



Junicode

the font for medievalists



specimens and user manual

for version 2.201



Contents

1	About Junicode	1
2	Specimens	3
3	Getting Started with Junicode	9
4	Feature Reference	14
4.1	Introduction	14
4.2	Case-Related Features	17
4.2.1	<code>smcp</code> – Small Capitals	17
4.2.2	<code>c2sc</code> – Small Capitals from Capitals	17
4.2.3	<code>pcap</code> – Petite Capitals	18
4.2.4	<code>c2pc</code> – Petite Capitals from Capitals	18
4.2.5	<code>case</code> – Case-Sensitive Forms	18
4.3	Numbers and Sequencing	18
4.3.1	<code>frac</code> – Fractions	18
4.3.2	<code>numr</code> – Numerators	19
4.3.3	<code>dnom</code> – Denominators	19
4.3.4	<code>nalt</code> – Alternate Annotation Forms	19
4.3.5	<code>tnum</code> – Tabular Figures	19
4.3.6	<code>onum</code> – Oldstyle Figures	19
4.3.7	<code>pnum</code> – Proportional Figures	20
4.3.8	<code>lnum</code> – Lining Figures	20
4.3.9	<code>zero</code> – Slashed Zero	20
4.3.10	<code>ss09</code> – Alternate Figures	20

4.4	Superscripts and Subscripts	20
4.4.1	sup s – Superscripts	20
4.4.2	sub s – Subscripts	21
4.5	Ornaments	21
4.5.1	orn m – Ornaments	21
4.5.2	Lady Junicode	21
4.6	Alphabetic Variants	22
4.6.1	cv01–cv52 – Basic Latin Variants	22
4.6.2	cv53–cv66, cv91 – Other Latin Letters	24
4.6.3	ss01 – Alternate thorn and eth	25
4.6.4	ss02 – Insular Letter-Forms	25
4.6.5	ss04 – High Overline	26
4.6.6	ss05 – Medium-High Overline	26
4.6.7	ss06 – Enlarged Minuscles	26
4.6.8	ss07 – Underdotted Text	26
4.6.9	ss08 – Contextual Long s	27
4.6.10	ss16 – Contextual r Rotunda	27
4.6.11	sal t – Stylistic Variants (medieval capitals)	27
4.6.12	cv68 – Variant of ? (U+0294, glottal stop)	27
4.7	Greek	27
4.7.1	ss03 – Alternate Greek	29
4.8	Punctuation	29
4.8.1	ss18 – Old-Style Punctuation Spacing	29
4.8.2	cv69 – Variants of ꝛꝛ (U+204A / U+2E52, Tironian nota)	29
4.8.3	cv70 – Variants of . (period)	29
4.8.4	cv71 – Variant of · (U+00B7, middle dot)	30
4.8.5	cv72 – Variants of , (comma)	30
4.8.6	cv73 – Variants of ; (semicolon)	30
4.8.7	cv74 – Variants of ∴ (U+2E4E, <i>punctus elevatus</i>)	30
4.8.8	cv75 – Variant of ! (exclamation mark)	30
4.8.9	cv76 – Variants of ? (question mark)	30
4.8.10	cv77 – Variant of ~ (ASCII tilde)	30
4.8.11	cv78 – Variant of * (asterisk)	30
4.8.12	cv79 – Variants of / (slash)	30

4.9	Abbreviations	31
4.9.1	cv80 – Variant of ꝛ (U+A75D, run abbreviation)	31
4.9.2	cv81 – Variants of ͇ (U+035B, combining zigzag above)	31
4.9.3	cv82 – Variants of spacing ʹ (U+A770)	31
4.9.4	cv83 – Variants of 3 (U+A76B, “et” abbreviation)	31
4.10	Combining Marks	31
4.10.1	cv84 – MUFI combining marks (variants of U+0304)	31
4.10.2	cv67 – Spacing zigzag (variant of U+00AF, spacing macron)	33
4.10.3	ss10 – Character Entities for Combining Marks	33
4.10.4	ss20 – Low Diacritics	34
4.10.5	cv85 – Variant of ͆ (U+1DD3, combining open a)	34
4.10.6	cv86 – Variant of ͇ (U+1DD8, combining insular d)	34
4.10.7	cv87 – Variant of ͈ (U+1DE3, combining r rotunda)	34
4.10.8	cv88 – Variant of combining dieresis (U+0308)	35
4.10.9	cv89 – Variant of ͊ (U+0305, combining overline)	35
4.10.10	cv90 – Variants of ͋ (U+035E, combining double macron)	35
4.10.11	cv92 – Variant of breve below (U+032E)	35
4.11	Currency and Weights	35
4.11.1	cv93 – Variants of ͌ (U+0044, generic currency sign)	35
4.11.2	cv94 – Variant of ͍ (U+2114)	36
4.11.3	cv95 – Variants of ͎ (U+00A3, British pound sign)	36
4.11.4	cv96 – Variant of ͏ (U+20B0, German penny sign)	36
4.11.5	cv97 – Variant of ͐ (U+0192, florin)	36
4.11.6	cv98 – Variant of ͑ (U+2125, Ounce sign)	36
4.12	Gothic	36
4.12.1	ss19 – Latin to Gothic Transliteration	36
4.13	Runic	36
4.13.1	ss12 – Early English Futhorc	36
4.13.2	ss13 – Elder Futhark	37
4.13.3	ss14 – Younger Futhark	37
4.13.4	ss15 – Long Branch to Short Twig	37
4.13.5	rtlm – Right to Left Mirrored Forms	37
4.14	Ligatures and Digraphs	37
4.14.1	hlig – Historic Ligatures	38

4.14.2	dlig – Discretionary Ligatures	39
4.14.3	ss17 – Rare Digraphs	39
4.15	Required Features	39
5	Non-MUFI Code Points	40
6	Entering characters with tags	42
7	Transcribing records	61
7.1	A preliminary note on transcription	61
7.2	1. Common combining marks	63
7.3	2. Spacing characters	65
7.4	3. Other formatting	67
7.5	4. On the web	67
8	The Enlarge Axis	69
9	Unicode on the Web	71
9.1	Subsetting Unicode	71
9.2	Unicode and CSS/HTML	74

1. About Junicode

Junicode is modeled on the Pica Roman type purchased by Oxford University in 1692 and used to set the bulk of the Latin text of George Hickes, *Linguarum vett. septentrionalium thesaurus grammatico-criticus et archaeologicus* (Oxford, 1703–5). This massive two-volume folio is not only a major work of scholarship on the languages and literatures of northern Europe in the Middle Ages, but also a fine example of the work of the Oxford 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.

Junicode also includes two other typefaces from the *Thesaurus*: Pica Saxon, used to set passages in the Old English language, and a typeface reproducing the Gothic alphabet (“Gothic” here being not the late medieval style, but rather the earliest extensively attested Germanic language). These were commissioned by the literary scholar Franciscus Junius (1591–1677) and bequeathed by him to the University. Examples of all these typefaces 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).

Junicode has two distinct Greek faces. The first, newly designed to harmonize with the roman face, is upright and modern. 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 “Junicode” quickly stuck, and it is now so well known that it can’t be changed.¹ The project has been active for its entire history, responding to frequent requests from users and changes in font technology; a particular focus of recent versions of Junicode (numbered 2.000 and higher) is the promotion of 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.

¹ An effort to change the name to “JuniusX” produced only confusion. If you find a font by the name JuniusX on a free font site, that is nothing more than an early version of Junicode 2.

2. Specimens

Old and Middle English

Wē æthrynon mid ūrum ārum þā yðan þæs dēopan wāles; wē gesāwon ēac þā muntas ymbe þære sealtan sē strande, and wē mid ādēnedum hrægle and gesundfullum windum þær gewicodon on þām gemārum þære fægerestan þeode. Þā yðan getācniað þisne dēopan cræft, and þā muntas getācniað ēac þā micelnyssa þisses cræftes. (Regular)

Siþen þe sege and þe assault 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 bicomē
Welneȝe of al þe wele in þe west iles. (SemiExpanded)

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

Her cýnepulſ benam riȝebryht hys riceȝ ȝ peſſeaxna ȝiotan ȝop unſryhtum deðū buton
hamtúnſcipe ȝ he hæfde þa oþ he ofſloȝ þone aldormon þe hi lengeſt punode ȝ hiene þa
cýnepulſ on andƿeð aðræfde ȝ h þær punaðe oþ þæt hine án ȝpán ofſtang æt ȝrýſeteſſlodan
ȝ he ȝrȝc þone aldormon cumbƿan ȝ ȝe cýnepulſ oft miclum ȝeſeohtum feaht uuiþ bƿetƿalū.
(SemiCondensed)

Old Irish

Fect n-oen do Ailill ȝ do Meidb iar n-dergud a rígleptha dóib i Cruachanráith Chonnacht, arrecaim

4 SPECIMENS

comrad chind-cherchailli eturru. Fírbriathar, a ingen, bar Ailill, is maith ben ben dagfír. Maith omm, bar ind ingen. Cid diatá latsú ón. Is de atá lim, bar Ailill, ar it ferr-su indiu indá in lá thucus-sa thu. (Condensed Medium)

For insular letter-forms, apply the OpenType feature ssoz (Stylistic Set 2), making sure the language is set to Irish.

ḃamaith-re nemut, ar Meḃb. Ír maith nach cualammar ḡ nach řetammar, ar Ailill, acht ḃo bithriu ar bantincup mnaa ḡ biḃba na cřich ba neřřom ḃuit oc břeith ḃo řlait ḡ ḃo chřech i řúatagh úait. Ní řamlaiḃ bářa, ar Meḃb, acht m'athair i n-arḃriři hEřenn .i. Eocho řeidlech mac řind meic řindomain meic řindeoin meic řindguni meic Rořein Rúaiḃ meic Riřéoin meic ḃlathachta meic ḃeothechtā meic Enna Ařniř meic Oengura řurbiř. bářar aice ře ingena ḃ'ingenaib: řeřbriu, Echi ḡ Éle, Clothriu, Muřain, Meḃb, meřři ba uarřiu ḡ ba upřaitiu ḃib. (Regular)

For a (somewhat) uncial look, try combining ssoz with smcp (Small Caps), adding other variants as you see fit.

ḃamaith-re nemut, ar Meḃb. Ír maith nach cualammar ḡ nach řetammar, ar Ailill, acht ḃo bithriu ar bantincup mnaa ḡ biḃba na cřich ba neřřom ḃuit oc břeith ḃo řlait ḡ ḃo chřech i řúatagh úait. Ní řamlaiḃ bářa, ar Meḃb, acht m'athair i n-arḃriři hEřenn .i. Eocho řeidlech mac řind meic řindomain meic řindeoin meic řindguni meic Rořein Rúaiḃ meic Riřéoin meic ḃlathachta meic ḃeothechtā meic Enna Ařniř meic Oengura řurbiř. bářar aice ře ingena ḃ'ingenaib: řeřbriu, Echi ḡ Éle, Clothriu, Muřain, Meḃb, meřři ba uarřiu ḡ ba upřaitiu ḃib. (Regular)

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 sso1 (Stylistic Set 1) feature.

Um haustit sendi Mörðr 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

Junicode has features for automated transliteration of Latin letters into various runic systems.

ÞÍΛΛ ÞÍΓΩΛ ΓΩΠΠ ΘΤ ΘΜRXΜΤΒΜRIX ΠΡΡΠ ΧΓ:ΣΡΙΛ ΧΡΡΡΤ ΠΡΡ ΝΜ ΘΤ ΧΡΜΝΤ
ΧΙΣΠΠΠ ΝΡΡΠΠΣ ΒΠΤ
ΡΡΠΠΠΠΠΠ ΠΤΩ ΡΜΝΠΠΠΠΠΠ ΤΠΩΧΜΤ ΧΙΒΡΠΠΡΡ ΡΡΩΩΩΩ ΩΙΠ ΡΛΠΠ ΠΤ ΡΡΠΠ
ΛΠΣΤΠ: ΠΠΠΠ ΛΤΤΜΧ (Expanded)

German

Ich sag üch aber / minen fründen / Fōzchtēd üch nit vo2 denen die den lyb tōdend / vnd darnach nichts habennd das fy mer thūgind. Ich wil üch aber zeigē vo2 welchem ir üch fōzchten follend. Fōzchtend üch vo2 dem / der / nach dem er tōdet hat / ouch macht hat zewerffen inn die hell: ja ich sag üch / vo2 dem selben fōzchtēd üch. Koufft man nit fünff Sparen v̄m zween pfennigꝝ (Condensed)

Latin

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

Adiuuanos dī salutaris noster & pp̄t glām nominis tui dnē libanof & ppitiuf esto peccatis nostris ppter nomen tuum. Ne forte dicant ingentib: ubi est dī eorum & innotescat innationib: corā oculis nr̄is. Posuerunt mosticina seruorū ruorū escaf uolatilib: celi carnes scōz tuoꝝ bestiis tenice. Facti sumꝰ obp̄brium uicinis nr̄is. (Light)

Gothic

jabai auk hwas gasaihvīþ 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)

6 SPECIMENS

Use ss19 to produce Gothic letters automatically from transliterated text.

