Junicode 2 Feature Reference

Peter S. Baker

October 10, 2021

Introduction

The OpenType features of Junicode version 2 and its variable counterpart JunicodeVF (hereafter "Junicode") have two purposes. One is to provide convenient access to the rich character set of the Medieval Unicode Font Initiative (MUFI) recommendation.

The other is to enable best practices in the presentation of medieval text, promoting accessibility in electronic texts from PDFs to e-books to web pages.

Each character in the MUFI recommendation has a code point[1] associated with it: either the one assigned by Unicode or, where the character is not recognized by Unicode, in the Private Use Area (PUA) of the Basic Multilingual Plane, a block of codes, running from U+E000 to U+F8FF, that are assigned no value by Unicode but instead are available for font designers to use in any way they please.

The problem with PUA code points is precisely their lack of any value. Consider, as a point of comparison, the letter a (U+0061). Your computer, your phone, and probably a good many other devices around the house store a good bit of information about this a: that it's a letter in the Latin script, that it's lowercase, and that the uppercase equivalent is A (U+0041). All this information is available to word processors, browsers, and other applications running on your computer.

Now suppose you're preparing an electronic text containing what MUFI calls LATIN SMALL LETTER NECKLESS A (a). It is assigned to code point U+F215, which belongs to the PUA. Beyond that, your computer knows nothing about it: not that it is a variant of a, or that it is lowercase, or a letter in the Latin alphabet, or even a character in a language system. A screen reader cannot read, or even spell out, a word with U+F215 in it; a search engine will not recognize the word as containing the letter a.

Junicode offers the full range of MUFI characters—you can enter the PUA code points—but also a solution to the problems posed by those code points. Think of an electronic text (a web page, perhaps, or a PDF) as having two layers: an underlying text, stable and unchanging, and the displayed text, generated by software at the instant it is needed and discarded when it is no longer on the screen. For greatest accessibility the underlying text should contain the plain letter a (U+0061) along with markup indicating how it should be displayed. To generate the displayed text, a program called a "layout engine" will (simplifying a bit here) read the markup and apply the OpenType feature cv02[5][2] to the underlying a, bypassing the PUA code point, with the result that readers see a—the "neckless a." And yet the letter will still register as a with search engines, screen readers, and so on.

This is the Junicode model for text display, but it is not peculiar to Junicode: it is widely considered to be the best practice for displaying text using current font technology.

The full range of OpenType features listed in this document is supported by all ma-

---

[1]A Unicode code point is generally expressed as a four-digit hexadecimal (or base-16) number with a prefix of 'U+'. The letter capital "A," for example, is U+0041 (65 in decimal notation), and lowercase "ȝ" (Middle English yogh) is U+021D.

[2]Many OpenType features produce different outcomes depending on an index passed to an application's layout engine along with the feature tag. Different applications have different ways of entering this index: consult your application's documentation. Here, the index is recorded in brackets after the feature tag.

jor web browsers, LibreOffice, XeTeX, LuaTeX, and (presumably) other document processing applications. All characters listed here are available in Adobe InDesign, though that program supports only a selection of OpenType features. Microsoft Word, unfortunately, supports only Stylistic Sets, ligatures (all but the standard ones in peculiar and probably useless combinations), number variants, and the Required Features. In terms of OpenType support, Word is the most primitive of the major text processing applications.

Many MUFI characters cannot be produced by using the OpenType variants of Junicode. These characters fall into three categories:

- Those with non-PUA code points. MUFI has done valuable work obtaining Unicode code points for medieval characters. All such characters (those with hexadecimal codes that *do not* begin with E or F) are presumed safe to use in accessible and searchable text. However, some of these are covered by Junicode OpenType features for particular reasons.

- Precomposed characters—those consisting of base character + one or more diacritics. For greatest accessibility, these should be entered not as PUA code points, but rather as sequences consisting of base character + one or more diacritics. For example, instead of MUFI U+E498 LATIN SMALL LETTER E WITH DOT BELOW AND ACUTE, use e + U+0323 COMBINING DOT BELOW + U+0301 COMBINING ACUTE ACCENT: ẹ́ (when applying combining marks, start with any marks below the character and work downwards, then continue with any marks above the character and work upwards. For example, to make ǭ̣́, place characters in this order: o, COMBINING OGONEK U+0328, COMBINING DOT BELOW U+0323, COMBINING MACRON U+0304, COMBINING ACUTE U+0301). Some MUFI characters have marks in unconventional locations, e.g. ȯ́ LATIN SMALL LETTER O WITH DOT ABOVE AND ACUTE, where the acute appears beside the dot instead of above. This and other characters like it should still be entered as a sequence of base character + marks (here o, COMBINING DOT ABOVE U+0307, COMBINING ACUTE U+0301). Junicode will position the marks correctly.

