Junicode

the font for medievalists

specimens and user manual

for version 2

Contents

About Junicode

The design of Junicode is based on scans of George Hickes, *Linguarum vett. septentrionalium thesaurus grammatico-criticus et archaeologicus* (Oxford, 1703–5). This massive two-volume folio is a fine example of the work of the Oxford University Press at this period: printed in multiple types (for every language had to have its proper type) and lavishly illustrated with engravings of manuscript pages, coins and artifacts.

The type used for Hickes's *Thesaurus* resembles those assembled by John Fell (1625–86) and bequeathed by him to the University of Oxford. To my eye, however, it looks more like the "Pica Roman" purchased by the University in 1692 than like any of Fell's. For printing in Old English, this type was supplemented by the "Pica Saxon" commissioned by the literary scholar Franciscus Junius (1591–1677) and bequeathed by him to the University. Specimens of both can be found in *A Specimen of the Several Sorts of Letter Given to the University by Dr. John Fell, Sometime Lord Bishop of Oxford. To Which Is Added the Letter Given by Mr. F. Junius* (Oxford, 1693). Junius's Pica Saxon was mixed freely with Pica Roman in printing the *Thesaurus*.

Junicode includes two Greek faces, one for the roman font and another for the italic. The first is newly designed to harmonize with Junicode, and the other, accompanying the italic face, is based on type designed by Alexander Wilson (1714–86) of Glasgow and used in numerous books published by the Foulis Press, most notably the great Glasgow Homer of

1756–58.

The Junicode project began around 1998, when the developer began to revise his older (early 1990s) "Junius" fonts for medievalists to take account of the Unicode standard, then relatively new. The font's name, a contraction of "Junius Unicode," was supposed to be a stopgap, serving until a more suitable name could be found, but the name "Junicode" is now so well known that it can't be changed. The project has been active for its entire history, responding to frequent requests from users and changes in font technology; the developer is now pushing towards the release of Junicode version 2, an extended font family of five weights and five widths, with both static and variable versions, which aims to promote best practices in the presentation of medieval texts, especially in the area of accessibility. This aspect of the font is explored in the Introduction to the Feature Reference.

Users need to be aware that Junicode version 2 is a beta, unstable and incomplete. Until the official release of version 2 (probably sometime in 2022), versions of Junicode numbered 1.NNN should be used for projects that require stability. The latest stable version can be downloaded here.

Specimens

Old and Middle English

Wē æthrynon mid ūrum ārum þā ȳðan þæs dēopan wǣles; wē ġesāwon ēac þā muntas ymbe þǣre sealtan sǣ strande, and wē mid āðēnedum hrǣġle and ġesundfullum windum þǣr ġewīcodon on þām ġemǣrum þǣre fæġerestan þēode. Þā ȳðan ġetācniað þisne dēopan cræft, and þā muntas ġetācniað ēac þā miċelnyssa þisses cræftes. (Regular)

Siþen þe sege and þe assaut watz sesed at Troye,
Þe borȝ brittened and brent to brondez and askez,
Þe tulk þat þe trammes of tresoun þer wroȝt
Watz tried for his tricherie, þe trewest on erthe:
Hit watz Ennias þe athel, and his highe kynde,
Þat siþen depreced prouinces, and patrounes bicome
Welneȝe of al þe wele in þe west iles. (SemiExpanded)

Apply the OpenType feature ss02 (Stylistic Set 2) for insular letter-forms.

Her cẏneƿulf benam sigebrẏht his rices ⁊ ƿestseaxna ƿiotan for unrẏhtum dędū buton hamtúnscire ⁊ he hæfde þa oþ he ofslog þone aldorman þe hī lengest ƿunode ⁊ hiene þa cẏneƿulf on andred adræfde ⁊ ħ þær ƿunade oþ þæt hine án sƿán ofstang æt prẏfetesflodan ⁊ he ƿręc þone aldorman cumbran ⁊ se cẏneƿulf oft miclum gefeohtum feaht uuiþ bretƿalū. (SemiCondensed)

Old Icelandic

For Nordic shapes of þ and ð in an English context, specify the appropriate language (e.g. Icelandic or Norwegian); or apply the OpenType ss01 (Stylistic Set 1) feature.

Um haustit sendi Mǫrðr Valgarðsson orð at Gunnarr myndi vera einn heimi, en lið alt myndi vera niðri í eyjum at lúka heyverkum. Riðu þeir Gizurr Hvíti ok Geirr Goði austr

yfir ár, þegar þeir spurðu þat, ok austr yfir sanda til Hofs. Þá sendu þeir orð Starkaði undir Þríhyrningi; ok fundusk þeir þar allir er at Gunnari skyldu fara, ok réðu hversu at skyldi fara. (SemiExpanded Medium)

Runic

ᚠᛁᛋᚴ ᚠᛚᚠᛞᚢᚴ ᚠᚿᚠᚠ ᚠᛏ ᚠᛖᚱᚷᛖᛏᛒᛖᚱᛁᚷ ᚦᚠᚱᚦ ᚷᚠ:ᛇᚱᛁᚴ ᚷᚱᚠᚱᛏ ᚦᚠᚱ ᚿᛖ ᚠᛏ ᚷᚱᛖᚢᛚᛏ ᚷᛁᛇᚹᚠᛖ ᚿᚱᚠᛏᚠᛇ ᛒᚠᛏ
ᚱᚠᛖᚹᚠᚴᚢᛇ ᚠᛏᛞ ᚱᛖᚢᛖᚹᚠᚴᚢᛇ ᛏᚹᛟᚷᚷᛖᛏ ᚷᛁᛒᚱᚠᚦᚠᚱ ᚠᚠᛟᛞᛞᚠ ᛞᛁᚠ ᚹᚣᚴᛁᚠ ᚠᛏ ᚱᚠᖻᚠᛁᚠᛇᛏᛁ:
ᚠᚦᚴᚠ ᚢᛏᛏᛖᚷᚷ (Expanded)

German

Ich ſag üch aber / minen fründen / Fō2chtēd üch nit vo2 denen die den lyb tōdend / vnd darnach nichts habennd das ſy mer thůgind. Ich wil üch aber zeigē vo2 welchem ir üch fō2chten ſollend. Fō2chtend üch vo2 dem / der / nach dem er tōdet hat / ouch macht hat zewerffen inn die hell: ja ich ſag üch / vo2 dem ſelben fō2chtēd üch. Koufft man nit fünff Sparen vm̄ zween pfennig℈ (Condensed)

Latin

Junicode contains the most common Latin abbreviations, making it suitable for diplomatic editions of Latin texts.