ԳԼԵՒԻ ԼՈՒԿ ՕՒՏ ԴՆՏԻՈՒՓ ՓՈՒ ՓԼՈՒ ԿԼԵՒՆԴԼՆ ԿՈՆՓԻ ԻՆ ԴԼԼԻՆԴԵ ՏԴԴԼ ԼՈՒԿՈՄԵԾԼՆԴԼՆ, ՈՒՆ ՄԻՓՎԻՏՏԵԻ ԻՏ ՏԻՆԿԻՏ ՎԻՏԼՆԴԻՆՏ ԴԻՄԿԾԼԴԼ ԴՆ ԴԼԻՆԴԼՆԴԼՆ ԴՆՏԻՈՒՓ ՄԼԴԾԼՆ? ԳԼԵՒԻ ԼՈՒԿ ՕՒՏ ԴՆՏԻՈՒՓ ՓՈՒ ՓԼՈՒ ԿԼԵՒՆԴԼՆ ԿՈՆՓԻ ԻՆ ԴԼԼԻՆԴԵ ՏԴԴԼ ԼՈՒԿՈՄԵԾԼՆԴԼՆ, ՈՒՆ ՄԻՓՎԻՏՏԵԻ ԻՏ ՏԻՆԿԻՏ ՎԻՏԼՆԴԻՆՏ ԴԻՄԿԾԼԴԼ ԴՆ ԴԼԼԻՆԴԼՆԴԼՆ ԴՆՏԻՈՒՓ ՄԼԴԾԼՆ? (SemiExpanded Bold)

Sanskrit Transliteration

mānaṁ dvividhaṁ viṣayadvai vidyātsāktyaśaktiṭaḥ
arthakriyāyāṁ keśadīrṇārtho 'narthādhimokṣataḥ
sadrśāsadrśatvācca viṣayāviṣayatvataḥ
śabdasyānyanimittānāṁ bhāve dhīśadasattvataḥ (SemiCondensed Medium)

International Phonetic Alphabet

hwan θat a:pril wiθ is ju:rəs so:tə θə druht of marƷ haθ pe:rsəd to: θə ro:te and ba:ðəd
 evri væin in swiƷ liku:r of hwiƷ vertiu endʒendrəd is θə flu:r hwan zefirus e:k wiθ his
 swe:tə bræ:θ (Regular)

Greek

βίβλος γενέσεως ἰησοῦ χριστοῦ υἱοῦ δαυιδ υἱοῦ ἄβραάμ. ἄβραάμ ἐγέννησεν τὸν ἰσαάκ, ἰσαάκ δὲ ἐγέννησεν τὸν ἰακώβ, ἰακώβ δὲ ἐγέννησεν τὸν ἰούδαν καὶ τοὺς ἀδελφοὺς αὐτοῦ, ἰούδας δὲ ἐγέννησεν τὸν φάρες καὶ τὸν ζάρα ἐκ τῆς θαμάρ, φάρες δὲ ἐγέννησεν τὸν ἑσρώμ, ἑσρώμ δὲ ἐγέννησεν τὸν ἄράμ, ἄράμ δὲ ἐγέννησεν τὸν ἀμιναδάβ, ἀμιναδάβ δὲ ἐγέννησεν τὸν ναασσών, ναασσών δὲ ἐγέννησεν τὸν σαλμών, σαλμών δὲ ἐγέννησεν τὸν βόες ἐκ τῆς ῥαχά (Regular)

βίβλος γενέσεως ἰησοῦ χριστοῦ υἱοῦ δαυὶδ υἱοῦ ἀβραάμ. ἀβραάμ ἐγέννησεν τὸν ἰσαάκ, ἰσαάκ δὲ ἐγέννησεν τὸν ἰακώβ, ἰακώβ δὲ ἐγέννησεν τὸν ἰούδαν καὶ τοὺς ἀδελφοὺς αὐτοῦ, ἰούδας δὲ ἐγέννησεν τὸν φάρες καὶ τὸν ζάρα ἐκ τῆς θαμάρ, φάρες δὲ ἐγέννησεν τὸν ἑσρῶμ, ἑσρῶμ δὲ ἐγέννησεν τὸν ἄράμ, ἄράμ δὲ ἐγέννησεν τὸν ἀμιναδάβ, ἀμιναδάβ δὲ ἐγέννησεν

τὸν ναασσών, ναασσών δὲ ἐγέννησεν τὸν σαλμών, σαλμών δὲ 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, wyrobijmy 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 Medium)

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.



3. Getting Started with Junicode

Junicode comes in two flavors—static and variable. Static fonts are the ones users are most familiar with, each font file supplying a single set of outlines that do not change except in size. By contrast, a single [variable font](#) file stores a set of outlines that can morph in various ways—for example, becoming bolder or lighter, narrower or wider, and sometimes undergoing more complex transformations. The static version of Junicode consists of thirty-eight font files, each providing a distinct variation of the font’s style; the variable version consists of only two (one each for roman and italic), but those two font files are capable of much more than the static version’s thirty-eight.

Because the static and variable versions of Junicode are differently named (“Junicode” and “Junicode VF”), both can be installed on the same system. However, you should choose one or the other for any particular project. Choose the static version if the application you are using does not yet support variable fonts. Such applications include Microsoft Word, Apple Pages, Quark Xpress, Google Docs, Affinity Publisher, and most flavors of T_EX (except for LuaT_EX—see below). Another reason to choose the static version is its familiarity: if you don’t need the advanced capabilities of the variable version, it is perfectly all right to stick with what you know.

All major web browsers (including browsers for mobile devices) support variable fonts, and there are good reasons to choose the variable version of Junicode for any web project. The greatest reason to go with the variable version is to speed the loading of web pages: users will never have to download more than two font files (the size of which can be radically reduced via subsetting, explained in Section 9 of this Manual). Additionally, however, variable fonts can make a page of text more dynamic and visually interesting. See Mozilla’s [Variable](#)

[Fonts Guide](#) for more information about using variable fonts on the web.

A growing number of desktop applications support variable fonts. Use the variable version of Junicode in Adobe InDesign (always with the World-Ready Paragraph Composer).¹ LibreOffice has supported variable fonts since version 7.5 (2023). LuaTeX has excellent support for variable fonts: make sure your TeX installation is up to date (since in recent years support for variable fonts has improved with every release), and always choose “harf” mode in your font-selection code. For an example of font-selection code for a variable font, see the file [language_samples.tex](#) in the “docs” directory of the GitHub Junicode site (you are welcome to copy and modify this code). A number of graphical design apps also support variable fonts, including Adobe Illustrator, PhotoShop, Figma, Sketch, and CorelDRAW.

The static version of Junicode has five weights and five widths, which are combined in many ways for a total of nineteen styles in both roman and italic. It is not necessary to install all of these; in fact, your life will be simplified (font menus easier to navigate) if you make a selection. You will probably want the traditional Regular, Bold, Italic, and Bold Italic fonts, but you should survey the styles displayed in the Specimen section of this booklet, choose the ones that look best to you, and install only those. A reasonable selection for many users will include the traditional four styles for main text, several SemiExpanded styles for footnotes, and SemiCondensed for titles.

Junicode’s static fonts come in two types, TrueType (files with a .ttf extension) and CFF (files with an .otf extension). These are functionally identical, but they may look subtly different on your computer’s screen because of the different technologies used to render glyphs. Choose the one you find most appealing.

With around 5,000 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

¹The choice of a Composer is well hidden in the “Justification” section of the “Paragraph Style Options” dialog. Use of the default “Adobe Paragraph Composer” with Junicode VF may cause InDesign to crash or otherwise misbehave. To prevent crashes when using the variable version of Junicode, it is also advisable to delete InDesign’s preferences when launching the program after a font upgrade. To do this, press Shift-Alt-Control (on Windows) or Shift-Control-Option-Command (on the Mac) all together, *immediately* after clicking to launch InDesign.

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 [MUF^I 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 *MUF^I Character Recommendation* or, alternatively, the *Junicode List of Encoded Characters* (packaged with the font) you can find out the **code points**² 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).³ But medieval manuscripts 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 **m̈**; **y** + ◌̄ (U+0304) + ◌̇ (U+0306) will make **ÿ**; **e** + ◌̣ (U+0323) + ◌̈ (U+1DE0) will make **ē**.

² 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 “3” (Middle English yogh) is U+021D.

³ Both Windows and the Mac OS come with international keyboards that make it easy to type special letters and diacritics. To find out how to enable these, search online for “Mac OS International Keyboard” or “Windows International Keyboard.”

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 Lua^AT_EX and X_YL^AT_EX. Adobe InDesign supports OpenType features only partially, but all features can be accessed via Roland Dreger’s [open-type-features](#) script, and InDesign provides access to all of Junicode’s characters via its “Glyphs” dialog.

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 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) instead of the modern Icelandic (þð), and in ancient Greek you will be able to type accents after vowels instead of looking up the codes for hundreds of polytonic vowel + accent combinations. 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, or wish to request an enhancement or other change, please open an [issue](#) at the font’s [development site](#).

4. Feature Reference

4.1. 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]¹ 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 major web browsers, LibreOffice, X_YTeX, 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:

¹ 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 X_YLaTeX 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.

- 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 + 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: **é** (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 **ő**, 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 positions, e.g. **ö** 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 in the manner prescribed by MUFI.
- 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:
 - **ſ** U+E8AF. This is a ligature of long **s** and **l** with stroke, but there are no base characters with this style of stroke.
 - **ŭŰ** U+EFD8 and U+EFD9. MUFI lists these as ligatures (corresponding to the historic ligatures **uU**), 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 **aA**.
 - **ṛṚ** U+EBE7 and U+EBE6, for the same reason.

- **Ǿ** 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.

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.

4.2. Case-Related Features

4.2.1. **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 **ť** to **ṭ**.

4.2.2. **c2sc** – Small Capitals from Capitals

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

Note: The variants of Ñ (U+014A—see [Other Latin Letters](#) below) have no lowercase equivalents. Their small capital forms can be accessed only through this feature.

4.2.3. **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 a number of other letters, but fewer than half of Junicode’s small caps have petite cap equivalents. *klmnop* → *KLMNOP*.

4.2.4. **c2pc** – Petite Capitals from Capitals

Produces petite capitals from capitals. Use with **pcap** to convert mixed-case texts to petite capitals. *PQRST* → *PQRST*.

Note: The variants of Ñ (U+014A—see [Other Latin Letters](#) below) have no lowercase equivalents. Their petite capital forms can be accessed only through this feature.

4.2.5. **case** – Case-Sensitive Forms

Produces combining marks that harmonize with capital letters: Š, Œ, etc. Use of this feature reduces the likelihood that a combining mark will collide with a glyph in the line above. Some applications turn this feature on automatically for runs of capitals, and precomposed characters (e.g. É U+00C9, Ū U+016A) already use case-appropriate combining marks. This feature also changes oldstyle to lining figures, since these harmonize better with uppercase letters.

4.3. Numbers and Sequencing

4.3.1. **frac** – Fractions

Applied to a slash and surrounding numbers, produces fractions with diagonal slashes. 6/9 becomes $\frac{6}{9}$, 16/91 becomes $\frac{16}{91}$.

4.3.2. `numr` – Numerators

Changes numbers to those suitable for use on the left/upper side of fractions with diagonal stroke (U+2044). This can be used, with `dnom`, to manually construct fractions, but for most users `frac` will be a better solution.

4.3.3. `dnom` – Denominators

Changes numbers to those suitable for use on the right/lower side of fractions with diagonal stroke (U+2044). This can be used, with `numr`, to manually construct fractions, but for most users `frac` will be a better solution.

4.3.4. `nalt` – Alternate Annotation Forms

Produces letters and numbers circled, in parenthesis, or followed by periods, as follows:

`nalt[1]`, circled letters or numbers: (a) (b) . . . (z); (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: 0 1 2 3 . . . 20.

