UTC #178 properties feedback & recommendations

Markus Scherer & Josh Hadley / Unicode properties & algorithms group, 2024-jan-17

Participants

The following people have contributed to this document:

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

1. Core spec

1.1 Unicode core spec improvements for variation selectors

L2/23-286 from Markus Scherer, Asmus Freytag, and other PAG members

Recommended UTC actions

1. Consensus: Improve the core spec chapters 3 & 23 text about variation selectors as proposed in L2/23-286. For Unicode 16.0. See L2/24-009 item 1.1.
2. Action Item for Markus Scherer, Asmus Freytag, PAG: Improve the core spec chapters 3 & 23 text about variation selectors as proposed in L2/23-286. For Unicode 16.0. See L2/24-009 item 1.1.

Summary

Proposed core spec changes for action items

- 152-A5a Ken Whistler, Mark Davis, EDC: Draft a new section for Chapter 3 on variation selectors and variation sequences, for Version 11.0. (retargeted to 13.0, 14.0, 15.0)
- 166-A61 Markus Scherer, Norbert Lindenberg, EDC: Propose changes to the specification of variation sequences in TUS chapter 23.4 and appropriate additions to chapter 3, based on document L2/21-012 item D2. The intent is to clarify the restrictions on initial characters in order to avoid issues under normalization. Include examples of characters and sequences that are excluded. See also action item 152-A5a.

2. UCD

2.1 Propose to Change BidiMirroring property for U+226D

L2/23-274 by CheonHyeong Sim (沈天珩)

Recommended UTC actions

We have reviewed this proposal and discussed it with SAH/UTC members, and agree with the SAH recommendation to change the Bidi_Mirrored property of U+226D NOT EQUIVALENT TO to Yes.

Summary

The glyph of U+226D should be mirrored, although neither of the characters in its Decomposition_Mapping are Bidi_Mirrored, and should not be.

2.2 Consistency of InSC and Alpha/Dia/Ext

From Ken Whistler & Robin Leroy, PAG

Recommended UTC actions

1. Consensus: Assign Extender=Yes to 3 characters: U+0A71 GURMUKHI ADDAK, U+0AFB GUJARATI SIGN SHADDA, U+11237 KHOJKI SIGN SHADDA. For Unicode 16.0. See L2/24-009 item 2.2.
2. Action Item for Robin Leroy, PAG: Assign Extender=Yes to 3 characters: U+0A71 GURMUKHI ADDAK, U+0AFB GUJARATI SIGN SHADDA, U+11237 KHOJKI SIGN SHADDA. For Unicode 16.0. See L2/24-009 item 2.2.
3. Consensus: Assign Diacritic=Yes to 5 characters: U+1BE6 BATAK SIGN TOMPI, U+10A38 KHAROSHTHI SIGN BAR ABOVE, U+10A39 KHAROSHTHI SIGN CAUDA, U+10A3A KHAROSHTHI SIGN DOT BELOW, U+1133B COMBINING BINDU BELOW. For Unicode 16.0. See L2/24-009 item 2.2.
4. Action Item for Robin Leroy, PAG: Assign Diacritic=Yes to 5 characters: U+1BE6 BATAK SIGN TOMPI, U+10A38 KHAROSHTHI SIGN BAR ABOVE, U+10A39 KHAROSHTHI SIGN CAUDA, U+10A3A KHAROSHTHI SIGN DOT BELOW, U+1133B COMBINING BINDU BELOW. For Unicode 16.0. See L2/24-009 item 2.2.
5. Consensus: Assign Diacritic=Yes to 7 characters: U+0E3A THAI CHARACTER PHINTHU, U+1734 HANUNOO SIGN PAMUDPOD, U+1BF2 BATAK PANGOLAT, U+1BF3 BATAK PANONGONAN, U+A806 SYLOTI NAGRI SIGN HASANTA, U+A82C SYLOTI NAGRI SIGN ALTERNATE HASANTA, U+11F41 KAWI SIGN KILLER. For Unicode 16.0. See L2/24-009 item 2.2.
6. Action Item for Robin Leroy, PAG: Assign Diacritic=Yes to 7 characters: U+0E3A THAI CHARACTER PHINTHU, U+1734 HANUNOO SIGN PAMUDPOD, U+1BF2 BATAK PANGOLAT, U+1BF3 BATAK PANONGONAN, U+A806 SYLOTI NAGRI SIGN HASANTA, U+A82C SYLOTI NAGRI SIGN ALTERNATE HASANTA, U+11F41 KAWI SIGN KILLER. For Unicode 16.0. See L2/24-009 item 2.2.
7. Consensus: Assign Diacritic=Yes to 3 characters: U+1A60 TAI THAM SIGN SAKOT, U+10A3F KHAROSHTHI VIRAMA, U+11F42 KAWI CONJOINER. For Unicode 16.0. See L2/24-009 item 2.2.
8. Action Item for Robin Leroy, PAG: Assign Diacritic=Yes to 3 characters: U+1A60 TAI THAM SIGN SAKOT, U+10A3F KHAROSHTHI VIRAMA, U+11F42 KAWI CONJOINER. For Unicode 16.0. See L2/24-009 item 2.2.

Initial exchange

Ken:

I don't think there is a real "explanation" for the discrepancies [in the assignment of the Extender property] other than insufficient analysis, occasional oversight, lack of expertise in the implications of some characters in scripts for which we are not all experts, and fuzziness of concept.

I'd be fine with adding the two new gemination marks (Garay, and Tulu-Tigalari) to Extender for 16.0, and then coming around again later to consider further consistency issues for the others.
The InSC value of "Gemination_Mark" was added much later, of course, and I don't think we've ever rigorously checked for consistency since then.

