Recommended UTC actions

1. **Action Item for** Robin Leroy, PAG: Remove the UnicodeSets below Table 3b from UAX #31. For Unicode Version 19.0. See L2/26-235 item 2.6.

PAG input

From Roozbeh Pournader, PAG:
UAX #31 Table 3a has a UnicodeSet defined below it that doesn't match it. UTC-176-A66 explicitly removed U+30FB KATAKANA MIDDLE DOT from Table 3a for Unicode 15.1, but when that action was done, we missed to update the UnicodeSet below the table that still includes U+30FB.

The UnicodeSets below that table are not very readable either, which could have been one of the reasons we missed to update one of them.

I think it would be an improvement to either remove those three UnicodeSets below the table from UAX #31, or convert them to use hex notation so they would be more readable.

2.7 UAX #24 rules for scx values are incomplete [#650]

Recommended UTC actions

1. **Consensus**: In UAX #24 Section 3.1, change the constraints C and D per L2/26-235 item 2.7 to add more restrictions to match the examples of disallowed sets in the same section. For Unicode Version 19.0. See L2/26-235 item 2.7.
2. **Action Item for** Ken Whistler, PAG: Update constraints C and D in UAX #24 Section 3.1 per L2/26-235 item 2.7. For Unicode Version 19.0. See L2/26-235 item 2.7.

PAG input

From Roozbeh Pournader, PAG:
UAX #24 rules about scx value are incomplete, and as such, they actually allow the first two rows of Table 8, which is an example list of things the rules are not supposed to allow.

Rules A–D in section 3.1 specify:

- A. Each code point is associated with exactly one non-empty set of values of the sc property. This set is known as the code point's scx set.
- B. The elements of the scx set consist of an unordered list of unique values of the Script (sc) property values.
- C. An scx set either contains a single implicit sc value or one or more explicit sc values.
- D. If the sc property value of a code point is explicit, then that value must be an element of the scx set for that code point as well.

In order to confirm this, I converted the rules A–D in section 3.1 to mathematical rules. For these, let:

- S be all Script values.
- I = {Common, Inherited, Unknown}, the implicit values.
- $E = S \setminus I$, the explicit values.
- s = sc(cp).
- X = scx(cp).

For each code point cp, the rules A–D translate to:

- A. $\emptyset \neq X \subseteq S$
- B. X is a set
- C. $(\exists\, i \in I : X=\{i\}) \vee (\emptyset \neq X \subseteq E)$
- D. $s \in E \implies s \in X$

With those expressed mathematically, it becomes clear that both of the following match all those rules, despite what the UAX claims:

- Row 1: X={Latn}, s = Unknown
- Row 2: X={Common}, s = Inherited

Minimal changes to rules A–D to actually reject the two rows would be changing C and D to:

- C′. $(s \in I \wedge X = \{s\}) \vee (\emptyset \neq X \subseteq E)$
- D′. $s \in E \cup \{\text{Unknown}\} \implies s \in X$

Reworded rules C and D would then be as follows, with "additional text in quotes":

- C. An scx set either contains a single implicit sc value "equal to the sc property value of the code point" or one or more explicit sc values.
- D. If the sc property value of a code point is explicit "or Unknown", then that value must be an element of the scx set for that code point as well.

2.8 Define what Age=Unassigned means [#655]

Recommended UTC actions

1. **Action Item for** Robin Leroy, PAG: In UCD file DerivedAge.txt, add comments to clarify that the code points for which the value of the Age property is Unassigned are the reserved code points. For Unicode Version 19.0. See L2/26-235 item 2.8.
2. **Action Item for** Ken Whistler, PAG: In Unicode Standard Annex #44, Unicode Character Database, in Section 5.14, Character Age, note that the value Unassigned exists and is assigned to reserved code points. For Unicode Version 19.0. See L2/26-235 item 2.8.

PAG input

From Markus Scherer, PAG:

We have this property value:

```
age; NA                    ; Unassigned
```

UAX #44 5.14 Character Age points out that “Unassigned” is special because it's not numeric, even though none of the other long aliases (e.g., `V17_0`) are numeric either.

And while it discusses parsing this value, I cannot find an explicit statement of what Age=Unassigned **means**.

I assume that it means the same set of code points as gc=Cn.

Our own online tools treat this as “all code points”. That seems wrong.

Discussion

The set of code points for which the value of the Age property is Unassigned, according to DerivedAge.txt, is equal to the set of *reserved code points*. For *reserved code points* see definition D15. These are the code points that have General_Category=Unassigned and do not have the Noncharacter_Code_Point property.

The latest draft of DUTS #61 clarifies that `\p{Age=Unassigned}` returns exactly those code points for which the Age property has the value Unassigned, instead of trying to compare Unassigned with age values that correspond to a version of The Unicode Standard.

2.9 L2/26-255 Some Changes to Indic_Syllabic_Category: A Report on UTC 158-A106 [#687]

Recommended UTC actions

1. **No Action**: PAG has no objection to the recommendation from the script encoding working group.

Document

L2/26-255 by Roozbeh Pournader: L2/26-255 Some Changes to Indic_Syllabic_Category: A Report on UTC 158-A106

This document proposes changes InSC assignments.

Background information / discussion

InSC has an effect on grapheme cluster segmentation, via the derivation of InCB: https://www.unicode.org/reports/tr44/#Derivation_InCB.
Changes to this property should therefore be reviewed by the PAG (this would have prevented the breakage of Zanabazar square 𑨯, see L2/26-154 item 8.3).

However, the proposed changes, being from InSC=Other to InSC=Consonant_Killer or InSC=Bindu, have no effect on this derivation, and therefore the proposed changes do not affect the behaviour of algorithms.

2.10 L2/26-256 Categorizing Some Uncategorized Indic Characters: A Report on UTC 164-A63 [#688]

Recommended UTC actions

1. **No Action**: PAG has no objection to the recommendation from the script encoding working group.

Document

L2/26-256 by Roozbeh Pournader: L2/26-256 Some Changes to Indic_Syllabic_Category: A Report on UTC 158-A106

This document proposes changes InSC and InPC assignments.

Background information / discussion

InPC has no effect on algorithms. InSC has an effect on grapheme cluster segmentation, via the derivation of InCB: https://www.unicode.org/reports/tr44/#Derivation_InCB.
Changes to this property should therefore be reviewed by the PAG (this would have prevented the breakage of Zanabazar square 𑨿, see L2/26-154 item 8.3).

However, the proposed changes, being from InSC=Other to InSC=Cantillation_Mark, have no effect on this derivation, and therefore the proposed changes do not affect the behaviour of algorithms.

3. Proposed new scripts & characters

PAG participants have reviewed the following character encoding proposals and prepared draft UCD data as described below. Where the descriptions below compare proposed characters with already-encoded ones, tests are in place to check that their property assignments are consistent.

Work has not yet been carried out on non-UCD data files, such as those for UTS #10 (collation), UTS #39 (security), or UTS #46 (IDNA). When these files are generated, the draft UCD properties may be revised in light of implications in these Unicode Technical Standards.

In the following, the term “propertywise” is a shorthand for “follows a similar pattern in character properties” where the comparison excludes properties that are necessarily distinct, such as Name or Age—the latter because the comparisons are to already-existing characters—, or ones where the differences are irrelevant for the behaviour of text, such as Block or Unicode_1_Name. The Properties and Algorithms Working Group uses a tool to compare tentative property assignments to those for characters that would be expected to have similar behavior and use and therefore similar property assignments.

- L2/25-156R: Unicode request for old-style IPA pitch and tonetic stress marks — Kirk Miller, Michael Ashby [#680]
    - Four modifier symbols.
        - One, DOUBLE GRAVE ACCENT, is a spacing clone of the combining double grave ◌̏, and is propertywise like the spacing U+02DD ˝ DOUBLE ACUTE ACCENT, except for East_Asian_Width and Line_Break (Neutral and Alphabetic rather than ea=Ambiguous, lb=Ambiguous for U+02DD, since the new character does not have legacy East Asian usage). In particular, Diacritic, and `<compat>`-decomposing to SPACE followed by the combining mark.
        - The others are a middle acute, used together with and similarly to the existing ˎ MODIFIER LETTER MIDDLE GRAVE ACCENT (See Figure 2 of the proposal), and arrow-shaped modifier letters, used together with ˅ MODIFIER LETTER DOWN ARROWHEAD. These three characters are propertywise like ˎ MODIFIER LETTER MIDDLE GRAVE ACCENT and ˅ MODIFIER LETTER DOWN ARROWHEAD, in particular Diacritic, as well as Word_Break=ALetter, see L2/16-336, 149-A51 (see also L2/25-183 p. 7); this requires a change to the derivation in UAX #29, proposed elsewhere in this report.
        - Note that DOUBLE GRAVE ACCENT should not be Word_Break=ALetter as it is a spacing clone of a combining mark; see L2/16-336, p. 2, item a.
- L2/25-157: Unicode request for stacked arrowheads — Kirk Miller [#653]
    - Four modifier symbols consisting of < or > (U+02C2 & U+2C3 MODIFIER LETTER LEFT and RIGHT ARROWHEAD) over ^ or ˅ (U+02C4 & U+2C5 UP or DOWN). These should be propertywise like these components, or like the LOW arrowheads [◌˯◌˰], in particular Diacritic, as well as Word_Break=ALetter, see L2/16-336, 149-A51 (see also L2/25-183 p. 7); this requires a change to the derivation in UAX #29, proposed elsewhere in this report. Note that U+02C4 ^ MODIFIER LETTER UP ARROWHEAD is East_Asian_Width=Ambiguous rather than Neutral like the others, see decision UTC-87-M20 and document L2/01-189; the new characters have no legacy East Asian usage, and should be Neutral.
- L2/25-224: Proposal to encode Sirmauri in Unicode — Anshuman Pandey, Biswajit Mandal [#242]
    - This is a script with a pure killer and no invisible stacker nor virama—this property of a script affects the segmentation properties of its letters and of any stacker/virama, in particular the derivation of InCB—; the current set of such scripts is {Ahom, Gurung Khema, Hanunoo, Kirat Rai, Lao, Rejang, Khudawadi, Tagalog, Thai, Tibetan}. It also has independent vowels (more than one, per L2/24-203), restricting the set of similar scripts to {Hanunoo, Khudawadi, Tagalog}. It also has a bindu, contrary to the Philippine scripts; the most similar script in terms of properties is thus Khudawadi.
        - The Sirmauri letters (in the sense of General_Category, i.e., the independent vowels and consonants) are propertywise like those of Khudawadi.
        - The Sirmauri halanta is propertywise like the Khudawadi virama (both pure killers), in particular Diacritic.
        - The Sirmauri anusvara is propertywise like the Khudawadi anusvara, in particular Alphabetic via Other_Alphabetic.