`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).

4.3.5. `tnum` – Tabular Figures

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

4.3.6. `onum` – Oldstyle Figures

Junicode’s default figures are oldstyle and proportional, harmonizing with lowercase characters: 0123456789. Use this feature to switch temporarily to oldstyle figures in a context where `lnum` is active.

4.3.7. `pnum` – Proportional Figures

Unicode’s default figures are proportionally spaced: unlike tabular figures, they are not all the same width: 0123456789. Use this feature to switch temporarily to proportional figures in a context where `tnum` is active.

4.3.8. `lnum` – Lining Figures

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

4.3.9. `zero` – Slashed Zero

Produces slashed zero in all number styles, including superscripts, subscripts, and fractions made with `frac`: 0 0 0 0, ¹⁰/₃₀.

4.3.10. `ss09` – Alternate Figures

In the manner of old typefaces, Unicode’s default number one is shaped like a small capital I and its zero is a plain ring. This feature provides more modern-looking figures: 01. Only oldstyle figures are affected by this feature.

4.4. Superscripts and Subscripts

4.4.1. `sups` – Superscripts

Produces superscript numbers and letters. Superscript numbers are in one of two styles: oldstyle proportional (from oldstyle numbers) and lining tabular (from lining numbers). 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.






























4.4.2. subs – Subscripts

Produces subscript numbers. Only produces oldstyle proportional and lining tabular figures: 2345 8901.

4.5. Ornaments

4.5.1. 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:

a, 1		i, 9		q, 17		y, 25	
b, 2		j, 10		r, 18		z, 26	
c, 3		k, 11		s, 19		A, 27	
d, 4		l, 12		t, 20		B, 28	
e, 5		m, 13		u, 21		C, 29	
f, 6		n, 14		v, 22			
g, 7		o, 15		w, 23			
h, 8		p, 16		x, 24			

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.

4.5.2. Lady Junicode

Lady Junicode cannot be produced by an OpenType feature, believing that it would be vulgar to make herself so accessible. She has, indeed, commanded that the author of this document not publish her code point, located in one of the more private corners of the Private Use Area. She has, however, given permission to publish her miniature:



If you encounter her while adventuring in her domains, greet her respectfully, and she will welcome you graciously.

4.6. Alphabetic Variants

4.6.1. cv01–cv52 – Basic Latin Variants

These features also affect small cap (`smcp`) and underdotted (`ss07`) forms, where available. Variants in **magenta** are also available via `ss06` “Enlarged Minuscules.” Use the `cvNN` features instead of `ss06` when you want to substitute an enlarged minuscule for a capital (or, less likely, a lowercase) letter everywhere in a text.

Variant of	cvNN	Variants
A	cv01	1=Ɽ, 2=ⱦ, 3=Ⱨ, 4=a
a	cv02	1=ɑ, 2=ⱥ, 3=ⱦ, 4=a, 5=Ⱨ, 6=a, 7=ⱨ, 8=Ⱪ, 9=ⱪ, 10=Ⱬ
B	cv03	1=b, 2=ⱬ
b	cv04	1=b
C	cv05	1=Ɱ, 2=c
c	cv06	1=Ɐ, 2=Ɒ
D	cv07	1=Ⱳ, 2=d, 3=ⱴ
d	cv08	1=Ⱶ, 2=ⱶ, 3=ⱷ, 4=d, 5=ⱸ (for 1, see also sso2)
E	cv09	1=ⱺ, 2=ⱻ, 3=e, 4=ⱽ
e	cv10	1=Ȿ, 2=e, 3=Ɀ, 4=e, 5=Ɀ
F	cv11	1=Ɀ, 2=f, 3=Ɀ
f	cv12	1=Ɀ, 2=Ɀ, 3=Ɀ, 4=Ɀ, 5=f, 6=Ɀ, 7=f, 8=Ɀ, 9=Ɀ
G	cv13	1=Ɀ, 2=Ɀ, 3=Ɀ, 4=g
g	cv14	1=Ɀ, 2=Ɀ, 3=g, 4=g, 5=g, 6=g, 7=g, 8=g, 9=Ɀ
H	cv15	1=Ɀ, 2=h, 3=Ɀ
h	cv16	1=Ɀ, 2=h, 3=h

Variant of	cvNN	Variants
I	cv17	1=İ, 2=I, 3=i
i	cv18	1=ı, 2=İ, 3=ı̇, 4=ij, 5=j, 6=i*
J	cv19	1=Ĵ, 2=j
j	cv20	1=J, 2=j̇, 3=j̇, 4=j̇
K	cv21	1=k
k	cv22	1=k, 2=k, 3=k̇, 4=k̇, 5=k̇, 6=k̇
L	cv23	1=l
l	cv24	1=ł, 2=l̇, 3=l̇, 4=l̇, 5=l̇, 6=l̇
M	cv25	1=Ṁ, 2=Ṁ, 3=Ṁ, 4=m
m	cv26	1=m, 2=ṁ, 3=ṁ, 4=ṁ
N	cv27	1=Ṅ, 2=n
n	cv28	1=ñ, 2=Ṅ, 3=ṅ, 4=ṅ, 5=ṅ
O	cv29	1=O, 2=O
o	cv30	1=O, 2=o
P	cv31	1=Ṗ, 2=p
p	cv32	1=p, 2=p**
Q	cv33	1=Ṗ, 2=q̇, 3=Q̇, 4=Q̇
q	cv34	1=q̇, 2=q̇
R	cv35	1=Ṗ, 2=Ṗ, 3=ṙ
r	cv36	1=ṙ, 2=Ṗ, 3=ṙ, 4=ṙ
S	cv37	1=Ṗ, 2=Ṗ, 3=ṡ, 4=Ṗ, 5=Ṗ, 6=Ṗ, 7=Ṗ
s	cv38	1=Ṗ, 2=Ṗ, 3=Ṗ, 4=Ṗ, 5=Ṗ, 6=Ṗ, 7=ṡ, 8=Ṗ, 9=Ṗ, 10=Ṗ, 11=Ṗ

Variant of	cvNN	Variants
T	cv39	1=Ț, 2=t
t	cv40	1=τ, 2=τ̇, 3=t
U	cv41	1=U
u	cv42	1=U
V	cv43	1=V
v	cv44	1=v̇, 2=v̇, 3=v̇, 4=v̇, 5=V
W	cv45	1=W, 2=W̃
w	cv46	1=W, 2=ŵ
X	cv47	1=X
x	cv48	1=ẋ, 2=ẋ, 3=ẋ, 4=ẋ, 5=X
Y	cv49	1=Ȳ, 2=y
y	cv50	1=y, 2=ẏ, 3=ẏ, y
Z	cv51	1=Z, 2=Z
z	cv52	1=ż, 2=ż, 3=Z

* **cv18[4]** changes ii to ij at the end of a word; **cv18[5]** changes i to j at the end of a word whether another i precedes or not. These variants are chiefly useful for roman numbers, but also for Latin words ending in -ii. The j produced by this feature is searchable as i.

** **cv32[2]** should be on in any edition or extensive quotation of the *Ormulum*. The feature produces a p that differs from the default only in the way it forms a double-p ligature with **hlig**: ð, not pp.

4.6.2. cv53-cv66, cv91 – 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**;

Variant of	cvNN	Variants
Ą (U+0104)	cv53	1=Ą, 2= Ą , 3=Ą̇
ą (U+0105)	cv54	1=ą, 2= ą
ǻ (U+A733)	cv55	1=ǻ, 2=ǻ, 3= ǻ
Æ (U+00C6)	cv56	1=Æ, 2= Æ
æ (U+00E6)	cv57	1=æ, 2= æ , 3=æ̇, 4=æ̈
Ɔ (U+A734)	cv58	1=Ɔ, 2= Ɔ , 3= Ɔ
ɔ (U+A735)	cv59	1=ɔ, 2=ɔ, 3= ɔ
ɹ (U+A739)	cv60	1=ɹ
đ (U+0111)	cv61	1=đ
Ǝ, ɛ ... (U+0118, U+0119)	cv62	1=Ǝ, ɛ ...; 2=Ǝ, ɛ ...
Ȝ, ȝ (U+021C, U+021D)	cv63	1=Ȝȝ, 2=Ȝȝ
Ŋ (U+014A)	cv91	1=Ŋ, 2=Ŋ
ø (U+A7C1)	cv65	1=ø, 2=ø, 3=ø, 4=ø
þ, þ (U+A765)	cv66	1=þ, þ

4.6.3. ss01 – Alternate thorn and eth

Produces Nordic thorn and eth (þǿ) when the language is English, and English thorn and eth (þǿ) with any other language, reversing the font’s ordinary usage. This also affects small caps, crossed thorn (þ þ—see also [cv66](#)), combining mark eth (U+1DD9, ǿ ǿ), and enlarged thorn and eth (see [ss06](#)). This feature depends on [loca](#) (Localized Forms), which in most applications will always be enabled.

4.6.4. ss02 – Insular Letter-Forms

Produces insular letter-forms, e.g. $\mathfrak{d}^{\text{f}}\mathfrak{g}\mathfrak{h}\mathfrak{i}\mathfrak{j}\mathfrak{k}$. The result is different, depending on whether the language is English or Irish (make sure the language for your document is set properly). In English text, capitals are not affected (except W), as these do not commonly have insular shapes in early manuscripts; instead,

enter the Unicode code points or use the Character Variant (**c_vNN**) features. In English text, **sso2** imitates the typography of the Old English passages of Hicke's *Thesaurus*, not the usage of Old English or Anglo-Latin manuscripts. In Irish texts, it imitates the distribution of insular characters but cannot imitate the style of particular scribal hands or typefaces.

4.6.5. **ss04 – High Overline**

Produces a high overline over letters used as roman numbers: $\overline{\text{cdijlmvx}} \overline{\text{CDI-JLMVXO}}$.

4.6.6. **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: $\overline{\text{bcdhijklmfvxp}}$.

4.6.7. **ss06 – Enlarged Minuscles**

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: $\text{abcde}^{\text{f}}\text{g}$. 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 minuscles 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.

4.6.8. **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. $\text{a}^{\text{b}}\text{c}^{\text{d}}\text{e}^{\text{f}}\text{g} \text{H}^{\text{I}}\text{J}^{\text{K}}\text{L}^{\text{M}}$. This also affects small caps, e.g. $\text{ABCDEF} \rightarrow \text{A}^{\text{B}}\text{C}^{\text{D}}\text{E}^{\text{F}}$. For letters without corresponding underdotted forms (e.g. U+A751, p), use U+0323, combining dot below ($\text{p}^{\text{}}$).

4.6.9. `ss08` – Contextual Long s

In English, French, and Latin text only, varies **s** and **f** according to rules followed by many early printers: *fports*, *effence*, *ftormy*, *difheveled*, *transfusions*, *flynefs*, *cliffside*. For this feature to work properly, `calt` “Contextual Alternates” must also be enabled (as it should be by default: see [Required Features](#) below). This feature does not work in Lua_T_EX, except in *harf* mode.

4.6.10. `ss16` – Contextual r Rotunda

Converts **r** to **ʀ** (lowercase only) following the most common rules of medieval manuscripts: *pʀiest*, *fʀimer*, *fʀost*, *oʀnament*. For this feature to work properly, `calt` “Contextual Alternates” must also be enabled (as it should be by default: see [Required Features](#) below). This feature does not work in Lua_T_EX, except in *harf* mode.

4.6.11. `salt` – Stylistic Variants (medieval capitals)

Junicode has two series of decorative capitals in medieval scripts. These affect only the letters A-Z and a-z. `salt[1]` provides rustic capitals, a script used for text in the late ancient and early medieval periods and for display until around the eleventh century: *ϜΛΖΙϞϞΙϞϞΙΝΣ ΛΙΒΥϞΟΣ ΔΥΧΙΤ ΚΑΡΤΗΛΑϞΟ ΤΡΙϞΜΡΗΟΣ*. `salt[2]` provides Lombardic capitals, a style used mainly for what are now called drop caps. Junicode’s Lombardic capitals are not suitable for running text, titles, or headers: **A B C D E F**

4.6.12. `cv68` – Variant of ? (U+0294, glottal stop)

ɿ=?

4.7. 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 αβγδεζ.

To set Greek properly (especially polytonic text) requires that both `loc1` 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) Α (U+1FBB), not Α (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 *prosogrammeni* (U+1FBE).²

For example, the sequence ω (U+03C9) ˘ (U+0313) ˘ (U+0301) Ͱ (U+0345) produces ω̂̂̂. Substitute capital Ω (U+03A9) and the result is Ω̂̂̂. 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 Ω̂̂̂̂.

