

# UTC #164 properties feedback & recommendations

Markus Scherer / Unicode properties & algorithms group, 2020-jul-21

## Properties & algorithms

We are a group of Unicode contributors who take an interest in properties and algorithms.

We look at relevant feedback reports and documents that Unicode receives, do some research, and submit UTC documents with recommendations as input to UTC meetings.

This group started with the UCD file and production tool maintainers, with Markus Scherer as the chair. Several UTC participants have requested and received invitations to join. So far, discussion has been via email and shared documents. We may use video meetings if needed.

## Participants

The following people have contributed to this document:

Markus Scherer (chair), Mark Davis, Christopher Chapman, Ken Lunde, Peter Constable, Asmus Freytag, Andy Heninger, Koji Ishii, Ken Whistler

## Public feedback

Feedback received via the Unicode reporting form, see L2/20-174 "Comments on Public Review Issues (April 20 - July 16, 2020)".

### F1: IdentifierType of Ainu Katakana characters

#### *Feedback (verbatim)*

Date/Time: Tue May 12 20:46:39 CDT 2020

Name: Manish Goregaokar

Report Type: Error Report

Opt Subject: IdentifierType of Ainu Katakana characters

In IdentifierStatus.txt:

```
31F0..31FF ; Technical # 3.2 [16] KATAKANA LETTER SMALL KU..KATAKANA LETTER SMALL RO
```

These are from the Katakana Phonetic Extensions block; which exists for writing the Ainu language. Ainu is apparently both written using the Latin and Katakana scripts, using these extensions.

According to UTS 39 Table 1[1], "Technical" is "Specialized usage: technical, liturgical, etc.", which doesn't seem to fit with code points that are actively used in a primary script for a language.

Should we be changing this to Recommended?

[1]: [https://www.unicode.org/reports/tr39/#Identifier\\_Status\\_and\\_Type](https://www.unicode.org/reports/tr39/#Identifier_Status_and_Type)

### *Recommended UTC actions*

1. Approve: Change the Identifier\_Type of Ainu Katakana characters 31F0..31FF from Technical to Obsolete.
2. AI: Change the Identifier\_Type of Ainu Katakana characters 31F0..31FF from Technical to Obsolete.

### *Background information / discussion*

UTS #39 describes the relevant Identifier\_Type values like this:

- Obsolete: Characters that are no longer in modern use.
- Technical: Specialized usage: technical, liturgical, etc.
- Uncommon\_Use: Characters whose status is uncertain, or that are not commonly used in modern text.
- Limited\_Use: Characters from scripts that are in limited use: with Script\_Extensions values containing a script in Table 7, Limited Use Scripts in [UAX31], and no explicit script from Table 5, Recommended Scripts.

[https://www.unicode.org/reports/tr31/#Table\\_Limited\\_Use\\_Scripts](https://www.unicode.org/reports/tr31/#Table_Limited_Use_Scripts)

- “Modern scripts that are in more limited use”
- “Recommended Scripts are generally recommended for use in identifiers. These are in widespread modern customary use, or are regional scripts in modern customary use by large communities.”

[https://en.wikipedia.org/wiki/Ainu\\_language](https://en.wikipedia.org/wiki/Ainu_language) “Only the Hokkaido variant survives ... Hokkaido Ainu is moribund, though attempts are being made to revive it.”

The Ainu letters fit the criteria for Obsolete but not for Technical nor Limited\_Use.

Note: The Identifier\_Status is for use in identifiers, not general text.

Note: We used to distinguish a particular class of Limited\_Use as “Aspirational”. We dropped that distinction when it became clear that there were no objective, measurable criteria for making that distinction.

## F2: uppercase of U+0587 ARMENIAN SMALL LIGATURE ECH YIWN

### *Feedback (verbatim)*

Date/Time: Thu Apr 23 13:30:44 CDT 2020

Name: Markus W Scherer

Report Type: Error Report

Opt Subject: uppercase of U+0587 ARMENIAN SMALL LIGATURE ECH YIWN

Maybe for the Script Ad Hoc?

We have received a bug report claiming that the uppercase form of U+0587 լ is wrong.

SpecialCasing.txt has

```
# <code>; <lower> ; <title> ; <upper> ; (<condition_list> ;)? # <comment>
0587; 0587; 0535 0582; 0535 0552; # ARMENIAN SMALL LIGATURE ECH YIWN
```

This means that the ligature small ech-yiwn uppercases to ԼԻ=capital ech+yiwn=0535+0552.

The report says that it should uppercase to ԼՎ=capital ech+vew=0535+054E.

I have asked for an authoritative reference and will report when I receive something.

In the meantime, I found this:

[https://en.wikipedia.org/wiki/Armenian\\_alphabet#endnote\\_h](https://en.wikipedia.org/wiki/Armenian_alphabet#endnote_h)

“The ligature լ has no majuscule form; when capitalized it is written as two letters ԼԻ (classical) or ԼՎ (reformed).”

Can someone confirm this?

If true, should we change SpecialCasing.txt to use the "reformed" uppercasing?  
Should implementers (e.g., ICU) offer both versions? Under what conditions?

Please advise.

