L2/23-160

UTC #176 properties feedback & recommendations

Markus Scherer / Unicode properties & algorithms group, 2023-jul-17

Participants

The following people have contributed to this document:

Markus Scherer (chair), Josh Hadley (vice chair), Asmus Freytag, Elango Cheran, Ken Whistler, Manish Goregaokar, Mark Davis, Ned Holbrook, Peter Constable, Rick McGowan, Robin Leroy

1. Core spec

1.1 Is « The [three] Unicode encoding forms » well-defined?

From Robin Leroy, liaison officer to ISO/IEC JTC 1/SC 22.

Recommended UTC actions

1. Action Item for Asmus Freytag, EDC: Change definition D79 to look like a definition, and add bullet points noting the use of the term to refer to the three UTF and suggesting the term “standard Unicode encoding forms” for clarity. Update mentions of “the [three] Unicode encoding forms” throughout the core specification as appropriate, for Unicode Version 16. See L2/23-160 item 1.1.
2. Action Item for Ken Whistler, EDC: In UTR #17, change mentions of “the [three] Unicode encoding forms” to use the phrase “standard Unicode encoding forms” as appropriate. See L2/23-160 item 1.1.
3. Action Item for Asmus Freytag, EDC: In the FAQ and Glossary, change mentions of “the [three] Unicode encoding forms” to use the phrase “standard Unicode encoding forms” as appropriate. See L2/23-160 item 1.1.

Feedback & discussion

Discussion on the ISO/IEC JTC 1/SC 22/WG 21/SG 16 mailing list:
While working on addressing a French NB comment on C++23, Corentin Jabot was looking for a term for « the UTF-8, UTF-16, and UTF-32 encoding forms ».
It was been pointed out by Jens Maurer that this definition in TUS, Section 3.9, is open-ended:

> D79 A *Unicode encoding form* assigns each Unicode scalar value to a unique code unit sequence.

That is, it suggests that any operation that has this property is a *Unicode encoding form*, whether or not it is specified in the Standard.

This is in contradiction to the use of the definite article and the number three in other places in the Standard (the Standard has 11 occurrences of *the Unicode encoding forms*, and 8 occurrences of *the three Unicode encoding forms*).

D79 does not look to me like a definition at all, rather like a statement about the encoding forms; indeed it does not have the normal structure of *defined term*: definition.

A number of alternatives were considered by the PAG.

Robin Leroy initially suggested changing D79 to mean only the three UTFs.

Asmus Freytag replied:

> We really must stop fiddling with the text of basic and long-standing definitions, foundational definitions, in the standard. Those should be considered off limits unless what they define is incorrect on their own terms. If someone needs a snappy term, if push comes to shove, let's invent a new one. How about something as simple as "A Unicode standard encoding form" (is one of ....).

Ken Whistler and Mark Davis objected to adding another definition, as this would be disruptive to existing specifications that are relying on the term *Unicode encoding form* to mean the three UTFs, like the Unicode Standard currently does.

The PAG instead decided to retain and endorse the practice of saying « the Unicode encoding forms » to mean the three Unicode encoding forms defined in the Unicode Standard, while also keeping the definition open-ended; in addition, a term is suggested (but not as a separate definition) in cases where there is a need to unambiguously refer to the three UTFs.

The proposed text for D79 is as follows:

> D79 *Unicode encoding form*: A mapping from each Unicode scalar value to a unique code unit sequence.
>
> - This standard defines three Unicode encoding forms; see D90, D91, and D92.
> - Unless otherwise stated, the term “Unicode encoding form” refers to one of those three forms. For clarity, they can be referred to as “standard Unicode encoding forms”.

The usages of “the [three] Unicode encoding forms” throughout the standards and the website should be updated as appropriate to take advantage of the new term where appropriate for clarity.

For example, the 2nd paragraph of TUS 2.2.2

> All Unicode encoding forms are self-synchronizing and non-overlapping.

which is not true of arbitrary Unicode encoding forms, could be rewritten as

> UTF-8, UTF-16, and UTF-32 are self-synchronizing and non-overlapping,

or as

> the standard Unicode encoding forms are self-synchronizing and non-overlapping,

On the other hand, the following statement in Restricted Interchange under TUS 2.4.1 is really true in general, and does not need to touched.

Surrogate code points cannot be conformantly interchanged using Unicode encoding forms. They do not correspond to Unicode scalar values and thus do not have well-formed representations in any Unicode encoding form.

1.2 Where is locale-independent simple case folding of strings defined?

From Robin Leroy, liaison officer to SC 22.

Recommended UTC actions

1. Action Item for Markus Scherer, Robin Leroy, PAG: Propose wording defining the phrase "*simple case folding"* in Section 3.13 of The Unicode Standard, and changing the misleading and undefined phrase "*locale-independent"* case folding in Table 4-3 and Section 5.18 to case folding (which is locale-independent). See L2/23-160 item 1.2. For Unicode Version 16.0.

Feedback & discussion

The liaison officer to SC 22 asked the following questions to the PAG:

> While trying to get rid of the « documents referenced » trick in Ada, I found that the Ada Reference Manual mentions
>
> > locale-independent simple case folding, as defined by documents referenced […]
>
> with the Annotated Ada Reference Manual note:
>
> > The "documents referenced" means Unicode, Chapter 4 (specifically, section 4.2 — Case).
>
> However, the documents referenced in the note in Section 1 of ISO/IEC 10646:2003 (also known as Unicode 4.0) do not really define that; the phrase « locale-independent case folding » occurs in Table 4-1,

File Name	Description
CaseFolding.txt	Contains data for performing locale-independent case folding, as described in "Caseless Matching," in Section 5.18, Case Mappings.

> and in 5.18,
>
> > The CaseFolding.txt file in the Unicode Character Database is used to perform locale-independent case folding.

In particular the phrase *locale-independent* does not appear in Section 3.13, where toCasefold is actually defined for strings.

This situation persists to this day, in the documents referenced in Clause 2 of ISO/IEC 10646:2020, and in Unicode 15.0.

Chapter 3 does have the sentence

> A common variant, for example, is to make use of simple case conversion, rather than full case conversion.

Questions:

1. Can locale-independent simple case folding of strings be said to be defined in Section 13.3?
   - alternatively, should one say that it is defined in the UCD, in https://www.unicode.org/Public/UCD/latest/ucd/CaseFolding.txt (see Usage)?
2. Should we add a bullet point under R4 saying that toCasefold is also known as locale-independent case folding?
3. Should we add a bullet point under R4 saying that the tailoring consisting of mapping C to the value of its Simple_Case_Folding property is also known as locale-independent simple case folding or simple case folding?

The PAG discussed and came to the following conclusions:

1. Core spec 3.13 is the place that should be referenced.
2. Core spec 3.13 should talk about "simple case folding" even if we don't define a toSimpleCasefold(X). Between R4 and the "A modified form..." intro for R5.

The PAG also noted that the phrase locale-independent case folding is misleading (contrary to case *mapping*, there is no locale-dependent case folding, only turkic case folding which is its own thing to be handled with great caution).

2. UCD

2.1 Changes for new characters in 16.0

Robin Leroy, Ken Whistler, et al., PAG.

Recommended UTC actions

1. Consensus: The UTC approves the ArabicShaping.txt entries in L2/23-160 item 2.1 (a), replacing those recommended by L2/22-116 and previously approved in UTC-172-C3.
2. Consensus: The UTC approves the code point change for TULU-TIGALARI LETTER DHA to U+113A4, from the conflicting U+113A5 approved by UTC-170-C9. TULU-TIGALARI LETTER NA remains at U+113A5.
3. Consensus: The UTC approves the change of canonical combining class for U+0897 ARABIC PEPET to ccc=230, from ccc=0 as approved by UTC-172-C3.

Feedback

The PAG spotted some issues while preparing the UCD changes for the Unicode Version 16.0 pipeline.

a) Joining_Type consistent with Joining_Group

```
Unset
10EC2; DAL WITH VERTICAL 2 DOTS BELOW; R; DAL
10EC3; TAH WITH VERTICAL 2 DOTS BELOW; D; TAH
10EC4; KAF WITH VERTICAL 2 DOTS BELOW; D; KAF
```

See comments on unicode-org/unicodetools@aef2638#comments.

b) Change the code point for TULU-TIGALARI LETTER DHA from U+113A5 to U+113A4.
Spotted by Ken while making a UnicodeData for 16.0 for TUS purposes.

c) Change U+0897 ARABIC PEPET from ccc=0 to ccc=230 because it is a combining mark above.
See discussion in unicode-org/unicodetools#435 (comment).

2.2 Katakana middle dots in XID_Continue

From Robin Leroy, liaison officer to ISO/IEC JTC 1/SC 22.