Adiuuanos dn̄ ſalutariſ noſter & ꝑꝑtī głam nominiſ tui dnē libanoſ· & ꝑꝑitiuſ eſto peccatiſ noſtriſ ꝑꝑter nomen tuum· Ne forte dicant ingentib: ubi eſt dn̄ eorum & innoteſcat innationib: corā oculiſ nr̄iſ· Poſuerunt moſticina ſeruorū ruorū eſcaſ uolatilib: cęli carneſ ſcō2ꝝ tuo2ꝝ beſtiiſ tenice· Facti ſum⁹ obꝑbrium uicíniſ nr̄iſ· (Light)

Gothic

jabai auk ƕas gasaihƀiþ þuk þana habandan kunþi in galiuge stada anakumbjandan, niu miþwissei is siukis wisandins timrjada du galiugagudam gasaliþ matjan? fraqistniþ auk sa unmahteiga ana þeinamma witubnja broþar in þize Xristus gaswalt. swaþ þan frawaurkjandans wiþra broþruns, slahandans ize gahugd siuka, du Xristau frawaurkeiþ. (SemiCondensed Light)

Use ss19 to produce Gothic letters automatically from transliterated text.

ᚷᚪᛒᚣᛁ ᚣᚢᚴ ᛟᚪᛊ ᚱᚪᛊᚪᛁᛟᛁᚠ ᚠᚢᚴ ᚠᚪᚿᚪ ᚻᚪᛒᚣᚿᛞᚪᚪᚿ ᚴᚢᚿᚹᛁ ᛁᚿ ᚱᚪᚪᛁᚢᚱ᛫ ᛊᛏᚪᛞᚪ ᚣᚿᚪᚴᚢᛗ-ᛒᚷᚪᚿᛞᚪᚪᚿ, ᚿᛁᚢ ᛗᛁᚠᚹᛁᛊᛊ᛫ ᛁᛊ ᛊᛁᚢᚴᛁᛊ ᚹᛁᛊᚪᚿᛞᚪᛁᚿᛊ ᛏᛁᛗᚴᚷᚪᛞᚪ ᛞᚢ ᚱᚪᚪᛁᚢᚱᚪᚱᚢᛞᚪᛗ

ᚴᛚᛋᛚᛚᛁᚠ ᛗᛚᛏᚷᛚᚴ? ᚷᛚᛒᚣᛚ ᛚᚢᚴ ᛜᛚᛋ ᚴᛚᛋᛚᛁᛜᛁᚠ ᚠᚢᚴ ᚠᛚᚢᛚ ᚺᛚᛒᚣᚢᛜᛚᚴ ᚴᚢᚢᚠᛁ ᛁᚢ ᚴᛚᛚᛁᚢᚴᛈ ᛋᛏᛚᛜᛚ ᚳᚢᛚᚴᚢᛗᛒᚷᚳᚢᛜᛚᚴ, ᚢᛁᚢ ᛗᛁᚠᚣᛁᛋᛋᛖᛁ ᛁᛋ ᛋᛁᚢᚴᛁᛋ ᚣᛁᛋᚳᚢᛜᛁᚢᛋ ᛏᛁᛗᚴᚷᚳᛚ ᚳᚢ ᚴᛚᛚᛁᚢᚴᚳᚴᚢᛜᚳᛚᛗ ᚴᛚᛋᛚᛚᛁᚠ ᛗᛚᛏᚷᛚᚴ? (SemiExpanded Bold)

Sanskrit Transliteration

mānaṁ dvividhaṁ viṣayadvai vidyātśaktyaśaktitaḥ
arthakriyāyāṁ keśadirnārtho 'narthādhimokṣataḥ

sadṛśāsadṛśatvācca viṣayāviṣayatvataḥ
śabdasyānyanimittānāṁ bhāve dhīsadasattvataḥ (SemiCondensed Semibold)

International Phonetic Alphabet

hwɑn θɑt ɑːprɪl wɪθ ɪs ʃuːrəs soːtə θə druxt ɔf mɑrʧ hɑθ peːrsəd toː θə roːte ɑnd bɑːðəd evrɪ væɪn ɪn swɪʧ lɪkuːr ɔf hwɪʧ vertɪu ɛnʤendrəd ɪs θə fluːr hwɑn zɛfɪrus eːk wɪθ hɪs sweːtə bræːθ (Regular)

Greek

βίβλος γενέσεως ἰησοῦ χριστοῦ υἱοῦ δαυὶδ υἱοῦ ἀβραάμ. ἀβραὰμ ἐγέννησεν τὸν ἰσαάκ, ἰσαὰκ δὲ ἐγέννησεν τὸν ἰακώβ, ἰακὼβ δὲ ἐγέννησεν τὸν ἰούδαν καὶ τοὺς ἀδελφοὺς αὐτοῦ, ἰούδας δὲ ἐγέννησεν τὸν φάρες καὶ τὸν ζάρα ἐκ τῆς θαμάρ, φάρες δὲ ἐγέννησεν τὸν ἑσρώμ, ἑσρὼμ δὲ ἐγέννησεν τὸν ἀράμ, ἀρὰμ δὲ ἐγέννησεν τὸν ἀμιναδάβ, ἀμιναδὰβ δὲ ἐγέννησεν τὸν ναασσών, ναασσὼν δὲ ἐγέννησεν τὸν σαλμών, σαλμὼν δὲ ἐγέννησεν τὸν βόες ἐκ τῆς ῥαχά (Regular)

