My website exploring language support and more in libre TrueType and OpenType fonts: `https://typosetting.co.uk`

Ken Moffat

Abstract

At `typosetting.co.uk` I have a website where I explore libre OpenType and TrueType fonts to try to see which languages they support (and to note mismapping of codepoints when I spot them). I am using XƎLaTeX to produce PDFs showing the glyphs in the fonts, the languages I think they support, and some other comparisons between fonts.

1 Introduction

For many years I have been interested in using fonts which render all the text I am likely to see on mailing lists (whether or not I can read it), and whatever text is presented in the various rabbit-holes I dive into on Wikipedia.

My previous websites for doing this began in or before 2013. I had a C++ program to list what was in a font, and various Bash and Perl scripts. For the PDFs I was using LibreOffice, and at that time it indicated missing codepoints. In PDFs which compared files I referenced the font numbers.

Later, I added a second table (to avoid renumbering) for some other fonts, and then in 2023 I revisited fonts and added a third table.

Along the way, perhaps in 2016, LibreOffice had changed to plucking missing glyphs from other fonts when a font lacked a codepoint. Before that I had picked up a little information for testing LaTeX etc. to check that from-source builds worked, and determined that XƎLaTeX mostly worked for me when displaying multiple languages.

For all languages I cover, I show Article 1 of the Universal Declaration of Human Rights to give a flavour of how text will look. For Latin and Cyrillic alphabets and monotonic Greek I attempt to show the whole alphabet including accented letters.

As a novice I used the default page layout with large margins and lots of space above and below the text. I ignored text that overflowed and did not think about any hyphenation.

My initial interest was only for on-screen output. At that time fontconfig was the tool that decided which glyphs to use. (Now browsers, office suites, and desktop environments can all override fontconfig.) As a result, I ignore any supporting files which might be on CTAN and can happily use a newer version of a font even when CTAN is out of date.

2 The new site: `typosetting.co.uk`

I prefer a readable dark-mode site without JavaScript. The PDFs have a white background as usual, and links to directories use a white background.

The three tables are now merged into one, keeping the same numbers. I have added indexes to link the fontconfig names to the table entries, and similarly to link my own identifiers (which omit spaces and are usually based on the filename). Figure 1 shows the first few rows of the table

That table is now very wide. I inserted a column showing when I revised the PDF for the supported languages, with links to the `.tex` file and a link for the small cap `.tex` file if there is one. The lines above the table, and on other pages, are much narrower, based on what looks good with my default sans font.

For fonts which do not cover Latin or Cyrillic alphabets or CJK languages I am not attempting to revise the pre-2024 documents. Although I have always used HarfBuzz with Graphite2, some of the old rendering of complex scripts might be poor [1].

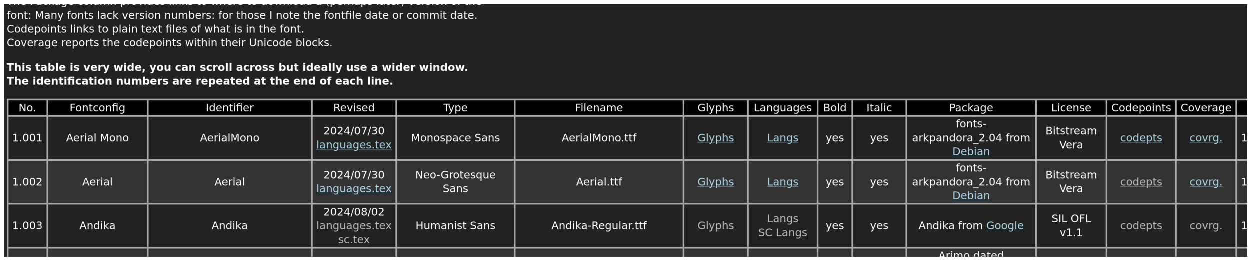
I have not yet revised several CJK fonts where I am aware of newer versions. For any file revised before 2024 there is a possibility that it will no longer compile in XƎLaTeX and there are probably typos in any Latin or Cyrillic alphabets, outdated Article 1 text in a few languages, and ugliness (formatting of CJK, and some of the TeX markup).