Recommended UTC actions

1. Consensus: Add the characters U+30FB KATAKANA MIDDLE DOT and U+FF65 HALFWIDTH KATAKANA MIDDLE DOT to Other_ID_Continue and its derivative properties, as described in L2/23-160 item 2.2, for Unicode Version 15.1.
2. Action Item for Robin Leroy, PAG: Add the characters U+30FB KATAKANA MIDDLE DOT and U+FF65 HALFWIDTH KATAKANA MIDDLE DOT to Other_ID_Continue and its derivative properties, as described in L2/23-160 item 2.2, for Unicode Version 15.1.
3. Action Item for Robin Leroy, PAG: Remove the table in Section 3.3.2 of DUTS #55, and change the last paragraph to say that there are no incompatibilities when using a definition based on earlier versions of the UCD. For Unicode Version 15.1.
4. Action Item for Robin Leroy, PAG: In Table 3a of UAX #31, remove the line for U+30FB KATAKANA MIDDLE DOT. For Unicode Version 15.1.

Summary

In Unicode Version 4.1, we accidentally removed U+30FB KATAKANA MIDDLE DOT and U+FF65 HALFWIDTH KATAKANA MIDDLE DOT from default identifiers (Note: we did the same to two Khmer inherent vowels (U+17B4 and U+17B5) in Unicode Version 4.0, but we added them back later on).

This was noted by the UTC, which recommended in UTC-174-C3 that the statement in the stability policy be corrected; the UTC believed that this was old enough that it affected no-one, and that no other action was needed.

However, as noted in L2/23-132, this incompatibility affects COBOL 2023, which has to work around it as part of its profile.

We have also found in the work on UTS #55 that several programming language standards still use the Unicode 3 property-based definition with an implementation-defined version of the UCD; in at least one case (Ada 2012 and later; Ada 2005 required Unicode 4.0 exactly) that implementation permission allows Unicode 4.0 (which is before the incompatibility in 4.1). The incompatibility thus complicates the compatibility considerations involved when migrating such standards and implementations to the recommended definitions, see the table at the bottom of https://www.unicode.org/reports/tr55/proposed.html#Evolution-Unicode-3.

We could resolve those issues by adding those two middle dots to XID_Continue, as we should have done in Unicode Version 4.1.
The SCWG discussed this issue on 2023-06-01 and is in favor of that.

Relevant precedents:

1. We have in the past fixed stability issues even across gaps, for instance Other_ID_Continue was only introduced in Unicode 4.0, but includes characters removed from default identifiers between Unicode 2 and Unicode 3, see UTC-92-C29.
2. XID_Continue contains several middle-dot-looking things, including the actual MIDDLE DOT.

2.3 specify more non-default case mapping behavior

Question from ICU4X team

Recommended UTC actions

1. Action Item for Markus Scherer, Mark Davis, PAG: In SpecialCasing.txt clarify that additional language-specific and orthography-specific case mapping behavior is or will be documented in CLDR, along the lines of suggested text in L2/23-160 item 2.3; for Unicode 16.0.
2. Action Item for Markus Scherer, Asmus Freytag, PAG: In the core spec, near Table 3-17 “Context Specification for Casing” and in chapter 5.18 “Case Mappings”, document that additional language-specific and orthography-specific case mapping behavior is or will be documented in CLDR; see also L2/23-160 item 2.3; for Unicode 16.0.

Feedback

The Unicode Standard includes non-default case mappings for certain characters and languages, with data in SpecialCasing.txt and spec text in the Standard. The UTC has been looking to CLDR for further language-specific behavior.

Some case mapping behaviors are currently implemented in ICU without specification in the Unicode Standard or in CLDR. Now that ICU4X is rounding out its case mapping implementation, it makes sense to specify the behavior.

Would the UTC consider adding further case mapping spec text, and possibly additional data, so that the spec & data is in one place for all languages, or should we add spec & data in CLDR?

Example:

Since ICU 58 (2016) we have had an implementation of Greek uppercasing to match user expectations:

- ICU-5456 “Uppercase formatting option results in accented capital letters - Invalid for Greek”
    - (duplicate: ICU-7423 “Modern Greek uppercasing behavior is to strip accents from Greek characters”)
- design doc from that plus search for documentation and discussions with linguists: https://icu.unicode.org/design/case/greek-upper

(FYI: We might need more research and possibly adjustment to the behavior: ICU-12845 “Greek Casing: breathers”)

Other examples: Dutch IJ titlecasing, Eastern Armenian uppercasing of ligature ech-yiwn

Background information / discussion

In PAG discussion, we concluded that additional language-specific and orthography-specific case mapping behavior should be documented in CLDR rather than in UTC specifications (UCD & core spec). In CLDR, this is now tracked via ticket CLDR-16849.

UCD SpecialCasing.txt already says:

```
Unset
# Note that the preferred mechanism for defining tailored casing operations is
# the Unicode Common Locale Data Repository (CLDR). For more information, see the
# discussion of case mappings and case algorithms in the Unicode Standard.
```

It also has these two contradictory statements:

```
Unset
# This file is a supplement to the UnicodeData.txt file. It does not define any
# properties, but rather provides additional information ...
```

vs.

```
Unset
# ... The data in this file, combined with
# the simple case mappings in UnicodeData.txt, defines the full case mappings
# Lowercase_Mapping (lc), Titlecase_Mapping (tc), and Uppercase_Mapping (uc).
```

The comments in the file should be resolved to something like this:

```
Unset
# This file is a supplement to the UnicodeData.txt file. The data in this file, combined with
# the simple case mappings in UnicodeData.txt, defines the full case mappings
# Lowercase_Mapping (lc), Titlecase_Mapping (tc), and Uppercase_Mapping (uc).
# For compatibility, the UnicodeData.txt file only contains simple case mappings
# for characters where they are one-to-one (and independent of context and language).
#
# For historical reasons, this file also provides additional information about the casing
# of Unicode characters for selected situations when casing is dependent on context or
locale.
#
# Note that the preferred mechanism for defining tailored casing operations is
# the Unicode Common Locale Data Repository (CLDR). For more information, see the
# discussion of case mappings and case algorithms in the Unicode Standard.
#
# All code points not listed in this file that do not have a simple case mappings
# in UnicodeData.txt map to themselves.
```

3. New Scripts & Characters

PAG members reviewed the following proposals and draft, provided feedback to SAH, and the feedback has been mostly addressed.

- L2/23-103R Proposal for ARABIC BIBLICAL END OF VERSE
- L2/23-122 Proposal to Encode Kashmiri Sharada Characters in Unicode
- L2/23-121 Proposal to encode Arabic Letter Thin Noon
- UTR #53 UNICODE ARABIC MARK RENDERING revision
    - We have reviewed the proposed update (Unicode-internal PDF). One of us has reported issues which should be fixed in the document before it is approved for publication.

Recommended UTC actions

1. Note: The ARABIC BIBLICAL END OF VERSE was originally proposed with gc=So. The consensus 175-C13 was based on the original proposal. This character should instead have gc=Po, and L2/23-103R from 2023-07-10 reflects that now.

4. Text Segmentation

4.1 WB4 rule for Prepend characters: use PCM property

From discussion during UTC-175 of PAG report (L2/23-079) item 5.2 UAX #29: WB4 should be expanded and clarified

Recommended UTC actions

1. Action Item for Josh Hadley, PAG: Change lb=Numeric to include the list of characters corresponding to [[:PCM:]-\u070F] and then remove the list of characters from the WB=Numeric definition. For Unicode 15.1.

Feedback

We recorded
[175-C] Consensus: Change the Word_Break property of U+0600–U+0605, U+06DD, U+0890, U+0891, U+08E2, U+110BD, and U+110CD from Format to Numeric, and the Word_Break property of U+070F from Format to ALetter, for Unicode Version 15.1. See L2/23-079 item 5.2.

The set of characters that move to Numeric can be expressed as [[:PCM:]-\u070F].
Since future characters that get added to PCM are likely to have similar functions, if the WB rule used this expression, they would automatically behave as expected in word segmentation. This is a useful default.

Consider changing UAX29 to say that characters in [[:PCM:]-\u070F] are assigned WB=Numeric.

Background information / discussion

WB=Numeric is derived from lb=Numeric with additions and exceptions.
The characters that are becoming WB=Numeric should also become lb=Numeric, and when they are, then they need not be listed in the definition of WB=Numeric any more.

Line_Break values are not derived, but the maintainers will add a programmatic test that lb=Numeric includes characters with the PCM property, with exceptions as appropriate (currently U+070F).

4.2 Text segmentation properties of Kirat Rai vowel signs

Recommended UTC actions

1. Consensus: Generalize conjoining behavior to include Kirat Rai vowel signs, and set the Grapheme_Cluster_Break property of the Kirat Rai vowel signs {E, AI, AA, O, AU} to "V". For Unicode 16.0. See L2/23-160 item 4.2.
2. Action Item for Robin Leroy, PAG: Set the Grapheme_Cluster_Break property of the Kirat Rai vowel signs {E, AI, AA, O, AU} to "V". For Unicode 16.0. See L2/23-160 item 4.2.
3. Action Item for Asmus Freytag, PAG: Generalize TUS 3.12 Conjoning Jamo Behavior, with Kirat Rai vowel signs as an example. For Unicode 16.0. See L2/23-160 item 4.2.
4. Action Item for Josh Hadley, Robin Leroy, PAG: In UAX29, generalize conjoining behavior to include Kirat Rai vowel signs, in descriptive text and in lists of code points for GCB values. For Unicode 16.0. See L2/23-160 item 4.2.