- Characters for which a base character (a Unicode character to which it can be linked) cannot be identified, or for which there may be an inconsistency in the MUFI recommendation. These include:

    - ﬀ U+E8AF. This is a ligature of long s and l with stroke, but there are no base characters with this style of stroke.

    - ǘⱰ U+EFD8 and U+EFD9. MUFI lists these as ligatures (corresponding to the historic ligatures uⱰ, but they cannot be treated as ligatures in the

3

font because a single diacritic is positioned over the glyphs as if they were digraphs like aꜳ.

- ꝥꝥ U+EBE7 and U+EBE6, for the same reason.
- ꝺ U+F159 latin abbreviation sign small de. Neither a variant of d nor an eth (ð), this character may be a candidate for Unicode encoding.

- Characters for which OpenType programming is not yet available. These will be added as they are located and studied. [Check: U+EBF1, and smcp version.]

These characters should be avoided, even if you are otherwise using MUFI's PUA characters:

- U+F1C5 combining curl high position. Use U+1DCE combining ogonek above. The positioning problem mentioned in the MUFI recommendation is solved in Junicode (and, to be fair, many other fonts with OpenType programming).

- U+F1CA combining dot above high position. Use U+0307 combining dot above. It will be positioned correctly on any character.

A. Case-Related Features

1. `smcp` – Small Capitals

Converts lowercase letters to small caps; also several symbols and combining marks. All lower- and uppercase pairs (with exceptions noted below) have a small cap equivalent. Lowercase letters without matching caps may lack matching small caps. fghij → fghij.

Note: Precomposed characters defined by MUFI in the Private Use Area have no small cap equivalents. Instead, compose characters using combining diacritics, as outlined in the introduction. For example, `smcp` applied to the sequence t + combining ogonek (U+0328) + combining acute (U+0301) will change ṭ to ṭ.

2. `c2sc` – Small Capitals from Capitals

Use with `smcp` for all-small-cap text. ABCDE → abcde.

3. `pcap` – Petite Capitals

Produces small caps in a smaller size than `smcp`. Use these when small caps have to be mixed with lowercase letters. The whole of the basic Latin alphabet is covered, plus several other letters. klmnoþ → klmnoþ.

4. `case` – Case-Sensitive Forms

Produces combining marks that harmonize with capital letters: Ř, X̌, etc. Use of this feature reduces the likelihood that a combining mark will collide with a glyph in the line above.

B. Numbers and Sequencing

5. `nalt` – Alternate Annotation Forms

Produces letters and numbers circled, in parenthesis, or followed by periods, as follows:
    `nalt[1]`, circled letters or numbers: ⓐ ⓑ . . . ⓩ; ⓪ ① ② . . . ⑳.
    `nalt[2]`, letter or numbers in parentheses: (a) . . . (z); (0) (1) . . . (20).
    `nalt[3]`, double-circled numbers: ⓪ ① . . . ⑩.
    `nalt[4]`, white numbers in black circles: ⓿ ❶ ❷ ❸ . . . ⓴.
    `nalt[5]`, numbers followed by period: ⓪ ① . . . ⑳.
For enclosed figures 10 and higher, `rlig` (Required Ligatures) must also be enabled (as it should be by default: see Required Features below).

6. `tnum` – Tabular Figures

Fixed-width figures: 0123456789 (default or with `lnum`), 0123456789 (with `onum`).

7. `onum` – Oldstyle Figures

Figures that harmonize with lowercase characters: 0123456789 (default or with `tnum`), 0123456789 (with `pnum`). When combined with `pnum`, this feature also affects subscripts and superscripts.

8. `pnum` – Proportional Figures

Proportionally spaced figures: 0123456789 (default or with `lnum`), 0123456789 (with `onum`). When combined with `onum`, this feature also affects subscripts and superscripts. Most applications (including MS Word) with any support of OpenType features will support this feature and `lnum` in such a way that you don't have to enter them manually.

9. `lnum` – Lining Figures

Figures in a uniform height, harmonizing with uppercase letters: 0123456789 (default or with `tnum`), 0123456789 (with `pnum`).

10. `zero` – Slashed Zero

Produces slashed zero in all number styles: 0 0 0 0. Includes superscripts and subscripts: 0 0 0 0.

C. Superscripts and Subscripts

11. `sups` – Superscripts

Produces superscript numbers and letters. Only affects lining tabular and oldstyle proportional figures. All lowercase letters of the basic Latin alphabet are covered, and most uppercase letters: 0123 4567 abcde ABDEG. Wherever superscripts are needed (e.g. for footnote numbers), use `sups` instead of the raised and scaled characters generated by some programs. With sups: 4567. Scaled: 4567.

12. `subs` – Subscripts

Produces subscript numbers. Only affects lining tabular (the default numbers) and oldstyle proportional figures (use `pnum` and `onum`): 8901 2345.

D. Ornaments

13. `ornm` – Ornaments

Produces ornaments (fleurons) in either of two ways: as an indexed variant of the bullet character (U+2022) or as variants of a-z, A-C (all fleurons are available by either method):

As a variant of •: 1=❀, 2=❦, 3=❁, 4=❧, etc., up to 29.