βίβλος γενέσεως ἰησοῦ χριστοῦ υἱοῦ αυὶ υἱοῦ ἀβραάμ. ἀβραὰμ ἐγέννησεν τὸν ἰσαάκ, ἰσαὰκ ἐ ἐγέννησεν τὸν ἰακώβ, ἰακὼβ ἐ ἐγέννησεν τὸν ἰού αν καὶ τοὺς ἀ ελφοὺς αὐτοῦ, ἰού ας ἐ ἐγέννησεν τὸν φάρες καὶ τὸν ζάρα ἐκ τῆς θαμάρ, φάρες ἐ ἐγέννησεν τὸν ἑσρώμ, ἑσρὼμ ἐ ἐγέννησεν τὸν ἀράμ, ἀρὰμ ἐ ἐγέννησεν τὸν ἀμινα άβ, ἀμινα ἀβ ἐ ἐγέννησεν τὸν ναασσών, ναασσὼν ἐ ἐγέννησεν τὸν σαλμών, σαλμὼν ἐ ἐγέννησεν τὸν βόες ἐκ τῆς ῥαχά (Italic)

Lithuanian

Lithuanian poses several typographical challenges. Make sure Contextual Alternates (calt) is turned on; for í, use i followed by combining dot accent (U+0307) and acute (U+0301).

Visa žemė turėjo vieną kalbą ir tuos pačius žodžius. Kai žmonės kėlėsi iš rytų, jie rado slėnį Šinaro krašte ir ten įsikūrė. Vieni kitiems sakė: Eime, pasidirbkime

plytų ir jas išdekime. – Vietoj akmens jie naudojo plytas, o vietoj kalkių – bitumą. Eime, – jie sakė, – pasistatykime miestą ir bokštą su dangų siekiančia viršūne ir pasidarykime sau vardą, kad nebūtume išblaškyti po visą žemės veidą. (Expanded)

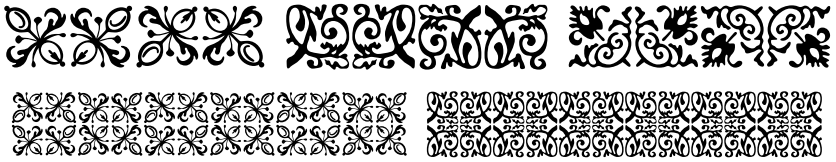
Polish

The default shape and position of ogonek in Junicode are suitable for modern Polish. For the medieval Latin e-caudata, consider using cv62.

Mieszkańcy całej ziemi mieli jedną mowę, czyli jednakowe słowa. A gdy wędrowali ze wschodu, napotkali równinę w kraju Szinear i tam zamieszkali. I mówili jeden do drugiego: Chodźcie, wyrabiajmy cegłę i wypalmy ją w ogniu. A gdy już mieli cegłę zamiast kamieni i smołę zamiast zaprawy murarskiej, rzekli: Chodźcie, zbudujemy sobie miasto i wieżę, której wierzchołek będzie sięgał nieba, i w ten sposób uczynimy sobie znak, abyśmy się nie rozproszyli po całej ziemi. (Condensed Bold)

Fleurons

Junicode contains a number of fleurons (floral ornaments) copied from a 1785 Caslon specimen book. Access these via the OpenType feature ornm. Fleurons have only one weight and width, and they are the same in roman and italic.



Getting Started with Junicode

With more than 4,700 characters, Junicode is a large font. Finding the things you want in a collection that size can be a challenge, and then entering them in your documents is another challenge. This document will help, but it presupposes a certain amount of knowledge—for example, how to install a font in Windows, Mac OS or Linux and how to install and use different kinds of software.

Medievalists will find the *MUFI Character Recommendation*, version 4.0 (2015) an essential supplement to this document. The *Recommendation* lists thousands of characters identified by the Medieval Unicode Font Initiative as being of interest to medievalists. Junicode contains all of these characters. There are two versions of the *Recommendation*: you will probably find the "Alphabetical Order" version most helpful.

From the *MUFI Character Recommendation* or, alternatively, the Junicode *List of Encoded Characters* (packaged with the font) you can find out the **code points**[1] of the characters you need. These code points can be used to enter characters in your documents when they cannot be typed on the keyboard.

To enter any Unicode character in a Windows application, type its four-digit code, followed by Alt-X. To do the same in the Mac OS, first install and switch to the "Unicode Hex Input" keyboard, then type the code while holding down the Option key. In most Linux distributions you can enter a code by typing Shift-Control-U, then the code followed by Return or Enter.

Combining marks (diacritics and certain abbreviation signs) can pose special problems for medievalists. Unicode contains a great many **precomposed characters** consisting of a base letter plus one or more marks. If these are all you need you're fortunate—especially if they can be typed on an international keyboard (not all can).[2] But medieval manuscripts

[1] A Unicode code point is a numerical identifier for a character. It 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] Both Windows and the Mac OS come with international keyboards that make it easy to type

frequently contain combinations of base + mark that are not used in modern written languages. For these, you'll have to enter bases and marks separately.

To position a mark correctly over a base character, first enter the base, followed by the mark or marks. The sequence **m** + ◌᷹ (U+1DD9) will make **m᷹**; **y** + ◌̵ (U+0304) + ◌̆ (U+0306) will make **y̵̆**; **e** + ◌̣ (U+0323) + ◌᷎ (U+1DE0) will make **e᷎̞**.

More than sixteen hundred characters in Junicode can only be accessed via OpenType features—that is, by way of the programming built into the font—and many others *should* be accessed that way for reasons explained in the Introduction to the Feature Reference section of this document.

For example, some programs (including Microsoft Word) produce small caps by scaling capitals down to approximately the height of lowercase letters. These always look too thin and light. But Junicode contains hundreds of TRUE SMALL CAPS designed to harmonize with the surrounding text. These can only be accessed via the OpenType `smcp` feature, which you can apply to a run of text much as you apply italic or bold styles: select some text and then apply the feature.

Unfortunately, not every program supports OpenType features, and some that do either support only a few or make them difficult to access. Programs that support Junicode's features fully include the free word processor LibreOffice Writer, all major browsers (Firefox, Chrome, Safari and Edge), and the typesetting programs LuaLaTeX and XeLaTeX. Adobe InDesign supports OpenType features only partially, but provides access to all of Junicode's characters through its own interface.