² Some applications will automatically reorder sequences of letters and accents, sparing you the trouble of remembering the canonical order.

4.7.1. 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.

4.8. 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 (Ꞣ U+2E4C), *punctus elevatus* (ꞣ U+2E4E), *virgula suspensiva* (⁄ U+2E4A), triple dagger (‡ U+2E4B).

4.8.1. ss18 – Old-Style Punctuation Spacing

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

4.8.2. cv69 – Variants of ꝛꝛ (U+204A / U+2E52, Tironian nota)

1=ꝛꝛ, 2=ꝛꝛ.

4.8.3. 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).

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

1=· (*distinctio*).

4.8.5. **cv72** – Variants of , (comma)

1=,, 2=’.

4.8.6. **cv73** – Variants of ; (semicolon)

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

4.8.7. **cv74** – Variants of ∴ (U+2E4E, *punctus elevatus*)

1=∴, 2=⋮, 3=⋮̇, 4=? (*punctus flexus*).

4.8.8. **cv75** – Variant of ! (exclamation mark)

1=! (*punctus exclamativus*).

4.8.9. **cv76** – Variants of ? (question mark)

1=¿, 2=⸐, 3=⸑. Shapes of the *punctus interrogativus*.

4.8.10. **cv77** – Variant of ~ (ASCII tilde)

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

4.8.11. **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*.

4.8.12. **cv79** – Variants of / (slash)

1=℄, 2=℅. The first of these is Unicode, U+2E4E.

4.9. Abbreviations

4.9.1. **cv80** – Variant of ꝛ (U+A75D, run abbreviation)

1=ꝛ.

4.9.2. **cv81** – Variants of ꝣ (U+035B, combining zigzag above)

1=ꝣ̈, 2=ꝣ̇, 3=ꝣ̆. Positioning of the zigzag can differ from that of other combining marks, e.g. Ț, ț, đ. 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. đ̈, ț̈, k̈. Enable **case** for forms that harmonize with capitals (Ă Ț Ć Đ), **smcp** for forms that harmonize with small caps (Ț ț Ć đ).

4.9.3. **cv82** – Variants of spacing ʹ (U+A770)

1=ʹ, 2=ᵹ. **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.

4.9.4. **cv83** – Variants of ꝥ (U+A76B, “et” abbreviation)

1=ꝥ, 2=ꝥ₂. **[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.

4.10. Combining Marks

4.10.1. **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**[30], [31] and [32]) that lack MUFI code points but are used to form MUFI characters, and three more ([2], [33], and [34]) that have no code points in Unicode or MUFI.

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**.

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 you had to apply **smcp** “Small Caps” to **U+1DE8** ^b to get **cv84**[11] ^b, it would be impossible to produce the sequence **nāa** (or the reverse case **NĀA**) with the diacritic properly positioned.

1=̇	8=̈	15=̇	22=̈
2=̈	9=̈	16=̇	23=̈
3=̇	10=̈	17=̇	24=̈
4=̇	11=̈	18=̈	25=̈
5=̇	12=̈	19=̈	26=̈
6=̇	13=̈	20=̈	27=̈
7=̈	14=̈	21=̈	28=̈

29=◌ ^{az}	32=◌ ^q	35=◌ ³
30=◌ ^o	33=◌ ^o	36=◌ ³
31=◌ ^o	34=◌ ^o	37=◌ ³

4.10.2. cv67 – Spacing zigzag (variant of U+00AF, spacing macron)

A spacing version of ◌^z (U+035B, combining zigzag) appears in John Hutchins, *The History and Antiquities of the County of Dorset* (London, 1774). It is not in Unicode or MUFL. In the future this feature may be used, as necessary, for other spacing marks of abbreviation.

4.10.3. ss10 – Character Entities 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 **ē**. 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; → ◌ ^{az}	&_eogo; → ◌ ^ē	&_oogo; → ◌ ^o
&_an; → ◌ ^{an}	&_emac; → ◌ ^ē	&_oslash; → ◌ ^o
&_ar; → ◌ ^{ar}	&_idotl; → ◌ ^o	&_omac; → ◌ ^o
&_arsc; → ◌ ^{ar}	&_j; → ◌ ^j	&_orr; → ◌ ^{or}
&_as; → ◌ ^{as}	&_jdotl; → ◌ ^j	&_oru; → ◌ ^{or}
&_bsc; → ◌ ^b	&_ksc; → ◌ ^k	&_q; → ◌ ^q
&_dsc; → ◌ ^d	&_munc; → ◌ ^m	&_ru; → ◌ ^r

$\&z_sa; \rightarrow \text{ſ̃}$	$\&z_y; \rightarrow \text{ŷ}$
$\&z_tsc; \rightarrow \text{ſ̈}$	$\&z_thorn; \rightarrow \text{þ̈}$

For another function of Stylistic Set 10, see [Chapter 6, Entering Characters with Tags](#).

4.10.4. **ss20** – Low Diacritics

The MUFI recommendation includes a number of precomposed characters with base letters b, h, k, þ, ð and ð and combining marks ̃ (U+0363), ̆ (U+0364), ̇ (U+0304/[cv84\[15\]](#)), ̈ (U+0366), ̉ (U+036C), ̊ (U+1DE2), ̋ (U+036D), ̌ (U+036E), ̍ (U+1DE6) and ̎ (U+0304/[cv84\[19\]](#)). Instead of being positioned above ascender height as usual (e.g. h̃), the MUFI glyphs have the marks positioned above the x-height (e.g. h̆). 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 (ð̈) that accommodates the combining mark will be substituted for the normal base character (but this is not necessary for the other base characters). Examples: þ̈ , ð̈ , h̆ , k̆ , þ̇ , ð̇ .

ss20 is intended for use only with the diacritics and base characters listed here; other base+diacritic combinations may be disrupted by the feature. You should therefore apply it only to relevant base+diacritic pairs (e.g. via a style in InDesign or a word processor or a command in LuaTeX).

4.10.5. **cv85** – Variant of ̃ (U+1DD3, combining open a)

$\text{I}=\text{̃̃̃}$.

4.10.6. **cv86** – Variant of ̈ (U+1DD8, combining insular d)

$\text{I}=\text{̈̈̈}$.

4.10.7. **cv87** – Variant of ̊ (U+1DE3, combining r rotunda)

$\text{I}=\text{̊̊̊}$.

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

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

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

1=◌̄.

4.10.10. cv90 – Variants of ◌̄◌̄ (U+035E, combining double macron)

1=◌̄◌̄, 2=◌̄◌̄.

4.10.11. 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.

4.11. Currency and Weights

4.11.1. cv93 – Variants of ₪ (U+0044, generic currency sign)

1=Π	8=₧	15=ℓ	22=₣
2=×	9=Φ	16=£	23=₧
3=¥	10=₪	17=₧	24=₧
4=₪	11=₧	18=₧	25=₧
5=₪	12=₧	19=₧	26=₪
6=₪	13=₧	20=₪	27=₪
7=₪	14=₪	21=₪	

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).

4.11.2. cv94 – Variant of ₣ (U+2114)

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

4.11.3. 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.

4.11.4. cv96 – Variant of ¢ (U+20B0, German penny sign)

1=¢. Same as MUFI U+F2F5.

4.11.5. cv97 – Variant of ₯ (U+0192, florin)

1=₯. Same as MUFI U+F2E8.

4.11.6. cv98 – Variant of ʒ (U+2125, Ounce sign)

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

4.12. Gothic**4.12.1. ss19 – Latin to Gothic Transliteration**

Produces Gothic letters from Latin: Warþ þan in dagans jainans → ʞʌʀϥ ϥʌN IN ȡʌȦʌNS ȡʌINʌNS. In web pages, the letters will be searchable as their Latin equivalents.

4.13. Runic**4.13.1. 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` → `ƿiƿk ƿiƿðk ƿnƿƿ`.

4.13.2. `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. `ABCDEFGF` → `ƿ<ðmƿx`.

4.13.3. `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. `ABCDEFGF` → `ƿƿtƿƿ`.

4.13.4. `ss15` – Long Branch to Short Twig

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

4.13.5. `rtlm` – Right to Left Mirrored Forms

Produces mirrored runes, e.g. `ƿƿðmƿx` → `xƿmðƿƿ`. This feature cannot change the direction of text or reverse its order.

4.14. Ligatures and Digraphs

Old-style fonts typically contain a standard collection of ligatures (conjoined letters), including `fi`, `fl`, `ff`, `ffi`, and `ffl`. Most software will display these ligatures automatically (except Microsoft Word, for which they must be enabled explicitly). Junicode has a large number of ligatures, including the standard `f`-ligatures, a similar set for long `s`, e.g. `fl`, `ff`, `ffi`, but also more specialized forms like `ft`, `ft`, `fp` (the last two with `ss02` and `cv38` [11]), and a few more. Most of Junicode's

ligatures, however, are not automatic, but belong to the set of either Historic Ligatures or Discretionary Ligatures, both of which must be invoked explicitly. These are listed in the following sections.

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

af→f	de→ðe	gg→gg	o2→o2	fch→fch
aƿ→ƿ	ea→ea	gg→gg	Oʒ→Oʒ	ftr→ftr
ag→aȝ	ec→ec	go→go	oʒ→oʒ	fþ→fþ
al→d	eƿ→eƿ	gp→gp	PP→PP	fþ→fþ
an→an	eȝ→eȝ	gr→g	pp→pp	ττ→ττ
aN→N	em→em	Hr→h	pp→pp	UE→UE
ap→p	en→en	hr→h	Ps→P	ue→ue
ar→r	eo→eo	hf→h	pe→pe	UU→UU
aR→R	eþ→eþ	hf→h	ps→P	uu→uu
aþ→þ	eƿ→eƿ	kr→k	Psi→P	pp→p
bb→b	eτ→eτ	kf→k	psi→h	þr→þ
bg→bȝ	ex→ex	kf→k	qp→qp	þf→þ
bo→bo	ey→ey	ll→ll	q3/q3→q3/q3	ðð→ðð
ch→ch	fä→fä	na→n	qq→qq	þþ→þþ
ck→ck	gd→d	Nf→N	Q2→Q2	pp→p
ðð→ðð	gð→ð	oc→α	q2→q2	þþ→þþ
de→ð	gð→ð	O2→O2	fä→fä	þþ→þþ

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 Junicon, 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 **liga**.

Note: For the ligature **ſ** to work properly, U+017F **f** must be entered directly, not by applying an OpenType feature to **s**.

4.14.2. **dlig** – Discretionary Ligatures

Produces lesser-used ligatures: **ct**, **fp**, **str**, **st**, **tr**, **tt**, **ty**.

4.14.3. **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 **þau** from **pau** allows the word to be searched as “þau.” The digraphs covered by this feature are **a**, **æ**, **ai**, **æ**, **g**, **oo**, **w**, 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. **ð**), you may have to place U+034F COMBINING GRAPHEME JOINER (see [cv84](#) above) between the second element of the digraph and the combining mark.

4.15. 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/mkmm** (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.