Robin:

This issue is part of the « coming around again later ».

Since we have a finer-grained characterization of the function of characters in scripts covered by Indic_Syllabic_Category, we might be able to use that to inform the assignment of the fuzzier more general properties.

Surely InSC=Gemination_Mark should entail Extender.

I noticed that most of `\p{InSC=Virama}\p{InSC=Pure_Killer}\p{InSC=Invisible_Stacker}` are Alpha=N; Dia=Y; Ext=N, which seems eminently sensible at least for Virama and Pure_Killer (I am not sure if `Invisible_Stackers` can really be said to *linguistically modify the meaning of another character to which they apply* though…).

See the exceptions.

Other thoughts: aren't bindus diacritic when used for vowel nasalization?

Discussion

We discussed possible correlations between properties.

Agreed with property changes to fix inconsistencies:

1. \p{InSC=Gemination_Mark} ⊆ \p{Extender}
   - Assign Extender=Yes to 3 characters: U+0A71 GURMUKHI ADDAK, U+0AFB GUJARATI SIGN SHADDA, U+11237 KHOJKI SIGN SHADDA
2. \p{InSC=Nukta} ⊆ \p{Diacritic}
   - Assign Diacritic=Yes to 5 characters: U+1BE6 BATAK SIGN TOMPI, U+10A38 KHAROSHTHI SIGN BAR ABOVE, U+10A39 KHAROSHTHI SIGN CAUDA, U+10A3A KHAROSHTHI SIGN DOT BELOW, U+1133B COMBINING BINDU BELOW
3. [\p{InSC=Virama}\p{InSC=Pure_Killer}] ⊆ \p{Diacritic}
   - Assign Diacritic=Yes to 7 characters: U+0E3A THAI CHARACTER PHINTHU, U+1734 HANUNOO SIGN PAMUDPOD, U+1BF2 BATAK PANGOLAT, U+1BF3 BATAK PANONGONAN, U+A806 SYLOTI NAGRI SIGN HASANTA, U+A82C SYLOTI NAGRI SIGN ALTERNATE HASANTA, U+11F41 KAWI SIGN KILLER
4. \p{InSC=Invisible_Stacker} ⊆ \p{Diacritic}

    - Assign Diacritic=Yes to 3 characters: U+1A60 TAI THAM SIGN SAKOT, U+10A3F KHAROSHTHI VIRAMA, U+11F42 KAWI CONJOINER

Agreed with an exception:

5. \p{InSC=Avagraha} ⊆ \p{Alphabetic}

- Make an exception for U+0F85 TIBETAN MARK PALUTA which is a punctuation character and thus not Alphabetic.

Not agreed:

6. \p{InSC=Syllable_Modifier}-\p{No} ⊆ \p{Diacritic}

- Syllable_Modifier characters are too varied

The UCD maintainers will add tests to their tooling for these correlations.

2.3 Review the assignment of the Extender property for gemination or length marks

From Ken Whistler & Robin Leroy, PAG

Recommended UTC actions

1. Action Item for Ken Whistler, Robin Leroy, PAG: Review more characters for whether they should have the Extender property, looking at characters with names including "GEMINATION", "SHADDA", "LENGTH", "LONG VOWEL", "PLUTA" and similar. For Unicode 17.0. See L2/24-009 item 2.3.

Initial exchange

Ken: [same as in the previous item]

Robin:

This issue is part of the « coming around again later ».

See
https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5Cp%7Bname%3D%2FGEMINATION%7CSHADDA%7CLENGTH%7CLONG%20VOWEL%7CPLUTA%2F%7D-%5Cp%7BSignwriting%7D&g=ext&i=gc+insc+alpha+dia
and look for more.

Also might be interesting to look at
https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5Cp%7Bname%3D%2FREDUPLICATION%7CREPETITION%7CITERATION%7CREPEAT%2F%7D-%5Cp%7Bblock%3DMusical+Symbols%7D&g=ext&i=gc+insc+alpha+dia.

2.4 Characters that do not show an explicit mark in a span of Japanese wakiten

From Addison Phillips, W3C I18N Core Working Group

Recommended UTC actions

1. Note: Asmus Freytag and Ken Whistler have edited core spec section 4.5, adding Japanese wakiten as another example/bullet item for characters with multiple interpretations. For Unicode 16.0. See L2/24-009 item 2.4.
2. Action Item for Robin Leroy, PAG: Propose a property for the kana mappings currently hardcoded in the UCA sifter. For Unicode version 17.0. See L2/24-009 item 2.4 and CLDR-17044.

Summary

The W3C I18N Core WG resolved to ask Unicode whether the General_Category of some characters, including ASCII characters, could be changed so that \p{P} would equal to the set of characters that do not show an explicit mark in a span of Japanese wakiten.