Microsoft Word, unfortunately, provides only limited support for OpenType features. It supports the Required Features discussed below, and also variant number forms and Stylistic Sets (though only one at a time). Many characters (for example, TRUE SMALL CAPS and those accessible only via Character Variant features) cannot be accessed at all. To activate Word's OpenType support, you must open the "Font" dialog, click over to the "Advanced" tab, and check the "Kerning" box. (Oddly, the "Kerning" box enables all other OpenType features.) Then, in the same tab, select Standard Ligatures, Contextual Alternates, and any other features you want. OpenType features are best applied to character styles rather than directly to text: this will save you from having to perform this operation repeatedly.

It is also good to set the language properly for the text you're working on. Programs like Word will automatically set the language to the default for your system. If you change to a language other than your own for a passage (or even a single word), you

---

special letters and diacritics. To find out how to enable these, search online for "Mac OS International Keyboard" or "Windows International Keyboard."

should set the language for that passage appropriately. This will unlock a number of capabilities. For example, in Old and Middle English, Word and other programs will use the English form of thorn and eth (þð), and in ancient Greek you will be able to type accents after vowels instead of looking up the codes for hundreds of polytonic vowels. But these and other capabilities are only available when you set the language correctly.

In LibreOffice and InDesign you can set the language with a drop-down menu in the "Character" dialog. In Word there is a separate "Language" dialog, accessible from the "Tools" menu.

If you have questions about any aspect of Junicode, post a query in the Junicode discussion forum. If you notice a bug, please open an issue at the font's development site. If you need help with programming, subsetting or other tasks, contact the developer directly.[3]

[3] b dot tarde at gmail dot com.

Feature Reference

Introduction

The OpenType features of Junicode version 2 and its variable counterpart (hereafter referred to together as "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 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]`[4] to the underlying **a**, bypassing the PUA code point, with the result that readers see **ɑ**—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 major 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 Unicode (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

[4]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. Users of fontspec (with XeLaTeX or LuaTeX) should also be aware that fontspec indexes start at zero while OpenType indexes start at one. Therefore all index numbers listed in this document must be reduced by one for use with fontspec.

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:
  - **ſƚ** U+E8AF. This is a ligature of long **s** and **l** with stroke, but there are no base characters with this style of stroke.
  - **ꝡ̋Ꝡ̋** U+EFD8 and U+EFD9. MUFI lists these as ligatures (corresponding to the historic ligatures **ꝡꝠ**, but they cannot be treated as ligatures in the font because a single diacritic is positioned over the glyphs as if they were digraphs like **ꜳꜲ**.
  - **ꝓ̈Ꝓ̈** 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 t̨́ to ṭ́.

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. frac – Fractions

Applied to a slash and surrounding numbers, produces fractions with diagonal slashes. 6/9 becomes ⁶⁄₉, 16/91 becomes ¹⁶⁄₉₁.

6. 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 ⓵ . . . 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).

7. tnum – Tabular Figures

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

8. 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.

9. 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.

10. lnum – Lining Figures

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

11. zero – Slashed Zero

Produces slashed zero in all number styles: ⌀ ⌀ 0̸ ⌀. Includes superscripts, subscripts, and fractions formed with frac: ⁰ ⁰ ₀ ₀ 10⁄30.

C. Superscripts and Subscripts

12. 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.

13. subs – Subscripts

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

D. Ornaments

14. 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

15. 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.

Variant of	cvNN	Variants
A	cv01	1=Ɑ, 2=Ⱥ, 3=A
a	cv02	1=ɑ, 2=ꞛ, 3=ɑ, 4=a, 5=ɑ
B	cv03	No variants available
b	cv04	No variants available
C	cv05	1=C
c	cv06	1=c
D	cv07	1=D
d	cv08	1=ꝺ, 2=ꝺ, 3=d (for 1, see also ss02)
E	cv09	1=Ɛ, 2=Ɛ
e	cv10	1=ɛ, 2=e, 3=e
F	cv11	1=Ꝼ
f	cv12	1=ꝼ, 2=ꝼ, 3=ꝼ, 4=ꝼ, 5=f, 6=f
G	cv13	1=Ᵹ, 2=Ꞡ, 3=G
g	cv14	1=ᵹ, 2=ꞡ, 3=ɡ, 4=g, 5=g, 6=ɡ, 7=g
H	cv15	1=Ꜧ
h	cv16	1=ꜧ, 2=h
I	cv17	1=İ, 2=I
i	cv18	1=ı, 2=ɩ, 3=ɨ
J	cv19	1=J̇

Variant of	cvNN	Variants
j	cv20	1=ȷ, 2=ɉ, 3=j
K	cv21	No variants available
k	cv22	1=k, 2=ꝁ, 3=ꝃ, 4=ꝅ
L	cv23	No variants available
l	cv24	1=ꞁ
M	cv25	1=ꝳ, 2=Ɱ, 3=M
m	cv26	1=m, 2=ɱ, 3=ꝳ
N	cv27	1=ꝴ
n	cv28	1=ŋ, 2=ɴ, 3=ꞥ, 4=ꝴ
O	cv29	1=Ꝋ
o	cv30	1=ꝋ
P	cv31	1=ꟼ
p	cv32	No variants available
Q	cv33	1=Ꝗ
q	cv34	1=ꝙ
R	cv35	1=Ꞃ
r	cv36	1=ꞃ, 2=ꝛ
S	cv37	1=Ꞅ, 2=Ƨ
s	cv38	1=ꞅ, 2=ꜱ, 3=ſ, 4=ʃ, 5=ẜ, 6=ẝ
T	cv39	1=Ꞇ
t	cv40	1=ꞇ, 2=ꜩ
U	cv41	No variants available
u	cv42	No variants available
V	cv43	No variants available

Variant of	cvNN	Variants
v	cv44	1=v̽, 2=v̾, 3=v̿, 4=v̀
W	cv45	No variants available
w	cv46	No variants available
X	cv47	No variants available
x	cv48	1=x́, 2=x̋, 3=x̧, 4=x̨
Y	cv49	1=Ý
y	cv50	1=ý, 2=y̋
Z	cv51	1=Ƹ
z	cv52	1=ƹ, 2=ƺ

16. 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=Ą́, 3=Ą̓
ą (U+0105)	cv54	1=ą, 2=ą́
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
ꝟ (U+A739)	cv60	1=ꝟ
đ (U+0111)	cv61	1=đ

Variant of	cvNN	Variants
Ę, ę … (U+0118, U+0119)	cv62	1=Ę, ę …; 2=Ę, ę …
Ȝ, ȝ (U+021C, U+021D)	cv63	1=Ȝ, ȝ
ꝥ (U+A749)	cv64	1=ꝕ
ø (U+00F8)	cv65	1=o, 2=ø, 3=o, 4=o
ꝧ, ꝧ (U+A765)	cv66	1=ꝧ, ꝧ
?	cv67	Reserved for future use

17. 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 (ꝧ ꝧ—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.