### *Recommended UTC actions*

1. Approve: Change SpecialCasing.txt for U+0587 ligature ech-yiwn to unconditionally titlecase and uppercase to ech+vew. (Option 2 in [L2/20-175 item F2, Discussion](#))
2. AI: Change SpecialCasing.txt for U+0587 ligature ech-yiwn to unconditionally titlecase and uppercase to ech+vew: title=0535 057E, upper=0535 054E
3. AI: Add an annotation for U+0587 noting that the titlecase and uppercase mappings disagree with the Decomposition\_Mapping.

### *Expert feedback*

[L2/20-143](#) “Uppercase of U+0587 ARMENIAN SMALL LIGATURE ECH YIWN” via Deborah Anderson

### *Background information*

Ligature ech-yiwn has a Decomposition\_Mapping to ech+yiwn which cannot be changed: <compat> 0565 0582

UnicodeData.txt:

```
0535;ARMENIAN CAPITAL LETTER ECH;Lu;0;L;;;;N;;;;0565;
054E;ARMENIAN CAPITAL LETTER VEW;Lu;0;L;;;;N;;;;057E;
```

0552;ARMENIAN CAPITAL LETTER YIWN;Lu;0;L;;;;;N;;;;;0582;  
0565;ARMENIAN SMALL LETTER ECH;Ll;0;L;;;;;N;;;;;0535;;0535  
057E;ARMENIAN SMALL LETTER VEW;Ll;0;L;;;;;N;;;;;054E;;054E  
0582;ARMENIAN SMALL LETTER YIWN;Ll;0;L;;;;;N;;;;;0552;;0552  
0587;ARMENIAN SMALL LIGATURE ECH YIWN;Ll;0;L;<compat> 0565 0582;;;;;N;;;;;

SpecialCasing.txt:

0587; 0587; 0535 0582; 0535 0552; # ARMENIAN SMALL LIGATURE ECH YIWN

There are two language codes/subtags for Armenian:

Type: language

Subtag: hy

Description: Armenian

Added: 2005-10-16

Suppress-Script: Armn

Comments: see also hyw

Type: language

Subtag: hyw

Description: Western Armenian

Added: 2018-03-08

Comments: see also hy

[https://en.wikipedia.org/wiki/Western\\_Armenian](https://en.wikipedia.org/wiki/Western_Armenian)

## *Discussion*

It appears that in Western Armenian (outside of what used to be USSR territories), the uppercase form is ech+yiwn but the lowercase ligature is not in common use; while in Eastern Armenian (reformed in Soviet times) the ligature continues to be used, the letter yiwn has mostly fallen out of use, and the ligature is understood as standing for ech+vev with a corresponding uppercase form.

Options:

1. Do nothing.
  - a. Pro: Respects the original identity of the ech-yiwn ligature.
  - b. Con: Does not fit usage where the ligature is in common use.
2. **Change SpecialCasing.txt to unconditionally map ech-yiwn to uppercase and titlecase ech+vev.**
  - a. **Pro: Simple change to fit common usage.**
  - b. **Pro: Does not impact most usage in Western Armenian.**
  - c. **Con: Does not fit rare usage of the ligature in Western Armenian.**
  - d. **Con: The default case mapping would disagree with the Decomposition\_Mapping.**
3. Change SpecialCasing.txt to conditionally map ech-yiwn to uppercase and titlecase ech+vev but only for language hy.
  - a. Pro: Limits change to Eastern Armenian.
  - b. Con: Conditional case mappings are not generally data-driven, thus tend to require library code changes.

- c. Con: Does not fit common behavior “out of the box” as a default mapping, requires the language to be specified for the case mapping function.
4. Change SpecialCasing.txt to **un**conditionally map ech-yiwn to ech+vev, and also conditionally map ech-yiwn to ech+yiwn but only for language hyw.
  - a. Pro: Fits common behavior “out of the box”.
  - b. Con: Conditional case mappings are not generally data-driven, thus tend to require library code changes.
  - c. Con: The default case mapping would disagree with the Decomposition\_Mapping.

## F3: IDNA test case error

### *Feedback (verbatim)*

Date/Time: Wed May 20 01:31:04 CDT 2020

Name: Trevor

Report Type: Error Report

Opt Subject: IDNA test case error

Hello,

I believe I have found 2 tests in <https://www.unicode.org/Public/idna/13.0.0/IdnaTestV2.txt> whose expected result are not possible to represent when using the ToASCII operation with Transitional\_Processing = true, CheckJoiners = false, and VerifyDnsLength = false.

This relates to tests whose source string is U+200C or U+200D. The U+200C and U+200D get mapped to an empty string due to the use of Transitional Processing and as a result, the expected output is an empty string. However, it is not possible to represent an empty string as the expected output for toAsciiT because an empty string means that toAsciiT "adopts" toAsciiN's value, which in this case is either 'xn--1ug' or 'xn--0ug'.

