Creating annotated Unicode-like font charts

Janusz S. Bień

Abstract

Printing annotated font tables in the layout following the Unicode standard documentation is discussed. Existing tools are presented and desirable but missing features are proposed.

1 Unicode charts

In this article, we will discuss the primary Unicode charts;[1] the secondary charts describing the variation sequences, in particular those belonging to Ideographic Variation Database, are outside the scope of our interest.

All the charts are produced with the Unibook formatting software,[2] supplied by ASMUS, Inc. (a company owned by Asmus Freytag, Technical Vice President of the Unicode Consortium from 1991 to 2007). A version of Unibook is available free of charge (but with a rather complicated license); it is used to prepare proposals of new characters, as well as the full standard. The program runs on several versions of MS Windows. The fonts used to print the characters themselves in the charts come from many sources and are not, in general, free in any sense.[3]

The charts proper are followed by various annotations. The source of annotations, at least in principle, is the `NamesList.txt`[4] file belonging to the Unicode Character Database. Its header states: *This file is semi-automatically derived from UnicodeData.txt and a set of manually created annotations using a script to select or suppress information from the data file.* The documentation of the format of the file is provided.[5] However, there are systematic discrepancies between the content of the file and the charts, so somewhere in the workflow additional processing is performed.

As of the standard version 15.0, there are the following annotations used (examples are given later):

- Lines starting with • (U+2022 BULLET) are just comment lines.
- Lines starting with = (U+003D EQUALS SIGN) are alias lines.
- Lines starting with ※ (U+203B REFERENCE MARK) are formal alias lines.[6]
- Lines starting with → (U+2192 RIGHTWARDS ARROW) contain cross-references to related characters.
- Lines starting with ≡ (U+2261 IDENTICAL TO) are used for precomposed characters and contain the list of characters which together compose the character in question.[7]
- Lines starting with ≈ (U+2248 ALMOST EQUAL TO) are used for so-called compatibility characters[8] and show the compatibility decomposition of the character (this relation is defined by enumeration).
- Lines starting with ~ (U+007E TILDE) describe registered variation sequences.

The file also contain commands to control printing titles, subtitles, block headers, various subheaders and notices.

2 The fntsample tool and its extension

In 2007 Eugeniy Meshcheryakov (the original spelling seems to be Евгений Мещеряков) released the first version of the fntsample program.[9] The tool was developed for the DejaVu Fonts project.

The charts it generates resemble those of the Unicode standard; see Fig. 1. In particular, they include glyphs for the characters which are invisible by definition, such as 𞄀 (U+FE00 VARIATION SELECTOR-1), if the font contains them.

In 2013 a student of mine, Paweł Parafiński, accepted the task to made the samples even more similar to the Unicode charts by supplementing them with annotations. He solved the problem in two steps. First he created a parser for the `NamesList.txt` file which converted it into simple XML. In the second step he extended fntsample to print the additional information from the XML file.

The program is orphaned, but the repositories are still available;[10] in particular, this allows reporting the issues. The most annoying problem is the inability to break long character names. There was also a problem with the parser, which was coded in the now-obsolete Python 2. I managed to adapt it to Python 3 (by trial and error, as I am not a programmer).

[1] `unicode.org/charts`
[2] `unicode.org/unibook`
[3] `unicode.org/charts/fonts.html`
[4] `unicode.org/Public/UNIDATA/NamesList.txt`
[5] `unicode.org/Public/UCD/latest/ucd/NamesList.html`
[6] Formal aliases are used for control characters, which have no glyphs and official names. Another use is correcting name mistakes; because of the stability principle, erroneous names are not removed from the standard, but the correct names are added as a formal alias.
[7] Some characters 'decompose' into another single character, but this is another story; the most well-known example is U+212B ANGSTROM SIGN, which 'decomposes' into U+00C5 LATIN CAPITAL LETTER A WITH RING ABOVE.
[8] See `en.wikipedia.org/wiki/Unicode_compatibility_characters`, for example.
[9] `github.com/eugmes/fntsample`
[10] Now at `github.com/jsbien/fntsample-with-comments` and `github.com/jsbien/ucd_xml_parser`.