18. ss02 – Insular Letter-Forms

Produces insular letter-forms, e.g. ð�something. 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.

19. ss04 – High Overline

Produces a high overline over letters used as roman numbers: c̄d̄ij̄l̄m̄v̄x̄ C̄D̄I̅J̅L̄M̄V̄X̄Ↄ̄.

20. 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: b̶c̶d̶h̶ı̶j̶k̶l̶m̶f̶v̶x̶ꝧ̶.

21. ss06 – Enlarged Minuscules

Letters that are lowercase in form but uppercase in function, and between upper- and lowercase in size. They are often used to begin sentences in medieval manuscripts:

abcdefg. The feature covers the whole of the basic Latin alphabet and a number of other letters that occur at the beginnings of sentences: consult the MUFI recommendation for details. Uppercase letters are also covered by this feature so that enlarged minuscules can, if you like, be searched as capitals.

If you are using the variable version of the font (JunicodeVF), consider using the Enlarged axis instead, for reasons of flexibility and accessibility.

22. 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. ạ̣ḅc̣ḍẹf̣g̣ ḤỊJ̣ḲḶṂ. This also affects small caps, e.g. ABCDEF → ᴀ̣ʙ̣ᴄ̣ᴅ̣ᴇ̣ꜰ̣. For letters without corresponding underdotted forms (e.g. U+A751, ꝑ), use U+0323, combining dot below (ꝑ̣).

23. 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, ſlyneſſ, cliffſide. For this feature to work properly, calt "Contextual Alternates" must also be enabled (as it should be by default: see Required Features below).

24. ss16 – Contextual r Rotunda

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

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

ɪ=ʔ.

26. cv99 – revert small cap ᴀ to lowercase a

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

F. Greek

Junicode has two distinct styles of Greek. In the roman face, it is upright and modern, especially designed to harmonize with Junicode's Latin letters. In the italic, it is slanted and old-style, being based on the eighteenth-century Greek type designed by Alexander Wilson and used by the Foulis Press in Glasgow. Both Greek styles include the full polytonic and monotonic character sets: αβγδεζ *αβγ εζ*.

To set Greek properly (especially polytonic text) requires that both `locl` and `ccmp` be active, as they should be by default in most text processing applications (but in MS Word they must be explicitly enabled by checking the "kerning" box on the "Advanced" tab of the Font Dialog).

Modern monotonic Greek should be set using only characters from the Unicode "Greek and Coptic" range (U+0370–U+03FF). When monotonic text is set in all caps, Junicode suppresses accents automatically (except in single-letter words, for which you must substitute unaccented forms manually). This substitution is not performed on text containing visually identical letters from the "Greek Extended" range (U+0F00–U+1FFF). Thus when setting polytonic Greek, one should use (for example) **Ά** (U+1FBB), not **Ά** (U+0386), though they look the same.

You can set polytonic Greek either by entering code points from the Greek Extended range or by entering sequences of base characters and diacritics. When using the latter method, you must first make sure the language for the text in question (whether a single word, a short passage, or a complete document) is set to Greek, and then enter characters in canonical order (that is, the sequence defined by Unicode as equivalent to the composite character). The order is as follows: 1. base character; 2. diacritics positioned either above or in front of the base character, working from left to right or bottom to top; 3. the *ypogegrammeni* (U+0345), or for capitals, if you prefer, the *prosgegrammeni* (U+1FBE).

For example, the sequence ω (U+03C9) ◌̓ (U+0313) ◌́ (U+0301) ◌ͅ (U+0345) produces **ᾧ**. Substitute capital Ω (U+03A9) and the result is **῾Ωι**. Note that in a number of applications the layout engine will perform these substitutions before Junicode's own programming is invoked. If either the layout engine or Junicode fails to produce your preferred result, try placing U+034F COMBINING GRAPHEME JOINER somewhere in the sequence of combining marks—for example, before the *ypogegrammeni* to make **῾Ω͏ͅ**.

27. ss03 – Alternate Greek

Provides alternate shapes of β γ θ π φ χ ω: β γ θ π φ χ ω. These are chiefly useful in linguistics, as they harmonize with IPA characters.

G. 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).

28. ss18 – Old-Style Punctuation Spacing

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

29. cv69 – Variants of �battuta (U+204A / U+2E52, Tironian nota)

1=⁊, 2=⁊.

30. 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).

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

1=· (*distinctio*).

32. cv72 – Variants of , (comma)

1=‚, 2=’.

33. cv73 – Variants of ; (semicolon)

1=; (*punctus versus*), 2=⁏, 3=⁝, 4=⁘, 5=⁖.

34. cv74 – Variants of ⁝ (U+2E4E, *punctus elevatus*)

1=⁝, 2=⁞, 3=⁝, 4=⁝ (*punctus flexus*).

35. cv75 – Variant of ! (exclamation mark)

1=*!* (*punctus exclamativus*).

36. cv76 – Variants of ? (question mark)

1=⸮, 2=⸘, 3=⸮. Shapes of the *punctus interrogativus*.

37. cv77 – Variant of ~ (ASCII tilde)

1=~ (same as MUFI U+F1F9, “wavy line”).

38. 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*.