        - The Sirmauri nukta is propertywise like the Khudawadi nukta (both InPC=Bottom), except that pursuant to L2/22-250, L2/22-248 §8, and L2/23-008 §2, its ccc is 0 (Not_Reordered) rather than 7 (Nukta), like the Kawi nukta (but the Kawi nukta is InPC=Top rather than Bottom). In particular, like all other nuktas, it is Diacritic. The proposal is missing an InSC for this character, it should have InSC=Nukta.
    - Sirmauri also has a visarga, which is propertywise like the Devanagari one, in particular Alphabetic via Other_Alphabetic.
    - Sirmauri also has a medial ra, which is propertywise like the Gurung Khema one, in particular Alphabetic via Other_Alphabetic.
    - Sirmauri has an ekam, which the proposal describes as similar to an anji. All three existing anji characters (Bengali U+0980 ঀ, Meetei Mayek U+AAF2 ꫲ, Tirhuta U+11480 𑒀), as well as the existing U+111DA 𑇚 SHARADA EKAM, have General_Category=Lo, so the proposed character should likewise be Lo rather than So as proposed. The Bengali anji differs from the other two in being InSC=Consonant_Placeholder rather than InSC=Other, because it can take a candrabindu, see L2/12-121. No evidence of such behaviour is given for the Sirmauri ekam, it should therefore be propertywise like the Tirhuta anji or the Sharada ekam: InSC=Other.
    - Sirmauri has digits; these are propertywise like the Takri digits.
    - The most interesting feature of this script, from a propertizing standpoint, lies in its dependent vowels.
        - The vowel signs II and U are straightforwardly propertywise like the Dogra ones, InPC=Right and Bottom respectively.
        - The vowel signs O (=E+AA) and AU (=AI+AA) are new entrants to the very exclusive club of decomposable InPC=Top_And_Right dependent vowels, joining the Kannada II (=I+length mark), EE (=E+length mark), AI (=E+AI length mark), and O (=E+UU) and the Balinese PEPET TEDUNG, LA LENGA TEDUNG, and RA REPA TEDUNG.
            - Sirmauri O (=E+AA) and AU (=AI+AA) are thus propertywise like Kannada O (=E+UU), Sirmauri E and AI like Kannada E, and Sirmauri AA like Kannada UU.
            - In particular, this means that the considerations behind decisions UTC-93-C25 and UTC-180-C19 apply to these characters:
                - the E and AI are Bidi_Class=L rather than Bidi_Class=NSM as proposed, even though they are General_Category=Mn, to preserve the behaviour of the Unicode Bidirectional Algorithm under canonical equivalence;
                - the O, AU, and AA are Other_Grapheme_Extend so that they are Grapheme_Cluster_Break=Extend even though they are General_Category=Mc, so that legacy grapheme clusters remain equivalent under canonical equivalence.

- L2/22-219: Proposal to encode MINIMIZING VIRAMA in Brahmi script — Srinidhi A, Sridatta A (1107E) [#192]
    - Another disunification of U+11046 BRAHMI VIRAMA (previously: U+11070 BRAHMI SIGN OLD TAMIL VIRAMA in Unicode Version 14.0, compare the sections on Old Tamil Brahmi in Unicode 14.0 and in Unicode 17.0). This character is a killer that does not form conjuncts and does not cause any reordering, so it is a Pure_Killer as recommended by the proposal (unusually, it is not a separate visible mark, instead it minimizes the consonant to which it applies). As usual for viramas, killers, and invisible stackers, it is Diacritic.
        - Propertywise the MINIMIZING VIRAMA is most like the BRAHMI SIGN OLD TAMIL VIRAMA (the other Brahmi Pure_Killer), except that its Indic_Positional_Category is Not_Applicable rather than Top (since it is not a mark), and except for its Line_Break property of Virama_Final rather than Combining_Mark (see below).
        - It is less similar to the BRAHMI VIRAMA from which it is being disunified, that its Indic_Positional_Category is Not_Applicable rather than Top, its Indic_Positional_Category is Pure_Killer rather than Virama, and its Line_Break property is Virama_Final rather than Virama.
        - It is not very similar to the KHAROSHTHI VIRAMA, which, as mentioned in the proposal, has the same minimizing behaviour in the absence of a following consonant: InSC=Pure_Killer to the Kharoshthi virama's Invisible_Stacker, as the latter forms conjuncts, Line_Break=Virama_Final to the Kharoshthi virama's Combining_Mark (Kharoshthi does not break lines at orthographic syllable boundaries), Indic_Conjunct_Break=Extend to the Kharoshthi virama's Linker (The Kharoshthi virama form conjuncts, and Kharoshthi conjuncts form grapheme clusters).
        - It is very unlike the existing characters with Line_Break=Virama_Final, BATAK PANGOLAT and BATAK PANONGONAN, as these are spacing marks and have a unique reordering behaviour.
        - The interesting property of this character is its Line_Break=Virama_Final assignment. Brahmi is a script that breaks lines at orthographic syllable boundaries, but a minimized consonant (*i.e.*, one to which the MINIMIZING VIRAMA is applied) sticks to the *preceding* akshara, rather than the following as for an ordinary virama. Line_Break=Virama_Final has precisely that effect by the last sub-rule of LB28a, so this is a perfect fit.
            - Interestingly, it is a much better fit than for Batak, where it is more of a workaround; as described in L2/22-086, p. 2 and https://unicode.org/versions/Unicode17.0.0/core-spec/chapter-17/#G27239, line breaks should be allowed before every spacing character, but the reordering behaviour means that the scope of reordering cannot be broken.

4. Bidi

4.1 Fast path in UAX #9 needs more restrictions [#607]

Recommended UTC actions

1. **Action Item for** Manish Goregaokar and Robin Leroy, PAG: Add text to UAX #9 Section 5.1 per L2/26-235 item 4.1 explaining that the first fast path suggested may only be used in some specific scenarios. For Unicode Version 19.0. See L2/26-235 item 4.1.

PAG input

From Roozbeh Pournader, PAG:
The first optimization suggestion in UAX #9, Section 5.1, is incorrect. It says "an effective optimization is to first test for right-to-left characters and invoke the Bidirectional Algorithm only if they are present." This fails for bc=AN characters (which are weak LTR, but can cause bidi reordering in a sequence such as AN ON AN) as well as explicit formatting characters (which are directionless themselves?). Also, if some higher level protocol sets the paragraph level or modifies the algorithm, we can't use that optimization.

Suggested clarification text to be added: "Note that such a fast path may be used only when the effective paragraph level is known to be zero, no higher-level protocol changes bidi processing, and the text contains no R, AL, AN, or directional formatting controls."

4.2 Five bugs in bidi reference implementations [#675]

Recommended UTC actions

1. **Action Item for** Roozbeh Pournader, PAG: Add the five reported bidi test cases to BidiTest.txt and BidiCharacterTest.txt. For Unicode Version 19.0. See L2/26-235 item 4.2.
2. **Action Item for** Ken Whistler, PAG: Fix the C version of the bidi reference implementation to correctly handle C1, C2, and C3 cases. See L2/26-235 item 4.2.
3. **Action Item for** Roozbeh Pournader, PAG: Fix the Java version of the bidi reference implementation to correctly handle J1 and J2 cases. See L2/26-235 item 4.2.

PAG input

From Roozbeh Pournader, PAG:
I found five differences in our bidi reference implementations. Each appear to be a bug in the implementations: three in the C implementation and two in the Java implementation. Each can produce incorrect reordering, not just different internal levels. The published conformance suites currently miss these five cases.

In all examples below, levels correspond to logical input positions, including `x` for characters removed by X9. Reordering lists give zero-based logical indices in visual order, excluding X9 characters. `×62` and `×64` mean repetition.

Finding	C	Java	Rule or definition
C1: wrong PDI used to join runs after an override	Fails	Passes	BD9, BD13, X10
C2: overflow isolate splits a level run	Fails	Passes	BD7, BD13, X5b, X10
C3: extra reversal at level 126	Fails	Passes	L2
J1: PDI does not receive its directional override	Passes	Fails	X6a
J2: bracket overflow preserves previously found pairs	Passes	Fails	BD16, N0

C1. Isolating run sequences can join to the wrong PDI after an override.

Source: brrule.c, `br_FirstSignificantBC()` at line 224, used by `br_UBA_IdentifyIsolatingRunSequences()`.