doi.org/10.47397/tb/44-1/tb136bien-unichart

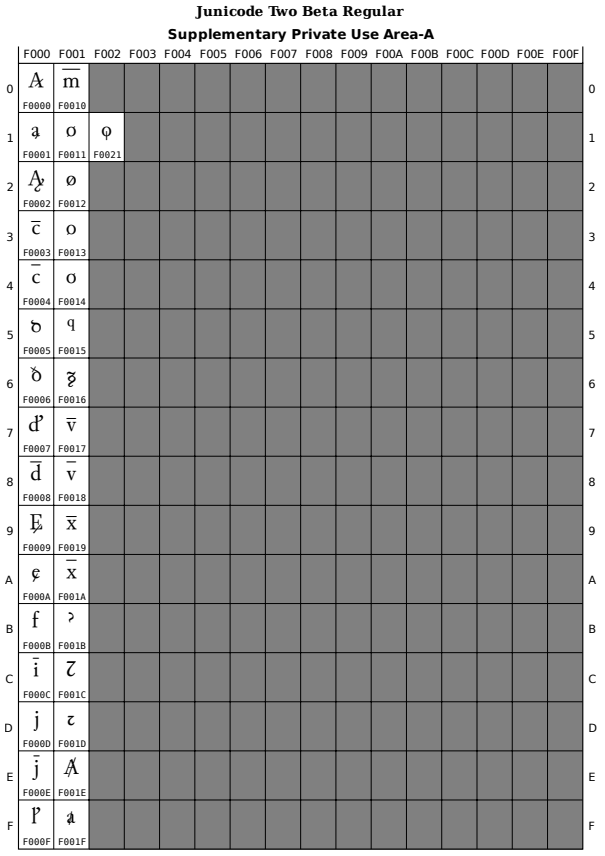

Figure 1: Output of
`fntsample -f JunicodeTwoBeta-Regular.ttf -i 0xf0000- -o sample.pdf`

Unfortunately the parser ignores some elements of the `NamesList.txt` file, as at the time of its design they seemed not to be needed.

3 Frank Mittelbach's unicodefonttable

Frank Mittelbach's unicodefonttable package is available from CTAN and the usual distributions; its source repository is `github.com/frankmittelbach/fmitex-unicodefonttable`. It provides a LaTeX style file to include a font table in a document and, an interactive standalone version to produce a font table as a separate document. Tables can be supplemented by block names and user captions; tables can be limited to selected blocks.

The unicodefonttable package is implemented using the fontspec package, which is an advantage, as this allows using all font features which the package supports. The source of the Junicode font manual [1], typeset with XeLaTeX, gives extensive examples of the usage of font features with the fontspec package.[11] They are also discussed in the manual [1, p. 13]:

[11] `github.com/psb1558/Junicode-font`

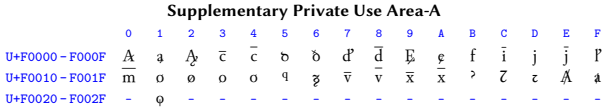

Figure 2: unicodefonttable output for plane 15 in `JunicodeTwoBeta-Regular.ttf`

> Many OpenType features produce different outcomes depending on an index passed to an application's layout engine along with the feature tag. Different applications have different ways of entering this index: consult your application's documentation. Here, the index is recorded in brackets after the feature tag. Users of fontspec (with XeLaTeX or LuaTeX) should also be aware that fontspec indexes start at zero while OpenType indexes start at one. Therefore all index numbers listed in this document must be reduced by one for use with fontspec.

For referencing a specific value of a feature we follow [1], e.g., cv02[1] means the feature cv02 (character variant number 2) with the index 1; on the other hand ss10 is an example of a feature (stylistic set number 10) which does not require an index but which is just on or off.

It is worth noting that fontspec, and hence also unicodefonttable, if used with LuaLaTeX, allow for the little known "raw feature" `-invisible`,[12] allowing printing glyphs for characters which are invisible in principle (mentioned earlier; see also, for example, [3]) if the font contains them.[13]

Figure 2 shows the output of the following:

```
\displayfonttable
  [range-start=F0000,range-end=FFFFF,
   nostatistics,noheader,hex-digits=block]
  {JunicodeTwoBeta-Regular.ttf}
```

4 David M. Jones' STIX charts

The charts for the STIX fonts (*OpenType Unicode fonts for Scientific, Technical, and Mathematical texts*[14]) are typeset by David M. Jones with XeLaTeX from source generated by a Perl script. The principal part of the charts has the same layout as the Unicode charts, so this technique could be used to replace fntsample (the additional part of the charts describes the OpenType features, which is only indirectly related to the present paper). Figure 3 shows the page for the Greek and Coptic block, starting at U+037.

[12] `github.com/latex3/luaotfload/issues/63`

[13] I learned this from the author; see `github.com/FrankMittelbach/fmitex-unicodefonttable/issues/5`.

[14] `github.com/stipub` and `www.stixfonts.org`

Figure 3: A page from `STIXTwoText-Regular.pdf` (headers and footers are omitted).