Feedback (verbatim)

Date/Time: Tue May 02 07:23:16 CDT 2023
ReportID: ID20230502072316
Name: Charlotte Buff
Report Type: Other Document Submission
Opt Subject: Text segmentation properties of Kirat Rai vowel signs

The vowel signs of the Kirat Rai script, which has been accepted for a future version of the Unicode Standard based on proposal document L2/22-043r, are slated to be implemented as spacing, stand-alone characters (gc=Lo) rather than as combining or spacing marks. While not explicitly stated, this would likely result in them being assigned the Grapheme_Cluster_Break property value Other (GCB=XX). Three of these vowel signs – AI, O, and AU – are visually sequences of other vowel signs and have therefore been given canonical decomposition mappings:

U+16D68 ≡ <U+16D67, U+16D67> AI ≡ <E, E>
U+16D69 ≡ <U+16D63, U+16D67> O ≡ <AA, E>
U+16D6A ≡ <U+16D69, U+16D67> AU ≡ <O, E>

These properties, however, do not maintain canonical equivalence. The vowel signs in question would be one grapheme cluster each in NFC, but two grapheme clusters each in NFD. This is forbidden by UAX #29, which states in section 2, “Conformance”:

»A boundary exists in text not normalized in form NFD if and only if it would occur at the corresponding position in NFD text.«

There are several possible approaches for resolving this issue:

1) Reclassify Kirat Rai vowel signs as spacing, combining marks
A minimal solution that preserves canonical equivalence for both legacy and extended grapheme clusters would involve U+16D67 KIRAT RAI VOWEL SIGN E and U+16D68 KIRAT RAI VOWEL SIGN AI being changed to Grapheme_Cluster_Break=Extend (GCB=EX). Though not strictly necessary, it would then also make sense to change their General_Category value to Spacing_Mark (gc=Mc).
This approach may not be desirable because it would prevent vowel signs E and AI from being used in isolation; they would always forcibly “glue” themselves to the preceding character such as a space or a punctuation mark and potentially cause problems for the text renderer. The stand-alone nature of the Kirat Rai vowel signs was quite a deliberate choice because of the similarities to the New Tai Lue script.

2) Invent new GCB rules for these vowel signs
The text segmentation algorithm would need to be amended to make Kirat Rai vowel signs similar in nature to Hangul Jamo – forming grapheme clusters with each other in certain configurations, but not with unrelated characters. For minimal impact, the new rule should be limited to the interaction between vowel signs E, AA, and O followed directly by vowel sign E, which covers all three decomposition mappings. It could look something like this:

```
[\u{16d63}\u{16d67}\u{16d69}] × \u{16d67}
```
Note that U+16D67 occurs on both sides of the rule because it is both the leading and the trailing codepoint in the decomposition mapping of U+16D68.

This approach is probably a cleaner solution because it gets rid of the problem without changing anything about the general nature of the script, but it also introduces a unique edge case into an otherwise quite straightforward algorithm for the sake of just a handful of characters.

3) Change decompositions from canonical to compatibility
There is no requirement for compatibility decompositions to preserve the text segmentation boundaries of their source strings. In practice, users of the script would always encounter the vowel signs in precomposed form because NFKC and NFKD are generally not used on the front end, while search and collation algorithms would still be able to recognise the weak equivalence.

However, it is questionable whether using mere compatibility equivalence for sequences that are truly identical in every sense is appropriate, especially in the context of security.

4) Do not encode compound vowel signs as separate characters
The characters U+16D68..U+16D6A would be removed from the Kirat Rai repertoire altogether and the only way to represent vowel signs AI, O, and AU would be through the use of sequences. Perhaps named character sequences could be defined as well if deemed useful.

This approach would circumvent the entire issue without side effects, but is also clearly the least desirable for actual users of the script who consider these vowel signs to be linguistic units regardless of their glyphic appearance. I do not think this would be an acceptable solution in practice.

5) Encode the vowel signs as atomic characters without decomposition mappings
This approach is the worst one in my view as it would necessitate the creation of dreaded Do Not Use tables for the Kirat Rai script, which goes against everyone’s interests. I strongly recommend against this solution.

Background information / discussion

We thank Charlotte Buff for finding this problem.

The encoding as such seems appropriate, and in any encoding where a logical unit can be represented by a sequence of letters (gc=L), we need to use an appropriate mechanism to avoid grapheme cluster breaks within such a unit.
We propose to extend the conjoining behavior of Hangul Jamo to the Kirat Rai vowel signs (UAX29 rule GB7).

We had considered using different GCB values for different vowel signs, for example E→V, AI→V, AA→LV, O→LV, AU→LV, in order to minimize accidental connections with adjacent Hangul syllables. However, in PAG and SAH discussions we concluded that this adds some amount of complication which we judged to not be necessary.

Suggested additional paragraph for TUS section 3.12 Conjoning Jamo Behavior:

There are several instances of characters that bind tightly into grapheme clusters, but that unlike combining characters don't depend on a base character. These characters are said to exhibit conjoining behavior. Historically, the first example of these were the Hangul Jamo, which means that the wording of the definitions and naming of character properties reflect the nature of Hangul syllables. Nevertheless, the concept of conjoining behavior can readily be extended to other characters. For example, the Grapheme_Cluster_Break property value V which originally matched Hangul_Syllable_Type V can be extended to implement conjoining behavior for characters such as the Kirat Rai vowels.

Suggested note to be added to UAX29:

Similar to Jamo clustering into Hangul Syllables, other characters bind tightly into grapheme clusters, that, unlike combining characters, don't depend on a base character. These characters are said to exhibit conjoining behavior. For the purpose of Grapheme_Cluster_Break, the property value V has been extended beyond characters of HST=V to cover them.

4.3 Should the set of scripts affected by GB9c be provided in the UCD?

Recommended UTC actions

1. Consensus: Create a new informative derived enumerated property Indic_Conjunct_Break (InCB), whose values Linker, Consonant, and Extend correspond respectively to the macros ConjunctLinker, LinkingConsonant, and ExtCccZwj from draft 2 of the the Proposed Update to Unicode Standard Annex #29, Unicode Text Segmentation, and with a default value of None (None), for Unicode Version 15.1.
2. Action Item for Robin Leroy, PAG: Add the new enumerated property Indic_Conjunct_Break (InCB), with values Linker, Consonant, and Extend to PropertyAliases.txt, PropertyValueAliases.txt, and DerivedCoreProperties.txt, for Unicode Version 15.1. See document L2/23-160 item 4.3.
3. Action Item for Josh Hadley, PAG: Update rule GB9c Unicode Standard Annex #29, Unicode Text Segmentation, to use the Indic_Conjunct_Break property instead of macros, for Unicode Version 15.1. See document L2/23-160 item 4.3.
4. Action Item for Ken Whistler, PAG: Add the informative property Indic_Conjunct_Break to the Property Table of Unicode Standard Annex #44, Unicode Character Database, for Unicode Version 15.1. This property should be listed as to be available in property APIs. See document L2/23-160 item 4.3.
5. Action Item for Mark Davis, PAG: Add Indic_Conjunct_Break to the list of Full Properties in Section 2.7 of UTS #18, Unicode Regular Expressions, for a future revision of that UTS. See document L2/23-160 item 4.3.

Summary & discussion

The grapheme cluster boundary rules are not stable, but compared to the fickle line breaking rules, they do not change very often.
They are also implemented in many more places than the line breaking rules (for instance, C++23 uses them as part of the std::format library; the Swift Character represents an EGC, etc.).

Thus, we prefer changing the properties rather than the rules; see for instance the discussion in #119. We can reasonably expect the list of scripts [\p{sc=Gujr}\p{sc=Telu}\p{sc=Mlym}\p{sc=Orya}\p{sc=Beng}\p{sc=Deva}] added by UTC-175-C26 to grow; indeed this was brought up by Roozbeh in plenary; but as specified, that would entail a trickle of minor changes to the EGC boundary rules, which would be a burden to implementers.

It would be more convenient to have an informative derived binary property corresponding to [\p{sc=Gujr}\p{sc=Telu}\p{sc=Mlym}\p{sc=Orya}\p{sc=Beng}\p{sc=Deva}] which can be updated to include more scripts as needed.

Bikeshedding the property name: Grapheme_Conjunct_Script?