As a variant of a-z, A-C: e=❀, f=❀, g=❧, h=❧, etc.

The method with letters of the alphabet is easier, but the method with bullets will produce a more satisfactory result when text is displayed in an environment where Junicode is not available or `ornm` is not implemented.

E. Alphabetic Variants

14. `cv01-cv52` – Basic Latin Variants

These features also affect small cap (`smcp`) and underdotted (`ss07`) forms, where available. Where Junicode has no variant for a Basic Latin letter, the expected `cvNN` feature

is skipped, being reserved for future development. These features also affect small cap (smcp) and underdotted (ss07) forms, where available.

Variant of	cvNN	Variants
A	cv01	1=Ɑ, 2=ᴀ, 3=Ꞛ
a	cv02	1=ɑ, 2=ᴀ, 3=ɑ, 4=a, 5=ᴀ
B	cv03	No variants available
b	cv04	No variants available
C	cv05	1=Ꞓ
c	cv06	1=ᴄ
D	cv07	1=Ꝺ
d	cv08	1=ð, 2=ꝺ, 3=ꝺ (for 1, see also ss02)
E	cv09	1=Ꞓ, 2=Ꞔ
e	cv10	1=ɛ, 2=e, 3=ᴇ
F	cv11	1=Ꝼ
f	cv12	1=ꝼ, 2=ꝼ, 3=ꝑ, 4=ꝑ, 5=f, 6=ſt
G	cv13	1=Ꝿ, 2=Ꝿ, 3=Ᵹ
g	cv14	1=ꝿ, 2=ꝿ, 3=ɡ, 4=ɡ, 5=ɡ, 6=ɡ, 7=ɡ
H	cv15	1=ꝲ
h	cv16	1=ꜧ, 2=ħ
I	cv17	1=İ, 2=Ɪ
i	cv18	1=ɪ, 2=ɪ, 3=ɨ
J	cv19	1=J̇
j	cv20	1=ȷ, 2=ɟ, 3=j
K	cv21	No variants available
k	cv22	1=ᴋ, 2=k, 3=ꝁ, 4=ꝁ
L	cv23	No variants available
l	cv24	1=ɩ
M	cv25	1=ꟽ, 2=ꟿ, 3=ℳ

Variant of	cvNN	Variants
m	cv26	1=ɱ, 2=ɱ, 3=ɱ
N	cv27	1=Ŋ
n	cv28	1=ŋ, 2=ɴ, 3=ñ, 4=n͡
O	cv29	1=Ọ
o	cv30	1=ọ
P	cv31	1=Ɋ
p	cv32	No variants available
Q	cv33	1=ꝗ
q	cv34	1=ꝗ
R	cv35	1=Ɲ
r	cv36	1=ɼ, 2=ɼ
S	cv37	1=ſ, 2=ẞ
s	cv38	1=ſ, 2=ʂ, 3=ſ, 4=ʃ, 5=ẞ, 6=ʃ
T	cv39	1=Ꞇ
t	cv40	1=τ, 2=τ
U	cv41	No variants available
u	cv42	No variants available
V	cv43	No variants available
v	cv44	1=ѵ, 2=v̌, 3=v̌, 4=v
W	cv45	No variants available
w	cv46	No variants available
X	cv47	No variants available
x	cv48	1=x, 2=x, 3=x, 4=x
Y	cv49	1=Ɏ
y	cv50	1=y, 2=ȳ
Z	cv51	1=Ꝫ

Variant of	cvNN	Variants
z	cv52	1=ꝫ, 2=ꝥ

15. `cv53-cv67` – Other Latin Letters

Some features affect both upper- and lowercase forms. `cv62` also affects combining e with ogonek, accessible via `ss10` with the entity reference `&_eogo;`. In this range, `cvNN` features are not reserved for future development, since Junicode already uses or reserves all of the available `cvNN` features.

Variant of	cvNN	Variants
Ą (U+0104)	cv53	1=Ą, 2=Ą, 3=Ą
ą (U+0105)	cv54	1=ą, 2=ą
ꜳ (U+A733)	cv55	1=ꜳ, 2=ꜳ
Æ (U+00C6)	cv56	1=Æ
æ (U+00E6)	cv57	1=æ, 2=æ, 3=æ
Ꜵ (U+A734)	cv58	1=Ꜵ
ꜵ (U+A735)	cv59	1=ꜵ, 2=ꜵ, 3=ꜵ
ꜹ (U+A739)	cv60	1=ꜹ
đ (U+0111)	cv61	1=đ
Ę, ę, ͕ (U+0118, U+0119)	cv62	1=Ę, ę, ͕
Ę, ę (U+0118, U+0119)	cv63	1=Ę, ę
Ȝ, ȝ (U+021C, U+021D)	cv64	1=Ȝ, ȝ
Ꝉ (U+A749)	cv65	1=Ꝉ
ø (U+00F8)	cv66	1=ø, 2=ø, 3=ø, 4=ø
Ꝥ, ꝥ (U+A765)	cv67	1=Ꝥ, ꝥ

16. `ss01` – Alternate thorn and eth

Produces Nordic thorn and eth (þðÞ) when the language is English, and English thorn and eth (þðÞ) with any other language, reversing the font's usual usage. This also affects small caps, crossed thorn (Ꝥ ꝥ—see also `cv67`), combining mark eth (U+1DD9, ͕ ͕), and enlarged thorn and eth (see `ss06`). This feature depends on `loca` (Localized Forms), which in most applications will always be enabled.