39. cv79 – Variants of / (slash)

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

H. Abbreviations

40. cv80 – Variant of ꝛ (U+A75D, rum abbreviation)

1=ꝝ.

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

1=◌͛, 2=◌͝, 3=◌͠. Positioning of the zigzag can differ from that of other combining marks, e.g. b͛, f͛, d͛. 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. d͝, p͝, k͝. Enable case for forms that harmonize with capitals (A͠ B͠ C͠ D͠), smcp for forms that harmonize with small caps (E͠ F͠ G͠ H͠).

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

1=ꝰ, 2=9. 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.

43. cv83 – Variants of ⁊ (U+A76B, "et" abbreviation)

1=;, 2=₃. [1] is 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. [2] produces a subscript version of the character, a common variant in printed books.

I. Combining Marks

44. cv84 – MUFI combining marks (variants of U+0304)

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+0304 (◌̄, COMBINING MACRON) 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[36], [37] and [38]) that lack MUFI code points but are used to form MUFI characters.

This feature may often appear to have no effect. When this happens it is because an application replaced a sequence like **a U+0304** with a precomposed character like **ā** (U+0101) before Junicode's OpenType programming had a chance to work. This process is called normalization, and it usually has the effect of simplifying text processing tasks,

but can sometimes prevent the proper functioning of OpenType features. To disable it, place the character U+034F COMBINING GRAPHEME JOINER (don't waste time puzzling over the name) between the base character and the combining mark (or the first combining mark). For example, to produce the combination **ú**, enter **u U+034F U+0304**. (Without U+034F, you would get **ú**).

These marks can sometimes be produced by other cvNN features, which may be preferable to cv84 as providing more suitable 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.

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] ◌̪, it would be impossible to produce the sequence **N̪AA** with the diacritic properly positioned.

1=◌̂ 2=◌́ 3=◌̇ 4=◌̷ 5=◌̸ 6=◌̓ 7=◌̈ 8=◌̀ 9=◌̃ 10=◌̄ 11=◌̑ 12=◌̪ 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=◌̀ 37=◌̧

45. ss10 – Entity References for Combining Marks

Instead of cv84 for representing non-Unicode combining marks, some users may wish to use XML/HTML-style entities. When ss10 is turned on (preferably for the entire text), these entities appear as combining marks and are correctly positioned over base characters. For example, the letter **e** followed by **&_eogo;** will yield **ę**. An advantage of entities is that they are (unlike the PUA code points or the indexes of cv84) mnemonic and thus easy to use. A disadvantage is that searches cannot ignore combining marks entered by this method as they can using the cv84 method. (Every method of entering non-Unicode combining marks has disadvantages: users should weigh these, choose a method, and stick with it.)

If you use any of these entities in a work intended for print publication, you should call your publisher's attention to them, since they will likely have their own method of representing them.

&_ansc; → ◌ᵆ
&_an; → ◌ᵃⁿ
&_ar; → ◌ᵃʳ
&_arsc; → ◌ᵆʳ
&_bsc; → ◌ᵇ
&_dsc; → ◌ᵈ
&_eogo; → ◌ᵉ
&_emac; → ◌ᵉ̄
&_idotl; → ◌ᵢ
&_j; → ◌ʲ
&_jdotl; → ◌ⱼ
&_ksc; → ◌ᵏ
&_munc; → ◌ᵐ
&_oogo; → ◌ᵒ
&_oslash; → ◌ᵒ̸
&_omac; → ◌ᵒ̄
&_orr; → ◌ᵒʳ
&_oru; → ◌ᵒᵆ
&_q; → ◌ᑫ
&_ru; → ◌ʳ
&_tsc; → ◌ᵀ
&_y; → ◌ʸ
&_thorn; → ◌ᵇ

46. ss20 – Low Diacritics

The MUFI recommendation includes a number of precomposed characters with base letters b, h, k, þ, ȡ and ð and combining marks ◌ᵃ (U+0363), ◌ᵉ (U+0364), ◌ᵢ (U+0304/cv84[18]), ◌ᵒ (U+0366), ◌ᵣ (U+036C), ◌ᵆ (U+1DE2), ◌ₜ (U+036D), ◌ᵛ (U+036E), ◌ᶻ (U+1DE6) and ◌ᵐ (U+0304/cv84[21]). Instead of being positioned above ascender height as usual (e.g. **hᵃ**), the MUFI glyphs have the marks positioned above the x-height (e.g. **ħᵃ**). 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: bᵒ, ðᵛ, hᵢ, kᵉ, þᵗ, ðʳ.

ss20 affects only the diacritics and base characters listed here; other combinations (e.g. **mᵉ**, **hᵢ́**) 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 ◌ᵆ (U+1DD3, combining open a)

1=◌ᵃ̄.

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

1=◌ᵇ.

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

1=◌᷏.

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

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

51. cv89 – Variant of ◌̅ (U+0305, combining overline)

1=◌̿.

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

1=◌̵, 2=◌̶, 3=◌̷. This character can be used with letters with ascenders or descenders, e.g. đ ƀ ꝑ ꝓ. `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`.

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

1=◌◌◌. 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`.

J. Currency and Weights

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

1=₤
2=₹
3=₹
4=₱
5=₹
6=₹
7=₹
8=₹
9=₱
10=₹
11=₹
12=₹
13=₹
14=₹
15=₹
16=₹
17=₹
18=₹
19=₹
20=₹
21=₹
22=₹
23=₹
24=₹
25=₹
26=₹
27=₹

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).

55. cv94 – Variant of ℔ (U+2114)

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

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

1=𞋿, 2=𞋾, 3=𞋽, 4=𞋼, 5=𞋻, 6=𞋺. Same as MUFI U+F2EA, F2EB, F2EC, F2ED, F2EE, F2EF, pound signs from various locales.

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

1=𞋹. Same as MUFI U+F2F5.

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

1=ﬔ. Same as MUFI U+F2E8.

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

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

K. Gothic

60. ss19 – Latin to Gothic Transliteration

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

K. Runic

61. 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 → fiᛋᛁᚴ flᚠᛒᚱᚢ ᚠᚾᚹᚠ.

62. 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.

63. 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 → ᚼᛒᚴᛏᛁᚠᚴ.