Ken Whistler replied thus:

> Looking at the pull request and discussion, it is clear that the basic contention is that the following characters from the Latin-1 subset of Unicode:
> # % & @ § ¶
>
> and then their various NFKD-equivalent kin and semantically related characters in other blocks, are miscategorized as punctuation (specifically gc=Po), when they should be treated as symbols (gc=So).
> This is in fact a very, very old contention, having to do with the ambiguity of punctuation versus symbol function of many characters. And the problem is especially acute for the very heavily overloaded functions of many of the original ASCII non-alphanumeric characters.
>
> The history of these particular characters is that for the first 4 (the ASCII ones) # % & @ the UTC settled on gc=Po back in 1996 for Unicode 2.0, and they have stayed that way ever since. The last 2 (the Latin-1 ones) § ¶, which are mostly seen in text contexts other than formal syntax usages, started out as gc=So for Unicode 2.0, but then were explicitly changed to gc=Po as part of a cleanup of some categories in 2012 for Unicode 6.1. Those two have been gc=Po since then.
>
> Note that part of the problem here is with over-expectations about the meaning of the General_Category property in the first place. The value of gc is not some truth about characters -- it was intended as a useful bucketing of major groupings of characters, but it was always understood as a) fuzzy around the edges, and b) needing to be augmented by other properties when dealing with specific behaviors of sets of characters in various algorithms and contexts. […]
> So when trying to define a class such as "characters which show an explicit mark in a span of Japanese wakiten", […] once can always start with with General_Category values, but then go on to find the most precise (and elegant) statement of the exception list that applies in a particular use case, and look for ways to future-proof that statement against possible further expansions of the supported repertoire of characters.

Robin provided illustrations of such « precise, elegant, and future-proof » derivations in the UCD, and noted that such derivations were usually not stable, and that this is why the UCD publishes data files for derived properties, so that implementers need only pick up new data files, and only UCD maintainers need to care about the changes to derivation.

Robin also asked:

Do you need a formal note from the UTC on the question of the General_Category of those characters?

To the first point Addison replied:

CSS doesn't want to be in the business of making lists of characters and their properties. I think you could read this as a request that *Unicode* make such a list/derived property. I note that there is also this CLDR issue we recently filed (which doesn't seem like a CLDR problem to me): https://unicode-org.atlassian.net/browse/CLDR-17044, about a mapping that CSS maintains of small kana to kana and which looks pretty similar to this.

To the second he replied:

Only if there is something materially different in that response, otherwise the CSS folks will get the gist of it from this thread.

Discussion

The General_Category is a partition and thus forces an either-or distinction. This cannot satisfy all contexts and use cases for characters with multi-faceted uses. Some contexts and use cases require additional data (overrides and exceptions). This is discussed in TUS section 4.5 "General Category". We should add Japanese wakiten as an example there.

PAG thinks that the UTC should not build and maintain this list of exceptions/overrides for a use case outside Unicode specifications.

2.5 misspelled name of U+1680B: BAMUM LETTER PHASE-A MAEMBGBIEE

Recommended UTC actions

1. Consensus: Create a formal alias BAMUM LETTER PHASE-A MAEMGBIEE of type "correction" for U+1680B BAMUM LETTER PHASE-A MAEMBGBIEE. For Unicode Version 16.0.
2. Action Item for Markus Scherer, PAG: In NameAliases.txt, add a formal alias BAMUM LETTER PHASE-A MAEMGBIEE of type "correction" for U+1680B BAMUM LETTER PHASE-A MAEMBGBIEE. For Unicode Version 16.0.

Feedback (verbatim)

Date/Time: Sat Nov 18 10:43:42 CST 2023
ReportID: ID20231118104342
Name: Mikhail Morozov
Report Type: Error Report
Opt Subject: The Unicode Standard, Version 15.1, Bamum Supplement Range: 16800–16A3F

There is a misspelling in the name of the character 𖠋 (U+1680B) BAMUM LETTER PHASE-A MAEMBGBIEE in https://www.unicode.org/charts/PDF/U16800.pdf.

The proposal for encoding Old Bamum script (https://www.loc.gov/rr/amed/pdf/proposal-for-encoding-bamum-script.pdf#page=20) has IPA, English and French transcriptions for the letters, and it seems that the English transcription should be spelled with one B instead of two, MAEMGBIEE. The source for the proposal, L'Écriture des Bamum: sa naissance, son évolution, sa valeur phonétique, son utilisation, by I. Dugast and M.D.W. Jeffreys (https://www.calameo.com/read/000061616e47e713325db) also supports this opinion.

2.6 Fix PropertyValueAliases for Teh_Marbuta_Goal / Hamza_On_Heh_Goal

From Mark Davis, PAG

Recommended UTC actions

1. Consensus: Make the proposed change to PropertyValueAliases.txt to fix the long value alias for jg=Teh_Marbuta_Goal. For Unicode 16.0. See L2/24-009 item 2.6.
2. Action Item for Mark Davis, PAG: Make the proposed change to PropertyValueAliases.txt to fix the long value alias for jg=Teh_Marbuta_Goal. For Unicode 16.0. See L2/24-009 item 2.6.

Feedback

Proposal

Change the line in PropertyValueAliases.txt

Unset
```
jg ; Teh_Marbuta_Goal ; Hamza_On_Heh_Goal
```

To match the format of the other lines for 'jg'.

Unset
```
jg ; Teh_Marbuta_Goal ; Teh_Marbuta_Goal ; Hamza_On_Heh_Goal
```

Background

There was an action to add Hamza_On_Heh_Goal as an alias for Teh_Marbuta_Goal. Unfortunately, there was a mistake in doing so (mea culpa), and rather than add an alias, the long property value name (3rd field) was changed to Hamza_On_Heh_Goal, leaving the short property value name (2nd field) as Teh_Marbuta_Goal: a bizarre situation.