17. `ss02` – Insular Letter-Forms

Produces insular letter-forms, e.g. ðᵹʒɲɾꝑ. Does not affect capitals (except W), as these do not not commonly have insular shapes in early manuscripts. For these, enter the Unicode code points or use the Character Variant (`cvNN`) features.

18. `hist` – Historical Forms

Changes **s** to **ſ** (longs).

19. `ss03` – Long s

Changes **s** to **ſ** (duplicating `hist`). see also `ss08`.

20. `ss04` – High Overline

Produces a high overline over letters used as roman numbers: c̄d̄īj̄l̄m̄v̄x̄ C̄D̄ĪJ̄L̄M̄V̄X̄Ↄ̄.

21. `ss05` – Medium-High Overline

Produces a medium-high overline over (or through the ascenders of) letters used as roman numbers, and some others as well: b̄c̄d̄h̄ī j̄k̄l̄m̄f̄v̄x̄þ̄.

22. `ss06` – Enlarged Minuscules

Lowercase letters that match the height of normal ones, but with a higher x-height, e.g. abcdefg. Covers the whole of the basic Latin alphabet and several other letters: consult the MUFI recommendation for details, and if you are using the variable version of the font (JunicodeVF), consider using the Enlarged axis instead, for reasons of flexibility and accessibility.

23. `ss07` – Underdotted Text

Produces underdotted text (indicating deletion in medieval manuscripts) for many letters (including the whole of the basic Latin alphabet and a number of other letters), e.g. ạḅc̣ḍẹf̣g̣ ḤIJ̣ḲḶṂ. This also affects small caps, e.g. ABCDEF → ạḅc̣ḍẹf̣. For letters without corresponding underdotted forms (e.g. U+A751, ꝑ), use U+0323, combining dot below (ꝑ̣).

24. ss08 – Contextual Long s

In English and French text only, varies s and ſ according to rules followed by many early printers: ſports, eſſence, ſtormy, diſheveled, transfuſions, flyneſs, cliffside. For this feature to work properly, calt "Contextual Alternates" must also be enabled (as it should be by default: see Required Features below).

25. ss11 – r Rotunda

In lowercase and small caps, substitutes r rotunda (ꝛ) for r. This features does not affect capital R: the uncommon Ꝛ (U+A75A) must be entered manually. See also ss16.

26. ss16 – Contextual r Rotunda

Converts r to ꝛ (lowercase only) following the most common rules of medieval manuscripts: pꝛiest, firmer, frost, oꝛnament. For this feature to work properly, calt "Contextual Alternates" must also be enabled (as it should be by default: see Required Features below). See also ss11.

27. cv68 – Variant of ʔ (U+0294, glottal stop)

1=ˀ.

28. cv99 – revert small cap a to lowercase a

1=a. This features reverts small cap ᴀ to a, enabling it to ligature with small cap ɴ or ʀ via hlig: ᴀɴ, ᴀʀ. Be sure to apply smcp, cv99 and hlig to both components of the ligature.

F. Punctuation

MUFI encodes nearly twenty marks of punctuation in the PUA. In Junicode these can be accessed in either of two ways: all are indexed variants of . (period), and all are associated with the Unicode marks of punctuation they most resemble (but it should not be inferred that the medieval marks are semantically identical with the Unicode marks, or that there is an etymological relationship between them). The first method will be easier for most to use, but the second is more likely to yield acceptable fallbacks in environments where Junicode is not available.

Marks with Unicode encoding are not included here, as they can safely be entered directly. In MUFI 4.0 several marks have PUA encodings, but have since that release been

assigned Unicode code points: *paragraphus* (⁊ U+2E4D), medieval comma (⹌ U+2E4C), *punctus elevatus* (⸮ U+2E4E), *virgula suspensiva* (⸰ U+2E4A), triple dagger (⹋ U+2E4B).

29. `ss18` – Old-Style Punctuation Spacing

Colons, semicolons, parentheses, quotation marks and several other glyphs are spaced as in early printed books.

30. `cv69` – Variants of ⁊⹒ (U+204A / U+2E52, Tironian nota)

1=⁊⹒, 2=⁊⹒.