The helper returns the current `bc`. X6a can already have changed a PDI's current type to L or R. The sequence-building code subsequently looks for `BIDI_PDI` using that helper, so it skips the actual matching PDI and may attach a later, unrelated PDI's level run instead. Matching must preserve the identity of the original formatting characters. See BD9, BD13, and X10.

Example, paragraph level 0:

```
Input:      202E 2067 05D0 2069 202C 202B 0061 2067 05D1 2069 0062
Meaning:    RLO  RLI  ALEF PDI  PDF  RLE  a    RLI  BET  PDI  b
Expected:   x    1    3    1    x    x    2    2    3    2    2
C actual:   x    1    3    1    x    x    2    1    3    1    2

Expected order: 6 7 8 9 10 3 2 1
C actual order: 10 9 8 7 6 3 2 1
```

The first RLI matches the PDI at index 3. C instead joins its run to the PDI at index 9, whose current type has not been overridden. This also changes the relative visual order of the visible `a`, BET, and `b`.

C2. An overflow isolate incorrectly splits a level run.

Source: brrule.c, `br_SpanOneRun()`, especially the unconditional break at line 2338.

The function terminates a run after every isolate initiator. When the depth limit prevents the initiator from raising the level, the following text can remain in the same level run. Splitting that run changes the context used by subsequent rules. See BD7, X5b, and X10.

Example, paragraph level 0:

```
Input: 202A ×62, followed by 05D0 2066 05D1
       LRE ×62, followed by ALEF LRI  BET

Expected final three levels: 125 125 125
C actual final three levels: 125 124 125
Expected order: 64 63 62
C actual order: 62 63 64
```

The 62 LREs establish explicit level 124. The LRI cannot raise it to 126 because the explicit limit is 125. Consequently ALEF, LRI, and BET are all in one level run at this stage; N1 resolves the intervening LRI to R. C prematurely ends the run at LRI and resolves it to L instead. The two Hebrew letters consequently appear in the wrong order.

C3. L2 performs an extra reversal when level 126 is present but no odd level exists.

Source: brrule.c, `br_UBA_ReverseLevels()`, lines 5213 and 5266.

The sentinel for an absent odd level is initialized to `maximum_depth + 1`, or 126. This is also a legitimate resolved level. When all surviving levels are even and the highest is 126, the reversal loop nevertheless executes once at level 126. Under L2, no reversal should occur when no odd level exists.

Example, paragraph level 0:

> Input: 202A ×62, followed by 0661 0662
> LRE ×62, followed by ARABIC-INDIC DIGIT ONE, ARABIC-INDIC DIGIT TWO
>
> Expected final two levels: 126 126
> C actual final two levels: 126 126
> Expected order: 62 63
> C actual order: 63 62

The level calculation is correct, but the digits are reversed.

J1. PDI does not receive the active directional override.

Source: BidiReference.java, `determineExplicitEmbeddingLevels()`, the `case PDI` branch ending at line 585.

The branch sets the PDI's level but omits the corresponding type override required by X6a. That final step applies even to an unmatched PDI.

Example, paragraph level 0:

> Input: 202A 0061 202C 202E 2069 202A 0062
> Meaning: LRE a PDF RLO PDI LRE b
> Expected: x 2 x x 1 x 2
> Java actual: x 2 x x 2 x 2
>
> Expected order: 6 4 1
> Java actual order: 1 4 6

PDI must become R at explicit level 1. Java leaves it neutral; its run has L for both sos and eos because the neighboring runs have explicit level 2. It therefore becomes L and rises to level 2. This changes the relative order of the visible `a` and `b`.

J2. Bracket-stack overflow preserves bracket pairs that BD16 requires it to discard.

Source: BidiPBAReference.java, `locateBrackets()`, lines 233, 234, and 235.

The overflow branch clears `openers` and returns, but leaves `pairPositions` populated. The caller then resolves those earlier pairs. BD16 requires an empty result for the entire isolating run sequence when an opening bracket overflows the 63-entry stack.

Example, paragraph level 1:

> Input: 0061 0028 0062 0029, followed by 0028 ×64
> a (b), followed by 64 opening parentheses
>
> Expected first four levels: 2 2 2 1
> Java actual first four levels: 2 2 2 2
> Remaining 64 levels: all 1 in both results
>
> Expected order: 67 66 ... 4 3 0 1 2
> Java actual order: 67 66 ... 4 0 1 2 3

The earlier `(b)` pair must be discarded on overflow. Without N0 pairing, the closing parenthesis at index 3 resolves to R; Java incorrectly retains the pair and makes it L. This changes its placement and its direction for mirroring. The overflow need not occur within that earlier pair: unmatched openings later in the same isolating run sequence suffice.

5. Normalization

5.1 UAX #15 buffering guidance is incorrect [#628]

Recommended UTC actions

1. **Action Item for** Markus Scherer, PAG: In UAX #15 Section 13.1 “Buffering with Unicode Normalization”, fix the condition from just NFC_QC=Yes to NFC_QC=Yes and ccc=0. For Unicode Version 19.0. See L2/26-235 item 5.1.

PAG input

From Roozbeh Pournader, PAG:
UAX #15 Section 13.1 (https://www.unicode.org/reports/tr15/#Buffering_with_Unicode_Normalization) uses the last QC=Y code point as a flush boundary.

Now consider:
<a, U+0315, U+0316> → <a, U+0316, U+0315>

All three characters are QC=Y for all normalization forms, ccc(a)=0, ccc(U+0315)=232, and ccc(U+0316)=220.

Now suppose the buffer becomes full after U+0316. Section 13.1 currently says to find the last QC=Y character. That is U+0316. It then says to normalize and flush everything before it:

Flushed: <a, U+0315>
Retained: <U+0316>

Normalizing the flushed portion does not move U+0315 because U+0316 is not yet included. Once the prefix has been emitted, U+0316 can no longer move before U+0315. The implementation therefore emits the non-normalized sequence <a, U+0315, U+0316>.

The safe boundary should require QC=Y and ccc=0, instead of just QC=Y.

Consider moving this section under section 14 Implementation Notes.

5.2 UAX #15 and UAX #41 refer to removed normalization charts [#681]

Recommended UTC actions

1. **Action Item** for Ken Whistler, PAG: Remove the mention of normalization charts from UAX #15 and UAX #41. For Unicode Version 19.0. See L2/26-235 item 5.2.

PAG input

From Roozbeh Pournader, PAG:
UAX #15 and UAX #41 refer to normalization charts that no longer exist. Their mention should be removed from both annexes.

6. Line Break

6.1 Dashes aren’t spaces [#582]

Recommended UTC actions

1. **No Action**: PAG recommends no action. This has been fixed in revision 57 (Unicode Version 18.0).

PAG input

From Charlotte Buff, PAG:

In proposed revision 56 of UAX #14, in the description of the `Break_After` line break class in section 5.1, the characters U+2012 FIGURE DASH and U+2013 EN DASH are erroneously listed as examples of “breaking spaces”. These should go under “Visible Word Dividers” instead.

6.2 Change line break property of Tulu-Tigalari independent vowels [#689]

Recommended UTC actions

1. **Consensus**: Change the Line_Break property of the Tulu-Tigalari independent vowels U+11380..U+11389, U+1138B, U+1138E, U+11390, and U+11391 from lb=AS (Aksara Start) to lb=AK (Aksara). For Unicode Version 19.0. See L2/26-235 item 6.2.
2. **Action Item for** Robin Leroy, PAG: Change the Line_Break property of the Tulu-Tigalari independent vowels U+11380..U+11389, U+1138B, U+1138E, U+11390, and U+11391 from lb=AS (Aksara Start) to lb=AK (Aksara). For Unicode Version 19.0. See L2/26-235 item 6.2.

PAG input

From Roozbeh Pournader, PAG, with some notes by Robin Leroy:
During some PAG discussions, Robin Leroy discovered that we have some lb=AS (Aksara_Start, meaning that they can start an orthographic syllable for line breaking, but not continue it) characters that are InCB=Consonant (meaning that they can continue an orthographic syllable for grapheme cluster breaking), and that seems like a mistake. These are the Tulu-Tigalari independent vowels that get categorized as InCB=Consonant because Tulu-Tigalari is an invisible stacker script, which means it doesn't have a dual-purposed virama like Devanagari does, but an invisible conjoiner (as well as two visible vowel killers). (In turn this means that the conjoiner followed by these independent vowels either is degenerate or means that the vowel is conjoined, so the proposed change to the Line_Break assignments either has no effect on nondegenerate text, or fixes a bug; a CVC syllable followed by but not conjoined to an independent vowel would use the visible vowel killer.)

While we don't seem to have evidence of Tulu-Tigalari independent vowels occuring after a conjoiner in the proposals that led to the encoding of the script, the generalized script grammar allows that to happen, so they should become lb=AK. (They take rephas, so it's not inconceivable that they can also occur after a conjoiner.)