At the end of 2023 I realised that showing overlong lines was poor. For many languages with Latin alphabets `polyglossia` can usually do an adequate job of fitting the available space. But for Cyrillic alphabets, Greek, CJK and others the font needs a specific OpenType tag (`cyrl`, `grek`, `hani`, etc.). For Northern Sami (supposedly supported, but did not hyphenate) I found some guidance online for splitting the words into syllables. Similarly, for several Cyrillic languages where I was unable to use polyglossia I found various suggestions on how to split syllables. The results may be ugly to native readers.

For a few languages, and for some small caps fonts, I have to reformat the Article 1 text. Mostly I now use shorter lines to avoid putting vast spaces between words. For monospace fonts, much of the output includes spaces added automatically to justify the lines, but occasionally I will add non-breaking spaces to replicate what happens in a graphical terminal such as `xfce4-terminal`.

3 Characteristics I have noted

I have grouped fonts into sans, serif, and monospace, and then for Latin alphabets tried to separate the styles (transitional serif, modern serif, etc.). For monospace and sans I am no longer sure that such a



font: Many fonts lack version numbers: for those I note the fontfile date or commit date.
Codepoints links to plain text files of what is in the font.
Coverage reports the codepoints within their Unicode blocks.

This table is very wide, you can scroll across but ideally use a wider window.
The identification numbers are repeated at the end of each line.

No.	Fontconfig	Identifier	Revised	Type	Filename	Glyphs	Languages	Bold	Italic	Package	License	Codepoints	Coverage	
1.001	Aerial Mono	AerialMono	2024/07/30 languages.tex	Monospace Sans	AerialMono.ttf	Glyphs	Langs	yes	yes	fonts-arkpandora_2.04 from Debian	Bitstream Vera	codepts	covrg.	1
1.002	Aerial	Aerial	2024/07/30 languages.tex	Neo-Grotesque Sans	Aerial.ttf	Glyphs	Langs	yes	yes	fonts-arkpandora_2.04 from Debian	Bitstream Vera	codepts	covrg.	1
1.003	Andika	Andika	2024/08/02 languages.tex sc.tex	Humanist Sans	Andika-Regular.ttf	Glyphs	Langs SC Langs	yes	yes	Andika from Google	SIL OFL v1.1	codepts	covrg.	1
										Arimo dated				

Figure 1: The start of the big table at `typosetting.co.uk`.

distinction is generally useful. In the past, particularly in my comparative "lipsum" files, I have made a similar distinction for types of Cyrillic fonts. For CJK fonts I am now tending towards merely labelling them as sans or serif — the terminology varies and is not always consistent for CJK fonts for a specific language.

Beyond that I show italics and bold weights for fonts offering Latin alphabets. For recent families I mention the range of weights, and briefly show Cyrillic italics.

For CJK fonts I hope to create language-specific files showing codepoints known to differ across languages, together with examples of each Noto CJK font and WenQuanYi Zen Hei (a preferred font in fontconfig). The first of these, `CJK-sans-sc`, is in the `files/PDF-cjk/` and `files/cjk-tex/` directories (`sc` for Simplified Chinese). I have few sans (Hei) sc fonts so this was an easier place to start, but many of the chosen codepoints look similar in sans fonts.

4 Validating my analysis, tools and templates

I doubt there is much of significance in any of my analysis of what a font covers, but in the last year I have read many articles online where research could not be validated, sometimes because the tools were not available. In addition, many fonts needed weird workarounds in some places, particularly for extended text in small caps. So I have attempted to make all my tools, and my templates, available.

4.1 Templates

For people using XƎLaTeX and/or `polyglossia` (and perhaps for users of LuaLaTeX) the templates are perhaps the most interesting thing (apart from the `.tex` files for any fonts of interest to them). They currently include:

4.1.1 `languages-full.tex`

This is for attempting to see which languages a font covers. It includes a few languages which are covered by FreeSans and FreeSerif.

Beyond that, it has a lot of comments for me about things to look at when I revise an old file, and reminders about mentioning styles (normal, italic) and weights — in the past I tended to talk of font "faces" — followed by sections for the various languages.