31. `cv70` – Variants of . (period)

1=·, 2=,, 3=', 4=;, 5=., 6=:, 7=;, 8=∴, 9=⸱, 10=⸲, 11=⸳, 12=⸴, 13=!, 14=⸵, 15=⸶, 16=∾, 17=~, 18=∴, 19=⸰, 20=⁄. This feature provides access to all non-Unicode MUFI punctuation marks. Some of them are available via other features (see below).

32. `cv71` – Variant of · (U+00B7, middle dot)

1=· (*distinctio*).

33. `cv72` – Variants of , (comma)

1=,, 2='.

34. `cv73` – Variants of ; (semicolon)

1=; (*punctus versus*), 2=,, 3=:, 4=;, 5=∴.

35. `cv74` – Variants of ⸮ (U+2E4E, *punctus elevatus*)

1=⸱, 2=⸲, 3=⸳, 4=⁊ (*punctus flexus*).

36. `cv75` – Variant of ! (exclamation mark)

1=! (*punctus exclamativus*).

37. `cv76` – Variants of ? (question mark)

1=⸴, 2=⸵, 3=∾. Shapes of the *punctus interrogativus*.

38. cv77 – Variant of ~ (ASCII tilde)

1=∼ (same as MUFI U+F1F9, "wavy line").

39. cv78 – Variant of * (asterisk)

1=∴. MUFI defines U+F1EC as a *signe de renvoi*. Manuscripts employ a number of shapes (of which this is one) for this purpose. Junicode defines it as a variant of the asterisk—the most common modern *signe de renvoi*.

40. cv79 – Variants of / (slash)

1=⫽, 2=⁄. The first of these is Unicode, U+2E4E.

G. Abbreviations

41. cv80 – Variant of ꝝ (U+A75D, rum abbreviation)

1=ꝝ.

42. cv81 – Variants of ̛ (U+035B, combining zigzag above)

1=ȏ, 2=ȏ, 3=ȏ. Positioning of the zigzag can differ from that of other combining marks, e.g. ɓ, ſ̛, d̛. If `calt` "Contextual Alternates" is enabled (as it is by default in most apps), variant forms of `cv81[2]` will be used with several letters, e.g. d̛, ſ̛, k̛. Enable `case` for forms that harmonize with capitals (Ã B̃ C̃ D̃), `smcp` for forms that harmonize with small caps (ᴇ̃ ꜰ̃ ɢ̃ ʜ̃).

43. cv82 – Variants of spacing ꝰ (U+A770)

1=ꝰ, 2=Ꝋ. `cv82[1]` produces the baseline *-us* abbreviation (same as MUFI U+F1A6). MUFI also has an uppercase baseline *-us* abbreviation (U+F1A5), but as there is no uppercase version of U+A770 to pair it with, it is indexed separately here.

44. cv83 – Variant of ꝫ (U+A76B, "et" abbreviation)

1=;. Identical in shape to a semicolon, but as it is semantically the same as U+A76B, it is preferable to use that character with this feature.

H. Combining Marks

45. cv84 – MUFI combining marks (variants of U+1DD1)

MUFI encodes a number of combining marks in the PUA (with code points between E000 and F8FF), but when these characters are entered directly, they can interfere with searching and accessibility, and some important applications fail to position them correctly over their base characters. To avoid these problems, enter U+1DD1 (�ᷝ, COMBINING UR ABOVE) and apply cv84, with the appropriate index, to both mark and base character. This collection of marks does not include any Unicode-encoded marks (from the "Combining Diacritical Marks" ranges), as these can safely be entered directly. It does include three marks (cv84[35], [36] and [37]) that lack MUFI code points but are used to form MUFI characters.

These marks can sometimes be produced by other cvNN features, which may be preferable to cv84 as providing fallbacks for applications that do not support Character Variant (cvNN) features.

For some marks with PUA code points, users may find it easier to use entities than this feature. Note that entities beginning &_ (ampersand + underscore) are for combining diacritics.

These marks are not affected by most other features. This is to preserve flexibility, given the rule that the feature that produces them must be applied to both the mark and the base character. For example, if smcp "Small Caps" changed cv84[11] ᷠ to [12] ᴮ, it would be impossible to produce the sequence NᴬA with the diacritic properly positioned.

1=ᷓ, 2=ᷙ, 3=ᷓ, 4=ᷔ, 5=ᷓ, 6=ᷕ, 7=ᷗ, 8=ᷓ, 9=ᷘ, 10=ᷘ, 11=ᷠ, 12=ᴮ, 13=ᷘ, 14=᷒, 15=ᷓ, 16=ᷓ, 17=ᷓ, 18=ᷘ, 19=ᷘ, 20=ᷘ, 21=ᷘ, 22=ᷓ, 23=ᷓ, 24=ᷓ, 25=ᷘ, 26=ᷘ, 27=ᷓ, 28=ᷓ, 29=ᷓ, 30=ᷘ, 31=ᷘ, 32=ᷓ, 33=ᷓ, 34=ᷓ, 35=ᷓ, 36=ᷓ, 37=ᷓ.