7. Segmentation

7.1 L2/26-135 Finite automata for line breaking and segmentation [#567]

Recommended UTC actions

1. **Consensus**: Accept the proposal in L2/26-135 to update Unicode Standard Annexes #14 and #29 to describe a state machine implementation, and to include data files in the UCD. For Unicode Version 19.0. See L2/26-235 item 7.1.
2. **Action Item** for Robin Leroy, PAG: Apply the changes described in L2/26-135 to Unicode Standard Annex #14, Unicode Line Breaking Algorithm. For Unicode Version 19.0. See L2/26-235 item 7.1.
3. **Action Item** for Robin Leroy, PAG: Apply the changes described in L2/26-135 to Unicode Standard Annex #29, Unicode Text Segmentation. For Unicode Version 19.0. See L2/26-235item 7.1.
4. **Action Item** for Robin Leroy, PAG: Add data files based on those for PRI-555 to the Unicode Character Database, for Unicode Version 19.0. Include the CSS language-independent tailorings, but not the language-specific tailorings. See L2/26-235 item 7.1.
5. **Action Item** for Ken Whistler, PAG: In Unicode Standard Annex #44, Unicode Character Database, consider updating Section 6.2 Segmentation Test Files and Documentation, to mention the new data files. For Unicode Version 19.0. See L2/26-235 item 7.1.
6. **Action Item** for Robin Leroy, PAG: Work with ICU-TC to address the issues with word breaking rule statuses and CSS tailorings of line breaking identified in L2/26-235 item 7.1. For Unicode Version 19.0.
7. **Action Item** for Michelle Perham, UTC: Close PRI #555. See L2/26-235 item 7.1.

Document

L2/26-135 by Robert Bastian and Robin Leroy.

The document proposes adding the definition of a DFA with lookaheads to UAX #14 and UAX #29, and the adding data files for the underlying symbols, states, and transitions to the UCD, to facilitate performant implementation of line breaking and segmentation. A Public Review Issue has been created for this proposal.

Background information / discussion

The group discussed this proposal, and agreed that there is an implementability problem to be solved, and that the proposal is an adequate solution. During the discussion, the question of the maintenance burden of the DFA-based implementation was raised. While maintaining this implementation is non-negligible work, this work already needs to be carried out in another TC (ICU-TC) within the Unicode Consortium; we expect that doing this implementation work together with the specification work will reduce the workload rather than increase it.

The group was satisfied that the implementation was principled and long-standing, so that it should remain maintainable and understandable going forward. In particular, the current maintainer, Robin Leroy, noted that he was not taught by his predecessor, Andy Heninger, but was able to pick up and improve the implementation. This is because, as noted in the document, this implementation is based on a classic computer science textbook [ASU86], and the extensions and improvements have been kept simple and principled (in particular, affecting only the generator for the DFA, not the logic of its execution).

Roozbeh Pournader identified the following issues in the behaviour of the tailored state machines and of the break statuses, which should be addressed in coordination with ICU-TC:

3. I think these two accepting word-break states need a final "Letter" in their last field:
   Hebrew_Letter Single_Quote ExtendmHanmComplexContext ; Yes ; ;
   Hebrew_Letter Single_Quote ZWJ ; Yes ; ;
4. It looks like CSS requirements before lb=CJ have changed relatively recently, in June 2026 (based on a decision in August 2024). In normal mode, it used to allow breaks before CJ, but it no longer does. So both LineBreakTailoring_normal.txt and LineBreakTailoring_normal_cj.txt need to be updated to reflect that. This is also used as an example in Section 9.3.3.1 of the PDF, which may also need to be updated.
5. For both "漢-漢" and "漢–漢", using U+2010 and U+2013 respectively, the generic loose file prohibits the break before the hyphen/dash, while the _cj loose file permits it. But CSS's permission when the preceding character has the class ID is not restricted to Chinese/Japanese. The generic loose profile needs the corresponding mappings too.
6. The Chinese/Japanese loose hyphen tailoring seems to remove an existing break: For <U+6F22 U+2010 U+0061>, or "漢-a", in default mode and generic loose we have "漢 × - ÷ a". In Chinese/Japanese loose we then allow a break before the hyphen but disallow it after the hyphen, making it "漢 ÷ - × a". I don't think this is intentional.

Robin Leroy replied:

[The approach of taking segmentation/line breaking as a tokenization problem] has the interesting consequence that breaks cannot depend on context before a break opportunity (this is the reason for decision 176-C27, the rule couldn't be implemented if it distinguished BK from ZW).
But we have LB20a (don't break after a word-initial hyphen), so (sot) - × a.
This is why 漢 ÷ - × a.

I think the fix here would be to remove LB20a in line-break:loose mode.

7. The PDF, on page 8, proposes this text for UAX#29: "If there is no type, the preceding segment is an interword segment, such as spaces and punctuation." And then "This classification can be used to extract the words from a piece of text." But running the machine seems to give an empty Break_Type after Han ideographs and Hiragana syllables. This means Han and Hiragana may get discarded if implementations follow the advice closely.

7.2 Add Malayalam example to GraphemeBreakTest.txt [#611]

Recommended UTC actions

1. **No Action**: PAG recommends no action. The proposed test case has already been added to GraphemeBreakTest.txt.

PAG input

From Roozbeh Pournader, PAG:
Our recent change to GB9c in UAX #29 (see UTC-187-C47) also fixed certain Malayalam conjuncts, as mentioned in the related PAG report (L2/26-096, Section 5.1). It would be great if we could add one such example to GraphemeBreakTest.txt. For example:

```
÷ 0D2F × 200C × 0D4D × 0D2F ÷
```
(ya, ZWNJ, virama, ya)

Which is the last row from Unicode 17.0 Core Spec Table 12-38 (https://www.unicode.org/versions/Unicode17.0.0/core-spec/chapter-12/#G706410).

7.3 More ALetters: A Report on UTC 162-A2 [#665]

Recommended UTC actions

1. **Consensus**: Change the Word_Break property of the following characters from Other to ALetter. For Unicode Version 19.0. See L2/26-235 item 7.3:
    - U+02D9 DOT ABOVE
    - U+0888 ARABIC RAISED ROUND DOT
    - U+1807 MONGOLIAN SIBE SYLLABLE BOUNDARY MARKER
    - U+180A MONGOLIAN NIRUGU
    - U+2E32 TURNED COMMA
    - U+2E35..U+2E39 TURNED SEMICOLON..TOP HALF SECTION SIGN
    - U+A700..U+A707 MODIFIER LETTER CHINESE TONE YIN PING..MODIFIER LETTER CHINESE TONE YANG RU
    - U+AB6A MODIFIER LETTER LEFT TACK and U+AB6B MODIFIER LETTER RIGHT TACK
    - U+10EC9 ARABIC SMALL BASELINE FATHA and U+10ECA ARABIC SMALL BASELINE DOTLESS HEAD OF KHAH.
2. **Action Item for** Robin Leroy, PAG: Change the Word_Break property of U+02D9, U+0888, U+1807, U+180A, U+2E32, U+2E35..U+2E39, U+A700..U+A707, U+AB6A, U+AB6B, U+10EC9, and U+10ECA to ALetter in the UCD data files. For Unicode Version 19.0. See L2/26-235 item 7.3.

3. **Action Item for** Josh Hadley, PAG: Add U+02D9, U+0888, U+1807, U+180A, U+2E32, U+2E35..U+2E39, U+A700..U+A707, U+AB6A, U+AB6B, U+10EC9, and U+10ECA to Table 3 (under ALetter additions) in UAX #29. For Unicode Version 19.0. See L2/26-235 item 7.3.
4. **Action Item for** Josh Hadley, PAG: Adjust UAX #29 text about modifier symbols not having WB=ALetter, along the lines of the editorial corrections in L2/26-212. For Unicode Version 19.0. See L2/26-235 item 7.3.
5. **Action Item for** Roozbeh Pournader, PAG: Adjust core spec text about modifier symbols not having WB=ALetter, along the lines of the editorial corrections in L2/26-212. For Unicode Version 19.0. See L2/26-235 item 7.3.

Document

L2/26-212, “More ALetters: A Report on UTC 162-A2” by Roozbeh Pournader

In response to a UTC action item, the document proposes changing some characters from wb=Other to wb=ALetter.

Background information / discussion

For future additions: PAG’s current practice of identifying a functionally similar encoded character and copying its properties should generally catch such cases. New modifier letters with General_Category=Lm will normally acquire ALetter automatically through Alphabetic=Yes. For spacing modifiers assigned another general category, such as Sk or Po, reviewers should explicitly consider whether the character occurs adjacent to letters as part of the same orthographic word or phonetic transcription and, if so, try to find a functionally similar character.

For example, the current process would have caught U+AB6A and U+AB6B by comparison with U+02D4 and U+02D5. Conversely, copying U+0888 for U+10EC9 and U+10ECA reproduced U+0888’s mistake.

A useful question for a proposal assignment checklist could be:

“Can this character occur immediately before, after, or between letters or syllabic characters without marking a word boundary?”

7.4 Reviewing Word_Break Open Issues: A Report on UTC 162-A3 [#670]

Recommended UTC actions

1. **Consensus**: Change the Word_Break property of U+058A ARMENIAN HYPHEN from ALetter to Other and the Word_Break property of U+FF0C FULLWIDTH COMMA and U+FF1B FULLWIDTH SEMICOLON from MidNum to Other. For Unicode Version 19.0. See L2/26-235 item 7.4.

2. **Action Item for** Robin Leroy, PAG: Change the Word_Break property of U+058A ARMENIAN HYPHEN, U+FF0C FULLWIDTH COMMA, and U+FF1B FULLWIDTH SEMICOLON to Other. For Unicode Version 19.0. See L2/26-235 item 7.4.
3. **Action Item for** Josh Hadley, PAG: Reflect the changes to the Word_Break properties of U+058A ARMENIAN HYPHEN, U+FF0C FULLWIDTH COMMA, and U+FF1B FULLWIDTH SEMICOLON in UAX #29 Table 3. For Unicode Version 19.0. See L2/26-235 item 7.4.

Document

L2/26-214, “Reviewing Word_Break Open Issues: A Report on UTC 162-A3” by Roozbeh Pournader

The document disposes various review notes from UAX #29 revision 36 and proposes Word_Break changes for three characters.