If a language supports Vietnamese I attempt to show the added accents for the various tones. The individual letters are precomposed, but I show a separate combining accent at the start of the line to indicate what is used. A few fonts do not support those combining accents, in those cases I show their missing glyph indication. More commonly, the combining accent is either in the gutter (negative spacing needed) or ends up being inset (monospaced fonts). None of the PDFs have well-aligned columns for all of these accented letters.

For my revisions after the site went live I now try to be specific about how missing codepoints are rendered, but most files use older, more general, language like "blank spaces", "nothing", or "indication of a missing glyph".

For quotation marks I show all supported variants, but for currencies and some other symbols such as copyright I list them all whether or not they exist in that font. The output messages usually end with missing codepoints. I now know that I need to review those messages, particularly where HarfBuzz might add an accent or simulate a non-breaking hyphen.

The CJK area starts with a series of codepoints which may have different forms in some of the languages. That is to help me decide at which language a font is targeted, and will be deleted once I've made the decision. It then includes some English text to help me know where the margins should be (in theory the XƎLaTeX CJK settings near the front of the file should fix that) which I will again delete.

In some cases, all I need to do for `fontspec` is use the name known to fontconfig; for XƎLaTeX, unlike LibreOffice, this is the first name, before any comma, if there is more than one name. But for several fonts, such as Carlito, FreeSerif, lmroman and Junicode, I had to add a `Path` statement within the `setmainfont` invocation to find one or more of the related font files.

4.1.2 `languages-sc.tex` for small caps

A very much cut-down template, for Latin and Cyrillic alphabets, and monotonic Greek. Many fonts with small caps omit certain less-common letters. For Azeri or Turkish (which use both dotted and dotless i) I was advised to use lowercase dotless i with combining dot above.

4.1.3 `languages-italic.tex`

A working document, to help me decide which languages have italics and/or small cap italics.

4.1.4 `Hangul-Hanja.tex`

Until recently I assumed that Hanja (South Korean Han) glyphs were no longer in common use. But it seems they are still used, both for place names on signs (probably only sans glyphs) and for both surnames and given names in official documents. I found a 2015 PDF of the list of permitted glyphs online, organized by Hangul syllable. But I cannot paste from that, and the only online pastable glyphs are mostly in Wikipedia, under Korean given names. Those pages are for notable people, so many of the permitted glyphs are not shown.

In addition, many of the glyphs are hard for me to distinguish unless I enlarge a text file until it is hard to navigate. As a result, these are only a subset of the permitted glyphs and probably contain some duplicates — certainly the initial files I created have a few duplicates.

The aim of this is to let me say "seems to adequately cover Hanja used in names", or "covers most", "covers many", . . .

4.2 Programs and scripts

These programs and scripts assume I am the only user on the machine, I make no attempt to use secure names for temporary files and am happy to write working files to known names in `/tmp`. These items have evolved since 2011, I do not normally touch most of them.

In `files/tools/to-compile` are directories for the following two tools and also source tarballs.

4.2.1 `get_codepoints`

The first requirement is to identify which codepoints are provided in a font. I found a `has_char` script by user repolho on github (`github.com/repolho/has_char`) which is public domain. It links to fontconfig and attempts to bypass the (obsolete?) rules that fontconfig uses to determine if a font has all the codepoints needed for the current locale.

I have been using this since 2016 with various releases of fontconfig and GNU C++, most recently `fontconfig-2.15.0` and `gcc-14.1.0`.

4.3 `getfonts`

That `get_codepoints` script reads a TTF (normally, a regular or medium file, I suppose other files could have fewer codepoints). But historically fonts for Chinese and Japanese have been supplied as TTC (TrueType Collection) files to save space and fit within a size limit. I used to use fontforge to extract an individual TTF, but it was always very difficult to use (tiny faint text, very hard to read, and prone to errors while trying to extract a single font to a TTF).

If I now need to repeat the process I found a repo called textile at github, designed for copying TTFs from a TTC on Mac OS X (`github.com/DavidBarts/getfonts`). The program is called `getfonts`; I had to fiddle about a bit to get it working. On the TTC I tested, it did not find the font name, so I had four numbered fonts. But looking at `fc-list` showed me the names of each so it is good enough.

I had hoped to fork the original repo, then work out what I had changed and add the changes as individual commits, but time is not on my side.