4.4 Edge cases of « quotation marks »

Recommended UTC actions

1. Consensus: Change rules LB15a and LB15b in UAX #14 so that they treat class ZW like BK, as described in L2/23-160 item 4.4, for Unicode Version 15.1.
2. Action Item for Robin Leroy, PAG: Make the changes to rules LB15a and LB15b in the Proposed Update for UAX #14 described in L2/23-160 item 4.4. For Unicode Version 15.1.
3. Action Item for Robin Leroy, PAG: Change the tooling that generates LineBreakTest.txt and LineBreakTest.html to adjust for the changes to the Proposed Update for UAX #14 described in L2/23-160 item 4.4. For Unicode Version 15.1.

Summary & discussion

From Robin Leroy, PAG:
I have been working on implementing the new LB15a and LB15b approved by UTC-175-C23 in ICU.

ICU uses a different way of expressing the breaking rules, based primarily on regexes matching unbroken strings rather than regexes providing context for sequentially-applied rules.

Rule LB15a is as follows in UAX14:
(sot | BK | CR | LF | NL | OP | QU | GL | SP) [\p{Pi}&QU] SP* ×

The pile of alternatives at the beginning is a heuristic to exclude cases where the quotation marks are being used »wie auf Deutsch«, where a Pi is actually final.

sot, BK, CR, LF, NL mean that the quotation mark is mandatorily at the beginning of the line, so it certainly cannot be final.
Spaces inside quotation marks are not used in »these styles«, so SP likewise means it is not final; the argument is the same for GL.

OP is not final, and the unresolved QU is there in case it is opening (to deal with "« this » kind of construct").

I tried this* in the ICU framework:
($OP | $QU | $GL) $CM* [\p{Pi} & $QU] $CM* $SP* .;

This means « don't break within (OP | QU | GL) CM* [\p{Pi} & QU] CM* SP* ALL », which covers the OP, QU, and GL alternatives (the CM* are for LB9).

^[\p{Pi} & $QU] $CM* $SP* .;

In the ICU syntax, ^ means « somewhere where we have a break », so this means « don't break within [\p{Pi} & QU] CM* SP* ALL » if it follows a break (either mandatory or allowed, ICU does not distinguish these).

I wrote it that way because the ICU rules cannot express context spanning a break for « keep-together » rules; it made sense because it covers all of sot (ICU counts the start of text as a break, contrary to UAX14), BK, CR, LF, NL, and SP.

At a higher level, it makes sense because if there is a break opportunity just before a quotation mark, that quotation mark is unlikely to be meant to be final.

The monkey tests (which compare the behaviour of the UAX14 rules with those of the ICU rules on random strings) promptly found a discrepancy:

ZW [\p{Pi} & QU] SP AL (for instance, <ZERO WIDTH SPACE>«<SPACE>A) breaks as
ZW ÷ [\p{Pi} & QU] × SP ÷ AL in the UAX 14 rules (LB 15a does not apply), but as
ZW ÷ [\p{Pi} & QU] × SP × AL in the ICU rules (^[\p{Pi} & $QU] $CM* $SP* .; applies).

Because this is an edge case (ZERO WIDTH SPACE is a manual override to the line breaking algorithm), and because it makes sense according to the above reasoning that

if there is a break opportunity just before a quotation mark, that quotation mark is unlikely to be meant to be final

I think we should amend LB15a to add ZW to the alternatives, thus
(sot | BK | CR | LF | NL | OP | QU | GL | SP | ZW) [\p{Pi}&QU] SP* ×.

For symmetry, we should correspondingly amend LB15b:
× [\p{Pf}&QU] (SP | GL | WJ | CL | QU | CP | EX | IS | SY | BK | CR | LF | NL | ZW| eot).

* Note: because of interactions with LB10, LB14, and limitations of rule chaining, the rules are really as follows ICU, but we can ignore the full complexity for that discussion.

```
# LB 15a
($OP $CM* $SP+ | [$OP $QU $GL] $CM*) ([\p{Pi} & $QU] $CM* $SP*)+ .;
($OP $CM* $SP+ | [$OP $QU $GL] $CM*) ([\p{Pi} & $QU] $CM* $SP*)+ $SP $CM+ $AL_FOLLOW?;
^([\p{Pi} & $QU] $CM* $SP*)+ .;
^([\p{Pi} & $QU] $CM* $SP*)+ $SP $CM+ $AL_FOLLOW?;

# LB 15b
$LB8NonBreaks [\p{Pf} & $QU] $CM* [$SP $GL $WJ $CL $QU $CP $EX $IS $SY $BK $CR $LF $NL $ZW
{eof}];
$CAN_CM $CM*  [\p{Pf} & $QU] $CM* [$SP $GL $WJ $CL $QU $CP $EX $IS $SY $BK $CR $LF $NL $ZW
{eof}];
^$CM+  [\p{Pf} & $QU] $CM* [$SP $GL $WJ $CL $QU $CP $EX $IS $SY $BK $CR $LF $NL $ZW {eof}];

# Messy interaction: manually chain between LB 15b and LB 15a on Pf Pi.
$LB8NonBreaks [\p{Pf} & $QU] $CM* ([\p{Pi} & $QU] $CM* $SP*)+ .;
$LB8NonBreaks [\p{Pf} & $QU] $CM* ([\p{Pi} & $QU] $CM* $SP*)+ $SP $CM+ $AL_FOLLOW?;
$CAN_CM $CM*  [\p{Pf} & $QU] $CM* ([\p{Pi} & $QU] $CM* $SP*)+ .;
$CAN_CM $CM*  [\p{Pf} & $QU] $CM* ([\p{Pi} & $QU] $CM* $SP*)+ $SP $CM+ $AL_FOLLOW?;
^$CM+  [\p{Pf} & $QU] $CM* ([\p{Pi} & $QU] $CM* $SP*)+ .;
^$CM+  [\p{Pf} & $QU] $CM* ([\p{Pi} & $QU] $CM* $SP*)+ $SP $CM+ $AL_FOLLOW?;
```

4.5 UAX29 Table 1c "ri-sequence" vs "RI-Sequence"

Recommended UTC actions

1. Action Item for Josh Hadley, PAG: Update UAX29 Table 1c for consistency of use of "ri-sequence"; for Unicode 15.1. See ReportID: ID20230616211348

Feedback (verbatim)

Date/Time: Fri Jun 16 21:13:48 CDT 2023
ReportID: ID20230616211348
Name: Eiso Chan
Report Type: Public Review Issue
Opt Subject: 469

In Table 1c, “ri-sequence” and “RI-Sequence” are both used.

Maybe all “ri-sequence” in Table 1c should be “RI-Sequence”.

Background information / discussion

PAG agreed to make these minor editorial changes.

4.6 UAX29 Consonant Clusters ExtCccZwj class

Recommended UTC actions

1. Action Item for Josh Hadley, PAG: Review the feedback from Norbert Lindenberg re: the set of characters with Indic_Conjunct_Break=Extend (formerly ExtCccZwj) for Unicode 16.0. See document L2/23-160 item 4.6 and ReportID: ID20230620135108

Feedback (verbatim)

Date/Time: Tue Jun 20 13:51:08 CDT 2023
ReportID: ID20230620135108
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

I’m happy to see some progress in fixing UAX 29 for Brahmic scripts, even if it’s initially only for 6 of the roughly 40 scripts that need a fix.

However, in the rule that defines consonant clusters, it’s not clear at all whether the class ExtCccZwj includes or excludes the right characters. The combining class for marks in Brahmic scripts (except for viramas and, up to now, nuktas) should generally be 0, and assignments of other values were in most cases mistakes that unfortunately can not be corrected. Trying to

derive meaning from ccc values in Brahmic scripts is almost certainly a
mistake. Why should variation selectors be excluded from consonant
clusters? Is the exclusion of three Gujarati nuktas intentional? Is the
inclusion of Vedic tone marks intentional?

If combining classes are really considered the appropriate basis for
selecting characters that can occur within a consonant cluster, then this
should be explained. If not, then the class should be defined so as to
include the right characters, independent of ccc values.

Background information / discussion

PAG discussed and agreed that although there are some weird edges, it has already been "battle tested" for four years. We did not think it was worthwhile to spend too much time on degenerate cases. We can refine things later but it is very late in the cycle to change anything for Unicode 15.1.

4.7 UAX29 set operators in regular expressions

Recommended UTC actions

1. Action Item for Robin Leroy, Mark Davis, PAG: Review regular expression syntax throughout the Unicode Standard & Annexes and provide recommendations to the UTC for Unicode 16.0. See L2/23-160 item 4.7 and ID20230621102117.

Feedback (verbatim)

Date/Time: Wed Jun 21 10:21:17 CDT 2023
ReportID: ID20230621102117
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

UAX 29 uses the set operators "&" and "-" in several regular expressions.
UTR 18 and Appendix A of The Unicode Standard have settled on "&&"
and "--". UAX 29 should follow.