Tests in question (source string escaped for readability):

```
\u200D; ; [C2]; xn--1ug; ; ; [A4_2] #
```

```
\u200C; ; [C1]; xn--0ug; ; ; [A4_2] #
```

### *Recommended UTC actions*

1. AI for Markus: Revise the syntax of IdnaTestV2.txt, adding the ability to represent an explicitly empty input or output string; and generate a new version of the data accordingly.
2. AI for Markus: Revise the generation of IdnaTestV2.txt, escaping certain characters like ZWJ and ZWNJ to make them visible.
3. AI for Markus: In IdnaTestV2.txt, fix the URL `.../tr46/proposed.html#Processing` to point to the latest version of UTS #46.
4. AI for Markus & the ed committee: Add a migration note to UTS #46 about incompatible changes to the test file format and contents.

## *Background information / discussion*

The test file format uses an abbreviation mechanism where several expected-output fields are documented with “A blank value means the same as the toASCIIINStatus value.” or similar. There is no syntax for an explicitly empty string.

The file format uses UTF-8, but “characters may be escaped using the \uXXXX or \x{XXXX} convention where they could otherwise have a confusing display. These characters include control codes and combining marks.” In the Unicode 13 version, no such escape sequences are used.

Markus: I suggest we use a pair of ASCII double quotes "" to indicate an explicitly empty string, and escape certain control codes as documented (as well as leading & trailing spaces, if there are any).

## F4: UTS#46 tests and URL delimiters

### *Feedback (verbatim)*

Date/Time: Fri May 29 16:49:03 CDT 2020

Name: Trevor

Report Type: Error Report

Opt Subject: UTS#46 tests and URL delimiters

Hello,

There are a number of tests[1] that contain labels that have a U+003F "?" question mark code point where the test expects the label containing the U+003F "?" question mark to remain in its Unicode form when performing the toASCII[2] operation on the domain. As far as I can tell, there is nothing in the UTS#46 specification that prevents the label from being converted into an ASCII label. The toASCII[2] operation converts all labels to ASCII unless punycode returns an error. Going through the Punycode spec, Punycode's encode[3] algorithm does not reject U+003F "?" question marks and as a result labels containing U+003F "?" question marks get converted to ASCII contrary to the test expectations.

I presume that that tests are trying to say that any label containing common URL delimiters such as ":/@.?!#[]" shouldn't be converted to an ASCII label, but I'm not really sure what the expected results are supposed to be. I suppose you could add a check for such common URL delimiters and skip punycode encoding labels that contain one assuming the test expectations are correct.

[1] <https://www.unicode.org/Public/idna/13.0.0/IdnaTestV2.txt>

[2] <https://www.unicode.org/reports/tr46/#ToASCII>

[3] <https://tools.ietf.org/html/rfc3492#section-6.3>

- Trevor

## *Recommended UTC actions*

1. AI for Markus: Re L2/20-175 item F4: Investigate revising the generation of IdnaTestV2.txt, writing expected output strings computed with UseSTD3ASCIIRules=false but retaining the error flags for when UseSTD3ASCIIRules=true is set.

## *Background information / discussion*

<https://www.unicode.org/Public/idna/13.0.0/IdnaMappingTable.txt>

U+003F “?” is among

003A..0040 ; disallowed\_STD3\_valid # 1.1 COLON..COMMERCIAL AT

IdnaTestV2.txt contains 129 test cases where the source string contains a question mark. The toASCII output strings are written using UseSTD3ASCIIRules=true.

The test instructions include this:

If the implementation converts illegal code points into U+FFFD [...] then the string comparisons need to account for that by treating U+FFFD in the actual value as a wildcard when comparing to the expected value in the test file.

However, Trevor’s situation is the reverse: He processes with UseSTD3ASCIIRules=false and gets a literal question mark, while the test file’s expected output string contains U+FFFD instead.

## **F5: UAX #50 orientation of Bopomofo tone marks**

### *Feedback (verbatim)*

Date/Time: Fri Apr 24 17:59:22 CDT 2020

Name: Elika J. Etemad

Report Type: Error Report

Opt Subject: UTR50 orientation of Bopomofo tone marks

Hello UTC,

I'm writing regarding the four tone marks used in bopomofo:

02C9 MODIFIER LETTER MACRON  
02CA MODIFIER LETTER ACUTE ACCENT  
02C7 CARON  
02CB MODIFIER LETTER GRAVE ACCENT  
02D9 DOT ABOVE

These are currently registered as R in UTR50, but they should probably be adjusted to U, consistent with the rest of the Bopomofo letters. (They're a bit more widely used than just within Bopomofo, but UTR50

is primarily targetted at CJK context, and within this context these modifier letters are much more likely to be used as Bopomofo tone marks than otherwise.)

See discussion thread at <https://lists.w3.org/Archives/Public/www-style/2015Aug/0315.html> for more context.

Thanks~  
~fantasai

### *Recommended UTC actions*

1. Approve: Change the Vertical\_Orientation property of 02C7, 02C9, 02CA, 02CB, 02D9 from R to U.
2. AI for Ken Lunde: Change the Vertical\_Orientation property of 02C7, 02C9, 02CA, 02CB, 02D9 from R to U.