46. ss20 – Low Diacritics

The MUFI recommendation includes a number of precomposed characters with base letters b, h, k, þ, ð and ꝺ and combining marks ᷓ (U+0363), ᷔ (U+0364), ᷓ (U+1DD1/ cv84[17]), ᷓ (U+0366), ᷓ (U+036C), ᷓ (U+1DE2), ᷓ (U+036D), ᷓ (U+036E), ᷓ (U+1DE6) and ᷓ (U+1DD1/cv84[21]). Instead of being positioned above ascender height as usual (e.g. h̊), the MUFI glyphs have the marks positioned above the x-height (e.g. h̊). Using the MUFI code points for these precomposed glyphs can interfere with searching and drastically reduce accessibility. Users of Junicode should instead use a sequence of base character + combining mark, and apply ss20 to the two glyphs. A variant shape of eth (ð) that accommodates the combining mark will be substituted for the normal base character (but this is not necessary for the other base characters). Examples: b̊, ð̊, h̊, k̊, þ̊, ꝺ̊.

ss20 affects only the diacritics and base characters listed here; other combinations (e.g. m̐, h́) are not affected. It will therefore probably be safe to apply this feature to the whole text if it is needed anywhere.

47. cv85 – Variant of ö̆ (U+1DD3, combining open a)

1=ȫ.

48. cv86 – Variant of ö̦ (U+1DD8, combining insular d)

1=ö̦.

49. cv87 – Variant of ö̓ (U+1DE3, combining r rotunda)

1=õ.

50. cv88 – Variant of combining dieresis (U+0308)

1=ö̈. This also affects precomposed characters on which this variant dieresis may occur, e.g. ä.

51. cv89 – Variant of ö͞ö (U+0305, two-letter overline)

1=ö͞ö.

52. cv90 – Variant of ō̄ (U+0304, combining macron)

1=õ.

53. cv91 – Variants of short horizontal stroke (U+0335)

1=ө, 2=ө, 3=ө. This character can be used with letters with ascenders or descenders, e.g. đ ħ þ ꝑ. cv91[1] widens the stroke, and cv91[2] and [3] offset the stroke to the right or left. Via calt "Contextual Alternates," this offset is performed automatically for many characters with ascenders and descenders, and so it should rarely be necessary to use an index with cv91.

54. cv92 – Variant of breve below (U+032E)

1=ọọọ. Position the mark after the middle of three glyphs, and apply cv92 to both the mark and (at least) the middle glyph. This mark is not available via cv84.

I. Currency and Weights

55. `cv93` – Variants of ¤ (U+0044, generic currency sign)

1=ℳ, 2=✗, 3=⅄, 4=ⅆ, 5=ȝ, 6=ṽ, 7=ℬ, 8=gℛ, 9=ɸ, 10=ⅎ, 11=ℬ, 12=ℍ, 13=ℭ, 14=ℒ, 15=ℒ, 16=£, 17=ℒ, 18=ℒ, 19=*m*ℒ, 20=ℳ, 21=ℭ, 22=ℳ, 23=*ҡ*, 24=ß, 25=ßℓ, 26=ȥ, 27=ℭ. All of MUFI's currency and weight symbols (those that do not have Unicode code points) are gathered here, but some are also variants of other currency signs (see below).

56. `cv94` – Variant of ℔ (U+2114)

1=℔. Same as MUFI U+F2EB (French Libra sign).

57. `cv95` – Variants of £ (U+00A3, British pound sign)

1=ℬ, 2=ℍ, 3=ℭ, 4=ℒ, 5=ℒ, 6=£. Same as MUFI U+F2EA, F2EB, F2EC, F2ED, F2EE, F2EF, pound signs from various locales.

58. `cv96` – Variant of ₰ (U+20B0, German penny sign)

1=*ҡ*. Same as MUFI U+F2F5.

59. `cv97` – Variant of ƒ (U+0192, florin)

1=ℬ. Same as MUFI U+F2E8.

60. `cv98` – Variant of ℥ (U+2125, Ounce sign)

1=℥. Same as MUFI U+F2FD, Script ounce sign.

J. Gothic

61. `ss19` – Latin to Gothic Transliteration

Produces Gothic letters from Latin: Warþ þan in dagans jainans → 𐍅𐌰𐍂𐌸 𐌸𐌰𐌽 𐌹𐌽 𐌳𐌰𐌲𐌰𐌽𐍃 𐌲𐌰𐌹𐌽𐌰𐌽𐍃. In web pages, the letters will be searchable as their Latin equivalents.

K. Runic

62. ss12 – Early English Futhorc

Changes Latin letters to their equivalents in the early English futhorc. Because of the variability of the runic alphabet, this method of transliteration may not produce the result you want. In that case, it may be necessary to manually edit the result. fisc flodu ahof → ᚠᛁᛋᚳ ᚠᛚᚩᛞᚢ ᚪᚺᚩᚠ.