Background information / discussion

U+058A ARMENIAN HYPHEN had changed from wb=Other to wb=ALetter in Unicode 13.0. PAG participants could not find any stated rationale for the change in any Unicode record. It appears to have been changed together with some other Armenian characters.

The discussion may have started by a piece of public feedback received on PRI-355 by David Corbett. It's one sentence:

Because the Armenian punctuation marks U+055B, U+055C, and U+055E occur within words, rather than occurring finally, they should have Word_Break = ExtendNumLet

This feedback led to the following UTC dispositions:

[155-C3] Consensus: Change the Word_Break property for U+055B, U+055C, and U+055E to ALetter for Unicode 11.0.

[155-A2] Action Item for Mark Davis: Update WordBreak property for U+055B, U+055C, and U+055E to ALetter for Unicode 11.0.

[155-A3] Action Item for Mark Davis, Editorial Committee: Update UAX #29 Table 3 with the changes to ALetter property for U+055B, U+055C, and U+055E.

[155-A4] Action Item for Mark Davis, Editorial Committee: Prepare a PUA (*sic*) UAX #29 for Unicode 12.0 including Armenian Punctuation U+055A, U+055F, U+058A; and characters from the blocks Modifier Tone Letters and Modifier Spacing Letters.

C2, A2, and A3 addressed the feedback—not quite as proposed—; A4 is recorded immediately afterwards, but pertains to characters not listed in the feedback. No discussion is recorded, as was common between 1999 and 2025—after UTC-79 and until UTC-183. PAG participants speculate that during UTC discussion, the question was raised of whether it should be generalized to the other Armenian Punctuation characters. The full stop was already classified, and the comma is not word-internal, so the remaining three were candidates.

The UTC was apparently less certain of these than of the three that were changed in Unicode 11, since it asked for these changes to be proposed in a later version, presumably to gather feedback. The UTC also did not specify what Word_Break assignments these three characters should receive.

Table 3 of Proposed Update UAX #29, Revision 36 lists U+058A under ALetter, and its modification history identifies that entry as an addition; this, together with the changes of U+055A to ALetter and U+055F to MidLetter, is the outcome of UTC-155-A4.

Note that while the UTC was less thorough about taking separate decisions for changes to property assignment in the past (which can also make it more difficult to trace the rationale for such changes), it is clear that the UTC was aware that it was approving these changes, as it instructed that the data file be updated accordingly, at the same meeting where it approved the UAX for publication:

[162-A4] Action Item for Mark Davis, Markus Scherer: Sync Wordbreak data file in the UCD to UAX #29 Table 3 update of revision 36. For Unicode 13.0.
[162-C17] Consensus: Approve the publication of the UAXes and UTS #10, UTS #39, UTS #46, and UTS #51 for Unicode version 13.0.

7.5 U+21AE breaks segmentation consistency with canonical equivalence [#682]

Recommended UTC actions

1. **Consensus**: Change the Extended_Pictographic property of U+21AE LEFT RIGHT ARROW WITH STROKE from No to Yes. For Unicode Version 19.0. See L2/26-235 item 7.5.
2. **Action Item for** Robin Leroy, PAG: Change the Extended_Pictographic property of U+21AE LEFT RIGHT ARROW WITH STROKE from No to Yes. For Unicode Version 19.0. See L2/26-235 item 7.5.

PAG input

From Roozbeh Pournader, PAG:

U+21AE is causing inconsistencies in grapheme breaks and word breaks. It is Extended_Pictographic=No, while it canonically decomposes to <U+2194, U+0338>, where U+2194 has Extended_Pictographic=Yes. This means that a sequence such as <U+1F600, U+200D, U+21AE> has two grapheme clusters, while its NFD form, <U+1F600, U+200D, U+2194, U+0338> is only one grapheme cluster.

Similarly, for the same character sequence, there is a word break after U+200D in the original sequence, but not in its NFD form.

The inconsistencies in these two algorithms across canonical equivalence for U+21AE and U+2194 started in Unicode 11.0. The easiest and cleanest fix appears to be changing the Extended_Pictographic property of U+21AE to Yes.

I should note that Unicode 10.0 had its own inconsistency issues for extended grapheme cluster breaks across normalization. For example, it had these different breaks for the following canonically equivalent sequences:

NFC: ÷ 1F44D × 0CC0 ÷ 1F3FB ÷
NFD: ÷ 1F44D × 0CBF × 0CD5 × 1F3FB ÷

That issue is since fixed.

Background information / discussion

The Unicode 10.0 extended grapheme cluster inconsistency illustrated by <U+1F44D, U+0CC0, U+1F3FB> and its NFD form was resolved in Unicode 11.0 by moving emoji modifiers from `Grapheme_Cluster_Break=E_Modifier` to `Extend` and removing rule GB10. This change was approved by UTC-154-C30, following Manish Goregaokar's PRI #355 feedback of January 2, 2018, with the disposition recorded in L2/18-055.

The underlying property inconsistency persisted until Unicode 16.0: U+0CC0 had `Grapheme_Cluster_Break=SpacingMark`, while both characters in its canonical decomposition, <U+0CBF, U+0CD5>, had `Grapheme_Cluster_Break=Extend`. UTC-180-C19 assigned `Other_Grapheme_Extend=Yes` to several Kannada and Balinese vowel signs, including U+0CC0, thereby changing their `Grapheme_Cluster_Break` values from `SpacingMark` to `Extend`.

The proposed change to U+21AE would make it the only assigned character with `Extended_Pictographic=Yes` and `Emoji=No`. There is substantial precedent for this combination: before Unicode 17.0, 672 assigned characters had those property values. Their `Extended_Pictographic` assignments were removed for Unicode 17.0 by UTC-182-C20. That history gives reason to expect that introducing a single such exception to restore segmentation consistency across canonically equivalent sequences would not cause new issues.

7.6 more GraphemeBreakTest.txt test cases (PRI-556) [#685]

Recommended UTC actions

1. **Action Item for** Robin Leroy, PAG: Consider adding GraphemeBreakTest cases from feedback ID20260929152114. For Unicode Version 19.0. See L2/26-235 item 7.6.

Feedback (verbatim)

PRI-556

Date/Time: Tue Sep 29 15:21:14 PT 2026
ReportID: ID20260929152114
Name: Don Hosek
Report Type: Report Error in Publication/Data
Opt Subject: Gaps in cluster text

I found some errors in my cluster handling code that are not currently covered by the tests in GraphemeBreakTest.txt In particular, the following should be added:

÷ 1F6D1 × 200D × 0308 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] ZERO WIDTH JOINER (ZWJ) × [9.0] COMBINING DIAERESIS (Extend_ConjunctExtender) ÷ [2.0]
÷ 1F6D1 × 200D × 200D ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] ZERO WIDTH JOINER (ZWJ) × [9.0] ZERO WIDTH JOINER (ZWJ) ÷ [2.0]
÷ 1F6D1 × 200D × 0903 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] ZERO WIDTH JOINER (ZWJ) × [9.1] DEVANAGARI SIGN VISARGA (SpacingMark) ÷ [2.0]
÷ 1F6D1 × 200D × 0308 ÷ 1F6D1 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] ZERO WIDTH JOINER (ZWJ) × [9.0] COMBINING DIAERESIS (Extend_ConjunctExtender) ÷ [999.0] OCTAGONAL SIGN (ExtPict) ÷ [2.0]
÷ 1F6D1 × 200D × 094D × 0915 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] ZERO WIDTH JOINER (ZWJ) × [9.0] DEVANAGARI SIGN VIRAMA (Extend_ConjunctLinker) × [9.3] DEVANAGARI LETTER KA (LinkingConsonant) ÷ [2.0]
÷ 1F6D1 × 0903 × 200D ÷ 1F6D1 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.1] DEVANAGARI SIGN VISARGA (SpacingMark) × [9.0] ZERO WIDTH JOINER (ZWJ) ÷ [999.0] OCTAGONAL SIGN (ExtPict) ÷ [2.0]
÷ 1F6D1 × 094D × 200D × 1F6D1 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] DEVANAGARI SIGN VIRAMA (Extend_ConjunctLinker) × [9.0] ZERO WIDTH JOINER (ZWJ) × [11.0] OCTAGONAL SIGN (ExtPict) ÷ [2.0]
÷ 1F6D1 × 0308 × 094D × 200D × 1F6D1 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] COMBINING DIAERESIS (Extend_ConjunctExtender) × [9.0] DEVANAGARI SIGN VIRAMA (Extend_ConjunctLinker) × [9.0] ZERO WIDTH JOINER (ZWJ) × [11.0] OCTAGONAL SIGN (ExtPict) ÷ [2.0]
÷ 1F6D1 × 094D × 0915 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] DEVANAGARI SIGN VIRAMA (Extend_ConjunctLinker) × [9.3] DEVANAGARI LETTER KA (LinkingConsonant) ÷ [2.0]
÷ 1F6D1 × 094D × 200D × 0915 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] DEVANAGARI SIGN VIRAMA (Extend_ConjunctLinker) × [9.0] ZERO WIDTH JOINER (ZWJ) × [9.3] DEVANAGARI LETTER KA (LinkingConsonant) ÷ [2.0]
÷ 1F6D1 × 094D × 200C ÷ 0915 ÷ # ÷ [1.0] OCTAGONAL SIGN (ExtPict) × [9.0] DEVANAGARI SIGN VIRAMA (Extend_ConjunctLinker) × [9.0] ZERO WIDTH NON-JOINER (ExtendmConjunctLinkermConjunctExtender) ÷ [999.0] DEVANAGARI LETTER KA (LinkingConsonant) ÷ [2.0]

7.7 Feedback L2/26-217 on Finite automata for line breaking and segmentation (PRI-555) [#679]

Recommended UTC actions

1. **Action Item for** Robin Leroy, PAG: Work with ICU-TC to understand which partition of the dictionary symbols would be generally useful for implementers, and partition the dictionary symbols accordingly. For Unicode Version 19.0. See L2/26-235 item 7.7 and L2/26-217 §5c.
2. **Action Item for** Robin Leroy, PAG: Document which UnicodeSet features are used in the data files for the segmentation and line breaking DFAs. For Unicode Version 19.0. See L2/26-235 item 7.7 and L2/26-217 §5d.