### *Copy of the referenced W3C email*

On 08/26/2015 12:05 AM, Xidorn Quan wrote:  
> On Wed, Aug 26, 2015 at 12:15 AM, fantasai  
> <fantasai.lists@inkedblade.net> wrote:  
>> On 08/25/2015 02:59 PM, Xidorn Quan wrote:  
>>> I propose that we should add "text-orientation: upright" to the  
>>> "rt:lang(zh-TW)" rule.  
>>>  
>>> The reason is that, all the tone marks in bopomofo (U+02CA, U+02C7,  
>>> U+02CB, U+02D9) have Vertical\_Orientation property "R" while the  
>>> bopomofo characters are all "U". It means, without explicitly setting  
>>> text-orientation to upright, the text run would break between them,  
>>> which makes it impossible to use font feature to place the tone mark  
>>> properly.  
>>  
>> I think it would make more sense to have Unicode update UTR50 to make  
>> these characters upright in mixed-orientation text (or otherwise tailor  
>> it within CSS as a whole). It's not just a problem with ruby.  
>  
> Well, that could be tricky, because those characters might also be  
> used with latin scripts. I'm not sure anyway.

Latin mostly uses the combining-mark form, rather than the modifier letter form. Also for mixed vertical text, we're biased mostly towards CJK usage (which bopomofo counts as). So I think we're okay.

Still would like Unicode signoff, too, of course.

~fantasai

## Background information / discussion

<https://www.unicode.org/reports/tr50/#vo>

- U: Characters which are displayed upright, with the same orientation that appears in the code charts.
- R: Characters which are displayed sideways, rotated 90 degrees clockwise compared to the code charts.
- Tu: Characters which are not just upright or sideways, but generally require a different glyph than in the code charts when used in vertical texts. In addition, as a fallback, the character can be displayed with the code chart glyph upright.
- Tr: Same as Tu except that, as a fallback, the character can be displayed with the code chart glyph rotated 90 degrees clockwise.

<https://www.unicode.org/Public/UCD/latest/ucd/VerticalOrientation.txt>

```
02B0..02C1      ; R # Lm    [18] MODIFIER LETTER SMALL H..
                MODIFIER LETTER REVERSED GLOTTAL STOP
02C2..02C5      ; R # Sk    [4] MODIFIER LETTER LEFT ARROWHEAD..
                MODIFIER LETTER DOWN ARROWHEAD
02C6..02D1      ; R # Lm    [12] MODIFIER LETTER CIRCUMFLEX ACCENT..
                MODIFIER LETTER HALF TRIANGULAR COLON
02D2..02DF      ; R # Sk    [14] MODIFIER LETTER CENTRED RIGHT HALF RING..
                MODIFIER LETTER CROSS ACCENT
02E0..02E4      ; R # Lm    [5] MODIFIER LETTER SMALL GAMMA..
                MODIFIER LETTER SMALL REVERSED GLOTTAL STOP
02E5..02E9      ; R # Sk    [5] MODIFIER LETTER EXTRA-HIGH TONE BAR..
                MODIFIER LETTER EXTRA-LOW TONE BAR
02EA..02EB      ; U # Sk    [2] MODIFIER LETTER YIN DEPARTING TONE MARK..
MODIFIER LETTER YANG DEPARTING TONE MARK
02EC            ; R # Lm    MODIFIER LETTER VOICING
02ED            ; R # Sk    MODIFIER LETTER UNASPIRATED
02EE            ; R # Lm    MODIFIER LETTER DOUBLE APOSTROPHE
02EF..02FF      ; R # Sk    [17] MODIFIER LETTER LOW DOWN ARROWHEAD..
                MODIFIER LETTER LOW LEFT ARROW

3105..3126      ; U # Lo    [34] BOPOMOFO LETTER B..BOPOMOFO LETTER ER
3127            ; Tu # Lo    BOPOMOFO LETTER I
3128..312F      ; U # Lo    [8] BOPOMOFO LETTER U..BOPOMOFO LETTER NN
31A0..31BF      ; U # Lo    [32] BOPOMOFO LETTER BU..BOPOMOFO LETTER AH
```

**Ken Lunde writes:** I agree with Erika that these should be U in UAX #50, which matches U+02EA and U+02EB that are also used with Bopomofo. In other words, in a vertical context, their use with Bopomofo outweighs other vertical uses, which should be somewhere between zero and none.

## F6: Vertical Text in UAX9 Mostly Irrelevant

### *Feedback (verbatim)*

Date/Time: Sat May 30 19:34:01 CDT 2020

Name: Erika J. Etemad

Report Type: Error Report

Opt Subject: Vertical Text in UAX9 Mostly Irrelevant

The rules in UAX9 6.2 Vertical Text [http://unicode.org/reports/tr9/#Vertical\\_Text](http://unicode.org/reports/tr9/#Vertical_Text) are presented as if this is what implementations are expected to do, but actually, most of them don't. RTL text is rendered bottom-to-top instead. The section should be removed, or rewritten to be an example of something that *could* be done with UAX9's algorithms (but isn't necessarily).

### *Recommended UTC actions*

1. Authorize a proposed update of UAX #9 for Unicode 14.0.
2. AI for Ken Whistler and the ed committee: Start a proposed update of UAX #9 for Unicode 14.0.
3. AI for Ken Whistler and the ed committee: Re L2/20-175 item F6: Update UAX #9 section 6.2 Vertical Text according to feedback from Elika Etemad and incorporating a suggested revision from Koji Ishii.