5. 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 <i>À</i>	U+F0012 <i>ø</i>	U+F0032 <i>Ĉ</i>	U+F0044 <i>×</i>
U+F0001 <i>ą</i>	U+F0013 <i>o</i>	U+F0033 <i>Ċ</i>	U+F0045 <i>Ÿ</i>
U+F0002 <i>Ȧ</i>	U+F0014 <i>o</i>	U+F0034 <i>ĥ</i>	U+F0046 <i>Ʒ</i>
U+F0003 <i>ċ</i>	U+F0015 <i>q</i>	U+F0035 <i>Ĳ</i>	U+F0047 <i>Ⓐ</i>
U+F0004 <i>ċ</i>	U+F0016 <i>ȝ</i>	U+F0036 <i>Ĵ</i>	U+F0048 <i>Ⓑ</i>
U+F0005 <i>ɔ</i>	U+F0017 <i>v̄</i>	U+F0037 <i>Ⓗ</i>	U+F0049 <i>Ⓒ</i>
U+F0006 <i>ð</i>	U+F0018 <i>v̄</i>	U+F0038 <i>l</i>	U+F004A <i>Ⓓ</i>
U+F0007 <i>đ</i>	U+F0019 <i>x̄</i>	U+F0039 <i>Ⓚ</i>	U+F004B <i>Ⓔ</i>
U+F0008 <i>ḋ</i>	U+F001A <i>x̄</i>	U+F003A <i>Ⓛ</i>	U+F004C <i>Ⓕ</i>
U+F0009 <i>Ẹ</i>	U+F001B <i>ʔ</i>	U+F003B <i>Ⓜ</i>	U+F004D <i>Ⓖ</i>
U+F000A <i>ɛ̣</i>	U+F001C <i>Ȥ</i>	U+F003C <i>Ⓝ</i>	U+F004E <i>Ⓢ</i>
U+F000B <i>f</i>	U+F001D <i>ɀ</i>	U+F003D <i>Ⓞ</i>	U+F004F <i>Ⓜ</i>
U+F000C <i>ī</i>	U+F001E <i>Ȧ</i>	U+F003E <i>Ⓟ</i>	U+F0050 <i>Ⓚ</i>
U+F000D <i>j</i>	U+F001F <i>Ȧ</i>	U+F003F <i>Ⓠ</i>	U+F0051 <i>Ⓛ</i>
U+F000E <i>j̄</i>	U+F0020 <i>ɔ</i>	U+F0040 <i>Ⓡ</i>	U+F0052 <i>Ⓜ</i>
U+F000F <i>Ṗ</i>	U+F0021 <i>ɔ</i>	U+F0041 <i>Ⓢ</i>	U+F0053 <i>Ⓝ</i>
U+F0010 <i>m̄</i>	U+F0030 <i>Λ</i>	U+F0042 <i>Ⓣ</i>	U+F0054 <i>Ⓞ</i>
U+F0011 <i>o</i>	U+F0031 <i>Ⓑ</i>	U+F0043 <i>Ⓤ</i>	U+F0055 <i>Ⓟ</i>

U+Foo56 **Q**
U+Foo57 **R**





































U+Foo58 **S**
U+Foo59 **C**


U+Foo5A **U**
U+Foo5B **X**


U+Foo5C **Y**
U+Foo5D **Z**


6. Entering characters with tags



Any character in Junicode that can be rendered using a Character Variant (**cvNN**) feature can also be rendered using a sequence consisting of a base character and two Unicode tags—that is, characters from the Unicode tag range. This range duplicates the ASCII character set (which consists, roughly, of things that can be typed on a U.S. English keyboard), but the characters it contains are normally invisible. They are used as modifiers for a preceding character (in other fonts, usually a flag symbol). Junicode contains a partial collection of tag characters:

 U+E0061	 U+E0062	 U+E0063
 U+E0064	 U+E0065	 U+E0066
 U+E0067	 U+E0068	 U+E0069
 U+E006A	 U+E006B	 U+E006C
 U+E006D	 U+E006E	 U+E006F
 U+E0070	 U+E0071	 U+E0072
 U+E0073	 U+E0074	 U+E0075
 U+E0076	 U+E0077	 U+E0078
 U+E0079	 U+E007A	 U+E0030
 U+E0031	 U+E0032	 U+E0033
 U+E0034	 U+E0035	 U+E0036
 U+E0037	 U+E0038	 U+E0039

When creating web pages or XML documents, you can enter these using character entities (e.g. **󠁳**; for ); along the same lines, you can use the

`\char` command (e.g. `\char"0E0073`) when composing TeX documents. But such entities and commands are not generally available in editors and word processors, and entering the tags themselves can be tricky because they are not only invisible, but also ignored by the application (the cursor skips over them). Instead of attempting to enter the code points for tags directly, use character entities consisting of **an ampersand, two underscores, the tag character, and a semicolon**. These will yield visible tags. For example, typing `&__6;` will yield the tag .

To use the tag method of entering characters, first type the base character from the table below, then the two tags. For example, to make a square C () , enter one of these sequences:

Web page:	<code>C&#xE0073;&#xE0071;</code>
TeX:	<code>C\char"0E0073\char"0E0071</code>
Word processor:	<code>C&__s;&__q;</code> (appears as C  )

Then apply the OpenType feature `ss10` (Stylistic Set 10) to the passage or passages containing the tags or, if tags occur throughout, to the whole document. The tags will disappear and the preceding characters (the base characters) will be transformed.

For variants of the combining macron (U+0304) and perhaps other combining marks, it will often be necessary to place the COMBINING GRAPHEME JOINER (U+034F) between the macron and the base character that precedes it. This will prevent normalization (the shaping engine changing sequences of character + combining mark into precomposed characters), which interferes with the operation of tags.

Most of the two-tag sequences documented here are designed to be mnemonic. For example, the sequence in the example above stands for “square.” However, two-tag sequences are not capable of describing characters in any detail, and in some cases, where the number of variants is large (especially for period, combining macron, and currency), the tags are not descriptive at all, but rather an index (the same numbers used in the corresponding `cvNN` features).

These tags are compatible with Junicode’s other OpenType features, including the `cvNN` features, and can be mixed with them. They will not interfere with

the placement of combining marks, which can come either before or after the tag-pair. Use a **cvNN** feature when a variant should appear throughout the text, repeatedly in a particular passage (for example, a block quotation), or in a style. A tag sequence may be preferable for isolated forms or to override an OpenType feature.
























In the list below, records for each character are color-coded as follows:

green	MUFI characters with PUA code points
yellow	Other characters with PUA code points
blue	Characters with Unicode code points
red	Characters without code points















































Tags or **cvNN** features are usually to be preferred to PUA code points, which should be used only where accessibility and searchability are not issues (mainly in printed texts). Unicode code points can safely be entered directly. Junicore makes a few of them accessible via **cvNN** features and tags because it may often be desirable to associate these characters with their bases rather than the Unicode code points. For example, the insular T (T) is sure to be searchable as T if entered with the sequence **T**_{TAG}_{TAG}, but if entered as **U+A786** it may or may not be searchable as T, depending on the application.















































Characters without code points can only be entered via tags or OpenType features.

Base	Sequence	Result	Description / Code point
period	. _{TAG} ₁ _{TAG}	·	Distinctio / F1F8
period	. _{TAG} ₂ _{TAG}	,	Comma positura / F1E2
period	. _{TAG} ₃ _{TAG}	⁂	High comma positura / F1E3
period	. _{TAG} ₄ _{TAG}	;	Punctus versus / F1EA
period	. _{TAG} ₅ _{TAG}	.,	Punctus with comma positura / F1E4















































Base	Sequence	Result	Description / Code point
period	. 	∴	Colon with middle comma positura / F1E5
period	. 	∵	Two dots over comma positura / F1F2
period	. 	∴̇	Three dots over comma positura / F1E6
period	. 	∴̸	Punctus elevatus diagonal stroke / F1F0
period	. 	∴̶	Punctus elevatus with high back / F1FA
period	. 	∴̷	Punctus elevatus with onset / F1FB
period	. 	∴̸̸	Punctus flexus / F1F5
period	. 	∴̸̸̸	Punctus exclamativus / F1E7
period	. 	∴̸̸̸̶	Punctus interrogativus / F160
period	. 	∴̸̸̸̷	Punctus interrogativus horizontal tilde / F1E8
period	. 	∴̸̸̸̸	Punctus interrogativus lemniskate / F1F1
period	. 	∴̸̸̸̸̸	Wavy line / F1F9
period	. 	∴̸̸̸̸̸̸	Signe de renvoi / F1EC
period	. 	∴̸̸̸̸̸̸̸	Virgula suspensiva / F1F4
period	. 	/	Short virgula / F1F7
U+0304	̄ 	̸̄	Combining curly bar above / F1CC
U+0304	̄ 	̸̸̄	Combining diagonal macron
U+0304	̄ 	̸̄̇	Combining bar above with dot / F1C0
U+0304	̄ 	̸̸̸̄	Combining angular zigzag / F1C7
U+0304	̄ 	̸̸̸̸̄	Combining curly zigzag / F1C8
U+0304	̄ 	̸̸̸̸̸̄	Combining vertical tilde / 033E
U+0304	̄ 	̸̸̸̸̸̸̄	Combining ligature an / F036
U+0304	̄ 	̸̸̸̸̸̸̸̸̄	Combining ligature aN / F03A















































Base	Sequence	Result	Description / Code point
U+0304	̄ [0 TAG] [9 TAG]	ṛ	Combining ligature ar / F038
U+0304	̄ [1 TAG] [0 TAG]	Ṛ	Combining ligature aR / F130
U+0304	̄ [1 TAG] [1 TAG]	Ḃ	Combining B / F013
U+0304	̄ [1 TAG] [2 TAG]	Ḑ	Combining D / F016
U+0304	̄ [1 TAG] [3 TAG]	Ḕ	Combining e with macron / F136
U+0304	̄ [1 TAG] [4 TAG]	Ḗ	Combining e with ogonek / F135
U+0304	̄ [1 TAG] [5 TAG]	Ḣ	Combining dotless i / F02F
U+0304	̄ [1 TAG] [6 TAG]	Ḧ	Combining j / F030
U+0304	̄ [1 TAG] [7 TAG]	Ḥ	Combining dotless j / F031
U+0304	̄ [1 TAG] [8 TAG]	Ḭ	Combining K / F01C
U+0304	̄ [1 TAG] [9 TAG]	Ṣ	Combining uncial m / F01F
U+0304	̄ [2 TAG] [0 TAG]	Ṡ	Combining o with macron / F13F
U+0304	̄ [2 TAG] [1 TAG]	Ṣ	Combining o with ogonek / F13E
U+0304	̄ [2 TAG] [2 TAG]	Ṣ	Combining o with stroke / F032
U+0304	̄ [2 TAG] [3 TAG]	Ṡ	Combining q / F033
U+0304	̄ [2 TAG] [4 TAG]	Ṣ	Combining rum abbreviation / F040
U+0304	̄ [2 TAG] [5 TAG]	Ṣ	Combining T / F02A
U+0304	̄ [2 TAG] [6 TAG]	Ṣ	Combining y / F02B
U+0304	̄ [2 TAG] [7 TAG]	Ṣ	Combining thorn / F03D
U+0304	̄ [2 TAG] [8 TAG]	Ṣ	Combining ligature o r rotunda / F03E
U+0304	̄ [2 TAG] [9 TAG]	Ṣ	Combining ligature letter o rum / F03F
U+0304	̄ [3 TAG] [0 TAG]	Ṣ	Combining diagonal dieresis
U+0304	̄ [3 TAG] [1 TAG]	Ṣ	Combining dot and acute

Base	Sequence	Result	Description / Code point
U+0304	◌̄  	◌̇	Combining ogonek and dot
U+0304	◌̄  	◌̆	Combining narrow macron
U+0304	◌̄  	◌̈	Combining macron with serifs
U+0304	◌̄  	◌̋	Combining et sign
U+0304	◌̄  	◌̌	Combining spiritus asper sign
U+0304	◌̄  	◌̍	Attached subscript a
A	A  	Ɑ	Insular A / F201
A	A  	Ɱ	Square A / F13A
A	A  	Ɐ	A with diagonal stroke / E8DA
A	A  	a	Enlarged minuscule A
a	a  	ⱱ	Insular a / F200
a	a  	Ⱳ	Uncial a / F214
a	a  	ⱳ	Open a / F202
a	a  	a	Closed a / F203
a	a  	a	Neckless a / F215
a	a  	a	Enlarged minuscule a / EEE0
B	B  	b	Enlarged minuscule B
B	B  	Ⱶ	Insular B
b	b  	b	Enlarged minuscule b / EEE1
C	C  	ⱶ	Square C / F106
C	C  	c	Enlarged minuscule C
c	c  	ⱷ	c with curl / F198
c	c  	c	Enlarged minuscule c / EEE2


















































Base	Sequence	Result	Description / Code point
D	D  	ð	Enlarged minuscule insular D
D	D  	Ð	Insular D / A779
D	D  	d	Enlarged minuscule D
d	d  	ð	Insular d default form / A77A
d	d  	ḋ	Insular d second form
d	d  	Ḍ	Enlarged minuscule insular d / EEE4
d	d  	ḏ	d with curl / F193
d	d  	d	Enlarged minuscule d / EEE3
E	E  	Ǝ	Uncial closed E / F217
E	E  	Ǝ	Uncial E / F10A
E	E  	e	Insular E
E	E  	e	Enlarged minuscule E
e	e  	Ǝ	Uncial e / F218
e	e  	e	e with bar / F219
e	e  	e	High e with bar / F21A
e	e  	e	Enlarged minuscule e / EEE6
F	F  	ƒ	Insular F / A77B
F	F  	f	Enlarged minuscule F / EEE7
F	F  	ƒ	Enlarged minuscule insular F
f	f  	ƒ	Insular f / A77C
f	f  	ƒ	Insular f with dotted hooks / F21C
f	f  	ƒ	Semi-closed insular f / EBD5
f	f  	ƒ	Closed insular f / EBD6















