Feedback (verbatim)

PRI-555

Date/Time: Sat Sep 19 22:35:02 PT 2026
ReportID: ID20260919223502
Name: Kip Cole
Report Type: PRI Feedback
Opt Subject: Feedback on PRI 555

I write and maintain unicode_string, a conformant implementation of UAX #14 and UAX #29 written in the Elixir
language.

This feedback comes from having implemented the proposal against Unicode Version 18.0 data, comparing it directly
with my existing hand-written implementation, and then adopting it: the published tables drive all four break types
in the released version.

Please find attached a PDF of my feedback, and a link to the same document in the github repo.

Thanks for the opportunity to contribute,

Regards, —Kip

Links:

- Feedback document: https://github.com/elixir-unicode/unicode_string/blob/main/pri555-feedback.pdf
- Implementation: https://github.com/elixir-unicode/unicode_string

Background information / discussion

The linked document has been submitted as L2/26-217.

The group noted that this document was generally positive, and confirmed much of the rationale behind the proposal in PRI-555: implementers struggle to produce implementations of line breaking that are both performant and conformant.

The issue described in §5a (missing eot symbol for grapheme and word segmentation) has been corrected in the PRI-555 data files.
It was noted that partitioning the dictionary symbols by script could be useful, since implementations typically used separate models depending on the script.

The group also agreed that defining the subset of UTS #61 features used would be useful. During discussions between the authors of L2/26-135, it was noted that not all implementations carry the data for the Name property; while it is useful for clarity and maintainability to use names in addition to hexadecimal code points, it would be possible to use only those named-elements that have constituent hexadecimal-digits; an implementation that does not have the Name property would then be able to interpret these files under the assumption that they are well-formed—though it would not be able to validate the names against the code points.

8. Collation

8.1 Tertiary level collation of characters secondary equal to ß [#663]

Recommended UTC actions

1. **Consensus**: In the DUCET, correct the tertiary weights for U+210F ℏ PLANCK CONSTANT OVER TWO PI and U+1D6A6 MATHEMATICAL ITALIC SMALL LIGATURE LONG S WITH DESCENDER S from 2 to 5. For Unicode Version 19.0. See L2/26-235 item 8.1.
2. **Action Item for** Ken Whistler, Robin Leroy, PAG: In the DUCET, correct the tertiary weights for U+210F ℏ PLANCK CONSTANT OVER TWO PI and U+1D6A6 MATHEMATICAL ITALIC SMALL LIGATURE LONG S WITH DESCENDER S from 2 to 5. For Unicode Version 19.0. See L2/26-235 item 8.1.

PAG input

From Robin Leroy, PAG, while working on collation folding.

The file allkeys.txt, Version 18.0.0, has the following lines:

```
0073  ; [.26FA.0020.0002] # LATIN SMALL LETTER S
[…]
1D6A6 ; [.26FA.0020.0002][.0000.011F.0002][.26FA.0020.0002] # MATHEMATICAL ITALIC SMALL LIGATURE LONG S WITH DESCENDER S
00DF  ; [.26FA.0020.0004][.0000.011F.0004][.26FA.0020.0004] # LATIN SMALL LETTER SHARP S
1DF95 ; [.26FA.0020.0004][.0000.011F.0004][.26FA.0020.0004] # LATIN SMALL LIGATURE LONG S WITH DESCENDER S
1E9E  ; [.26FA.0020.000A][.0000.011F.0004][.26FA.0020.000A] # LATIN CAPITAL LETTER SHARP S
```

In other words, the last four characters are all primary equal to "ss", and they are all secondary equal to each other but secondary different from "ss", as expected.
They are then tertiary ordered as follows:

MATHEMATICAL ITALIC SMALL LIGATURE LONG S WITH DESCENDER S <<< LATIN SMALL LETTER SHARP S === LATIN SMALL LIGATURE LONG S WITH DESCENDER S <<< LATIN CAPITAL LETTER SHARP S

This seems weird; considering

```
0061  ; [.2485.0020.0002] # LATIN SMALL LETTER A
[…]
1D44E ; [.2485.0020.0005] # MATHEMATICAL ITALIC SMALL A
```

whence LATIN SMALL LETTER A <<< MATHEMATICAL ITALIC SMALL A, one would expect LATIN SMALL LIGATURE LONG S WITH DESCENDER S <<< MATHEMATICAL ITALIC SMALL LIGATURE LONG S WITH DESCENDER S.

This also means that when collation folding at the second level to the first element of the equivalence class (under the expectation that this is the least marked element), in Unicode Version 18.0, U+00DF

ß starts folding to MATHEMATICAL ITALIC SMALL LIGATURE LONG S WITH DESCENDER S, which seems a lot more marked, whereas it would previously fold to itself.

Background

The tertiary weights are fixed based on types of characters, see 10.2 Tertiary Weight Table.

The math small ß, being `<font>` decomposable, should have tertiary 05, which would still sort less-than the capital ß. It does have `<font>`, but doesn't come out as 05. The reason is that this character is one of two that are `<font>` decomposable and have a recursive decomposition for collation (including ß decomposing to ss) that is a sequence.

The code that handles the tertiary weights of sequences did not handle `<font>` decompositions, leading to the assignment of the default tertiary 2 for both U+1D6A6 and U+210F ℏ PLANCK CONSTANT OVER TWO PI (`<font>` decomposable to U+0127 ħ, itself treated in collation as if it canonically decomposed to h followed by U+0335 COMBINING SHORT STROKE OVERLAY, as it used to in Unicode Version 1.1.5).

The proposed change reorders MATHEMATICAL ITALIC SMALL LIGATURE LONG S WITH DESCENDER S tertiary after ß=LATIN SMALL LIGATURE LONG S WITH DESCENDER.

It moves the distinction between ħ and ℏ from the identical to the tertiary level.

8.2 Notes on the normalization of collation folding [#680]

Recommended UTC actions

1. **Action Item for** Markus Scherer, PAG: In Unicode Technical Standard #10, Unicode Collation Algorithm, add a note in Section 11.1, Collation Folding, explaining that the result of collation folding is not normalized and should not be normalized before comparison. For Unicode Version 19.0. See L2/26-235 item 8.2.

PAG input

From Robin Leroy, PAG:

Consider U+018D ƍ LATIN SMALL LETTER TURNED DELTA. In UCA 18.0, its collation elements are [.27C3.0020.0004][.2793.0020.0004].

The letter ƍ is an archaic spelling of /zʷ/, and accordingly its expansion collates primary- and secondary-equal to "zw" [.27C3.0020.0002][.2793.0020.0002].

At the tertiary level, something interesting happens. Since the encoding of U+1ABF in Unicode Version 13.0, an implementation of collation folding that folds to the least element of the equivalence class, resolving the identical level in code point order, would fold the expansion to the sequence

U+1DE6 COMBINING LATIN SMALL LETTER Z, U+1ABF COMBINING LATIN SMALL LETTER W BELOW.

This sequence is unnormalized: canonical reordering puts the mark below first. That is not a problem *per se*: collation folding *starts* with NFD per https://www.unicode.org/reports/tr10/#Collation_Folding but is not followed by it, so the behaviour will be correct (\u1DE6\u1ABF = \u1ABF\u1DE6 = \u1ABF\u034F\u1DE6 ≠₁ o̱ = \u1DE6\u034F\u1ABF, with the first three folding to wz, wz, and \u1ABF\u1DE6 and the last two to zw, zw, and \u1DE6\u1ABF at levels 1 through 3) and collation-folded strings are not meant for general interchange where they might get normalized, they're going to look like barely-intelligible gobbledygook anyway).

But it may be worth calling out in case someone puts their keys somewhere that needs normalization (normalizing the keys would result in incorrect equivalences). I think if you want your keys to be normalization-resistant, you need to treat these expansions like contractions, and insert something like a CGJ in there, and preserve the ignorable (folding all ignorables to the same one, and a sequence to a single one, of course) when folding \u1DE6\u034F\u1ABF.

9. Security

9.1 Bad Java code in UTS #39 [#594]

Recommended UTC actions

1. **No Action**: PAG recommends no action: the issue has been corrected in Version 18.0 of UTS #39.

PAG input

From Roozbeh Pournader, PAG:

The Java code in Section 5.3 of UTS #39 looks wrong. Specifically, in the loop it has `i += Character.charCount(i)` which looks very wrong: it is looking at the UTF-16 length of the index itself, as opposed to the UTF-16 length of the code point `cp` at position `i` of the string.

9.2 Fast path in UTS #39 skeleton calculation is incorrect [#598]

Recommended UTC actions

1. **Action Item** for Josh Hadley and Roozbeh Pournader, PAG: Update UTS #39 to include four more bidi classes (AN, RLE, RLO, and RLI) in the fast path optimization for bidiSkeleton. For Unicode 19.0. See L2/26-235 item 9.2.

PAG input

From Roozbeh Pournader, PAG:

The fast path specified in UTS #39 for computing skeletons appears incorrect to me.