These charts use some additional conventions, e.g., red (grayscaled for print) is used for characters not directly supported but synthesized by a Unicode-aware shaping engine; this is shown with characters U+03D3 and U+03D4 in the figure; U+0374 is another example of a character which "decomposes" into (is substituted by) another single character.[15]

At present neither the XeLaTeX source nor the Perl script are available publicly.[16]

[15] `github.com/stipub/stixfonts/issues/248`
[16] `github.com/stipub/stixfonts/issues/247`

5 Typesetting character annotations

Although in the Unicode standard the character annotations occur immediately after the relevant glyph tables, it is not necessary to follow the standard in this respect. Separately provided annotations can be even more convenient.

Typesetting individual annotations in a way approximating the Unicode charts is not difficult. After some research I decided to use xltabular, which handles multipage tables (unfortunately it doesn't work in a multicolumn environment). The examples in figures 4–6 are typeset using code like this (showing just the first line; formatting has been slightly changed):

```
\begin{xltabular}{1.0\linewidth}%
                    {lll>{\raggedright}X}
0000 & &\multicolumn{2}{l}{<control>}\\
&    & &*& NULL\\
```

The Unicode conventions can be used easily to describe Private Use Area characters, e.g., those from Medieval Unicode Font Initiative[17] (other interesting initiatives are CONSCRIPT[18] and LINCUA[19]), or from a specific font. Fig. 5 shows some examples.

The line format for the registered variation sequences can be easily adapted to tag variation sequences (see, for example, [2]) and font specific information, as shown in Fig. 6.

The information about the font can be skipped when not needed.

6 Providing character annotations

In addition to the ttx program discussed later, the other tool I know of to list font features in a human-readable form is layout-features.py,[20] but its simple output does not contain the information needed for our purposes (see Fig. 7).

In 2018 a feature request was submitted to the fntsample repository entitled *Suggestion: enable printing glyphs not assigned to a unicode slot*,[21] but there was no follow up. In 2021 a similar feature request was made for unicodefonttable.[22] The answer, in my view correct, was that this should be a separate project.

We focus here on discussing the annotations in ttx's XML form (but the fork of layout-features.py may

[17] `mufi.info`
[18] `www.kreativekorp.com/ucsur`
[19] `bit.ly/2XVTzRL-LINCUA`
[20] `github.com/fonttools/fonttools/blob/8ab6af03c89726cf80ca3c4b755ae1bd0038b5da/Snippets/layout-features.py`; see also `github.com/fonttools/fonttools/discussions/2873`.
[21] `github.com/eugmes/fntsample/issues/11`
[22] `github.com/FrankMittelbach/fmitex-unicodefonttable/issues/2`

0000 <control>
※ NULL
0030 0 DIGIT ZERO
~ 0030 FE00 0 short diagonal stroke form
0040 @ COMMERCIAL AT
= at sign
0104 Ą LATIN CAPITAL LETTER A WITH OGONEK
≡ 0041 A 0328 ◌̨
2105 ℅ CARE OF
≈ 0063 c 002F / 006F o
2B7A ⬺ LEFTWARDS TRIANGLE-HEADED ARROW WITH DOUBLE HORIZONTAL STROKE
※ LEFTWARDS TRIANGLE-HEADED ARROW WITH DOUBLE VERTICAL STROKE
A7C1 ꝋ LATIN SMALL LETTER OLD POLISH O
• used in Old Polish as a nasal vowel
→ 00F8 ø LATIN SMALL LETTER O WITH STROKE

Figure 4: Typesetting standard character annotations.

F1C8 ᳪ COMBINING ABBREVIATION MARK ZIGZAG ABOVE CURLY FORM
• MUFI since v.2
E8AF ﬠ LATIN SMALL LIGATURE LONG S L WITH STROKE
• MUFI in v.4 at E8DF, later moved to E8AF
• used in old Polish
F0001 ą LATIN SMALL LETTER A WITH STROKE THROUGH TERMINAL
• supported in Junicode Beta since Jun. 30, 2020
• used in old Polish
→ 0105 ą LATIN SMALL LETTER A WITH OGONEK
→ 2C65 ⱥ LATIN SMALL LETTER A WITH STROKE

Figure 5: Some examples from Unicode's Private Use Area.