Background information / discussion

Too late for Unicode 15.1 but will consider for 16.0

4.8 UAX29: consistency use of "default" and "defaults"

Recommended UTC actions

1. Action Item for Mark Davis, PAG: Change Section 3 of Unicode Standard Annex #29 to clarify the usage of the term default and to remove references to ancient terminology, as described in L2/23-160 item 4.8. See ReportID 20230623113010 on PRI-469. For Unicode Version 15.1.

Feedback (verbatim)

Date/Time: Fri Jun 23 11:30:10 CDT 2023
ReportID: ID20230623113010
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

The proposed update of UAX 29 states twice in new text that “the default
grapheme clusters are also known as extended grapheme clusters”, and that
legacy grapheme clusters are defined as a profile. On the other hand,
existing text talks about a “key feature of default Unicode grapheme
clusters (both legacy and extended)”, notes that “default [i.e., extended]
Unicode grapheme clusters were previously referred to
as ‘locale-independent graphemes’” even though that note predates the
invention of extended grapheme clusters, has a section “Default Grapheme
Cluster Boundary Specification” that covers both legacy and extended
grapheme clusters, and requires “When citing the Unicode definition of
grapheme clusters, it must be clear which of the two alternatives are being
specified: extended versus legacy” as if there were no default.

The use of “default” and defaults with respect to grapheme clusters should
be reviewed and made consistent.

Background information / discussion

Proposed text changes:

> A key feature of ~~default~~ Unicode grapheme clusters (both legacy and extended)
>
> The Unicode <u>specification</u> ~~definitions~~ of grapheme clusters ~~are defaults: not meant to exclude the use of~~ <u>allows for</u> more sophisticated <u>profiles</u> ~~definitions of tailored grapheme clusters~~ where appropriate.
>
> Note: ~~The default Unicode grapheme clusters were previously referred to as “locale-independent graphemes.”~~ The term cluster is used to emphasize that the term grapheme is used differently in linguistics. ~~For simplicity and to align terminology with Unicode Technical Standard #10, “Unicode Collation Algorithm” [UTS10], the terms default and tailored are preferred over locale-independent and locale-dependent, respectively.~~

4.9 UAX29 boundary between default clusters

Recommended UTC actions

1. Action item for Josh Hadley, PAG: Remove note in UAX29 Section 3 about boundaries for Unicode 15.1. See ReportID: ID20230623113048

Feedback (verbatim)

Date/Time: Fri Jun 23 11:30:48 CDT 2023
ReportID: ID20230623113048
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

UAX 29 has a note claiming that “The boundary between default Unicode grapheme clusters can be determined by just the two adjacent characters”. Looking at rules GB9c, GB11, GB12, and GB13, I don’t believe this is true.

Background information / discussion

PAG discussed and agreed to remove the note as suggested.

4.10 UAX29 Table 2a macro definitions

Recommended UTC actions

1. Action Item for Josh Hadley, PAG: Correct the description of UAX29 table 2a according to feedback. See L2/23-160 item 4.10 and ID20230623113150; for Unicode 15.1.

Feedback (verbatim)

Date/Time: Fri Jun 23 11:31:50 CDT 2023
ReportID: ID20230623113150
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

The description of Table 2a states “each macro represents a repeated union of the basic Grapheme_Cluster property values”. This seems to be incorrectly adapted from the descriptions of other tables. In reality, the table uses intersection and difference rather than union, and uses several other Unicode properties besides Grapheme_Cluster_Break (the real name of “Grapheme_Cluster”).

The other macro tables in UAX 29 consider “represents” clear enough without a “=“ sign; I think this would work here too.

Background information / discussion

PAG discussed and agreed to make this change.

4.11 UAX29 rule GB9c rendering

Recommended UTC actions

1. No action: The document has already been modified.

Feedback (verbatim)

Date/Time: Fri Jun 23 11:32:29 CDT 2023
ReportID: ID20230623113229
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

When rule GB9c is rendered in a narrow view (such as a printed page), it appears as

LinkingConsonant ExtCccZwj* × LinkingConsonant
ConjunctLinker ExtCccZwj*

which invites a reading very different from the intended one.

The rendering could be improved by using “vertical-align: bottom” on the
last two cells of the row.

4.12 UAX29: clarify that word/line boundary relationship is script-specific

Recommended UTC actions

1. Action Item for Josh Hadley, PAG: In UAX29, update the paragraph below figure 2 to indicate that the relationship of line break/word break boundaries is script-specific; for Unicode 15.1. See document L2/23-160 item 4.12.
2. Action Item for Norbert Lindenberg, PAG: For UAX29, propose improved wording for the last paragraph of the introduction of Section 4; for Unicode 16.0. See document L2/23-160 item 4.12.

Feedback (verbatim)

Date/Time: Fri Jun 23 11:33:23 CDT 2023
ReportID: ID20230623113323
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

The introduction to word boundaries in UAX 29 has a paragraph on the relationship between word boundaries and line boundaries. It should be clarified that this relationship exists only in some scripts, not in others. In Chinese, Japanese, Balinese, Brahmi, etc. line breaking pays no attention to words. Also, thanks to hyphenation engines for languages where words do matter for line breaking, line breaks within words are far more common than the statement on SHY would imply.

The last paragraph in the same section mentions three Line_Break property values and then states “that means that satisfactory treatment of languages like Chinese or Thai requires special handling”. Chinese uses none of the three Line_Break property values, and while word breaking for Chinese requires special handling, that has nothing to do with its line breaking.

Background information / discussion

PAG discussed and agreed to make a change to figure 2 for Unicode 15.1 and to request proposed improved wording for the last paragraph of the introduction of Section 4 from the submitter for consideration for Unicode 16.0

4.13 UAX29 Section 3 line boundaries with emoji modifier on non-standard base

Recommended UTC actions

No new action: We already have an action item to address such inconsistencies (see UTC-160-A73).

Feedback (verbatim)

Date/Time: Sun Jun 25 06:15:35 CDT 2023
ReportID: ID20230625061535
Name: Charlotte Buff
Report Type: Public Review Issue
Opt Subject: 469

Section 3, “Grapheme Cluster Boundaries”, states:

»Word boundaries, line boundaries, and sentence boundaries should
not occur within a grapheme cluster: in other words, a grapheme
cluster should be an atomic unit with respect to the process of
determining these other boundaries.«
This does not actually hold true for line boundaries when an emoji modifier
is applied to a non-standard base character. For example, the
sequence <U+1F9DF, U+1F3FB> 🧟🏻 (ZOMBIE, EMOJI MODIFIER FITZPATRICK
TYPE-1-2) is a single grapheme cluster because emoji modifiers have
Grapheme_Cluster_Break=Extend, but nonetheless a line break is
theoretically allowed between the two characters because ZOMBIE has
Emoji_Modifier_Base=False and line break rule LB30b applies only to
characters with Emoji_Modifier_Base=True or unassigned code points with
Extended_Pictographic=True. In fact, Chromium-based web browsers will break
lines in the middle of these sequences.

Background information / discussion

PAG discussion revealed that there is already an open Action Item to address this.

4.14 UAX14 LB28b, LB15b

Recommended UTC actions

1. No action: Robin Leroy has already addressed both of these in [tr14/tr14-50.html (draft 8)]

Feedback (verbatim)

Date/Time: Wed May 24 06:47:37 CDT 2023
ReportID: ID20230524064737
Name: Charlotte Buff
Report Type: Public Review Issue
Opt Subject: 461

In UAX #14, the new “Do not break inside the orthographic syllables of Brahmic scripts” line break rule is numbered LB28b. Because there is no rule LB28a (and there never has been as far as I can tell), it should instead be LB28a.

Furthermore, the new rule LB15b accidentally lists its number twice.

Background information / discussion

Edits to TR14 had already been made that address the feedback.

4.15 conjunctCluster: allow ExtCccZwj after each LinkingConsonant

Recommended UTC actions

1. Action Item for Josh Hadley, PAG: In Unicode Standard Annex #29, ensure the definition of the conjunctCluster regular expression incorporates the first sequence of Extend characters in the parenthesized group. See document L2/23-160 item 4.15. For Unicode Version 15.1.

Feedback (verbatim)

Date/Time: Fri Jun 30 07:45:06 CDT 2023
ReportID: ID20230630074506
Name: Charlotte Buff
Report Type: Public Review Issue
Opt Subject: 469

Table 1c defines the following regex pattern:

conjunctCluster := LinkingConsonant ExtCccZwj* (ConjunctLinker ExtCccZwj* LinkingConsonant)+
If we expand the “(ConjunctLinker ExtCccZwj* LinkingConsonant)+” part, we
get a sequence pattern where ExtCccZwj can occur only after a
ConjunctLinker but not before it:

ConjunctLinker ExtCccZwj* LinkingConsonant ConjunctLinker ExtCccZwj* LinkingConsonant
ConjunctLinker ExtCccZwj* LinkingConsonant ...
This does not match rule GB9c which accounts for ExtCccZwj in both
positions, which is necessary because Indic scripts make use of combining
marks with CCC values both smaller and greater than 9 (Virama). Therefore I
think the definition should actually be:

conjunctCluster := LinkingConsonant ExtCccZwj* (ConjunctLinker ExtCccZwj* LinkingConsonant
ExtCccZwj*)+

Background information / discussion

This should be
conjunctCluster := LinkingConsonant (ExtCccZwj* ConjunctLinker ExtCccZwj* LinkingConsonant)+
moving the opening parenthesis left.