The header of PropertyValueAliases.txt reads:

Unset
```
# Second Field: The second field is the short name for the property value.

# It is typically an abbreviation, but in a number of cases it is simply

# a duplicate of the "long name" in the third field.

#

# Third Field: The third field is the long name for the property value,

# typically the formal name used in documentation about the property value.
```

So that means that Teh_Marbuta_Goal is the short name, and Hamza_On_Heh_Goal is the long name. So where does this strange behavior come from? We find in the header of ArabicShaping.txt:

Unset
```
# Note: The property value now designated [Joining_Group = Teh_Marbuta_Goal]

#   used to apply to both of the following characters

...

#   To avoid destabilizing existing Joining_Group property aliases, the

#   prior Joining_Group value for U+06C3 (Hamza_On_Heh_Goal) has been

#   retained as a property value alias, despite the fact that it

#   no longer applies to its namesake character, U+06C2.

#   See PropertyValueAliases.txt.
```

3. UCDXML

PRI #486 Stabilization of UAX #42, Unicode Character Database in XML (UCDXML)

Recommended UTC actions

1. Consensus: Close PRI #486
2. Consensus: Withdraw actions related to the previous recommendation to stabilize UAX #42, including 177-A105
3. Action Item for Rick McGowan, UTC: Respond on PRI #486 thanking respondents for their comments and noting that the feedback was instrumental in securing continued maintenance of UAX #42. See L2/24-009 item 3.
4. Action Item for Rick McGowan, UTC: Close PRI #486 with a note indicating that UAX #42 will not be stabilized and will continue to be maintained as long as we have willing volunteers. See L2/24-009 item 3.
5. Note: Continued maintenance of UAX #42 and UCDXML is dependent on volunteers continuing maintenance efforts. Should we find ourselves without a maintainer again, PAG could recommend stabilization once again. See L2/24-009 item 3.

Summary

Owing to considerable activity and comments on PRI #486 in late 2023 – early 2024 and other communications, PAG was asked to review and reconsider the previous recommendation to stabilize UAX #42 and the UCDXML data and make recommendations to the UTC for UTC #178. Related to this activity, a volunteer has come forward and has committed to learning the tooling & processes necessary to maintain the UAX and data. As a result: at the January 11 PAG meeting, the group agreed that we should recommend that the UTC:

- close the PRI
- rescind the previous recommendation to stabilize UAX #42
- proceed with updates and maintenance for Unicode 16.0 and beyond.

4. New Scripts & Characters

PAG members reviewed the following proposals, provided feedback to SAH, and the feedback has been addressed.
No further recommended actions from our side.

- L2/23-205 Reordering virama
- L2/23-206R Unicode request for Harrington diacritics (revised)
- L2/23-208 Unicode request for compound tone diacritics II
- L2/23-253 Working draft of Proposed Draft Unicode Standard Annex # 57: Egyptian Hieroglyph Database
- L2/23-272 Propose to Add Script_Extension for some CJK Punctuations
- L2/23-193R2 Proposal for Ten Chemical Symbols (revised)
- L2/23-276 Unicode request for Stein-Zimmermann quartertone Accidentals
- L2/23-277 Unicode request for Unicode request for numbers with slashes used in figured bass
- L2/24-024 Addition of Kannada to Script Extensions of U+1CD3 VEDIC SIGN NIHSHVASA

5. Normalization

5.1 Kirat Rai & Tulu-Tigalari vowel signs AI, Gurung Khema U: trouble with normalization

From Markus Scherer, PAG

Recommended UTC actions

1. Action Item for Mark Davis, PAG: Modify the descriptions of the normalization quick check values in UAX #44 and UAX #15 to reflect the possibility of characters that have decomposition mappings and may change in NFxC normalization depending on context, and clarify that the NFxC_Quick_Check values are chosen for the quickCheck algorithm to yield accurate results. For Unicode 16.0. See L2/24-009 item 5.1.
2. Action Item for Robin Leroy, Markus Scherer, PAG: Set NFxC_Quick_Check=Maybe for characters like U+16D68 KIRAT RAI VOWEL SIGN AI which may change in NFxC normalization depending on context. For Unicode 16.0. See L2/24-009 item 5.1.
3. Action Item for Robin Leroy, Markus Scherer, PAG: Add test cases to NormalizationTest.txt that exercise composition with the components of U+16D68 KIRAT RAI VOWEL SIGN AI and similar characters. For Unicode 16.0. See L2/24-009 item 5.1.
4. Action Item for Markus Scherer, Ken Whistler, PAG: In the Unicode 16.0 alpha PRI and on the 16.0 beta & landing pages, point out the possibility of characters that have decomposition mappings and may change in NFxC normalization depending on context. For Unicode 16.0. See L2/24-009 item 5.1. Note: subject to implementation issues in ICU being resolved.

Summary

We have approved the encoding of Kirat Rai vowel signs with unusual, multi-level canonical equivalence between several of them. We have worked through complications for segmentation (where we treat these like Hangul/Jamo) and collation (needing more contractions).

I have discovered complications for normalization processes. In short:

- The full decomposition of several characters ends with a vowel sign E which is also the first character in the decomposition of vowel sign AI.
- Therefore, vowel sign AI needs to have NFC_Quick_Check=Maybe, although it does not occur as a second character in any canonical decomposition.
  - It appears to be the first character for which this is true.
- It is also the first character which would have NFC_Quick_Check=Maybe as well as a Decomposition_Mapping.
- For implementations that take a string and compute a list of canonically equivalent strings, the overlapping equivalences with vowel sign AI on a sequence of three or more vowel signs E will yield a large number of output strings.

We have also approved the encoding of Tulu-Tigalari with a similar overlap between the decompositions of several characters ending with vowel sign EE and the decomposition of vowel sign AI starting with vowel sign EE. As well as Gurung Khema where vowel sign U=AA+AA and the decompositions of several vowel signs start with AA.

I do not see a violation of the Unicode normalization algorithms, but this creates problems for implementations, including and especially those not under our control. Implementers of normalization and related processes would need to carefully check that their implementations yield correct results for this new combination of properties.