### *Background information / discussion*

From UAX #9:

#### 6.2 Vertical Text

In the case of vertical line orientation, the Bidirectional Algorithm is still used to determine the levels of the text. However, these levels are not used to reorder the text, because the characters are usually ordered uniformly from top to bottom. Instead, the levels are used to determine the rotation of the text. Sometimes vertical lines follow a vertical baseline in which each character is oriented as normal (with no rotation), with characters ordered from top to bottom whether they are Hebrew, numbers, or Latin. When setting text using the Arabic script in vertical lines, it is more common to employ a horizontal baseline that is rotated by 90° counterclockwise so that the characters are ordered from top to bottom. Latin text and numbers may be rotated 90° clockwise so that the characters are also ordered from top to bottom.

The Bidirectional Algorithm is used when some characters are ordered from bottom to top. For example, this happens with a mixture of Arabic and Latin glyphs when all the glyphs are rotated uniformly 90° clockwise. The Unicode Standard does not specify whether text is presented horizontally or vertically, or whether text is rotated. That is left up to higher-level protocols.

**Koji Ishii:** I think the current text is correct, leaving up to higher-level protocols, but it's true that one may argue that it is recommending one of the options which is not implemented in major browsers. I'm fine with any of these options:

1. Remove. This is easiest for us. Unlike when this was written, CSS Writing Modes can provide one possible implementation in detail.
2. Stay in UAX#9, with some modifications.
3. Move to UAX#50 with some modifications.

Here's my try to modify this section:

In the case of vertical line orientation, there are multiple ways to represent bidirectional text. Some methods use the Bidirectional Algorithm, some don't. The Unicode Standard does not specify whether text is presented horizontally or vertically, or whether text is rotated. It is left up to higher-level protocols. For example, one of the common approaches is to rotate all the glyphs uniformly 90° clockwise. The Bidirectional Algorithm is used with this method. While some characters are ordered from bottom to top, this method can represent a mixture of Arabic and Latin glyphs in the same way as horizontal line orientation.

Another possible approach is to render the text uniformly from top to bottom. This method has multiple variations to determine the orientation of characters. One of them uses the Bidirectional Algorithm to determine the level of the text, but these levels are not used to reorder the text. Instead, the levels are used to determine the rotation of the text. Sometimes vertical lines follow a vertical baseline in which each character is oriented as normal (with no rotation), with characters ordered from top to bottom whether they are Hebrew, numbers, or Latin. When setting text using the Arabic script in vertical lines, it is more common to employ a horizontal baseline that is rotated by 90° counterclockwise so that the characters are ordered from top to bottom. Latin text and numbers may be rotated 90° clockwise so that the characters are also ordered from top to bottom.

## F7: UAX14 quotation marks vs ID

### *Feedback (verbatim)*

Date/Time: Mon Jun 22 16:19:50 CDT 2020

Name: fantasai

Report Type: Error Report

Opt Subject: UAX14 quotation marks vs ID

The UAX14 rules concerning QU are too strict, and don't work for Chinese by default, because they rely on spaces to be a reasonable default. This can probably be solved by allowing breaks between ID + Pi and between Pf + ID. See <https://github.com/w3c/clreq/issues/245> for more info.

### *Recommended UTC actions*

1. AI for the properties & algorithms group: Discuss L2/20-175 item F7 (line break quotation marks vs. ID) further and make a recommendation in time for UTC #165.

### *Background information / discussion*

**Mark:** I recommend putting out a PRI for this (and maybe other issues) to get feedback from a broader community.

**Andy:** I don't know much about Chinese, so I'll take the description of the problem on faith.

Splitting the character class QU into Pf and Pi would carry a cost in data size.

The change would be to LB-19, currently  
× QU

QU ×  
Something like  
[<sup>^</sup>ID] × Pi  
Pf × [<sup>^</sup>ID]

Which raises the question of whether any other character categories beyond ID would want to have this behavior.

And what should be done with quotes that are neither Pf or Pi, like ASCII quotes?

In any event, I don't think anything should go into UAX-14 without first being implemented and released in ICU, to see what unanticipated complications turn up. Maybe it could be put out as a CLDR tailoring.

**Koji Ishii:** Using ID doesn't solve the case presented in <https://github.com/unicode-org/icu/pull/223#issuecomment-435642278> as Andy pointed out above.

ASCII quotes are fine, they're not full-width in Chinese/Japanese and that authors will put spaces before or after appropriately. The problem of U+201C/201D/etc. is that they're full-width in C/J fonts, and they have embedded spacing in their glyphs, similar to full-width parenthesis (see [http://unicode.org/reports/tr50/#table\\_2](http://unicode.org/reports/tr50/#table_2) for example glyphs of full-width parenthesis,) so we can't expect space characters to exist.

Given Andy's comment on the data size, maybe similar approach as LB30 can solve?

## F8: ARABIC DATE SEPARATOR class error

### *Feedback (verbatim)*

Date/Time: Tue Jun 2 06:23:29 CDT 2020  
Name: Bahman Eslami  
Report Type: Error Report  
Opt Subject: ARABIC DATE SEPARATOR class error

Hello,

The error is that the character ARABIC DATE SEPARATOR is classified as Bidi Category "AL" which would imply strong right-to-left direction. This makes it's impossible to apply kerning between Arabic script numbers and ADS. Please take a look at the following issue on github:

<https://github.com/googlefonts/ufo2ft/issues/384>

I think bi-directional type of the ADS should be LTR or Neutral.