63. ss13 – Elder Futhark

Changes Latin letters to their equivalents in the Elder Futhark. Because of the variability of the runic alphabet, this method of transliteration may not produce the result you want. In that case, it may be necessary to manually edit the result. ABCDEFG → ᚨᛒᚲᛞᛖᚠᚷ.

64. ss14 – Younger Futhark

Changes Latin letters to their equivalents in the Younger Futhark. Because of the variability of the runic alphabet, this method of transliteration may not produce the result you want. In that case, it may be necessary to manually edit the result. ABCDEFG → ᛏᛒᛦᛏᛁᚠᛦ.

65. ss15 – Long Branch to Short Twig

In combination with ss14, converts long branch (the default for the Younger Futhark) to short twig runes: ᛏᛒᛦᛏᛁᚠᛦ → ᚤᚠᛦᚤᛁᚠᛦ.

66. rtlm – Right to Left Mirrored Forms

Produces mirrored runes, e.g. ᚠᛒᚱᛞᛗᚠᚷ → ᚷᚤᛗᛒᚱᚤᚠ. This feature cannot change the direction of text.

L. Ligatures and Digraphs

67. hlig – Historic Ligatures

Produces ligatures for combinations that should not ordinarily be rendered as ligatures in modern text.[3] Most of these are from the MUFI recommendation, ranges B.1.1(b)

---

[3]Some fonts define hlig differently, as including all ligatures in which at least one element is an archaic character, e.g. those involving long s (ſ). In Junicode, however, a historic ligature is defined not by the characters it is composed of, but rather by the join between them. If two characters should not be joined

and B.1.4. This feature does not produce digraphs (which have a phonetic value), for which see ss17. The ligatures:

af→af	ck→ck	hr→ħ	PP→PP	fþ→fþ
aꝼ→aꝼ	ꝺꝺ→ꝺꝺ	hſ→hſ	pp→pp	ττ→ττ
ag→ag	ey→ey	hf→hf	ꝓp→ꝓp	UE→UE
al→al	fä→fä	kr→kr	Ps→Ps	
an→an	gd→gd	kſ→kſ	ps→ps	ue→ue
aɴ→aɴ	gð→gð	kf→kf	Psi→Psi	UU→UU
ap→ap	gꝺ→gꝺ	ll→ll	Pi→Pi	uu→uu
ar→ar	gg→gg	ɴſ→ɴſ	qp→qp	pp→β
aʀ→aʀ	gg→gg	oc→oc	q3→q3	þr→þr
aþ→aþ	go→go	OꝜ→OꝜ	QꝜ→QꝜ	þs→þs
bb→bb	gp→gp	o2→o2	q2→q2	þſ→þſ
bg→bg	gr→gr	OꝛⱿ→OꝛⱿ	fä→fä	þþ→þþ
ch→ch	Hr→Hr	oⱿ→oⱿ	ſtr→ſtr	

Notes: For aɴ and aɴ see cv99 above. For the ligature ɴſ to work properly, U+017F ſ must be entered directly, not by applying an OpenType feature to s.

68. `dlig` – Discretionary Ligatures

Produces lesser-used ligatures, but also roman numbers, e.g. ii, II, xi, XI. The lesser-used ligatures: ct, sp, ſtr, ſt, tr, tt, ty.

69. `ss17` – Rare Digraphs

By "digraph" we mean conjoined letters that represent a phonetic value: the most common examples for western languages are æ and œ (though these, because they are so common, are not included in this feature). Use of this feature in web pages enables easier searches: for example, producing þaɪ from þau allows the word to be searched as "þau." The digraphs covered by this feature are aɪ, ao, aʊ, aʏ, ay, oo, uy, plus capital and small cap equivalents and digraph + diacritic combinations anticipated in the MUFI recommendation. To produce such a digraph + diacritic combination, either type a letter + diacritic combination as the second element of the digraph or type the diacritic after the second element. For example, a + ú yields áu, and so does a + u + U+0301 (combining acute accent). To produce a digraph + diacritic combination not covered by MUFI (e.g. aò), you may have to enter the digraph directly and not via ss17 (rare diacritics, however,

___
except in certain historic contexts, they form a historic ligature. If they should be joined in all contexts (though they may be archaic), the ligature is not historic and should be defined in liga.

are usually safe to use with this feature—e.g. ǻ). Note that `ss17` must be applied to the two elements of the digraph and any following diacritic.

M. Required Features

Required features, which provide some of the font's most basic functionality— ligatures, support for other features, kerning, and more—include `ccmp` (Glyph Composition/Decomposition), `calt` (Contextual Alternates), `liga` (Standard Ligatures), `loca` (Localized Forms), `rlig` (Required Ligatures), `kern` (Horizontal Kerning), and `mark`/`mkmk` (Mark Positioning). In MS Word these features have to be explicitly enabled on the Advanced tab of the Font dialog (Ctrl-D or Cmd-D: enable Kerning, Standard Ligatures, and Contextual Alternates, and the others will be enabled automatically), but in most other applications they are enabled by default.