Base	Sequence	Result	Description / Code point
f	f _{TNG} _i _{TNG} _e	ƒ	Enlarged minuscule insular f / EEFF
f	f _{TNG} _n _{TNG} _a	f	Narrow f / F000B
f	f _{TNG} _c _{TNG} _u	ƒ	f with curl / F194
f	f _{TNG} _e _{TNG} _n	f	Enlarged minuscule f / EEE7
G	G _{TNG} _i _{TNG} _n	Ȣ	Insular G / A77D
G	G _{TNG} _o _{TNG} _r	Ū	Ormulum G with bar / A7D0
G	G _{TNG} _s _{TNG} _q	Γ	Square G / F10E
G	G _{TNG} _e _{TNG} _n	g	Enlarged minuscule G
g	g _{TNG} _i _{TNG} _n	Ȣ	Insular g / 1D79
g	g _{TNG} _o _{TNG} _r	ǵ	Ormulum g with bar / A7D1
g	g _{TNG} _s _{TNG} _c	g	Script g / 0261
g	g _{TNG} _c _{TNG} _u	gꝛ	g with curl / F196
g	g _{TNG} _e _{TNG} _n	g	Enlarged minuscule g / EEE8
g	g _{TNG} _c _{TNG} ₁	g	Closed g with separate loops / F21D
g	g _{TNG} _c _{TNG} ₂	g	Closed g with large lower loop / F21E
g	g _{TNG} _c _{TNG} ₃	g	Closed g with small lower loop / F21F
H	H _{TNG} _u _{TNG} _n	Ȣ	Uncial H / F110
H	H _{TNG} _e _{TNG} _n	h	Enlarged minuscule H
h	h _{TNG} _d _{TNG} _e	h	h with descender / F23A
h	h _{TNG} _a _{TNG} _u	h	h-shaped <i>autem</i> abbreviation / E8A3
h	h _{TNG} _e _{TNG} _n	h	Enlarged minuscule h / EEE9
I	I _{TNG} _d _{TNG} _a	I	I with dot above / 0130
I	I _{TNG} _d _{TNG} _e	I	I with descender / A7FE

Base	Sequence	Result	Description / Code point
I	I  	i	Enlarged minuscule I
i	i  	ı	Dotless i / 0131
i	i  	ŀ	Long I / F220
i	i  	ï	i with double bar / E8A1
i	i  	ij	i final form (when i precedes)
i	i  	j	i final form
i	i  	ï	Enlarged minuscule i / EEEA
J	J  	Ĵ	J with dot above / E15C
J	J  	j	Enlarged minuscule J
j	j  	ȷ	Dotless j / 0237
j	j  	ĵ	j with double bar / E8A2
j	j  	ĵ	j with dot accent / F000D
j	j  	ĵ	Enlarged minuscule j / EEEB
K	K  	k	Enlarged minuscule K
k	k  	k	Uncial k / F208
k	k  	k	Closed k, one / F221
k	k  	ķ	Closed k, two / F209
k	k  	ķ	k with curl / F195
k	k  	k	Enlarged minuscule k / EEEC
L	L  	l	Enlarged minuscule L
l	l  	ł	Descending l / F222
l	l  	l	Enlarged minuscule l / EEED
l	l  	ŀ	l with high stroke / A749















































Base	Sequence	Result	Description / Code point
l	l[][	l̥	l with high stroke ending with flourish / F000F
l	l[][	l	Insular l
M	M[][	ℳ	Uncial M with descender / F224
M	M[][	ℳ	Uncial M / F11A
M	M[][	ℳ	Epigraphic M / A7FF
M	M[][	m	Enlarged minuscule M
m	m[][	ℳ	Uncial m with descender / F23D
m	m[][	ℳ	Uncial m / F23C
m	m[][	ℳ	m with descender / F223
m	m[][	m	Enlarged minuscule m / EEEE
N	N[][	ℳ	N with descender / F229
N	N[][	n	Enlarged minuscule n / EEEF
n	n[][	ℳ	n with descender / F228
n	n[][	ℳ	n with bar / E7B2
n	n[][	ℳ	n with curl / F19A
n	n[][	n	Enlarged minuscule n / EEEF
n	n[][	ℳ	Small cap n with descender / F22A
n	n[][	ℳ	n with attached subscript a
O	O[][	o	Enlarged minuscule O
o	o[][	o	Enlarged minuscule o / EEFO
P	P[][	ℳ	Reversed P / A7FC
P	P[][	p	Enlarged minuscule P
p	p[][	p	Enlarged minuscule p / EEFO














































52 ENTERING CHARACTERS WITH TAGS





































Base	Sequence	Result	Description / Code point
p	p  	p	Alternate p for Ormulum
Q	Q  	q	Q with stem / F22C
Q	Q  	Q 	Q with long tail, one
Q	Q  	Q 	Q with long tail, two
Q	Q  	q	Enlarged minuscule Q
q	q  	q 	q with diagonal stroke / E8B4
q	q  	q	Enlarged minuscule q / EEf2
R	R  	ſ	Insular R / A783
R	R  	ꝛ	R rotunda / A75A
R	R  	r	Enlarged minuscule R
r	r  	ꝛ	Insular r / A782
r	r  	ꝛ	r rotunda / A75B
r	r  	ꝛ	r with curl / F19B
r	r  	r	Enlarged minuscule r / EEf3
S	S  	ſ	Insular S / A784
S	S  	ſ	Sigmoid S / A7D8
S	S  	s	Enlarged minuscule S
S	S  	ſ	S with diagonal stroke (Sanctus abbrev)
S	S  	ſ	Middle Scots S
s	s  	ſ	Insular s / A785
s	s  	s	Sigmoid s / A7D9
s	s  	f	Long s / 017F
s	s  	s	Enlarged minuscule s / EEf4

Base	Sequence	Result	Description / Code point
s	s  	ſ	Long s with descender / F127
s	s  	ſ̃	Long s with flourish / E8B7
s	s  	ſ̸	Long s with diagonal stroke / E8B8
s	s  	ſ̊	Middle Scots s
T	T  	Ƨ	Insular T / A786
T	T  	t	Enlarged minuscule T
t	t  	Ƨ	Insular t / A787
t	t  	Ƨ	t with curl / F199
t	t  	t	Enlarged minuscule t / EEF5
U	U  	u	Enlarged minuscule U
u	u  	u	Enlarged minuscule u / EEF7
V	V  	v	Enlarged minuscule V
v	v  	v̇	Enlarged minuscule v / EEF8
v	v  	Ƨ	v with diagonal stroke / E8BA
v	v  	Ƨ	v with bar / E74E
v	v  	v	Enlarged minuscule v / EEF8
v	v  	v̈	v with two bars / E8BC
W	W  	w	Enlarged minuscule W
W	W  	Ƨ	W Anglicana
w	w  	w	Enlarged minuscule w / EEF9
w	w  	Ƨ	w Anglicana
X	X  	x	Enlarged minuscule X
x	x  	x	Enlarged minuscule x / EEFA

Base	Sequence	Result	Description / Code point
x	x _{TAG} _{TAG}	Ƶ	x with long left leg / AB57
x	x _{TAG} _{TAG}	ƶ	x with diagonal stroke (upper left) / E8BD
x	x _{TAG} _{TAG}	Ʒ	x with diagonal stroke (lower right) / E8BE
x	x _{TAG} _{TAG}	Ƹ	x with two diagonal strokes (lower right) / E8CE
Y	Y _{TAG} _{TAG}	ƹ	Y with diagonal stroke (Hymnus abbrev) / E8DB
Y	Y _{TAG} _{TAG}	y	Enlarged minuscule Y
y	y _{TAG} _{TAG}	ƿ	y with main right stroke / F233
y	y _{TAG} _{TAG}	ƿ̄	y with bar / E77B
y	y _{TAG} _{TAG}	y	Enlarged minuscule y / EEFB
Z	Z _{TAG} _{TAG}	Ʒ	Visigothic Z / A762
Z	Z _{TAG} _{TAG}	z	Enlarged minuscule Z
z	z _{TAG} _{TAG}	Ʒ	Visigothic z / A763
z	z _{TAG} _{TAG}	Ʒ	Middle High German z / F238
z	z _{TAG} _{TAG}	z	Enlarged minuscule z / EEFC
U+0104	A _{TAG} _{TAG}	Ȧ	A with short diagonal stroke / F0000
U+0104	A _{TAG} _{TAG}	Ȧ	A with long diagonal stroke / F001E
U+0104	A _{TAG} _{TAG}	Ȧ	A with flourish / F0002
U+0105	a _{TAG} _{TAG}	Ȧ	a with short diagonal stroke / F0001
U+0105	a _{TAG} _{TAG}	Ȧ	a with long diagonal stroke / F001F
U+A733	aa _{TAG} _{TAG}	aa	Uncial aa
U+A733	aa _{TAG} _{TAG}	aa	Closed aa / EFA0
U+A733	aa _{TAG} _{TAG}	aa	Enlarged minuscule aa / EFDF
U+00C6	Æ _{TAG} _{TAG}	Ɔ	Neckless Æ / EFAE

Base	Sequence	Result	Description / Code point
U+00C6	Æ  	æ	Enlarged minuscule Æ
U+00E6	æ  	æ	Neckless æ / EFA1
U+00E6	æ  	æ	Enlarged minuscule æ / EAF1
U+00E6	æ  	æ	Open æ / F204
U+00E6	æ  	æ	Uncial æ
U+A734	AO  	Œ	Neckless AO / F205
U+A734	AO  	œ	Enlarged minuscule AO
U+A734	AO  	œ	Enlarged minuscule AO with smaller O
U+A735	ao  	œ	Enlarged minuscule ao / EFDE
U+A735	ao  	œ	Enlarged minuscule ao with smaller o / EAF2
U+A735	ao  	œ	Neckless ao / F206
U+A735	ao  	œ	Uncial ao
U+A739	av  	av	Neckless av / EFA2
U+0111	đ  	đ	d with flourish / F0007
U+0118	Ě  	Ě	E with centered ogonek
U+0118	Ě  	Ě	E with diagonal stroke / F0009
U+0118	Ě  	Ě	E with dot, acute, and centered ogonek
U+0118	Ě  	Ě	E with dot, acute, and diagonal stroke
U+0119	ě  	ě	e with centered ogonek
U+0119	ě  	ě	e with diagonal stroke / F000A
U+0119	ě  	ě	e with dot, acute, and centered ogonek
U+0119	ě  	ě	e with dot, acute, and diagonal stroke
&_eogo; ◌&_eogo;  		◌̊	Combining e with centered ogonek

Base	Sequence	Result	Description / Code point
&_eogo;	◌&_eogo;  	◌ ^e	Combining e with diagonal stroke
U+021C	3  	3	Yogh with flat top
U+021C	3  	Ʒ	Yogh with insular shape
U+021D	3  	ƶ	yogh with flat top
U+021D	3  	Ʒ	yogh with insular shape
U+014A	N  	Ŋ	Rounded N with low hook
U+014A	N  	Ŋ	Rounded N with baseline hook
U+A7C1	  ¹	o	Old Polish o with broken slash / F0011
U+A7C1	 	ø	Old Polish o with short slash / F0012
U+A7C1	 	o	Old Polish o with lower left slash / F0013
U+A7C1	 	o	Old Polish o with upper right slash / F0014
U+A765	þ  	þ	thorn with stroke with different slant / F149
U+A765/ENG	þ  	þ	thorn with stroke with different slant
U+0241	?  	ʔ	Alternate glottal stop / F001B
U+204A	ɀ  	ɀ	Tironian et sign later form / F001D
U+204A	ɀ  	ɀ	Tironian et sign later form with bar
U+2E52	ɀ  	ɀ	Tironian Et sign later form / F001C
U+2E52	ɀ  	ɀ	Tironian Et sign later form with bar
U+00AF	˘  	˘	Spacing zigzag
U+035E	◌   ◌	◌◌	Double macron with serifs
U+035E	◌   ◌	◌◌	Shorter double macron with serifs
U+00B7	·  	·	Distinctio / F1F8
comma	,  	,	Comma positura / F1E2