Thanks,  
Bahman

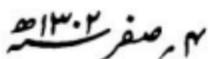
## Recommended UTC actions

We recommend not to make any changes.

## Background information / discussion

**Mark:** I am very leery of this, since it changes the bidi ordering. That incompatibility would be far more destructive than lack of kerning. (Also, it sounds like there is a technical limitation imposed by OT, not intrinsic; is there no way to relax that?)

**Peter Constable:** U+060D ARABIC DATE SEPARATOR (ADS) was proposed by Jonathan Kew in [L2/01-425](#) and [L2/01-426](#), based on usage in Pakistan for Urdu, Balochi and other languages. Per Figure 1 in L2/01-426, “In Pakistan Baluchistan... [d]ates are written from right to left: the numeral denoting the day of the month is followed by [ADS], followed by the name of the month, and finally the year.” The following is an example (from Figure 1 in L2/01-426):

Handwritten Urdu text: ۲ صفر ۱۳۰۲ھ (2 Safar 1302 AH)

The “Bidi Committee” that existed at that time reviewed this proposal and prepared recommendations to UTC in [L2/02-061](#); those recommendations included assigning Bidi Category = AL. Based on the example, AL isn’t obviously unreasonable. For comparison, there are other Arabic punctuation characters that are AL (061B, 061E, 061F, 066D, 06D4), though some are not (e.g., 0609: ET, 060C: CS). The rationale given in the feedback is that it “makes [it] impossible to apply kerning”. That seems to be a limitation either in certain rendering implementations, font formats or font tooling: in principle, there’s no reason why character-level properties should hinder ability to kern visually-adjacent glyphs.

## F9: Missing Indic shaping properties for U+0300 and U+0301

### Feedback (verbatim)

Date/Time: Fri Jun 26 19:06:23 CDT 2020

Name: Norbert Lindenberg

Report Type: Error Report

Opt Subject: Missing Indic shaping properties for U+0300 and U+0301

The Unicode Standard 13.0, page 466, recommends the characters U+0300 combining grave accent and U+0301 combining acute accent for use with the Devanagari script. However, these characters do not have Indic syllabic categories defined for them, so it’s not clear how they would be used and where they would fit into Devanagari syllables.

## Recommended UTC actions

1. AI for the ed committee: Re L2/20-175 item F9: In standard section 12.1 Devanagari, clarify use of 0953/0954/0300/0301 in Latin transliteration vs. in Devanagari text, and clarify that none of these 4 combining marks should be used in Devanagari text.

## *Background information / discussion*

Standard text referenced in the feedback ([chapter 12](#) “South and Central Asia-I”, section “12.1 Devanagari”)

U+0953 DEVANAGARI GRAVE ACCENT and U+0954 DEVANAGARI ACUTE ACCENT were originally encoded for Latin transliteration of Sanskrit text. However, such use is now discouraged, in favor of the generic combining marks, U+0300 COMBINING GRAVE ACCENT and U+0301 COMBINING ACUTE ACCENT. U+0953 and U+0954 should not be used with the Devanagari script; they have no Indic shaping properties.

The standard recommends using 0300/0301 “for *Latin* transliteration of Sanskrit text”. We need not define *Indic* shaping properties for them.

Markus pointed this out to Norbert and asked him to elaborate. He responded:

I may have overlooked that specific subcontext within the context of a section on the Devanagari script and on combining marks within that script. If none of the four characters is intended for use with the Devanagari script, then that paragraph should be moved into a separate section “Deprecated characters” at the end of section 12.1, and the last sentence should start with “None of the four characters should be used with the Devanagari script.” Indic shaping properties indeed are not needed in that case.

Peter Constable believes that the 0953/0954 marks were in ISCII for Latin transliteration and came into Unicode that way, but without clarity on their purpose. The text in the standard is intended to address this lack of clarity.

## F10: Missing Indic shaping properties for Devanagari and Vedic characters

### *Feedback (verbatim)*

Date/Time: Tue Jul 7 17:43:56 CDT 2020

Name: Norbert Lindenberg

Report Type: Error Report

Opt Subject: Missing Indic shaping properties for Devanagari and Vedic characters

A number of Devanagari and Vedic characters are missing Indic syllabic or positional category definitions in the Unicode 13.0 data:

– 0950, 0971, A8F4..A8F7, A8FB, A8FD don't have an Indic syllabic category (as letters they don't need a positional category).

– 1CE2..1CE8, 1CED don't have syllabic categories (they do have positional categories).

– 1CF8..1CF9 don't have a positional category (they do have a syllabic category).

For the first set, I can imagine that some of the characters don't participate in forming Devanagari syllables, and therefore the default value Other for the syllabic category is actually correct. If that's the case, however, I think it would be preferable to explicitly provide the value, both to make clear that it's intentional and to remind users of the data that this value can occur with Brahmic scripts (the specification of the Universal Shaping Engine currently does not handle this case).