0062 b LATIN SMALL LETTER B
~ supported in Junicode Beta since Aug. 25, 2022 with ss10 on: 0062 b E0070 E0073
ƀ old Polish *b quadratum*
0105 ą LATIN SMALL LETTER A WITH OGONEK
• Polish, Lithuanian, ...
≡ 0061 a 0328 ◌̨
~ supported in Junicode Beta since Jun. 30, 2020:
0104 ą cv02[1] ą LATIN SMALL LETTER A WITH STROKE THROUGH TERMINAL

Figure 6: Adapting output to tag variation sequences.

also be considered). How to process this information further is another question outside the scope of this note. One possible approach is to continue the work on the fntsample fork. Another possibility is to use XSLT to convert to LaTeX (running under XeTeX or LuaTeX); this is a general recommendation of David Carlisle, the author of xmltex.[23]

The Unicode standard is updated every year, so we need a way to handle the update conveniently. Conversion to XML with Parafiński's parser, after some minor improvements of the program, seems a satisfactory solution.

In the Private Use Area, the updates to Medieval Unicode Font Initiative recommendation, for example, don't have a stable specific form. The data, e.g., as prepared for Parafiński's program, have to be updated by hand.

One way to extract the interesting information from the font which seems to be worth consideration

[23] tex.stackexchange.com/questions/562856

```
Table: GSUB
  Script: DFLT
    Language: default
      Feature: aalt
        Lookups: 0,1
      Feature: c2sc
        Lookups: 204
...
      Feature: ccmp
        Lookups: 50,53,55,56,57,58,59,60,61,...
      Feature: cv01
        Lookups: 110
      Feature: cv02
        Lookups: 111
...
```

Figure 7: Some of the output from `layout-features.py JunicodeTwoBeta-Regular.ttf`

consists of using the `ttx` plain text format of OpenType/TrueType fonts produced by the program of the same name.[24]

To make the output more human readable, by default `ttx` uses the character names from the file `UnicodeData.txt` belonging to the Unicode Character Database. There is, however, an option to provide a different file for the data. One occasion for this is a new version of the standard which has not yet migrated to the relevant software libraries. The second, more important for us, is to provide names of the PUA characters, including MUFI, prepared already in the appropriate format by Rebecca G. Bettencourt[25] (updates are probably needed).

OpenType/TrueType fonts consist of several tables and subtables, which `ttx` converts to XML (in the examples below, the formatting is slightly changed). Variation and tag sequences are represented in the same way as other ligatures. For example, in `NotoSans-Regular.ttf` the slashed zero variation sequence (surprisingly few fonts support this variation sequence), stored in the `GSUB` (Glyph Substitution) table, looks like this:

```
<GSUB>
  <Version value="0x00010000"/>
  ...
  <LookupList>
    <!-- LookupCount=43 -->
    <Lookup index="2">
      ...
      <!-- SubTableCount=1 -->
      <LigatureSubst index="0" Format="1">
        <LigatureSet glyph="zero">
          <Ligature components="uniFE00"
                    glyph="zero.slash"/>
        </LigatureSet>
      </LigatureSubst>
    </Lookup>
    ...
  </LookupList>
</GSUB>
```

To discover what `uniFE00` definitively means, we consult the `cmap` (Character to Glyph Index Mapping) table:

```
<cmap>
  <tableVersion version="0"/>
  <cmap_format_4 platformID="0" platEncID="3"
                 language="0">
    ...
      <map code="0xfe00" name="uniFE00"/>
           <!-- VARIATION SELECTOR-1 -->
  </cmap_format_4>
  ...
</cmap>
```

The comments are provided by the `ttx` program.

As for `zero.slash`, we can use the auxiliary table `GlyphOrder` produced by `ttx` to find the font slot number of the character:

```
<GlyphOrder>
  <!-- The 'id' attribute is only for humans;
       it is ignored when parsed. -->
  ...
  <GlyphID id="2581" name="zero.slash"/>
  ...
</GlyphOrder>
```

The slot number can be used to typeset the character; in XeTeX the appropriate command is `\XeTeXglyph`.[26]

Let's now consider another example, namely *b quadratum* from the Junicode font mentioned above. The tag sequence is active only when stylistic set 10 is selected, so the ligature has to be embedded in the appropriate `FeatureElement`:

```
<GSUB>
  <Version value="0x00010000"/>
  ...
  <FeatureList>
    <!-- FeatureCount=155 -->
    ...
    <FeatureRecord index="140">
      <FeatureTag value="ss10"/>
      <!-- Character Entities
           for Combining Marks -->
      <Feature>
        <FeatureParamsStylisticSet>
          <Version value="0"/>
```

[24] `github.com/fonttools/fonttools`; see also [4, p. 22].