Note: ICU uses
$LinkingConsonant $ExtCccZwj* $Virama $ExtCccZwj* $LinkingConsonant; with chaining on
$LinkingConsonant, thus
$LinkingConsonant ($ExtCccZwj* $Virama $ExtCccZwj* $LinkingConsonant)+;.

Note that the exact text interacts with the new property Indic_Conjunct_Break (InCB) from the item “Should the set of scripts affected by GB9c be provided in the UCD?”.

4.16 adjust UAX29 wording now that default rules handle aksaras

Recommended UTC actions

1. Action Item for Josh Hadley, PAG: Correct UAX29 as described in document L2/23-160 item 4.16, for Unicode Version 15.1. See ID20230704173903.

Feedback (verbatim)

Date/Time: Tue Jul 04 17:39:03 CDT 2023
ReportID: ID20230704173903
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

The discussion of Aksaras in UAX 29 states that “consonant cluster aksaras
are not incorporated into the default rules”. That’s no longer correct;
such aksaras are now incorporated for six scripts, and more will hopefully
follow.

The same paragraph mentions “additional prefixed consonants”. That seems to
reflect a Devanagari-centric view, as in many other scripts the additional
consonants are better described as “subjoined” or in other terms. I suggest
removing the word “prefixed”.

Background information / discussion

Change the paragraph mentioned in the feedback as follows:

> However, aksaras may also include one or more additional ~~prefixed~~ consonants, typically with a virama (halant) character between each pair of consonants in the sequence. ~~Such~~ Some consonant cluster aksaras are not incorporated into the default rules for extended grapheme clusters, in part because not all such sequences are considered to be single “characters” by users. Another reason is that additional changes to the rules are made when new information becomes available. Indic scripts vary considerably in how they handle the rendering of such aksaras—in some cases stacking them up into combined forms known as consonant conjuncts, and in other cases stringing them out horizontally, with visible renditions of the halant on each consonant in the sequence. There is even greater variability in how the typical liquid consonants (or “medials”), ya, ra, la, and wa, are handled for display in combinations in aksaras. So tailorings for aksaras may need to be script-, language-, font-, or context-specific to be useful.

4.17 UAX29 boundaries vs. normalization

Recommended UTC actions

1. PAG recommends no action.

Feedback (verbatim)

Date/Time: Tue Jul 04 17:39:43 CDT 2023
ReportID: ID20230704173943
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

The proposed update of UAX 29 states “Boundaries never occur within a combining character sequence or conjoining sequence, so the boundaries within non-NFD text can be derived from corresponding boundaries in the NFD form of that text.” Unfortunately, the stated condition is not sufficient.
It would also be ncessary that normalization didn’t reorder characters out of character pairs that should not be broken up. As the section “Compatibility with normalization” of L2/23-141 discusses, it sometimes does, and workarounds are necessary to achieve the desired results in normalized text.

Background information / discussion

- This does not seem to be a defect in the current set of UAX29 rules.
- We are elsewhere recommending a review of the document’s proposal.
- Note that UAX29 has an explicit constraint to “Ignore degenerates”.

4.18 Setting expectations for grapheme clusters

Recommended UTC actions

1. Action Item for Manish Goregaokar, Josh Hadley, PAG: Revise UAX29: Retain the phrase “user-perceived characters” to anchor the concept to the reader's intuition, but clarify that this is a useful approximation in most cases. There are some cases where the clusters are not related to “characters”. There are many situations, including different use cases, where tailoring is expected. See L2/23-160 item 4.18 and L2/23-140; for Unicode 16.0.

Feedback (verbatim)

Date/Time: Tue Jul 04 17:40:02 CDT 2023
ReportID: ID20230704174002
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 469

Document L2/23-140, Setting expectations for grapheme clusters, is intended
to be feedback to PRI 469.

Background information / discussion

The language "user-perceived characters" is still useful as it provides some form of anchoring for what grapheme clusters as a concept are attempting to do (and thus informing the use cases they may make sense for, though we should also provide examples).

We should express that we're attempting "an approximation of a notion of user perceived characters that makes some sense across writing systems and languages". Basically, highlight that we are trying to do a best-effort thing that has uniformity across all human text without having further language/font metadata about that text. We already do have such hedging language in the spec, this is mostly a matter of restructuring it a bit so it includes claims about user-perceived characters.

Saying to the spec user "we know you want such a notion even if it does not generalize across scripts. Here is a way of doing that anyway which is not entirely terrible". For that to work we kinda need to give them a familiar term to anchor to, and then load it up with caveats.

Sometimes the GCB segmentation is really not a wider sense of a "character", user-perceived or not, and "approximate" or not. Any new hedging language should allow for the notion to utterly fail for some scripts, or some clusters in some scripts, even if it is a good approximation for others. (Approximation instead claims that it's largely close, with limited infidelity or precision).

The requirement for tailoring seems to arise both for use cases, as well as perhaps for certain writing systems. We know that different text operations use different segmentation at the lowest level, and that could be made more prominent. In context, the feedback implies that the default GCB may be a poor match for some writing systems as a whole, or some types of clusters in them. We might mention that as well.

4.19 Required conjunct forms in extended grapheme clusters

Recommended UTC actions

1. Action Item for Robin Leroy, Manish Goregaokar, Mark Davis, PAG: Work with CLDR to review the proposal in L2/23-141 to prevent grapheme cluster breaks within certain conjunct forms, without complications for handling degenerate sequences, and to enable appropriate implementations. See document L2/23-160 item 4.19.

Summary

From the document's proposal section:

> This document proposes to update the definition of extended grapheme clusters in UAX 29, Unicode Text Segmentation, to include required conjunct forms that Unicode represents by a sequence of a virama-like character followed by a consonant or independent vowel, and to tie Myanmar kinzi glyphs to the base characters they belong to.

Background information / discussion

As with the change in grapheme cluster break rules for aksaras before, these kinds of changes should be implemented in CLDR+ICU before UAX29 is changed. Briefly:

1. Incorporate the rules into CLDR/ICU, but initially empty properties (a noop with no scripts).
2. Once the rules are in place, implementers can easily activate them by changing the character classes in the rule sets.
3. Then add scripts over time as we get test cases (like the existing scripts).
4. For each script, after it has been in a release of ICU for more than some time (1 year? 2 years?), add to UAX#29.

5. IDNA

5.1 UTS #46: domain labels can be empty

Recommended UTC actions

1. Action Item for Markus Scherer, PAG: In UTS #46, limit all of the Validity Criteria tests to non-empty labels; for Unicode 15.1. See L2/23-160 item 5.1.

Feedback (verbatim)

Date/Time: Mon Jan 23 05:11:04 CST 2023
Name: Anne van Kesteren
Report Type: Error Report
Opt Subject: UTS46

Steps don't always consider that domain labels can be empty, e.g., when CheckBidi is true the first subrule of "The Bidi Rule" inspects the first character of a label. I think that might also apply to CheckJoiners and potentially other steps. (I initially thought the problem here was VerifyDnsLength not being considered, but that check happens much later on in the processing model so it's something more fundamental.)

Discussion

We could make this small insertion at the beginning of 4.1 Validity Criteria: “Each of the following criteria must be satisfied for a non-empty label”

Alternatively we could clarify this just for CheckBidi with this insertion: “If CheckBidi, and if the domain name is a Bidi domain name, and if the label is not empty, then the label must satisfy ...”

There is no such problem with CheckJoiners because https://www.rfc-editor.org/rfc/rfc5892.html#appendix-A processes a label with `For All Characters`. On an empty label, this is an empty loop.

5.2 UTS #46 mapping of capital sharp s

From a German Google user

Recommended UTC actions

1. Consensus: In UTS #46 map U+1E9E capital sharp s to U+00DF small sharp s instead of to "ss", for Unicode 15.1.
2. Action Item for Mark Davis, Markus Scherer, PAG: In UTS #46 Mapping Table Derivation step 1 (base mapping) substep 1 (exceptional characters), map U+1E9E capital sharp s to U+00DF small sharp s. Add a note that this changes the mapping of the capital sharp s. For Unicode 15.1.
3. Action Item for Mark Davis, Markus Scherer, PAG: In IdnaMappingTable.txt map U+1E9E capital sharp s to U+00DF small sharp s instead of to "ss", for Unicode 15.1.