This includes Unicode's own implementations: The UCD generation tools have generated incorrect preliminary data, the tools have not automatically added relevant test cases, and ICU cannot currently handle these cases in normalization.

There is also a complication for collation and other processes where canonical equivalence should be preserved.

Consider text like E+AI+E which "weird", but is canonically equivalent with E+E+E+E and AI+AI.

In the UCA, we do NFD first, but we also do provide DUCET mappings for non-NFD characters and some sequences ("canonical closure"). ICU relies on that and tries to avoid NFD normalization, for performance. The problem is, if there is misaligned text like this, how can we effectively detect that we need to decompose before lookup, and how far back and forward in the text do we need to decompose? AI=E+E forms a contraction in both Kirat Rai and Tulu-Tigalari. (Also finding the Kirat Rai AU=AA+E+E contraction in AA+E+AI.)

This is harder than finding combining marks out of order. In that case, we can collect enough context to reorder a whole combining sequence. But these vowel signs have ccc=0, so we have to glean from the contractions data (unsafe-backwards set) how far to go back. But we don't look at that data for *triggering* decomposition.

We should consider

- whether to move ahead with the approved encodings; fix our tools, data, and libraries; and prominently warn implementers
- or whether to revisit the Kirat Rai and Tulu-Tigalari encodings of vowel signs, such as possibly withdrawing vowel signs AI and other composite-character vowel signs.

Note that we also want to avoid more "do not use" situations. This means that we do not want to simply remove the decomposition mappings from the composite vowel signs, because that would yield atomic vowel signs that look just like sequences of others.

Details

At a minimum, we should modify the descriptions of the normalization quick check values in UAX #44 and UAX #15 to reflect this possibility and clarify that the values are chosen for the quickCheck algorithm to yield accurate results.

If we move ahead with the approved encodings, then we need to at least

- fix our tools so that this combination of properties is detected and DerivedNormalizationProps.txt set NFC_Quick_Check(U+16D68 AI)=Maybe
    - Same for Tulu-Tigalari vowel signs U+113C5 AI, U+113C7 OO, U+113C8 AU
    - Same for most of the Gurung Khema vowel signs
- add relevant, tricky test cases to NormalizationTest.txt including
    - 16D69 16D68 (AA+AI --NFD--> AA+E+E --NFC--> AU)
    - 16D69 16D67 16D68 (AA+E+AI --NFD--> AA+E+E+E --NFC--> AU+E)
- change the ICU normalization data structure which does not currently support characters with NFC_QC=Maybe *and* NFD_QC=No
    - see the color-coded table at https://icu.unicode.org/design/normalization/custom
    - check that ICU's own computation of derived normalization properties yields NFC_QC=Maybe and not hasCompBoundaryBefore; this would be a Maybe that does not directly combine-back
- prominently describe this situation
    - in UAX #15
    - on the Unicode 16.0 landing page
    - on the Unicode 16.0 beta page
    - in the Unicode 16.0 alpha PRI

If we avoid this issue for now and withdraw the relevant characters, then we should add a test in our tooling that detects this situation and alerts us for future proposed characters.

For the encoding rationale see page 5 of L2/22-043 “Proposal to Encode Kirat Rai script in the Universal Character Set”, and page 10 of L2/22-031 “Updated proposal to encode the Tulu-Tigalari script in Unicode”.

Draft UnicodeData.txt:

```
Unset
16D63;KIRAT RAI VOWEL SIGN AA;Lo;0;L;;;;;N;;;;;
16D67;KIRAT RAI VOWEL SIGN E;Lo;0;L;;;;;N;;;;;
16D68;KIRAT RAI VOWEL SIGN AI;Lo;0;L;16D67 16D67;;;;N;;;;;
16D69;KIRAT RAI VOWEL SIGN O;Lo;0;L;16D63 16D67;;;;N;;;;;
16D6A;KIRAT RAI VOWEL SIGN AU;Lo;0;L;16D69 16D67;;;;N;;;;;
```

Draft DerivedNormalizationProps.txt:

```
Unset
16D67         ; NFC_QC; M # Lo       KIRAT RAI VOWEL SIGN E
```

That is:

- AI=E+E
- O=AA+E
- AU=O+E
- AU=AA+E+E which is canonically equivalent with AA+AI
- AU+E=AA+E+E+E which is canonically equivalent with AA+AI+E and AA+E+AI

UAX #15 section 9 Detecting Normalization Forms:

- MAYBE: The code point can occur, subject to canonical ordering, but with constraints. In particular, the text may not be in the specified Normalization Form depending on the context in which the character occurs.

UAX #44 Table 16. Quick_Check Property Values:

- Characters that may occur in the respective normalization, depending on the context.

In the preliminary data, NFC_Quick_Check(U+16D68 AI)=Yes, which means that

- quickCheck_NFC(AA AI)=Yes which is wrong because toNFC(AA AI)=AU
- quickCheck_NFC(E AI)=Yes which is wrong because toNFC(E AI)=AI E

Similar for Tulu-Tigalari; draft UnicodeData.txt:

```
Unset
1138B;TULU-TIGALARI LETTER EE;Lo;0;L;;;;;N;;;;;
1138E;TULU-TIGALARI LETTER AI;Lo;0;L;1138B 113C2;;;;N;;;;;
11390;TULU-TIGALARI LETTER OO;Lo;0;L;;;;;N;;;;;
11391;TULU-TIGALARI LETTER AU;Lo;0;L;11390 113C9;;;;N;;;;;

113B8;TULU-TIGALARI VOWEL SIGN AA;Mc;0;L;;;;;N;;;;;
```

```
113C2;TULU-TIGALARI VOWEL SIGN EE;Mc;0;L;;;;;N;;;;;
113C5;TULU-TIGALARI VOWEL SIGN AI;Mc;0;L;113C2 113C2;;;;N;;;;;
113C7;TULU-TIGALARI VOWEL SIGN OO;Mc;0;L;113C2 113B8;;;;N;;;;;
113C8;TULU-TIGALARI VOWEL SIGN AU;Mc;0;L;113C2 113C9;;;;N;;;;;
113C9;TULU-TIGALARI AU LENGTH MARK;Mc;0;L;;;;;N;;;;;
```