64. ss15 – Long Branch to Short Twig

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

65. rtlm – Right to Left Mirrored Forms

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

L. Ligatures and Digraphs

66. hlig – Historic Ligatures

Produces ligatures for combinations that should not ordinarily be rendered as ligatures in modern text.[5] Most of these are from the MUFI recommendation, ranges B.1.1(b)

[5] 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 (though modern) should not be joined except in certain historic contexts, they form a historic ligature. If they should be joined in all contexts (even if archaic), the ligature is not historic and should be defined in

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

af→ff
aɾ→ɾɾ
ag→ɑg
al→ɬ
an→ɑn
aɴ→ɑɴ
ap→ɑp
ar→ɾ
aʀ→ɑʀ
aþ→ɸþ
bb→ɮb
bg→ɮg
ch→ʗ
ck→ʞk
ðð→ꝺð
ey→ɐy
fä→fü
gd→ɠ
gð́→gð́
gð→ɠ
gg→gg
gg→gg
go→gɒ
gp→gp
gr→gɾ
Hr→ℏ
hr→ℏ
hſ→ɦ
hf→ɦ
kr→ƙ
kſ→ƙ
kf→ƙ
ll→ll
nſ→ɳ
oc→ɔc
OƦ→ƆƦ
o2→ɔ2
OƷ→ƆƷ
oƷ→ɔƷ
PP→ꟼP
pp→ꝑp
ꝓp→ꝓp
Ps→Pˢ
ps→pˢ
Psi→Pˢi
psi→pˢi
qp→qp
q3/q3→q3/q3
QƦ→QƦ
q2→qƨ
ſä→ſä
ſch→ſʗ
ſtr→ſtr
ſp→ſp
ʊʊ→ɷ
UE→ᵾE
ue→ᵾ
UU→ᵾU
uu→ᵾ
pp→ꝑ
þr→þr
þs→þs
þf→þf
þþ→ꝑ

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

67. dlig – Discretionary Ligatures

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

68. 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 þaı from þau allows the word to be searched as "þau." The digraphs covered by this feature are ɑ, ɑo, ɑı, ɑɾ, ɑy, ɔo, ɯy, 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 aı́, and so does a + u + U+0301 (combining

liga.

acute accent). To produce a digraph + diacritic combination not covered by MUFI (e.g. ȣ̀), you may have to place U+034F COMBINING GRAPHEME JOINER (see cv84 above) between the second element of the digraph and the combining mark. Without U+034F: ȣ̄. With U+034F: ȣ̄.

N. 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.

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 ⍰
U+F0001 ꝏ
U+F0002 Ȇ̌
U+F0003 Ę́
U+F0004 ꝍ
U+F0005 Ꝉ
U+F0006 Ę
U+F0007 Ꝺ
U+F0008 ɔ
U+F0009 ꟈ
U+F000A ꝶ
U+F000B ɑ
U+F000C ᵪ
U+F000D Æ
U+F000E Є
U+F000F Ę́
U+F0010 Ɲ
U+F0011 Ꜧ
U+F0012 Ꞑ́
U+F0013 Є
U+F0014 Ꞑ̈
U+F0015 Ꞑ̌
U+F0016 a
U+F0017 aa
U+F0018 á
U+F0019 æ
U+F001A b
U+F001B c
U+F001C d
U+F001D ð
U+F001E ɔ
U+F001F e

Junicode on the Web

If you are using Junicode on a web page, you should prefer the variable fonts (those with "VF" in the family name and filename) to the static fonts. One variable font file can do the work of many traditional font files. For example, the Test of High-Level CSS Properties web page displays Junicode in regular, bold and semicondensed styles. It used to be that your user would have to download three font files, one for each style, but one variable font will display all three.

But you may be thinking, *That font is big*! It's true: even the compressed webfont (.woff2) is nearly a megabyte in size—enough to seriously slow down the loading of a web page.

To solve this problem, you'll need to subset the font—that is, produce a copy that contains only what you need. The subsetted font that downloads with the property test web page is approximately 275k in size—almost thirty percent of the size of the full webfont. It's still a pretty big download, but that's because the page displays a lot of the font's features. If I were displaying, say, a diplomatic transcript of a Latin text, the font would be much smaller.

So the first section of this chapter will talk about how to subset the Junicode font.

Subsetting Junicode

First the legalities. It is perfectly all right to create a modified version of Junicode via subsetting, compress it into a webfont (almost certainly in woff2 format), and host it on your web server. This is because "Junicode" is not a "Reserved Font Name" (which complicates web use of many fonts licensed under the Open Font License). If you are nervous about the legal requirements of the Open Font License, you can change the font name to something arbitrary with the `--obfuscate-names` option of the pyftsubset program, and you can embed the Open Font License, or a link to it, in your CSS. These steps should settle any ambiguity about whether you are in compliance with the license.

Generating a subsetted version of Junicode should be one of the last tasks you perform before deploying your web page(s). Until then, it is recommended that you work with an unmodified copy of Junicode. After subsetting, review your pages thoroughly to make sure everything is displayed properly. If you have forgotten to include a glyph in your subsetted font, you will see little boxes where characters should be or, perhaps, the correct characters displayed in the wrong typeface. If you have omitted features, you will see default instead of transformed characters.

There are many subsetting programs, some online and very easy to use. But for maximum control (and thus the smallest fonts), you should choose pyftsubset, a part of the fontTools library, which runs under Python 3.7 or higher. This is a command-line tool which takes a long list of arguments; you should create a shell script to run it.

Here is the script used to create the subsetted font for the property test web page mentioned above:

```
#!/bin/zsh
pyftsubset JunicodeTwoBetaVF-Roman.ttf \
--flavor=woff2 --output-file=JunicodeVFsubset.woff2 \
--recommended-glyphs \
--text="jq" --text-file=Junicode-2-feature-test.html \
--layout-features+=liga,ss01,ss02,ss03,ss04,ss05,ss06,ss07,ss08,\
ss10,ss12,ss13,ss14,ss15,ss16,ss17,ss18,ss19,ss20,cv01,cv02,cv05,\
cv06,cv07,cv08,cv09,cv10,dlig,hlig,onum,pnum,pcap,smcp,c2sc,subs,\
sups,zero \
--layout-features-=rlig
```

For those unfamiliar with shell scripts, the first line specifies the shell the script is to run under (in this case the default shell for Mac OS, but `bash` is another possible choice), and the backslashes mean that the command continues on the next line. The rest of the file is a list of arguments passed to pyftsubset. Let's walk through them.