UTS #39 currently says: "The bidiSkeleton is costlier to compute than the internalSkeleton, as the bidirectional algorithm must be applied. However, a fast path can be used: if d=LTR and X has no characters with bidi classes R or AL, bidiSkeleton(d, X) = internalSkeleton(X)."

But it appears to me that there are some other bidi classes that can cause reordering too. For example, the sequence X=<RLO, a, b, PDF> appears to be a counterexample to that fast path to me. bidiSkeleton(LTR, X) = <b, a> while internalSkeleton(X) = <a, b>. I believe these four bidi classes should be added to the list of two classes above: AN, RLE, RLO, and RLI.

9.3 Wrong claim about idempotency of skeleton() in UTS #39 [#595]

Recommended UTC actions

1. **No Action**: The text of UTS #39 has already been updated for Unicode 18.0.

PAG input

UTS #39, in its Section 4, claims that: "Implementations that use the confusable data do not have to recursively apply the mappings, because the transforms are idempotent. That is, *skeleton(skeleton(X)) = skeleton(X)*". I think that is no longer true since Unicode 15.1, when bidiskeleton was added and skeleton was redefined to be bidiSkeleton(LTR, X).

My example is the one-character string X=<U+FB4F HEBREW LIGATURE ALEF LAMED>. It is mapped by confusables.txt to two Hebrew characters, <U+05D0 HEBREW LETTER ALEF, U+05DC HEBREW LETTER LAMED>. Then we have skeleton(X)=bidiSkeleton(LTR,X)=internalSkeleton(X)=<U+05D0, U+05DC>. But skeleton(<U+05D0, U+05DC>)=bidiSkeleton(LTR,<U+05D0, U+05DC>)=<U+05DC, U+05D0>.

The idempotency claim is still true about internalSkeleton(), which is the pre-15.1 version of skeleton(). The text of UTS #39 should be updated to say *internalSkeleton(internalSkeleton(X)) = internalSkeleton(X)*.

9.4 Discrepancy between UTS #39 and RFC 6532 about local part of email addresses [#601]

Recommended UTC actions

1. **Consensus**: In UTS #39, Section 3.3 Email Security Profiles for Identifiers, change the requirement for the local-part of an email address from NFKC to NFC. For Unicode Version 19.0. See L2/26-235 item 9.4.
2. **Action Item for** Josh Hadley and Roozbeh Pournader, PAG: Change the requirement for the local-part of an email address in UTS #39 from NFKC to NFC. For Unicode Version 19.0. See L2/26-235 item 9.4.

PAG input

From Roozbeh Pournader, PAG:

UTS #39 specifies that the local-part of an email address "must be in NFKC format". This is in contradiction with RFC 6532 "Internationalized Email Headers" that says: "normalization form NFC SHOULD be used. Actually, if one is going to do internationalization properly, one of the most often cited goals is to permit people to spell their names correctly. Since many mailbox local parts reflect personal names, that principle applies to mailboxes as well. The NFKC normalization form SHOULD NOT be used because it may lose information that is needed to correctly spell some names in some unusual circumstances."

Apart from who is correct here, this is a discrepancy that should be addressed in UTS #39. My recommendation is switching to NFC and making the two standards consistent. Otherwise, valid and conforming EAIs could be rejected unnecessarily.

Update: Lushootseed uses U+02B7 ʷ as an orthographic marker of labialization. The Tulalip Tribes write a certain personal Indigenous name as słukʷałuli̓ca? (see https://tulaliplushootseed.com/kylee-sohappy/) and NFKC(słukʷałuli̓ca?) = słukwałuli̓ca?: Unicode gives ʷ the compatibility decomposition <super> 0077, so NFKC replaces it with ordinary w. Nuučaaʔuł likewise uses xʷ in its practical orthography.

Chatino uses uppercase-shaped modifier letters as tone letters. Unicode now encodes U+A7F1 □, U+A7F2 ᶜ, and U+A7F3 ᶠ specifically for Chatino, with these NFKC mappings: □ -> S, ᶜ -> C, ᶠ -> F. The encoding proposal says that these letters belong to a community orthography developed in Oaxaca and are used in teaching materials (L2/20-251). NFKC therefore turns an orthographic tone letter into an ordinary baseline capital letter and makes the original unrecoverable.

Thus these are direct examples supporting RFC 6532's concern: NFKC can destroy information required to spell Indigenous words and names correctly. Under the current UTS #39 email profile, such a local part cannot satisfy the requirement that it be NFKC.

9.5 UAX #31: consider withdrawing the Excluded Scripts table [#645]

Recommended UTC actions

1. **Consensus**: In UAX #31, replace the contents of Table 4, Excluded Scripts, with references to scripts that are not Excluded. For Unicode Version 19.0. See L2/26-235 item 9.5.
2. **Action Item for** Markus Scherer, PAG: In UAX #31, replace the contents of Table 4, Excluded Scripts, with references to scripts that are not Excluded. For Unicode Version 19.0. See L2/26-235 item 9.5.

PAG input

From Markus Scherer, PAG

UAX #31 has the following tables:

- Table 4. Excluded Scripts
- Table 5. Recommended Scripts
- Table 7. Limited Use Scripts

(Note: Table 6. Aspirational Use Scripts has been withdrawn.)

The Excluded Scripts table is very long -- over 100 rows, and listing the majority of the encoded scripts -- and most newly encoded scripts are getting added here.

This seems unnecessary and is a burden to maintain. I propose that we replace the contents table of Excluded Scripts with something like this:

Property Notation	Description
all Script property values except:	those listed in Table 5, Recommended Scripts
	those listed in Table 7, Limited Use Scripts
	Zzzz (unassigned code points)
	Brai (only symbols)
	Hrkt (script code alias, no characters)

... and adjusting surrounding text as necessary.

Include a suitable link to PropertyValueAliases.txt and mention its `sc ;` lines.

10. Confusables

10.1 UTS #39: Better description of non-goals for scope of confusables [#478]

Recommended UTC actions

1. **Action Item** for Josh Hadley and Roozbeh Pournader, PAG: Add a section on "Scope and Non-Goals of the Confusables Data" to UTS #39. For Unicode 19.0. See L2/26-235 item 10.1.

PAG input

From Asmus Freytag, PAG:
We need to define the scope of UTS #39 by talking more explicitly about what's not included.

L2/25-228 (UTC #185 properties feedback & recommendations) section 6.1 UTS #39 refers to the stabilized UTR #36 for gatekeeper-confusable strings:

"This is just one of several types of confusables that we are not collecting: things to slip past gatekeepers; but there are also semantic confusables, underlining vs. macron below, confusable with uppercase, italics (Cyrillic т with Latin m), etc."

We should elaborate that into a short subsection.

Other specifications exist in the confusables space which explicitly include some of the ones mentioned in this excerpt from L2/25-228/6.1. Being explicit about the non-goals helps people in evaluating and reviewing the Unicode Confusables set in context.

From Roozbeh Pournader, PAG:
Here is proposed text to be added to UTS #39:

"**Scope and Non-Goals of the Confusables Data**

The confusables data provide a curated, non-exhaustive set of mappings for detecting strings that may be mistaken for one another on the basis of visual appearance. The data are collected primarily (but not exclusively) for characters whose Identifier_Status is Allowed, using a selection of common fonts, particularly fonts commonly used in user interfaces. The absence of a mapping does not imply that two strings are distinguishable in every font, style, size, rendering environment, or application context. Alternatively, a mapping may be absent simply because the maintainers of this Technical Standard have been unaware of the confusabilty.

The following are not, by themselves, criteria for adding a mapping. A mapping associated with one of these cases may nevertheless be included when it also serves the primary visual-confusability goal.

- *Filter-evasion obfuscation.* Text modified so that it remains recognizable to a person while evading an automated filter or other text-recognition system, rather than to make one identifier appear to be another.
- *Semantic equivalence.* Characters or strings related by meaning, pronunciation, or orthographic convention, without sufficient visual similarity.
- *Presentation-dependent similarity.* There is no attempt to cover every similarity that arises only under a particular typographic presentation or transformation. Examples include underlining that may make a string resemble one containing U+0331 COMBINING MACRON BELOW; an italic rendering in which U+0442 т CYRILLIC SMALL LETTER TE resembles Latin lowercase m; and similarities that arise only after case conversion or display in uppercase or small capital forms. The data contain mappings involving uppercase characters, but do not systematically enumerate relations created by changing case or style.

Applications whose threat models include the cases listed above should supplement the confusables data with context-specific data or tests."