Base	Sequence	Result	Description / Code point
comma	,  	’	High comma positura / F1E3
semicolon	;	;	Punctus versus / F1EA
semicolon	;	.,	Punctus with comma positura / F1E4
semicolon	;	;	Colon with middle comma positura / F1E5
semicolon	;	;	Two dots over comma positura / F1F2
semicolon	;	;	Three dots over comma positura / F1E6
U+2E4E	⋈  	⋈	Punctus elevatus diagonal stroke / F1F0
U+2E4E	⋈  	⋈	Punctus elevatus with high back / F1FA
U+2E4E	⋈  	⋈	Punctus elevatus with onset / F1FB
U+2E4E	⋈  	⋈	Punctus flexus / F1F5
exclam	!  	!	Punctus exclamativus / F1E7
question	?  	?	Punctus interrogativus / F160
question	?  	?	Punctus interrogativus horizontal tilde / F1E8
question	?  	?	Punctus interrogativus lemniskate / F1F1
asciitilde	~  	~	Wavy line / F1F9
asterisk	*  	∴	Signe de renvoi / F1EC
slash	/  	/	Virgula suspensiva / F1F4
slash	/  	/	Short virgula / F1F7
U+A75D	ꝛ  	ꝛ	Alternate run abbreviation / F0016
U+035B	◌  	◌	Combining angular zigzag / F1C7
U+035B	◌  	◌	Combining curly zigzag / F1C8
U+035B	◌  	◌	Combining vertical zigzag (tilde) / 033E
U+A770	ꝥ  	ꝥ	Baseline spacing us abbreviation / F1A6

Base	Sequence	Result	Description / Code point
U+A770	ʹ ^a _{TAG} ² _{TAG}	Ɔ	Uppercase us abbreviation / F1A5
U+A76B	ʒ ^a _{TAG} ¹ _{TAG}	;	Semicolon-like et abbreviation / F1AC
U+A76B	ʒ ^a _{TAG} ² _{TAG}	₳	Subscript et abbreviation
U+1DD3	ˆ ^a _{TAG} ¹ _{TAG}	ā̂	Combining flattened a with macron / F1C1
U+1DD8	ˆ ^a _{TAG} ¹ _{TAG}	ḃ	Alternate combining insular d / F0005
U+1DE3	ˆ ^a _{TAG} ¹ _{TAG}	ō̂	Combining ur abbreviation lemniskate / F1C2
U+00E4	ä ^d _{TAG} ⁱ _{TAG}	ä̈	a with diagonal dieresis / E8D5
U+00F6	ö ^d _{TAG} ⁱ _{TAG}	ö̈	o with diagonal dieresis / E8D7
U+0308	ö ^d _{TAG} ⁱ _{TAG}	ö̈	Combining diagonal dieresis
U+0305	̄ ^a _{TAG} ¹ _{TAG}	ō̄	Combining horizontal stroke with dot / F1C0
U+032E	◌ ^a _{TAG} ¹ _{TAG} ◌	◌◌◌◌	Breve below three letters / F1FC
U+00A4	⊠ ⁰ _{TAG} ¹ _{TAG}	ℙ	Latin as libralis sign / F2E0
U+00A4	⊠ ⁰ _{TAG} ² _{TAG}	℥	Latin small capital letter x with bar / F2E2
U+00A4	⊠ ⁰ _{TAG} ³ _{TAG}	℥	Latin small capital letter y with bar / F2E3
U+00A4	⊠ ⁰ _{TAG} ⁴ _{TAG}	℥	Latin small capital letter d with slash / F2E4
U+00A4	⊠ ⁰ _{TAG} ⁵ _{TAG}	℥	Pharmaceutical dram sign / F2E6
U+00A4	⊠ ⁰ _{TAG} ⁶ _{TAG}	℥	Ecu sign / F2E7
U+00A4	⊠ ⁰ _{TAG} ⁷ _{TAG}	℥	Floren sign with loop / F2E8
U+00A4	⊠ ⁰ _{TAG} ⁸ _{TAG}	℥	Groschen sign / F2E9
U+00A4	⊠ ⁰ _{TAG} ⁹ _{TAG}	℥	Helbing sign / F2FB
U+00A4	⊠ ¹ _{TAG} ⁰ _{TAG}	℥	Krone sign / F2FA
U+00A4	⊠ ¹ _{TAG} ¹ _{TAG}	℥	Dutch libra sign / F2EA
U+00A4	⊠ ¹ _{TAG} ² _{TAG}	℥	French libra sign / F2EB

Base	Sequence	Result	Description / Code point
U+00A4	☐ _{TNG} ₁ ☐ _{TNG} ₃	℔	Italian libra sign / F2EC
U+00A4	☐ _{TNG} ₁ ☐ _{TNG} ₄	℔	Flemish libra sign / F2ED
U+00A4	☐ _{TNG} ₁ ☐ _{TNG} ₅	℔	Lira nuova sign / F2EE
U+00A4	☐ _{TNG} ₁ ☐ _{TNG} ₆	℔	Lira sterlina sign / F2EF
U+00A4	☐ _{TNG} ₁ ☐ _{TNG} ₇	℔	Old mark sign / F2F0
U+00A4	☐ _{TNG} ₁ ☐ _{TNG} ₈	℔	Old flourish mark sign / F2F1
U+00A4	☐ _{TNG} ₁ ☐ _{TNG} ₉	m℔	Marked small letter m sign / F2F2
U+00A4	☐ _{TNG} ₂ ☐ _{TNG} ₀	℔	Flourished small letter m sign / F2F3
U+00A4	☐ _{TNG} ₂ ☐ _{TNG} ₁	℔	Pharmaceutical obelus sign / F2F4
U+00A4	☐ _{TNG} ₂ ☐ _{TNG} ₂	℔	Penning sign / F2F5
U+00A4	☐ _{TNG} ₂ ☐ _{TNG} ₃	℔	Old Reichstaler sign / F2F6
U+00A4	☐ _{TNG} ₂ ☐ _{TNG} ₄	℔	German schilling sign / F2F7
U+00A4	☐ _{TNG} ₂ ☐ _{TNG} ₅	℔	German script schilling sign / F2F8
U+00A4	☐ _{TNG} ₂ ☐ _{TNG} ₆	℔	Scudi sign / F2F9
U+00A4	☐ _{TNG} ₂ ☐ _{TNG} ₇	℔	Script ounce sign / F2FD
U+2114	℔ _{TNG} _a ☐ _{TNG} ₁	℔	French libra sign / F2EB
U+00A3	℔ _{TNG} _a ☐ _{TNG} ₁	℔	Dutch libra sign / F2EA
U+00A3	℔ _{TNG} _a ☐ _{TNG} ₂	℔	French libra sign / F2EB
U+00A3	℔ _{TNG} _a ☐ _{TNG} ₃	℔	Italian libra sign / F2EC
U+00A3	℔ _{TNG} _a ☐ _{TNG} ₄	℔	Flemish libra sign / F2ED
U+00A3	℔ _{TNG} _a ☐ _{TNG} ₅	℔	Lira nuova sign / F2EE
U+00A3	℔ _{TNG} _a ☐ _{TNG} ₆	℔	Lira sterlina sign / F2EF
U+20B1	℔ _{TNG} _a ☐ _{TNG} ₁	℔	Penning sign / F2F5

Base	Sequence	Result	Description / Code point
U+0192	f [̄] _̇ _̈	ƒ	Floren sign with loop / F2E8
U+2125	3 [̄] _̇ _̈	㉟	Script ounce sign / F2FD

¹Junicode entities (like &__a;) following U+A7C1 (Old Polish o) are not properly displayed or resolved for technical reasons related to U+A7C1 having very recently been added to Unicode. However, tags entered directly (as here) or via HTML/XML entity references and TeX commands will still work, and the Junicode entity will work once applications are updated with the latest additions to the Unicode standard.

7. Transcribing records

This chapter provides guidance for persons transcribing records (laws and other public documents) in the style of Charles Trice Martin's *The Record Interpreter*, *Statutes of the Realm*, and similar guides and editions. Unlike most editions of early texts, these retain (or recommend retaining) the capitalization, punctuation, and abbreviations of their manuscript sources.

7.1. A preliminary note on transcription

Here are a few observations, based on a long career as a scholarly editor of medieval and eighteenth-century texts.

Before embarking on the task of transcribing an old document, ask yourself what value you want to add to the document as it already exists, because different kinds of transcription add different kinds of value. The kind of transcription that adds the least is that which aims at the exact *visual* reproduction of a document. A transcript is not a facsimile: it needs to do something that a photograph can't do.

Converting a document from visual image to Unicode-encoded text adds a good bit of value all by itself, but only if done with due regard for the semantics of Unicode characters. Every Unicode character has a meaning, and that meaning is a help to readers. Using the wrong character is a hinderance to readers, even if it *looks* right.

For example, in transcribing a Middle English text, you may decide that the Unicode ezh (ʒ, U+0292) looks more like the yogh in your source than the Unicode yogh (ȝ, U+021D) and therefore decide to use it for yogh. But the ezh is not a yogh! It is a character in the International Phonetic Alphabet and a

letter in the alphabets of several minor languages—but not a letter in the Middle English language. If you use it where the yogh is called for, it will make your text less accessible and less searchable. Indexing, concordance and bibliographical programs may be misled by it; screen readers will misinterpret it. To solve one problem (that of visual representation), you may well have introduced a host of far more serious problems.

Fortunately, Junicode offers a solution for this particular problem. The OpenType feature **cv63** substitutes for the yogh a character that *looks* like the ezh but is semantically a yogh and therefore will be handled correctly by applications. But neither Junicode nor any other font can solve every problem of this kind. Sometimes you will have to call to mind the important principle stated above: *A transcript is not a facsimile*. It is much more important that it should have the same *meaning* as the original than that it should have the same *look*.

This document concerns the transcription of texts in Latin (and to some extent, other archaic languages, e.g. Old and Middle English, Old French). It is long-standing custom, when transcribing certain kinds of documents, to retain marks of abbreviation—for example, the **ꝑ** you may find in a manuscript or printed edition representing the word *propterea*. This is okay—and Junicode can help with the task. But when dealing with the abbreviations, punctuation, and diacritics of an old text it is more important than ever that you use semantically correct characters for your transcription, as this will help readers who already face significant challenges.

For example, the abbreviation **ꝑ** as printed here consists of an underlying sequence of Unicode characters: **ꝑ** (U+A753, the common abbreviation for *pro*) + **ꝑ** + **◌̣** (U+0363, the combining small *a*). The OpenType feature **hlig** (Historical Ligatures) has been applied to this sequence, changing its appearance but not its underlying value. That underlying value is intelligible to computer applications in the sense that they can recognize each character.

This doesn't mean, though, that computer programs can correctly interpret **ꝑ** as *propterea*. Many (probably *most*) Latin abbreviations are ambiguous: this one, for example, can mean *propterea* or *propria*. Some abbreviations (most notoriously **◌̣** U+035B) can mean many things, depending on context. It takes a human being with a knowledge of Latin to interpret them correctly.

So another way you can add value in your transcript is by interpreting ab-

breviations like ꝥ^a and supplying expansions of them. Fortunately, systems for representing texts often offer ways to handle this task gracefully. For example, in a TEI (Text Encoding Initiative) text, you would use this construction:

```
<choice>
  <abbr rend="hlig">ꝥa</abbr>
  <expansion>propterea</expansion>
</choice>
```

This kind of structure can be approximated in HTML, with supporting CSS and scripting to allow readers to choose between a “diplomatic” version, with unexpanded abbreviations, and a “reading” version, with expanded abbreviations and perhaps other amenities, such as modern punctuation and capitalization.

There are other ways to add value to a transcript—for example, by correcting errors, annotating the content, or writing textual notes. Each of these operations takes your transcript farther from the facsimile and closer to the edition.

7.2. 1. Common combining marks

A **combining mark** is a character that combines with another character (called the **base**) to form a character with accent (e.g. é) or an abbreviation (e.g. ꝥ for *prae*). Unicode and the Medieval Unicode Font Initiative (MUFI) offer code points for many precomposed combinations of base + combining mark, but it is also possible to place any mark over any base character by entering first the base and then the combining mark. It is also possible to place a combining mark over another combining mark. For example, to produce q̄̇, enter this sequence: q (U+0071) + U+0363 + U+0304.

JunICODE 2 contains many variants of combining marks: for example the curly zigzag ȝ is a variant of Unicode’s angular zigzag ȝ (U+035B), produced by applying the OpenType feature **cv81[2]** to **both the base character and the combining mark**. Sometimes the combination of base + combining mark + OpenType feature will not produce the desired effect. When this happens, place U+034F (a special invisible combining mark, included in Unicode for exactly this purpose) between the base and the (visible) mark.