First after the program name comes the name of the unsubsetted, uncompressed font file. After that, the `--flavor` argument tells the program that you want a webfont in woff2 format, and `--output` is the name of the font file you want the program to save.

Having taken care of this preliminary business, we tell pyftsubset what we want the font to contain.

`--recommended-glyphs` includes a few characters that every font should have, according to the OpenType specification—though in fact modern browsers don't care. It's

best, however, to conform to the specification, since it's impossible to say with absolute certainty that no program will ever reject the font because of the absence of these few characters.

`--text-file` is the name of a file to treat as a list of characters that *must* be included in the font. In this case I have simply used the html file for the web page for this purpose. If your site contains multiple web pages, your job will be more complicated. You must make sure the text file contains all the characters used on the site—either that or supplement the text file with a `--text` argument (which here adds two lowercase letters that don't appear in the web page—just in case). The text file will contain only encoded characters—you don't have to worry here about unencoded characters produced by OpenType features.

`--layout-features+` tells the program which OpenType features you want to retain in the font. All others, except for the Required Features, are discarded. All of the characters referenced in these features will also be included in the output file, as long as those characters are variants of characters in your text file. For example, the `smcp` (Small Caps) feature has many more small caps than there are letters of the alphabet, but most of them are not included in the subsetted font. The program's parsimony with characters keeps the font file as small as possible. Note that some features are included automatically: `ccmp`, `locl`, `calt`, `liga`, `rlig`, `kern`, `mark`, and `mkmk`.

`--layout-features-` tells the program which OpenType features to omit. Normally, `rlig` (Required Ligtures) is automatically included in fonts by pyftsubset, but as it has no relevance to this web page, it can be omitted.

These are the most useful arguments, but there are many more. Type `pyftsubset --help` for a complete list. Once you have written your script, run it (in Mac OS or Linux you need to make the file executable by typing `chmod +x mysubsetscript` on the command line).

Before you put your subsetted font to work, check it carefully in a program like FontGoggles, which lets you preview the font and test all its OpenType features. If you find errors, revise your script and run it again.

Junicode and CSS/HTML

This section assumes a basic knowledge of HTML (Hypertext Markup Language, used to construct web pages) and CSS (Cascading Style Sheets, used to format them). If you want to learn about these subjects, the number of good books and online tutorials is so great that it makes no sense to try to list them. Just make sure that the instructional

materials you choose are of recent vintage, because the relevant standards are always changing.

In the CSS for your web page, the `@font-face` at-rule for a variable font is a little different from the one for a static font in that the range of possible values for each axis can be declared:

```
@font-face {
    font-family: "Junicode Two Beta VF";
    src: url("./webfiles/JunicodeVFsubset.woff2");
    font-weight: 300 700;
    font-stretch: 75% 125%;
    font-style: normal;
}
```

These ranges are not strictly necessary, but they will prevent your supplying invalid values for `font-weight` and `font-stretch` (that is, `width`) in other CSS rules.

Once you have declared the font, you can invoke it in setting up classes. For example:

```
body {
  font-family: "Junicode Two Beta VF";
  font-size: 28px;
  font-weight: normal; /* that is, 400 */
  font-stretch: 112.5%; /* that is, semiexpanded */
}
h1 {
  font-family: "Junicode Two Beta VF";
  font-size: 125%;
  font-weight: 600; /* that is, semibold */
  font-stretch: 112.5%; /* that is, semiexpanded */
}
.annotation {
  font-size: 90%;
  font-weight: 300; /* that is, light */
  font-stretch: 87.5%; /* that is, semicondensed */
}
```

These classes should be tested in all browsers. If any fail to display text properly, you can use `font-variation-settings` instead of the high-level `font-weight` and `font-stretch`:

```
body {
  font-family: "Junicode Two Beta VF";
  font-size: 28px;
  font-variation-settings: "wght" 400, "wdth" 112.5;
}
h1 {
  font-family: "Junicode Two Beta VF";
  font-size: 125%;
  font-variation-settings: "wght" 600, "wdth" 112.5;
}
.annotation {
  font-size: 90%;
  font-variation-settings: "wght" 300, "wdth" 87.5;
}
```

To accommodate older browsers, you should make a selection of Junicode static fonts, subset them, and include them in your CSS. For example, if you need normal and bold weights of Junicode roman, your `@font-face` at-rule may look like this:

```
@font-face {
    font-family: "Junicode Two Beta VF";
    src: url("./webfiles/JunicodeVFsubset.woff2");
    font-weight: 300 700;
    font-stretch: 75% 125%;
    font-style: normal;
}
@font-face {
    font-family: "Junicode Two Beta";
    src: url("./webfiles/JunicodeTwoBeta-Regular.woff2");
    font-weight: 400;
    font-style: normal;
}
@font-face {
    font-family: "Junicode Two Beta";
    src: url("./webfiles/JunicodeTwoBeta-Bold.woff2");
    font-weight: 700;
    font-style: normal;
```

```
}
```

Now use `@supports` in your CSS rules to determine which font gets downloaded:

```
body {
  font-family: "Junicode Two Beta", serif;
}
@supports (font-variation-settings: normal) {
  body {
    font-family: "Junicode Two Beta VF", serif;
  }
}
b {
  font-weight: 700;
}
@supports (font-variation-settings: "wght" 700) {
  b {
    font-variation-settings: "wght" 700;
  }
}
```

Now the variable version of Junicode will be downloaded only if the browser supports it, and the static version will be downloaded only if needed.

This document was set in 12pt Junicode VF SemiExpanded
using the LuaTEX typesetting system with fontspec for font management.
The source for the document, Feature_Reference.tex, is available at
https://github.com/psb1558/Junicode-font.