[25] `kreativekorp.com/charset/PUADATA/PUBLIC/MUFI`

[26] This was kindly pointed out to me by Ulrike Fischer on the XeTeX mailing list (`tug.org/pipermail/xetex/2022-October/028105.html`). [Editor's note: For LuaTeX, some Lua code can achieve the same result; see `tex.stackexchange.com/questions/120736`.]

```
          <UINameID value="256"/>
                    <!-- Entities -->
        </FeatureParamsStylisticSet>
        <!-- LookupCount=3 -->
        ...
        <LookupListIndex index="2"
                         dopiskivalue="71"/>
      </Feature>
    </FeatureRecord> ...
  </FeatureList>
  <LookupList>
    <!-- LookupCount=234 -->
    ...
    <Lookup index="71">
      <LookupType value="4"/>
      <LookupFlag value="16"/>
                 <!-- useMarkFilteringSet -->
      <!-- SubTableCount=1 -->
      <LigatureSubst index="0" Format="1">
        ...
        <LigatureSet glyph="b">
          <Ligature components="e.tag,n.tag"
                    glyph="b.enlarged"/>
          <Ligature components="p.tag,l.tag"
                    glyph="b.p02"/>
          <Ligature components="p.tag,s.tag"
                    glyph="b.p01"/>
        </LigatureSet>
        ...
      </LigatureSubst>
      <MarkFilteringSet value="1"/>
    </Lookup> ...
  </LookupList>
</GSUB>
```

In general the glyph selection can depend on a script, a language, and user-selected features, which make the font structure quite complicated. A program intended to extract all the information about a font has to take everything into account.

7 Final remark

I hope this note will provide inspiration to a reader or readers with appropriate skills and in some future we will see a tool for printing annotated font tables in a nice fashion.

References

[1] P. Baker. Junicode — the font for medievalists. specimens and user manual for version 2, 2022. github.com/psb1558/Junicode-font/

[2] J.S. Bień. Representating Parkosz's alphabet in the Junicode font. *TUGboat* 43(3):247–251, 2022. tug.org/TUGboat/tb43-3/tb135bien-parkosz.pdf

[3] M. Davis, K. Whistler. Default ignorable issues. L2/02-368, 2002. unicode.org/L2/L2002/02368-default-ignorable.pdf

[4] Y. Haralambous. *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between.* O'Reilly Media, 2007.

◇ Janusz S. Bień
Warsaw, Poland
jsbien (at) uw.edu.pl
sites.google.com/view/jsbien
ORCID 0000-0001-5006-8183

Production notes

Karl Berry

Almost all of the characters in Janusz's article could be typeset with no particular trouble. But two needed special attention: the character a (U+F0001 LATIN SMALL LETTER A WITH STROKE THROUGH TERMINAL) and the visible glyph 𛀀 (U+FE00 VARIATION SELECTOR-1).

For the former, XeLaTeX has no problems typesetting U+F0001 from the Junicode font:

```
\newfontfamily{\Junicode}
  {JunicodeTwoBeta-Regular.ttf} % for XeTeX
\newcommand{\sgl}[1]
  {{\Junicode #1}}
\newcommand{\Fzerosone}{\sgl{...}}
```

However, I wanted to use LuaLaTeX to typeset the article, because its support for microtype's font expansion feature avoided several overfull lines, and it typeset some other character. It turns out (github.com/latex3/luaotfload/issues/244) that setting the HarfBuzz rendering mode is what's needed. (This is not the default in lualatex, even though it uses the luahbtex engine.)

```
% For LuaTeX:
\newfontfamily{\Junicode}
  [Renderer=HarfBuzz]{JunicodeTwoBeta-Regular.ttf}
```

For the latter character: ordinarily, Unicode prescribes that variation selectors are invisible, but a few fonts also provide a visible glyph; the one here (found by Janusz) is from NotoSansManichaean-Regular.ttf, following what is printed in the Unicode font charts.

XeLaTeX could handle this with its \XeTeXglyph primitive, which can be used to typeset any glyph from a font, whether mapped to an input character or not; in this case, \XeTeXglyph 58. (The ttx program can be used to discern such internal information in any OpenType or TrueType font.)

For LuaLaTeX, however, it was necessary both to use the Base rendering mode, and a bit of Lua code devised by Henri Menke (thank you Henri, and thank you search engines), which emulates many XeTeX primitives in LuaTeX (gist.github.com/hmenke/6e8ff7c90a5e5df3c4895f60059a2ef7):

```
\ifx\undefined\XeTeXglyph % LuaTeX case:
  \def\XeTeXglyph{%
    \directlua{...}}%
\fi
\newfontfamily{\NSM}
  [Renderer=Base]{NotoSansManichaean-Regular.ttf}
\newcommand{\VSone}{{\NSM\XeTeXglyph 58}}
```

Happy Unicode typesetting.

◇ Karl Berry
github.com/TeXUsersGroup