11. Emoji

11.1 Standardize East_Asian_Width and Vertical_Orientation for new emoji [#347]

Recommended UTC actions

1. **Consensus**: Change the Vertical_Orientation of U+2795 ➕ HEAVY PLUS SIGN, U+2796 ➖ HEAVY MINUS SIGN, U+2797 ➗ HEAVY DIVISION SIGN, U+27B0 ➰ CURLY LOOP, and U+27BF ➿ DOUBLE CURLY LOOP from R to U. For Unicode Version 19.0. See L2/26-235 item 11.1.
2. **Consensus**: Change the Vertical_Orientation of unassigned characters in the range U+1FC00..U+1FFFD (reserved for future emoji blocks) from R to U and their East_Asian_Width from Neutral to Wide. For Unicode Version 19.0. See L2/26-235 item 11.1.
3. **Consensus**: Change the East_Asian_Width of the unassigned characters in the block Symbols and Pictographs Extended-A (U+1FA70..U+1FAFF) from Neutral to Wide. For Unicode Version 19.0. See L2/26-235 item 11.1.
4. **Action Item** for Ken Lunde, PAG: Add text to UAX #50 explaining that characters displaying as emoji, including when Emoji_Presentation=No characters that have an emoji variation sequence defined for them are followed by an emoji variation selector, should display as if they have Vertical_Orientation=U. For Unicode Version 19.0. See L2/26-235 item 11.1.
5. **Action Item** for Ken Whistler, PAG: In Unicode Standard Annex #44, Unicode Character Database, ensure the documentation of complex default values is consistent with the changes to East_Asian_Width and Vertical_Orientation defaults. For Unicode Version 19.0. See L2/26-235 item 11.1.
6. **Action Item** for Robin Leroy, PAG: Change the Vertical_Orientation of U+2795 ➕ HEAVY PLUS SIGN, U+2796 ➖ HEAVY MINUS SIGN, U+2797 ➗ HEAVY DIVISION SIGN, U+27B0 ➰ CURLY LOOP, and U+27BF ➿ DOUBLE CURLY LOOP from R to U. For Unicode Version 19.0. See L2/26-235 item 11.1.
7. **Action Item for** Robin Leroy, PAG: Change the Vertical_Orientation of unassigned characters in the range U+1FC00..U+1FFFD (reserved for future emoji blocks) from R to U and their East_Asian_Width from Neutral to Wide. For Unicode Version 19.0. See L2/26-235 item 11.1.
8. **Action Item for** Robin Leroy, PAG: Change the East_Asian_Width of the unassigned characters in the block Symbols and Pictographs Extended-A (U+1FA70..U+1FAFF) from Neutral to Wide. For Unicode Version 19.0. See L2/26-235 item 11.1.

PAG input

From Ned Holbrook, PAG:

Based on https://www.unicode.org/reports/tr51/#Display and recent assignments, I believe it is safe to standardize on ea=W and vo=U for emoji in the future.

12. Math

12.1 Approve UTR #25 [#691]

Recommended UTC actions

1. **Consensus**: Approve UTR #25 for publication as Revision 17, based on the current Proposed Update (Revision 16, draft 5). See L2/26-235 item 12.1.
2. **Action Item for** Robin Leroy, PAG: Publish UTR #25, Revision 17, and the associated data files. See L2/26-235 item 12.1.

PAG input

UTR #25 has undergone a major change in format (from PDF produced by Microsoft Word to HTML with MathML based on an XML source file), and has an expanded editorial team. The proposed update adequately documents the repertoire for Unicode Version 17.0. We believe it should be published in its current state to provide a baseline for documenting the additions in Unicode Version 18.0.

With the expanded editorial team and the new framework, it seems feasible for this document to become a synchronized document at some later point; in particular, we expect to be able to produce a revision synchronized with Unicode Version 18.0 before the release of Unicode Version 19.0.

13. Other

13.1 Draft UTS #61 editorial changes [#583]

Recommended UTC actions

1. **No Action**: This has been fixed in a recent draft revision.

Feedback (verbatim)

PRI-523

Date/Time: Sun July 12 08:38:15 PDT 2026
ReportID: ID20260712083815
Name: Philippe Verdy
Report Type: Public Review Issue

Minor editorial fix:

> 4 Conformance
>
> [...]
> Examples: The following are pure extensions:
> Accepting a final $ in a Union and interpreting it as representing the character U+FFFF.

Should be:

> Accepting a final $ in a Union and interpreting it as representing the non-character U+FFFF.

U+FFFF is a code point not mapped to any character.

13.2 CSSWG shares feedback on the handling of Hangul for text-autospace (PRI-510) [#589]

Recommended UTC actions

1. **No Action**: PAG agrees with the request, and the author of PD-UTR #59 has updated their working draft as requested, treating Hangul as "Other" (O).

Feedback (verbatim)

PRI-510

Date/Time: Wed Aug 12 13:43:16 PDT 2026
ReportID: ID20260812134316
Name: Elika J. Etemad

Report Type: Public Review Issue
Opt Subject: 510

CSSWG received the following feedback on the handling of Hangul for text-autospace, which I think also is of concern for UTR#59 (PRI 510), so forwarding it along:
https://github.com/w3c/csswg-drafts/issues/9979

The W3C issue includes this summary of the working group's position:

I am writing to provide feedback regarding the categorization of Hangul (Script=Hang) under UTR #59 (East Asian Spacing).

Currently, Hangul is classified under the "Wide" (W) property value, meaning layout engines will automatically insert a typographic gap between Hangul and "Narrow" (N) characters (such as Latin letters and numerals). While this behavior is highly beneficial for Chinese and Japanese, it is generally undesirable for Korean.

I would like to request that Hangul be treated as "Other" (O) by default, because:

1. Unlike Chinese and Japanese, modern Korean uses word spacing.
2. In Korean grammar, particles must be attached directly to the preceding noun with no intervening space. When a noun is written in Latin script and the particle in Hangul (e.g., "W3C를" or "Unicode는"), injecting an automatic typographic gap between the Latin character and the Hangul particle violates standard Korean orthography.
3. The "Requirements for Hangul Text Layout and Typography" (KLReq)[1] states that "where words written in the Latin script are followed by grammatical particles in the Korean language, there is no space between the Latin text and the following Korean characters."

[1] https://www.w3.org/International/klreq/#h_segmentation

Background information / discussion

PAG, CJK, and the author of PD-UTR #59 agree with the feedback, specifically with the GitHub issue that states that there should be no space between Hangul syllables and Han ideographs.

13.3 Joiners in UTS #58: A report on UTC 183-A76 [#657]

Recommended UTC actions

1. **Action Item for** Mark Davis and Markus Scherer, PAG: Add a paragraph about escaping joining controls in URLs per L2/26-235 item 13.3 to Section 4 of UTS #58. For Unicode Version 19.0. See L2/26-235 item 13.3.
2. **Action Item for** Mark Davis and Markus Scherer, PAG: Add test cases for joining controls in URLs to LinkFormattingTest.txt and LinkDetectionTest.txt per L2/26-235 item 13.3. For Unicode Version 19.0. See L2/26-235 item 13.3.

PAG input

From Roozbeh Pournader, PAG:
I was assigned an action item, UTC-183-A76, to "Review linkification feedback PRI-509 ID20250326055934 from Ebrahim Byagowi to see if changes are needed. See L2/25-087 item 6.1." Here's a report, with some suggestion for improvements to UTS #58:

The feedback reported that Chrome has displayed percent-encoded ZWNJ instead of letting it shape the surrounding Persian text; and Safari has sometimes double-encoded it after address-bar editing. The same class of error affects emoji sequences containing ZWJ. This is in scope for UTS #58 where it defines the minimally escaped Path, Query, and Fragment strings. Cursor behavior, address-bar UI, etc are out of scope; Host display remains governed by UTS #46.

No normative changes appear necessary. The specification appears to be handling ZWJ and ZWNJ correctly. The following editorial changes would both help explain how UTS #58 handles joiners, and provide clarification and guidance for implementations.

1. In Section 4, "URL Minimal Escaping": insert after the bullet ending "tooltip or statusbar" and before Section 4.1:

The minimal escaping algorithm does not percent-escape a code point merely because it has no glyph, has zero advance width, or has Default_Ignorable_Code_Point=Yes. In particular, U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER have Link_Term=Include and are literal code points in the minimally escaped form of a Path, Query, or Fragment. They can change the rendering of surrounding characters, as in Persian words and emoji ZWJ sequences.

2. Add a “Joining controls” block to LinkFormattingTest.txt with these source/result pairs (the first result contains ZWNJ; the second contains ZWJ):

https://example.com/%D9%85%DB%8C%E2%80%8C%D8%B1%D9%88%D9%85
https://example.com/می‌روم

https://example.com/%F0%9F%91%A9%F0%9F%8F%BD%E2%80%8D%F0%9F%92%BB
https://example.com/ 👩🏽‍💻

3. In LinkDetectionTest.txt, under “Various Path/Query/Fragment cases”, add:

See ¦https://example.com/می‌روم¦ on…
See ¦https://example.com/ 👩🏽‍💻 ¦ on…

13.4 Approve UTS #61 [#683]

Recommended UTC actions

1. **Consensus**: Approve Draft UTS #61 for publication as Unicode Technical Standard #61, Unicode Set Notation. See L2/26-235 item 13.4.
2. **Action Item for** Michelle Perham, UTC: Close PRI #523. See L2/26-235 item 13.4.
3. **Action Item for** Robin Leroy, PAG: Shepherd the publication of UTS #61.

PAG input

After extensive back-and-forth between PAG and ICU-TC, DUTS #61 has been implemented in ICU 79, and is referenced in the ICU API documentation and user guide (see https://unicode-org.github.io/icu/userguide/strings/unicodeset.html). CLDR-TC would like to be able to refer to it in UTS #35. We believe this document is now ready for publication.

14 Feedback not addressed in this document

The following public feedback was received, but is not covered in this report. PAG plans to address these for UTC #190.

1. L2/26-137 Left is Right, Right is Left: Joining_Type for Left-to-right Scripts [#568]
2. ID20260813083821 – Ben Scarborough – Bidi_Class of U+218A and U+218B, TURNED DIGIT TWO & THREE (PRI-556) [#588]
3. ID20261010164428 — Oliver Kuederle — Wrong Result in Default Work_Break Test Case (PRI-561) [#692]
4. IETF idna-update list discussion – Patrik Fältström, John Klensin, et al – IETF Cannot reliably tell status and locations of IDNA2008 derived file [#590]
5. Discussion with the IETF – Asmus Freytag – IETF considering asking for stability policy for IDNA2008Derived data [#591]
6. ID20260917211722 – Yasuo Kida – Feedback on UTR #59 [#678]