### *Recommended UTC actions*

1. AI for Andrew Glass and other experts: Re L2/20-175 item F10: Investigate what the right Indic shaping properties should be for certain Devanagari and Vedic characters.

### *Background information / discussion*

Please discuss!

## Documents

### D1: Proposal to merge three Identifier\_Types in UTS #39

[L2/20-054](#) from Roozbeh Pournader

#### *Summary*

Proposal to merge Obsolete and Technical into Uncommon\_Use

#### *Full text*

UTS #39 defines three different property values for Identifier\_Type that are neither well-defined nor followed properly in their assignments to characters. On top of that, the text for version 13.0 of UTS #39 includes:

"For the qualifiers on usage, Obsolete, Uncommon\_Use and Technical, the distinctions among the Identifier\_Type values is not strict and only one might be given."

The author recently tried to determine the best property values for a few characters new to Unicode 13.0, and had a hard time drawing the line between these for some characters. Since there is no practical distinction among the three property values for any algorithm defined by the Unicode Consortium, the author suggests merging the two property values "Obsolete" and "Technical" into "Uncommon\_Use". The definition of "Uncommon\_Use" can be expanded to include the definitions of the other two property values.

## Recommended UTC actions

1. AI for Roozbeh: Re L2/20-054 and L2/20-175 item D1: Analyze which characters are marked as Identifier\_Type=Technical but are only Obsolete, and vice versa.

## Background information / discussion

[https://www.unicode.org/reports/tr39/#Identifier\\_Status\\_and\\_Type](https://www.unicode.org/reports/tr39/#Identifier_Status_and_Type)

- ...
- Obsolete: Characters that are no longer in modern use.
- Technical: Specialized usage: technical, liturgical, etc.
- Uncommon\_Use: Characters whose status is uncertain, or that are not commonly used in modern text.
- Limited\_Use: Characters from scripts that are in limited use ...
- ...

**Mark:** The advantage of separate categories is that customizations can use them to filter. Eg allow Technical but not Obsolete. The only reason to merge is if we think the distinction can't be reliably maintained.

**Asmus:** Agree with Mark. We have external specs that now cite these distinctions. OK to fine tune if something is major off-kilter, but generally it's better to know whether stuff is "in use, but for limited scope" vs. "no longer in use".

## D2: Proposal to fix Hebrew in UAX #14 by ruling out LB21a..b

[L2/20-087](#) from Marcel Schneider

### Summary

In [L2/20-087](#), the author argues that rule LB21a ("Do not break after Hebrew + Hyphen" "HL (HY | BA) ×") does not match the original requirement to address the problem that "With <hebrew hyphen non-hebrew>, there is no break on either side of the hyphen." He points out that the current rule prevents line breaking after the hyphen when a Hebrew character follows the hyphen and notes that it is common to break there in Biblical Hebrew. He also notes that the rule includes spaces in the BA class, which was not part of the original requirement.

The author provides a "pro forma solution" of changing the rule to "HL (HY | BA) × [^HL]" and removing spaces from the BA class (which he covers in [L2/20-088](#)).

The author also provides an "actual solution" of removing rules LB21a and LB21b, merging the HL class back into the AL class, and "fixing" spaces (which he covers in [L2/20-088](#)).

## Recommended UTC actions

Wait for the author to provide a single, consolidated, final document to replace L2/20-005, L2/20-087, L2/20-088, and the work-in-progress recent submissions.

## *Background information / discussion*

Note that the author of documents 087 & 088 has presented further documents on the same topic in 2020 July, and has indicated that this topic is a work in progress.

**Chris:** It's not clear to me how the "actual solution" solves the original problem of "With <hebrew hyphen non-hebrew>, there is no break on either side of the hyphen." Do any of you see a way this would work?

## D3: Proposal to adjust space characters in UAX #14

[L2/20-088](#) from Marcel Schneider

### *Summary*

The author began making a case for moving spaces out of the BA class in [L2/20-087](#) and continues it in this document ([L2/20-088](#)).

The author provides a "solution" of moving the space characters U+1680 and U+2000..U+2003 to the SP class and the spaces U+2004..U+200A to the GL class. (Note that U+2007 is already in the GL class.)

### *Recommended UTC actions*

Wait for the author to provide a single, consolidated, final document to replace L2/20-005, L2/20-087, L2/20-088, and the work-in-progress recent submissions.

## *Background information / discussion*

Note that the author of documents 087 & 088 has presented further documents on the same topic in 2020 July, and has indicated that this topic is a work in progress.

**Chris:** The author asserts that "there are virtually zero existing documents that would be disrupted when upgrading from Unicode 13.0.0 to Unicode 14.0.0 while the line breaking property values of fixed-width spaces are changed for Unicode 14.0.0 as suggested". Do any of you have counterexamples to this?

**Mark:** I am very, very hesitant about these changes, since they change line breaks. We'd have to be sure that either (a) they are so little used in text that it doesn't matter, or (b) that there is a strong reason for it.

Moreover, we run the risk that implementations would ignore this change. SP has special handling in implementations because it occurs so often, and has special handling at the ends of lines.

**Andy:** I'd have to refresh myself on UAX#14, but if BA doesn't work, it may be better to pop a new class ? My hunch is that Mark is right: Some classes, like SP, shouldn't acquire new members.