N. Entities

70. `ss10` – Character Entity References

In XML and HTML, characters that can't easily be typed on a keyboard can be expressed as entity references consisting of an ampersand, a name, and a semicolon. The XML standard defines a few entities and HTML many more, but document authors can define as many as they like in a DTD. Junicode anticipates the need of medievalists for entities in addition to those defined in XML and HTML and defines more than a hundred of them. A few of these overlap with the collection of HTML entities, but most are peculiar to Junicode, emphasizing characters that are widely used in medieval texts and those that have only PUA code points (especially combining marks, which present special technical difficulties).

In applications that support Stylistic Sets, `ss10` makes these entities appear as the characters they represent. But as such entities can interfere with searching and accessibility when embedded in a web page, you should think of them as a convenient set of mnemonics to be used when typing, to be replaced by their Unicode equivalents before publication. If you use any non-Unicode characters (highlighted in the list below), you should call your publisher's attention to them, since they will present difficulties to typesetters.

&aa; → aa	&ao; → ao	&av; → av
&AA; → AA	&AO; → AO	&AV; → AV
æ → æ	&au; → au	&ay; → ay
Æ → Æ	&AU; → AU	&AY; → AY

&co; → ꝯ
ð → ð̆
Ð → Đ
&et; → ɜ
&ti; → ꜩ
&yo; → ꝫ
&YO; → Ꝫ
&kl; → ꝁ
&ob; → ꝋ
&OB; → Ꝋ
&OO; → ꝏ
&oo; → ꝏ
≺ → ꝑ
&po; → ꝓ
&q1; → ꝗ
&q2; → ꝙ
&rr; → ꝛ
&ru; → ꝝ
&is; → ſ
&sd; → ꝝ
&US; → ꝰ
&vy; → ꝩ
&VY; → Ꝩ
&wn; → ꝩ
&WN; → Ꝟ
þ → þ
Þ → Þ
&ct; → ꝥ
&_ZZ; → ◌
&_zz; → ◌
&_us; → ◌
&_ol; → ◌◌

&_a; → ◌ᵃ
&_oa; → ◌
&_ansc; → ◌
&_an; → ◌
&_ar; → ◌
&_arsc; → ◌
&_ao; → ◌
&_av; → ◌
&_aelig; → ◌
&_b; → ◌
&_bsc; → ◌
&_c; → ◌
&_ccedil; → ◌
&_d; → ◌
&_dsc; → ◌
&_dins; → ◌
&_eth; → ◌
&_e; → ◌
&_eogo; → ◌
&_emac; → ◌
&_f; → ◌
&_g; → ◌
&_gsc; → ◌
&_h; → ◌
&_i; → ◌
&_idotl; → ◌
&_j; → ◌
&_jdotl; → ◌
&_k; → ◌
&_ksc; → ◌
&_l; → ◌
&_lsc; → ◌

&_m; → ◌
&_msc; → ◌
&_munc; → ◌
&_n; → ◌
&_nsc; → ◌
&_o; → ◌
&_oogo; → ◌
&_oslash; → ◌
&_omac; → ◌
&_orr; → ◌
&_oru; → ◌
&_p; → ◌
&_q; → ◌
&_r; → ◌
&_rr; → ◌
&_rsc; → ◌
&_ur; → ◌
&_ru; → ◌
&_s; → ◌
&_longs; → ◌
&_t; → ◌
&_tins; → ◌
&_tsc; → ◌
&_u; → ◌
&_v; → ◌
&_w; → ◌
&_x; → ◌
&_y; → ◌
&_thorn; → ◌
&_z; → ◌

O. Non-MUFI Code Points

Characters in Junicode that do not have Unicode code points should be accessed via OpenType features whenever possible. MUFI/PUA code points should be used only in applications that do not support OpenType, or that support it only partially (for example, MS Word). For certain characters that lack either Unicode or MUFI code points, code

points in the Supplementary Private Use Area–A (plane 15) are available.

U+F0000 Ꜳ	U+F0008 d̄	U+F0010 m̄	U+F0018 v̄
U+F0001 ą	U+F0009 Ę	U+F0011 ơ	U+F0019 x̄
U+F0002 Ą	U+F000A ę	U+F0012 ø	U+F001A x̄
U+F0003 c̄	U+F000B f	U+F0013 o	U+F001B ˀ
U+F0004 c̄	U+F000C ī	U+F0014 o	U+F001C ƻ
U+F0005 ꝺ	U+F000D j	U+F0015 ꝗ	U+F001D ƶ
U+F0006 ꝺ	U+F000E j̄	U+F0016 ẑ	U+F001E Ꜵ
U+F0007 ꝺ	U+F000F ꝉ	U+F0017 v̄	U+F001F ą