Draft DerivedNormalizationProps.txt:

```
Unset
113B8         ; NFC_QC; M # Mc       TULU-TIGALARI VOWEL SIGN AA
113BB         ; NFC_QC; M # Mn       TULU-TIGALARI VOWEL SIGN U
113C2         ; NFC_QC; M # Mc       TULU-TIGALARI VOWEL SIGN EE
113C9         ; NFC_QC; M # Mc       TULU-TIGALARI AU LENGTH MARK
```

In other words:

- Letter AI = letter EE + vowel sign EE
- Vowel sign AI = vowel sign EE + vowel sign EE

Resulting NFC string normalization:

- Letter EE + vowel sign {AI, OO, AU} → letter AI + {vowel sign EE, vowel sign AA, AU length mark}
- Vowel sign EE + vowel sign {AI, OO, AU} → vowel sign AI + {vowel sign EE, vowel sign AA, AU length mark}

Therefore we need NFC_QC(vs{AI, OO, AU})=Maybe

Gurung Khema vowel signs:

- U=AA+AA
- UU=AA+length
- E=AA+I
- EE=length+I
- AI=AA+II
- O=U+I=AA+AA+I
- OO=UU+I=AA+length+i
- AU=U+II=AA+AA+II

Thus, overlaps of dm(U) with several other vowel signs, and overlap of dm(UU) with dm(EE) via the length mark.

6. Text Segmentation

6.1 UAX #14 LB28a -- confusing use of a literal value

From a discussion on the public unicode list

Recommended UTC actions

1. Action Item for Robin Leroy, PAG: In rule LB28a of UAX #14, replace the use of the literal ◌ with the character class [◌], and add a note clarifying that the class contains the single character U+25CC DOTTED CIRCLE. For Unicode Version 16.0. See L2/23-009 item 6.1.

Feedback

Daniel Bünzli:
I can’t figure out what the ◌ character classification represents in:
https://www.unicode.org/reports/tr14/proposed.html#LB28a

Robin Leroy:
Itself: U+25CC DOTTED CIRCLE.

Daniel:
Thanks.
I think it would be better if that was written \u{255C} as per regexp notation. Like that it’s highly ambiguous as to what it represents since in these rules a class C itself represent \p{lb=C} and some of the characters are distinguished syntax.
Also it would be nicer for certain implementations if that was somehow integrated as a character class in the rules like e.g. ZJW is.

Sławomir Osipiuk:
It's definitely confusing. At first glance it certainly appears to be some kind of special marker or syntax, not a simple literal character. It needs at least a note somewhere because this WILL cause confusion and this question will come up again elsewhere.

Asmus Freytag:
Correct, we don't have a notation for "literal" and we need one.

Discussion

PAG discussed and agreed that although this is a small matter, it’s important and necessary to fix for the next version.

6.2 UAX #14: LB9 is unclear about CM|ZWJ

Recommended UTC actions

1. No Action: PAG recommends no action. This feedback has been addressed editorially.

Feedback (verbatim)

Date/Time: Tue Nov 07 14:09:48 CST 2023
ReportID: ID20231107140948
Name: Joe Hildebrand
Report Type: Error Report
Opt Subject: UAX #14

Summary: LB9 is unclear that the CM|ZWJ character is treated as if it does not exist for the purpose of matching subsequent rules

LB9 currently states:

Unset
```
LB9 Do not break a combining character sequence; treat it as if it has the line
breaking class of the base character in all of the following rules. Treat ZWJ
as if it were CM.

Treat X (CM | ZWJ)* as if it were X.

where X is any line break class except BK, CR, LF, NL, SP, or ZW.

At any possible break opportunity between CM and a following character, CM
behaves as if it had the type of its base character. Note that despite the
summary title, this rule is not limited to standard combining character
sequences. For the purposes of line breaking, sequences containing most of the
control codes or layout control characters are treated like combining
sequences.
```

When combined with the new rule LB28a:

```
Unset
LB28a Do not break inside the orthographic syllables of Brahmic scripts.

AP × (AK | ◌ | AS)

(AK | ◌ | AS) × (VF | VI)

(AK | ◌ | AS) VI × (AK | ◌)

(AK | ◌ | AS) × (AK | ◌ | AS) VF
```

and the following test from line 10287 of https://www.unicode.org/Public/15.1.0/ucd/auxiliary/LineBreakTest.txt:

```
Unset
× 1B18 ÷ 1B27 × 1B44 × 200C × 1B2B × 1B38 ÷ 1B31 × 1B44 × 1B1D × 1B36 ÷ # ×
[0.3] BALINESE LETTER CA (AK) ÷ [999.0] BALINESE LETTER PA (AK) × [28.12]
BALINESE ADEG ADEG (VI) × [9.0] ZERO WIDTH NON-JOINER (CM1_CM) × [28.13]
BALINESE LETTER MA (AK) × [9.0] BALINESE VOWEL SIGN SUKU (CM1_CM) ÷ [999.0]
BALINESE LETTER SA SAPA (AK) × [28.12] BALINESE ADEG ADEG (VI) × [28.13]
BALINESE LETTER TA LATIK (AK) × [9.0] BALINESE VOWEL SIGN ULU (CM1_CM) ÷ [0.3]
```

it becomes clear that the 200C in the input (linebreak class CM, affected by LB9), should not just be treated as if it had the linebreak class VI, but should not be included at ALL when trying to match LB28a.