## D4: Identifier\_Status of Limited\_Use scripts in UTS #39

[L2/20-164](#) from Manish Goregaokar

### *Summary*

Move Identifier\_Type=Limited\_Use from Identifier\_Status=Restricted to Identifier\_Status=Allowed for support-by-default of aspirational-use scripts; for example, Javanese and Balinese.

Manish had first proposed and discussed this on the Unicode members mailing list on 2020-jun-29/30 under the subject "UTS 39: Inclusion of Identifier\_Type=Limited\_Use amongst Identifier\_Status=Restricted".

### *Recommended UTC actions*

We recommend not to make any changes.

### *Background information / discussion*

[https://www.unicode.org/reports/tr31/#Table\\_Limited\\_Use\\_Scripts](https://www.unicode.org/reports/tr31/#Table_Limited_Use_Scripts)

**In the email discussion**, Asmus Freytag argued for the status quo, worried about increasing the "attack surface" for spoofing, skeptical about how widely used the Limited\_Use scripts like Javanese are at this point, and pointing out that certain hurdles for use (fonts, keyboards, etc.) are more important than domain names and are not hindered by the Limited\_Use classification. Mark agrees with that, and notes that implementations are free to customize and add additional scripts to Allowed; the important feature is what Asmus noted, that this is insignificant in comparison to lack of fonts, keyboards, core locale data, etc. Richard Wordingham also seemed to agree with a cautious approach.

Manish recommends defaulting to permit Limited\_Use scripts and "maybe have a cautionary thing saying that you should be wary about confusables and that this increases the attack surface", rather than keeping them default-restricted and leaving it to products to discover (e.g., via bug reports) certain scripts that they may wish to allow.

Side issue: Restoring Aspirational

**Asmus:** I think it was wrong to remove the "aspirational" scripts category. I'd favor evaluating a proposal to bring that one back.

I am emphatically opposed to changing the status of limited\_use away from restricted. If something isn't limited\_use, then it should get a new (revived) category (with "recommended to treat as restricted").

That way, we can flexibly handle scripts that are (or may be) in the middle of a status change from defunct to actually in use. (That is, if people feel that there actually are aspirational scripts).

The key is to realize that there are external specifications that build on these classifications and a generic watering down of the meaning of limited\_use is more than unhelpful.

From a DNS perspective, recommendations would depend on the DNS zone. The root zone would be limited to recommended scripts, while second and lower levels may be more flexible. Recommendations for the second level are being formulated and it is in that area that it's conceivable that some country might allow a zone with an aspirational script.

Having Unicode make that determination would be preferable, because in the other arena, there's no way to really draw a boundary, and the minute some aspirational script has to be supported, if it is "limited\_use" then it undermines the case for restricting any other limited-use scripts.

Leaving things as they are is fine only as long as nobody succeeds in a real script revival to where a script may actually be competing with Latin or Arabic. At that point, the pressure of overriding limited\_use will be there and having a third category would add a firewall.

**Mark:** I'm against restoring "Aspirational".

We used to distinguish a particular class of Limited\_Use as "Aspirational".

We dropped that distinction when it became clear that there were no objective, measurable criteria for making that distinction.

We could only reinstate that if we were able to come up with such criteria, and a process for determining which scripts meet them.

IMO the criteria would at least include that a script is in common daily use (reading and writing) by a significant population — excluding scholars (think E. hieroglyphs) and hobbyists (think runes).

Then we would have to put in place a process for people to provide evidence that that is the case.

For all of that we would need a detailed proposal.

**Asmus:** Clarification: if we had some in-between classification for scripts in transition, where they don't meet (yet) the criteria for a recommended script, but are with some probability expected to do so, then we could resolve proposals like L/20-164 by focusing on whether the evidence is sufficient for assigning this third category.

Consumers of the Unicode specifications, like those setting policy for various DNS zones, would benefit from having a "watch list" of scripts that have a chance of becoming main-stream and also some guidance that support for them may be appropriate in certain zones (but not, by default) everywhere.

---

I generally agree with the need for a definition, not just a label, for some "in-between" status like "aspirational".

"Daily common use (non-specialized)" is effectively what we used for the DNS Root Zone for further selecting characters from the overall repertoire of recommended scripts.

Therefore, while a requirement like this would nicely differentiate a script from a limited-use one, it does make it difficult to distinguish a script from a fully recommended one.

The distinction could come, perhaps from a sense of trajectory. An aspirational script would be one that is experiencing significant changes in use, towards (or close to) a state where it is / will be in "daily common use". Another aspect could be that the script is in competition with an existing script for the same language community (or a recent development for a language community not well served by use of existing scripts).

With a definition like that, we would have a place for people like the current proposers to go to instead of requesting wholesale changes to existing classifications.

---

"Significant population" is always tricky. For the DNS we tried to abstract away from mere size, but to use a scale like EGIDS for the supported languages which describes how well a language is being transmitted across generations, resp. how close to extinction it is. Scripts that support native languages of low EGIDS values would qualify, even if the number of people speaking them was "small" (think Iceland or the Maldives) but not "miniscule" - after factoring in literacy rates.