Feedback

A user noticed that in domain names U+1E9E ẞ capital sharp s maps to "ss".
In IDNA2003 and in UTS #46 “transitional” processing, U+00DF ß lowercase sharp s also maps to "ss".
However, now that all major browser implementations have switched to “nontransitional” processing, ß=ss no longer matches ß.
The mapping for lowercase sharp s was conditional on whether to use “transitional” processing, but the mapping for capital sharp s was unconditional.
It has been this way since the beginning of UTS #46 (Unicode 5.2 IdnaMappingTable.txt).

For example, GIEẞEN.DE goes to giessen.de which, when using “nontransitional” processing, is (potentially) different from gießen.de.

Background information / discussion

IdnaMappingTable.txt:

```
Unset
00DF          ; deviation                ; 0073 0073     # 1.1  LATIN SMALL LETTER SHARP S
1E9E          ; mapped                   ; 0073 0073     # 5.1  LATIN CAPITAL LETTER SHARP S
```

We cannot simply add a deviation mapping for capital sharp s, because unlike the existing deviation characters this one needs a mapping for either processing option, rather than mapping vs. pass-through.

It does not seem worth making the mapping conditional on the processing option, since we are deprecating “transitional” processing.

5.3 UTS #46 Processing.Map should not record an error

From Markus Scherer, ICU

Recommended UTC actions

1. Consensus: In UTS #46 section 4 Processing step 1 Map, do not record an error for disallowed characters, for Unicode 15.1.
2. Action Item for Markus Scherer, PAG: In UTS #46 section 4 Processing step 1 Map, do not record an error for disallowed characters. Instead, note that the 4.1 Validity Criteria include a check for disallowed characters. For Unicode 15.1. (These changes have been applied during the beta period.)
3. Action Item for Markus Scherer, PAG: Modify IdnaTestV2.txt so that it never records error P1 from the Processing.Map step, for Unicode 15.1. (These changes have been applied during the beta period.)

Feedback

While testing Unicode 15.1 beta in ICU, I found that Consensus UTC-175-C29 was based on an incomplete analysis and yields inconsistent results. The characters ≠ ≮ ≯ are now valid regardless of the UseSTD3ASCIIRules option, but when the input contains their decomposed forms (=\u0338 etc.), the Processing algorithm's Map step records an error when UseSTD3ASCIIRules=true, which is listed as a P1 error in IdnaTestV2.txt.

The Processing.Map step should not record errors for disallowed characters, because after mapping the rest of the string and then normalizing, the string may no longer include disallowed characters. The later check of the validity criteria will catch disallowed characters.

See
https://www.unicode.org/reports/tr46/#Processing
1. Map. ... disallowed: Leave the code point unchanged in the string, and record that there was an error.

https://www.unicode.org/reports/tr46/#Validity_Criteria
6. Each code point in the label must only have certain status values according to Section 5, IDNA Mapping Table:
   1. For Transitional Processing, each value must be valid.
   2. For Nontransitional Processing, each value must be either valid or deviation.

6. Collation

6.1 ISO 12199 Quo vadis?

L2/23-125 from Håvard Hjulstad

Recommended UTC actions

1. Consensus: The position of the Unicode Consortium is that ISO 12199 should be withdrawn in its entirety with no replacement (option 4 in L2/23-125).
2. Action Item for Peter Constable, UTC: Communicate to ISO/TC 37/SC 2 that the position of the Unicode Consortium is that ISO 12199 should be withdrawn in its entirety with no replacement (option 4 in L2/23-125).

Summary

ISO 12199 Alphabetical ordering of multilingual terminological and lexicographical data represented in the Latin alphabet

- dormant standard recently updated editorially
- small subset of ISO 14651/UCA/CLDR
- annexes with extensions (word-by-word ordering, ordering rules for chemical names, ...)

“Formally, ISO 12199:2023 is a valid International Standard until July 2027, unless ISO/TC 37/SC 2 in the meantime decides to

1. extend its validity for another five years, a process that may be repeated an indefinite number of times
2. revise the entire document
3. withdraw ISO 12199 as such, but develop a new document based on parts of the current document, in particular informative parts
4. withdraw ISO 12199 in its entirety with no replacement”

Background information / discussion

Several PAG members reviewed this document and discussed it in detail.

6.2 MySQL utf8mb4_0900_ai_ci 요 = ㅇㅛ

Recommended UTC actions

1. Action item for Markus Scherer, PAG: Respond to Jae Woong Lee about ReportID: ID20230613081245 with information from doc L2/23-160 item 6.2.

Feedback (verbatim)

Date/Time: Tue Jun 13 08:12:45 CDT 2023
ReportID: ID20230613081245
Name: Jae Woong Lee
Report Type: Error Report
Opt Subject:

Hello,

I am using unicode 9.0 with mysql 8.0 database.
collation name: utf8mb4_0900_ai_ci
I can't get the desired result when I compare the Korean string using unicode 9.0.
unicode 9.0 considers separated characters and combined characters as the same thing.

ex)

- 요 = 요 -> result True : correct
- 요 = ㅇㅛ -> result True : This is an invalid result.

But if I use other collations, utf8mb4_general_ci, utf8mb4_unicode_ci,
I get the correct result.

ex)

- 요 = 요 -> result True : correct
- 요 = ㅇㅛ -> result False : corrent

It seems that the Korean comparison method is different from 9.0. I'm
wondering why characters that look different to Koreans are called the same
in unicode 9.0. Is this by design or is it a bug and can it be fixed? I
contacted mysql, but they told me that it's not a mysql issue, but to
contact the unicode association because they used unicode 9.0 as it is.

[9 Jun 16:10] MySQL Verification Team
Hi,

You can observe that collating the constants changing the result.
You can try different COLLATE expressions.
Regarding Koreans language, we are not experts on this.
We just implemented the UTF standard, to the last point.
Hence, you should contact the people that define Unicode standards.
Also, do not forget that two strings with different grapheme clusters

can be considered identical, as per standard. There are many examples in
the textbooks on this subject.

Not a bug.

Regards,
Jae.

Background information / discussion

Hangul syllable 요 is canonically equivalent to the sequence of conjoining Jamo, and sorts tertiary-before the sequence ㅇㅛ of compatibility Jamo (see https://www.unicode.org/charts/PDF/U3130.pdf).

The MySQL collation options ai (“accent-insensitive”) and ci (“case-insensitive”) might be implemented via collation using strength=primary, which would ignore the tertiary difference.

The UTC does not have the bandwidth to serve as a second-level support team for implementers.

7. Security

7.1 UTS #39 PU: 'Z' undefined in definition of bidiSkeleton

PRI #463

Recommended UTC actions

1. No action, this has already been fixed in the document.

Feedback (verbatim)

Date/Time: Mon Apr 17 16:44:35 CDT 2023
Name: Peter G Constable
Report Type: Public Review Issue
Opt Subject: 463

In section 4 of PU UTS #39, there is a block of new text defining bidiSkeleton. The following is included in the steps to derive a bidiSkeleton:

Apply rule L3 of the UBA: move combining marks after their base in Z; this yields the sequence R′.

But "Z" has not been defined in the prior steps.

Background information / discussion

UTS # 39 PU section 4

7.2 align UTS #39 conformance with other UTSs and UAXs

PRI-463

Recommended UTC actions

1. Consensus: Update the conformance clauses in UTS #39 as proposed in L2/23-160 item 7.2. For Unicode 15.1.
2. Action Item For Mark Davis, Michel Suignard, PAG: Update the conformance clauses in UTS #39 as proposed in L2/23-160 item 7.2. For Unicode 15.1.

Feedback (verbatim)

Date/Time: Thu May 25 18:27:09 CDT 2023
ReportID: ID20230525182709
Name: Asmus Freytag
Report Type: Public Review Issue
Opt Subject: 463