When the 200C is treated as VI, the sequence would read: AK VI VI AK, and would NOT match the third line of LB28.

When the 200C is ignored entirely, the sequence would read: AK VI AK, and WOULD match the third line of LB28, as the test states.

Both of these are potentially-valid readings of the current text in LB9. Before the addition of LB28a, there were no cases I can think of where the difference mattered.

In a future version of the spec, the language in LB9 could be clarified to make interoperable implementation easier.

6.3 Error in UAX #29 STerm definition

From Mark Davis, PAG

Recommended UTC actions

1. Consensus: In UAX #29, change the definition of SB=STerm by excluding SB=ATerm, in order to match the data file. For Unicode 16.0. See L2/24-009 item 6.3.
2. Action Item for Mark Davis, PAG: In UAX #29, change the definition of SB=STerm by excluding SB=ATerm. For Unicode 16.0. See L2/24-009 item 6.3.
3. Consensus: Give U+2024 ONE DOT LEADER the Sentence_Terminal property. For Unicode 16.0. See L2/24-009 item 6.3.
4. Action Item for Mark Davis, PAG: In PropList.txt, give U+2024 ONE DOT LEADER the Sentence_Terminal property. For Unicode 16.0. See L2/24-009 item 6.3.

Feedback

I happened to notice that STerm is defined incorrectly in UAX #29 Table 4. Sentence_Break Property Values, as

Sentence_Terminal = Yes

This is incorrect, as you see here:
https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5Cp%7BSentence_Terminal%7D%5Cp%7Bsb%3DATerm%7D%5Cp%7Bsb%3DSTerm%7D&g=sentence_terminal+sb

- Sentence_Terminal=No, Sentence_Break=ATerm, items: 1 // that is, U+2024 (.) ONE DOT LEADER
- Sentence_Terminal=Yes, Sentence_Break=ATerm, items: 3
- Sentence_Terminal=Yes, Sentence_Break=STerm, items: 153

1. We can fix the problem by adding one line:
- Sentence_Terminal=Yes
- and not ATerm
2. However, that still leaves Sentence_Terminal as slightly different than STerm + ATerm. It would be cleaner and less surprising in in behavior to also:
- change Sentence_Terminal by adding U+2024 ONE DOT LEADER, thus making it consistent with STerm+ATerm.

There is no good reason to go the other direction, to remove ONE DOT LEADER from ATerm:

- One Dot Leader is indistinguishable from Period
- We have other compatibility equivalents in each of Sentence_Terminal, STerm and ATerm
- Sentence break is widely used in implementations, whereas Sentence_Terminal is mostly a contributory property

6.4 KIRAT RAI VOWEL SIGN AU will be added to Grapheme_Cluster_Break=V

PRI #494

Recommended UTC actions

1. Action Item for Rick McGowan, PAG: Reply to Charlotte Buff pointing out the presence of two points in U+16D67..U+16D6A. See L2/24-009 item 6.4.

Feedback (verbatim)

Date/Time: Sun Jan 07 09:10:23 CST 2024
ReportID: ID20240107091023
Name: Charlotte Buff
Report Type: Public Review Issue
Opt Subject: 494

Currently it is stated in table 2 that U+16D6A KIRAT RAI VOWEL SIGN AU (together with two other characters) will be added to Grapheme_Cluster_Break=V. However, instead of AU it should be U+16D69 KIRAT RAI VOWEL SIGN O because AU decomposes into O+E, while AU itself does not appear in the decomposition of any other character.

Background information / discussion

PAG members reviewed and noted that the claim was incorrect.

6.5 Legacy grapheme clusters are inconsistent with canonical equivalence

From Robin Leroy, PAG, spotted in discussion with Steve Canon

Recommended UTC actions

1. Consensus: Assign the Other_Grapheme_Extend property to eighteen characters (fourteen spacing viramas U+1715, U+1734, U+1B44, U+1BAA, U+1BF2, U+1BF3, U+A953, U+A9C0, U+111C0, U+11235, U+1134D, U+116B6, U+1193D, and U+11F41, two Vietnamese alternate reading marks U+16FF0 and U+16FF1, and two musical symbols U+1D166 and U+1D16D) in order to make all non-starters GCB=Extend. For Unicode Version 16.0. See L2/24-009 item 6.5.
2. Action Item for Robin Leroy, PAG: In PropList.txt, assign the Other_Grapheme_Extend property to eighteen pre-existing characters (U+1715, U+1734, U+1B44, U+1BAA, U+1BF2, U+1BF3, U+A953, U+A9C0, U+111C0, U+11235, U+1134D, U+116B6, U+1193D, U+11F41, U+16FF0, U+16FF1, U+1D166, U+1D16D), as well as to the Tulu-Tigalari vowel signs and looped viramas, and update derived properties. For Unicode Version 16.0. See L2/24-009 item 6.5.

Summary

Consider the strings
S = <U+0300 COMBINING GRAVE ACCENT, U+1D166 MUSICAL SYMBOL COMBINING SPRECHGESANG STEM> and
S′ = <U+1D166 MUSICAL SYMBOL COMBINING SPRECHGESANG STEM, U+0300 COMBINING GRAVE ACCENT>.
These strings are canonically equivalent (the characters have different nonzero CCC).
U+0300 is GCB=Extend, U+1D166 is GCB=Spacing_Mark.