- a. For a straight stroke over any letter, use the COMBINING MACRON (U+0304):

ōnis *omnis*; oñis *omnis*; dāpna *dampna*; dampā *dampna*.

The combining macron can also be applied above superscripts and combining marks. Apply the OpenType feature `cv84[33]` for a narrower macron:

antiquā *antiquam*; q̄ *quam*.

For the superscript *a*, use the OpenType feature `sup`s (see r. below).

b. For a straight stroke through a tall letter, use the COMBINING SHORT STROKE OVERLAY (U+0335): **ƒđ†**. But Unicode also has precomposed versions of **đ**, **†** and other characters with stroke, e.g. **đ** (U+0111), **†** (U+019A).

c. For **~** above any character, use the COMBINING TILDE (U+0303):

ã *ac*, *apud*; ã *alias*.
 dñs *dominus*; cařina *carmina*; fčis *factis*.
 pōita *posita*.

d. For **~** through a vertical stroke, use the TILDE OVERLAY (U+0334): **†đ** (U+0303) would be positioned above the letter, e.g. **ĩ**, **đ̃**. For the ligatures **††**, **†đ**, and **đđ**, type the sequence for **†**, etc. twice.

e. For the tilde positioned above two letters, use COMBINING DOUBLE TILDE (U+0360) between the letters. It is automatically repositioned to clear tall characters: **cõ tõ dõ ol̃**. The same is true of DOUBLE BREVE (U+035D) **cô dô**, DOUBLE MACRON (U+035E) **cō dō**, DOUBLE INVERTED BREVE (U+0361) **cŏ dŏ**, and DOUBLE CIRCUMFLEX (U+1DCD) **cô dô**.

f. The figure used to represent *er* (and other similar combinations) is a common medieval abbreviation which takes many forms. The semantically correct Unicode character is the COMBINING ZIGZAG (ᶑ, U+035B), but the best match in Junicon 2 for the figure as it appears in the *Record Interpreter* and the *Statutes* is a gothic variant of this, which MUFI encodes as U+F1C8 (the curly form zigzag). However, because for technical reasons many applications will not position the MUFI character correctly over the base, that code point should be avoided. The best way to access this variant is to apply `cv81[2]` to U+035B, as here:

deᶑbe *debere*; inᶑter *inter*; Prᶑ *ferrum*; gᶑno *generatio*; pᶑ; *prae*; serᶑve *servire*.

The curly form of the combining zigzag may be attached to any letter, and it may change shape depending on the letter it is attached to (including caps, for which use the `case` feature, and small caps: $\tilde{A}\tilde{B}\tilde{C}\tilde{D}$).

g. All letters a–z, and several others too, have combining forms. You may access these via their code points, when they are standard Unicode, via the `cv84` feature, or via Junicode’s special entity references. For details, see [4.10.3, Character Entities for Combining Marks](#).

q̇ *quo*; q̇ *qui*; quat̃ *quattuor*.

7.3. 2. Spacing characters

h. The symbol for *is*, *es* and a number of other abbreviations is the IS-SIGN (U+A76D):

forꝑ *foris*; omꝑ *omnes*; 9tꝑ *competentes*; infꝑ *infortunium*.

This character will sometimes ligature with the preceding letter. The italic version differs from the roman stylistically (*for^ℓ om^ℓ 9t^ℓ inf^ℓ*), but it will be intelligible to informed readers.

i. There are two characters for *-us* in Unicode: SPACING US U+A770 (do not confuse this with CON U+A76F) and COMBINING US U+1DD2. The *Record Interpreter* and *Statutes* appear to use only the spacing character:

iꝑiꝑ *ipsius*; ũsꝑ *uersus*; pꝑtea *postea*; pꝑ *post*.

j. The three-like sign is the ET SIGN (ꝓ, U+A76B, also used for *us*—do not confuse this with Middle English yogh: ȝ, U+021D):

quibꝓ *quibus*; licꝓ *licet*; sꝓ *sed*.

k. For *-rum* the Unicode RUM ROTUNDA (U+A75D) is like the one in MUFI/Junicode. The one in the *Record Interpreter* and *Statutes* is a late stylized version of this. Use U+A75D and apply OpenType feature `cv8o` to obtain the correct shape:

ããž *animarum*; cožpere *corrumpere*; beatož *beatorum*.

- l. For *cum*, *con*, etc. use SMALL LETTER CON (U+A76F):

ꝑꝑutus *computus*; ꝑa *contra*; ꝑnouit *cognouit*.

- m. For *per* (or sometimes *par* and other similar sequences), use P WITH STROKE U+A751:

ꝑsōa *persona*; ꝑꝑet *comparet*.

- n. For *pro*, use P WITH FLOURISH U+A753:

ꝑꝑeres *proceres*.

- o. For *prae*, *præ*, *pre*, there is no separate character; use a variant of the ZIGZAG (f. above) with **p**:

ꝑsēs *praesens*.

- p. For **q** with stroke through the descender, there are two Unicode points: U+A757 for a straight stroke, and U+A759 for a diagonal stroke (the *Record Interpreter* appears to use only the former, and neither is listed among the *Statutes* abbreviations):

ꝑ ꝑuod; ꝑd ꝑuid; ꝑbꝑ ꝑuibꝑ.

- q. For *quae*, *que*, use **q** followed by ET (U+A76B) with or without **hlig**: **qꝑ qꝑ**. For the semicolon-like ET sign (**q;**), use **cv83[1]**; for the subscripted version (which can also form a ligature via **hlig**), use **cv83[2]**: **qꝑ qꝑ**.

- r. All of the letters a–z are available in superscript form. Access with the **sup**s OpenType feature:

ꝑ^os *quos*; cⁱlo *circulo*; capⁱ *capituli*.

The basic Latin letters a–z have anchors that allow you to position combining marks over them (see a. above)

- s. Tironian ET sign ꝑ U+204A, cap ꝑ U+2E52. With **cv69[1]** ꝑꝑ; with **cv69[2]** ꝑꝑ.

- t. For *est*, use ÷ U+223B HOMOTHETIC. Use of a mathematical sign for this purpose is not ideal, but Unicode offers no better solution.

- u. For *tz* (Old French), use **z** U+01B6 Z WITH STROKE.
- v. For an abbreviation for *Rex*, use **R** U+211E or **R̄** U+211F.
- w. At least one edition uses a spacing version of the COMBINING ZIGZAG (f. above). Neither Unicode nor MUFI has a matching character: with Junicode, apply **cv67** to the spacing MACRON (U+00AF): **◌̄**.

7.4. 3. Other formatting

- x. For underdotted text, use Stylistic Set 7, Underdotted. For letters that lack an underdotted form, use U+0323 COMBINING DOT BELOW.

7.5. 4. On the web

Because Junicode is a very large font, web pages should use a subsetted version to speed loading (see [Chapter 9, Junicode on the Web](#), for instructions). The variable version of the font is better for web use than the static fonts, since one variable font file can do the work of many static font files.

All major web browsers (Firefox, Chrome, Safari, Edge) are capable of accessing all of Junicode's characters via OpenType features, use of which promotes accessibility and searchability. When building a web page, study which features will be needed and write them into the appropriate element or class definition of the page's CSS style sheet. For example, if you use the curly form of the zigzag (U+035B) anywhere, you are likely to want it everywhere, and so it should be included in the CSS styling for the `<body>` element:

```
body {
  font-family: Junicode;
  font-feature-settings: "cv81" 2;
}
```

But the **hlig** feature, if applied to the whole text, will produce many unwanted effects, so it should be included in a class definition to be used in a `` applied just to the target sequence:

```
.que {
  font-feature-settings: "hlig" on;
}
filio<span class="que">q&#xA76B;</span>
```

The illustrations here use the low-level CSS font-feature-settings property. There are higher-level properties for some OpenType features, but as these are not (yet) universally supported by browsers, and some implementations are buggy, it is best to stick with font-feature-settings for now.

For the purposes addressed in this document, the font-feature-settings for the <body> element should probably be as follows:

```
font-feature-settings: 'cv69' 2, 'cv80' 1, 'cv81' 2;
```

And the following classes should be defined:

```
.super {
  font-feature-settings: 'supr' on, 'cv84' 39;
}

.que {
  font-feature-settings: 'hlig' on;
}

.deleted {
  font-feature-settings: 'ss07' on;
}
```

8. The Enlarge Axis

The character recommendation of the Medieval Unicode Font Initiative (MUFI) includes a class of characters called “Enlarged Minuscules,” for representing characters that are lowercase in shape but intermediate between lowercase and uppercase in size: these are often used to begin sentences in medieval manuscripts. MUFI encodes these characters in the Private Use Area, posing accessibility and searchability problems, as explained in the introduction to the “Feature Reference” chapter of this manual.

Junicode provides a solution to these problems via the OpenType feature Stylistic Set 6 (ss06, “Enlarged minuscules”). This feature also works in Junicode VF, the variable version of Junicode, which in addition offers a far more flexible way of representing enlarged minuscules—the Enlarge axis.

An “axis” is an aspect of a font that can be varied along a numerical range. A family of traditional fonts like Times New Roman has a weight axis with a font file on either end: Regular and Bold. Other font families have more weights along this axis: for example, Light, Medium, ExtraBold. Most variable fonts also have a weight axis, but all weights are contained in a single file, and users are not restricted to just a few weights, but can select any weight between the extremes.

Because almost every font family has at least two weights, Weight is the most familiar axis. But several other axes are frequently found in both variable fonts and extended font families. Junicode has Weight and Width axes (Width varying from 75 Condensed to 125 Expanded, with 100 Regular in the middle), and the variable font also has an Enlarge axis, which can vary the size of many lowercase letters from that of the font’s capitals to that of the lowercase letters: Just as the size of these sentence-initial letters varies widely in manuscripts, so it can vary on web pages and in print (though few applications for producing

Đñs Đñs Đñs Đñs đñs

printed documents currently support variable fonts). Notice that the letters are not simply scaled: the proportions change and the weight remains consistent (a lowercase letter scaled up would look too heavy, but a letter scaled via the Enlarged axis will have its original weight at the lower end of the axis and the same weight as a capital at the top).

The Enlarge axis runs from 0 to 100. You can choose any number in that range: to match the effect of sso6 precisely, choose 32. To ensure that the xheight of all letters matches, choose 47 or less: above that value, the xheight of letters like **e** increases at a higher rate than that of letters like **b**.

To use the axis in a web page, declare a CSS class specifying the value for the axis. For example, the second of the examples in the figure above has the axis set to 75:

```
.SentenceInitial {
  font-variation-settings: "wght" 400, "wdth" 100, "ENLA" 75;
}
```

In the text, enclose the first letter of a sentence in a `` with the class “SentenceInitial” (the entity is for insular d):

```
<span class="SentenceInitial">&#xA77A;</span>ñs
```

The result will be an abbreviation that begins with an “Enlarged Minuscule” insular d, precisely matching the look of the second example in the figure above.

9. 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 subsetting 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.

9.1. 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 the unmodified font. 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 JunicodeVF-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,\
supers,zero \
--layout-features-=rli
```

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`, `rli`, `kern`, `mark`, and `mk`.

`--layout-features-` tells the program which OpenType features to omit. Normally, `rli` (Required Ligatures) 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.

9.2. 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 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 VF";
    font-size: 28px;
    font-weight: normal; /* that is, 400 */
}
```



```

    font-stretch: 112.5%; /* that is, semiexpanded */
}
h1 {
    font-family: "Junicode 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 VF";
    font-size: 28px;
    font-variation-settings: "wght" 400, "wdth" 112.5;
}
h1 {
    font-family: "Junicode 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 VF";
    src: url("../webfiles/JunicodeVFsubset.woff2");
    font-weight: 300 700;
    font-stretch: 75% 125%;
    font-style: normal;
}
@font-face {
    font-family: "Junicode";
    src: url("../webfiles/Junicode-Regular.woff2");
    font-weight: 400;
    font-style: normal;
}
@font-face {
    font-family: "Junicode";
    src: url("../webfiles/Junicode-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", serif;
}
@supports (font-variation-settings: normal) {
    body {
        font-family: "Junicode VF", serif;
    }
}
b {
    font-weight: 700;
}
@supports (font-variation-settings: "wght" 700) {
    b {
        font-variation-settings: "wght" 700;
    }
}

```

}

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
using the Lua \LaTeX typesetting system with fontspec for font management.
The source for the document, JunicodeManual.tex, is available at
<https://github.com/psb1558/Junicode-font>.*