Unset
```
The statement of conformance for UTS #39 should be brought into better
alignment with the way conformance is stated in other UTSs and UAXs.

Currently:

C1 An implementation claiming to implement the General Profile for
Identifiers shall do so in accordance with the specifications in Section
3.1, General Security Profile for Identifiers.

    Alternatively, it shall declare that it uses a modification, and provide
    a precise list of characters that are added to or removed from the
    profile.

C1.1 An implementation claiming to implement the IDN Security Profiles for
Identifiers shall do so in accordance with the specifications in Section
3.2, IDN Security Profiles for Identifiers.

    Alternatively, it shall declare that it uses a modification, and provide
    a precise list of characters that are added to or removed from the
    profile.

C1.2 An implementation claiming to implement the Email Security Profiles for
Identifiers shall do so in accordance with the specifications in Section
3.3, Email Security Profiles for Identifiers.

    Alternatively, it shall declare that it uses a modification, and provide
    a precise list of characters that are added to or removed from the
    profile.
```

```
(italics lost in plain text).

Note that the phrases starting with "Alternatively" are needlessly
repetitive. And, because of the term "Alternatively" they can be
interpreted as overriding the *entire* "shall" clause in the preceding
sentence, erasing any reference to Sections 3.1, 3.2 or 3.3, which cannot
be intended. Also, except for the use of italics, the presentation as a
separate indented paragraph fatally resembles the informative notes present
on so many definitions, rules and clauses elsewhere, and thus casting into
doubt the force and formal nature of these phrases.

It would be better to consolidate them into their own clause which
generically covers the conformance to a modified profile. And used the
phrase "in addition" to make sure that specifying the modifications alone
without reference to the underlying profile is not sufficient.

Finally, conformance without reference to version is meaningless for these
profiles, so a clause remedying that omission is also proposed.

While these proposals do not intend to make any effective changes to the
de-facto requirements for conformance, they clarify the language and
therefore preclude certain unintended interpretations. This means a formal
change to normative language and is therefore proposed here so it can be
reviewed by UTC.

PROPOSED: C1 An implementation claiming to implement the General Profile for
Identifiers shall do so in accordance with the specifications in Section
3.1, General Security Profile for Identifiers.

C1.1 An implementation claiming to implement the IDN Security Profiles for
Identifiers shall do so in accordance with the specifications in Section
3.2, IDN Security Profiles for Identifiers.

C1.2 An implementation claiming to implement the Email Security Profiles for
Identifiers shall do so in accordance with the specifications in Section
3.3, Email Security Profiles for Identifiers.

C1.3 An implementation claiming conformance according to the profiles in C1,
C1.1, or C1.2, but wishing to use a modification, shall additionally
provide a precise list of characters that are added to or removed from the
profile.

C1.4. An implementation that claims to conform to this specification must
reference the version it conforms to.

    The version number of this document is synchronized with the version of
    the Unicode Standard which specifies the repertoire covered. See also
    The Unicode Standard, Section 3.1 Versions of the Unicode Standard.
```

```
(All but the very last paragraph in italics)

Another question that might be considered is whether to rename the "C"
clauses by a unique prefix "UTS39-", which would bring them more in line
with e.g. UTS10 or the UAXs.
```

Background information / discussion

UTS #39, like UAX #31, is set up to allow explicit claims of conformance to the default Profile as specified, or to allow a claim that is augmented by a detailed accounting for the differences to the default, as the result of tailoring.

For UAX #31, UTC just decided to split all conformance clauses into a sub-part -1 and -2, while the existing text of UTS #39 uses an ambiguous formulation using "Alternatively,".

The easiest fix would be to split the conformance clauses on a similar model as it was done for UAX #31. The downside is that UTS #39 has a larger number of conformance clauses that would have to be duplicated. However, because of the use of "Alternatively,.." the necessary amount of text is not too different from what is there.

Theoretically, it should have been possible to use a single clause describing generically how to claim conformance for an implementation that applies a tailoring to the default.

Doing so would seem to better 'factor' the text, but it would come with a loss of specificity on two levels. One, the instructions would have to be generic, because different conformance clauses allow different kinds of tailoring and that affects the type of data or description a tailored conformance claim would need to supply. Two, unlike the scheme from the revised UAX #31 it would not be possible to pinpoint the type of conformance by using the full clause number, as the same clause number would apply to both tailored and non-tailored conformance. (Note, this is the current situation, which we are trying to fix).

Having considered this issue, the PAG decided to propose a revision based on the pattern set by UAX #31. Like UAX #31, the conformance clauses will be prefixed with the numeric shorthand for the specification, in this case "UTS39-". While this formally changes the string used to index the clause, the change is not material, as there's no confusion or ambiguity between the statements "clause C1 in UTS #39" and "clause UTS-39-C1".

Editorially, we recommend styling the common prefix to be less prominent than the Cn-m part of the number. This could be done with color, but is shown here with font weight.

PROPOSED

UTS-39-C1 *An implementation claiming to implement the General Profile for Identifiers shall do so by conforming to either UTS-39-C1-1 or UTS-39-C1-2.*

UTS-39-C1-1 *The Implementation shall be in accordance with the specifications in Section 3.1, General Security Profile for Identifiers, without change.*

UTS-39-C1-2 *The implementation shall provide a precise list of characters that are added to or removed from the profile, but otherwise be in accordance with the specifications in Section 3.1, General Security Profile for Identifiers.*

UTS-39-C1.1 *An implementation claiming to implement the IDN Security Profiles for Identifiers shall do so by conforming to either UTS-39-C1.1-1 or UTS-39-C1.1-2.*

UTS-39-C1.1-1 *The Implementation shall be in accordance with the specifications in Section 3.2, IDN Security Profiles for Identifiers, without change.*

UTS-39-C1.1-2 *The implementation shall provide a precise list of characters that are added to or removed from the profile, but otherwise be in accordance with the specifications in Section 3.2, IDN Security Profiles for Identifiers.*

UTS-39-C1.2 *An implementation claiming to implement the Email Security Profiles for Identifiers shall do so by conforming to either UTS-39-C1.2-1 or UTS-39-C1.2-2.*

UTS-39-C1.2-1 *The Implementation shall be in accordance with the specifications in Section 3.3, Email Security Profiles for Identifiers, without change.*

UTS-39-C1.2-2 *The implementation shall provide a precise list of characters that are added to or removed from the profile, but otherwise be in accordance with the specifications in Section 3.3, Email Security Profiles for Identifiers.*

UTS-39-C2 *An implementation claiming to implement any of the following confusable-detection functions for Identifiers defined in Section 4, Confusable Detection shall do so by conforming to either UTS-39-C2-1 or UTS-39-C2-2.*

1. *X and Y are single-script confusables*
2. *X and Y are mixed-script confusables*
3. *X and Y are whole-script confusables*
4. *X has whole-script confusables in set of scripts S*

UTS-39-C2-1 *The Implementation of the function shall be in accordance with the specifications in Section 4, Confusable Detection, without change.*

UTS-39-C2-2 *The implementation shall provide a precise list of character mappings that are added to or removed from those provided, but otherwise be in accordance with the specifications in Section 4, Confusable Detection.*

UTS-39-C3 *An implementation claiming to detect mixed scripts shall do so by conforming to either UTS-39-C3-1 or UTS-39-C3-2.*

UTS-39-C3-1 *The Implementation shall be in accordance with the specifications in Section 5.1, Mixed-script Detection, without change.*

UTS-39-C3-2 *The implementation shall provide a precise description of changes in behavior, but otherwise be in accordance with the specifications in Section 5.1, Mixed-Script Detection.*

UTS-39-C4 *An implementation claiming to detect Restriction-Levels shall do so by conforming to either UTS-39-C4-1 or UTS-39-C4-2.*

UTS-39-C4-1 *The Implementation shall be in accordance with the specifications in Section 5.2, Restriction-Level Detection, without change.*

UTS-39-C4-2 *The implementation shall provide a precise description of changes in behavior, but otherwise be in accordance with the specifications in Section 5.2, Restriction-Level Detection.*

UTS-39-C5 *An implementation claiming to detect mixed numbers shall do so by conforming to either UTS-39-C5-1 or UTS-39-C5-2.*

UTS-39-C5-1 *The Implementation shall be in accordance with the specifications in Section 5.3, Mixed-Number Detection, without change.*

UTS-39-C5-2 *The implementation shall provide a precise description of changes in behavior, but otherwise be in accordance with the specifications in Section 5.3, Mixed-Number Detection.*

7.3 SCWG update

From Robin Leroy, SCWG

Recommended UTC actions

1. Consensus Approve Draft UTS #55 for publication as Unicode Technical Standard #55, Unicode Source Code Handling.
2. Action Item for Robin Leroy, PAG: Shepherd the publication of test cases for the conversion to plain text algorithm for UTS #55.

Summary

The SCWG does not plan to produce a separate report document this time around, as there have only been minor changes to UAX #31 and UTS #55. However, the UTC should take note of these changes.

The non-editorial modifications since UTC-175 looked at these documents are:

- UTS #55: Added a note clarifying that while identifiers should be equivalent under normalization, string literals should not be silently normalized: https://www.unicode.org/reports/tr55/tr55-2.html#Normalization-Case.
- UTS #55: Switched from levels of show hidden to a simpler model of independent options, and included non-NFC text as an option: https://www.unicode.org/reports/tr55/tr55-2.html#Show-Hidden-Options.
- UAX #31: In the note under UAX31-R3c, clarified that the definition excludes unassigned code points with the Pattern_Syntax property: https://www.unicode.org/reports/tr31/proposed.html#User-Defined_Operators.

In addition, the SCWG recommends that we publish test cases for the conversion to plain text algorithm, that is, pairs of input and output files in languages where conversion to plain text is generally applicable (right now these are Ada and Rust; the SC 22 liaison officer hopes that this can be extended to C++ as a Defect Report soon enough after 15.1).

Input	**Output**
Ada	Ada
Rust	Rust

The feedback from https://www.unicode.org/review/pri474/feedback.html#ID20230703084734 has been addressed editorially.