Apply the rules from https://www.unicode.org/reports/tr29/#Grapheme_Cluster_Boundary_Rules for Legacy Grapheme Clusters.

GB9 applies to S′, but not to S. S is two LGCs, S′ is one.

The issue here is that in the derivation of GCB=Extend, the « few General_Category = Spacing_Mark needed for canonical equivalence » are missing those:
https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5CP%7Bccc%3D0%7D-%5Cp%7Bgcb%3DExtend%7D&g=gc&i=ccc%2Cgcb.

Character	Code Point	Name
□	U+1715	TAGALOG SIGN PAMUDPOD
◌᜴	U+1734	HANUNOO SIGN PAMUDPOD
◌᭄	U+1B44	BALINESE ADEG ADEG
᮪	U+1BAA	SUNDANESE SIGN PAMAAEH
◌᯲	U+1BF2	BATAK PANGOLAT
◌᯳	U+1BF3	BATAK PANONGONAN
꥓	U+A953	REJANG VIRAMA
◌꧀	U+A9C0	JAVANESE PANGKON
□	U+111C0	SHARADA SIGN VIRAMA
□	U+11235	KHOJKI SIGN VIRAMA
□	U+1134D	GRANTHA SIGN VIRAMA
□	U+116B6	TAKRI SIGN VIRAMA
□	U+1193D	DIVES AKURU SIGN HALANTA
□	U+11F41	KAWI SIGN KILLER
□	U+16FF0	VIETNAMESE ALTERNATE READING MARK CA

□	U+16FF1	VIETNAMESE ALTERNATE READING MARK NHAY
□	U+1D166	MUSICAL SYMBOL COMBINING SPRECHGESANG STEM
□	U+1D16D	MUSICAL SYMBOL COMBINING AUGMENTATION DOT

We should have an invariant that GCB=Extend contains all nonstarters.

Note that this does not affect users of extended grapheme clusters (which is probably most users at this point), since those are GCB=SM, to which the same thing happens in GB9a.

We also need to make the Tulu-Tigalari vowel signs GCB=Extend (for the same reason we needed to make the Kirat Rai vowel signs GCB=V), and to make the Tulu-Tigalari looped virama GCB=Extend (it falls in the same category as the eighteen above), but those are new in 16.0 so no need for a decision here.

7. IDNA

7.1 U+19DA inconsistent IDNA2008 status

From personal communication by ICANN experts

Recommended UTC actions

1. Note: The idna2008derived files for versions 7.0 to 15.1 (inclusive) had U+19DA as PVALID which was in conflict with its IdnaMappingTable.txt IDNA2008 Status of XV8. The idna2008derived files have been corrected with 19DA as DISALLOWED.

Summary

The IDNA2008 status of U+19DA in idna2008derived files (https://unicode.org/Public/idna/idna2008derived/Idna2008-6.1.0.txt) is PVALID while in the IANA registry of IDNA parameters https://www.iana.org/assignments/idna-tables-12.0.0/idna-tables-12.0.0.xhtml it is DISALLOWED. See also discussion in RFC 9233 https://www.rfc-editor.org/rfc/rfc9233.html .

The `Status` in the IDNA Mapping Table files is `valid`, but the `IDNA2008 Status` is `XV8`.

- https://unicode.org/Public/idna/15.1.0/IdnaMappingTable.txt
- UTS #46 Table 2b. Data File Fields

The goal is to have the derived file match the IANA file by marking exceptions as necessary to match discrepancies from values derived algorithmically.

Also, the mapping file status should agree after applying `IDNA2008 Status` values of `NV8` and `XV8`.

8. Security

8.1 UTS #39 typos

From Markus Scherer, PAG

Recommended UTC actions

1. No Action: PAG recommends no action: this feedback has been addressed editorially.

Feedback

https://www.unicode.org/reports/tr39/#Data_Files

The format for IdentifierStatus.txt follows the normal conventions for UCD data files, and is described in the header of that file. All characters not listed in the file default to Identifier_Type=Restricted.

Typo: It's Identifier_Status=Restricted, not Identifier_Type.

The format for IdentifierType.txt ... This new convention allows the values to be used for more nuanced filtering. For example, if an implementation wants to allow an Exclusion script, it could still exclude Obsolete and Deprecated characters in that script.

Typo/misleading: Deprecated is one of the solo types, so it's misleading to put it in a context of possible combination with Exclusion. I suggest replacing it here with another type, like Not_XID.

All characters not listed in the file default to Identifier_Type=Recommended.

No. As the file says:

```
Unset
# All code points not explicitly listed for Identifier_Type
# have the value Not_Character.

# @missing: 0000..10FFFF; Not_Character
```

Addendum 1:

https://www.unicode.org/reports/tr39/#Version_Correspondance

The date for revision-03 is earlier than that for revision-02. The actual files in https://www.unicode.org/Public/security/revision-03/ are from 2010-04-12.

Addendum 2:

https://www.unicode.org/reports/tr39/#General_Security_Profile

An implementation following the General Security Profile does not permit any characters in \p{Identifier_Status=Restricted}, unless it documents the additional characters that it does allow. Such documentation can specify characters via properties, such as \p{Identifier_Status=Technical},

This needs to be \p{Identifier_Type=Technical}

or by explicit lists, or by combinations of these. Implementations may also specify that fewer characters are allowed than implied by \p{Identifier_Status=Restricted}; for example, they can restrict characters to only those permitted by [IDNA2008].

This is not wrong, but a bit confusing. It would read better if it said “... fewer characters are allowed than implied by \p{Identifier_Status=Allowed}; for example, they can allow only characters permitted by [IDNA2008].”