4.4 `create-codepoint-files`

This script is how I get the lists of codepoints in a file. At times in the past I have had problems with things not found by `get_codepoints`, so as a backup I install `ttfconfig.pl` which is shipped in the examples part of Perl module `Font-TTF-Scripts` (`metacpan.org/dist/Font-TTF-Scripts`).

I have had special code to deal with known TTC files, but it looks as if that is no longer used and only a variable remains.

Then I merge both files, converting the codepoints to 0xXXXX or 0xXXXXX format so that I can sort them, then write them as U+XXXX or U+XXXXX. After that I run the `font-contains` script, described next, to create the coverage file which lists the codepoint ranges of this font within their blocks.

4.5 `font-contains`

This sources the `unicode-blocks` script (described below) and then converts the codepoint to decimal for less-slow shell maths — on a big font such as the main (Latin, Cyrillic, Greek) Noto or Source fonts, or the Noto CJK fonts, it will take several minutes. Unicode values with five hex digits are written to a temporary file for a second pass (probably unnecessary). It then uses the data from `unicode-blocks` to find which block it should be in and writes lines of the contiguous codepoints present within that block.

4.6 `generate-all-characters`

This Bash script attempts to write every character in a font to a text file. That text file can then be opened in LibreOffice Writer, changed to use the required font, and formatted with page headings, footings and repeating the block names after the page breaks. If the 32 characters for a line (or to the end of the block if sooner) have gaps, spaces are used as padding.

For combining characters they are applied to a space, which usually works fairly well for Latin and Cyrillic combining accents. For more complex scripts the results may be poor (again see [1]) With CJK fonts I typically notice that Han glyphs are double-width, but that Hangul (Korean) seems to be an intermediate width.

4.7 Determining coverage of small caps

Most fonts with small caps include them as variants of the normal lowercase letters. Sometimes I can read what is in the file and generate the codepoints with some degree of confidence. I also report things I'm not sure about, which led me down rabbit holes for old Polish and for the Ukrainian yi-yi combination.

4.7.1 `find-small-caps`

A Bash script using the `otfinfo` program from Eddie Kohler's `lcdf-typetools` [2].

4.7.2 `merge-sc-codepoints`

This Bash script then merges the list of small caps codepoints into an updated codepoint file, adding '`+sc`' wherever there is a small cap available.

4.8 `unicode-blocks`

This is the data used by `generate-all-characters`: a list of the decimal values for the end of each block (strictly, the first value after this), a value for `MAXBLOCK` (378 as of Unicode 15.1.0 : unassigned blocks may be split in the future), and the block names with a list of their Unicode ranges. The process to create this is discussed in the next section.

4.9 Updating Unicode

When a coverage report shows codepoints in an unassigned area, it implies that Unicode has been extended. To update the data I use the following scripts:

4.9.1 `Blocks.txt`

I use the current list of Unicode assigned blocks. It can be downloaded from `unicode.org/Public/UCD/latest/ucd/`.

4.9.2 `update-blocks.sh`

This script processes `Blocks.txt` to create the file `unicode-blocks`. Although it invokes `/bin/sh` it might be Bash-specific.

It creates a temporary file, `assignments.txt`, and uses the `parse` script to get blocks and to create `unassigned` blocks to fill the gaps. Then it concatenates the files to get the `unicode-blocks` file.

4.9.3 `parse`

This invokes `/usr/bin/perl` to read through the `assignments.txt` file. At the end it appends the number of blocks.

5 Acknowledgements

Bob Tennent for suggesting `otfinfo -g` to list the small caps.

Don Hosek for suggesting dotless i with combining dot above for Turkish small caps.

Both Bruno Voisin and Herbert Voss suggested alternative approaches to listing the small caps; these are in the `tex-live` list archives for October 2023 (`tug.org/pipermail/tex-live/2023-October/`).

David Carlisle for explaining how HarfBuzz will add combining accents to base characters or alternatively use precomposed characters when combining accents are added.

Max Chernoff pointed to an alternative approach to typesetting a grid of characters showing all codepoints using LuaTEX; in the `tex-live` list archives for May 2024.

Werner Lemberg prompted me to dive deeper into CJK formatting. After discovering how to fix XƎLATEX for this, I later found draft W3C recommendations.

To all these people, and anyone I've missed — thanks for your help.

References

[1] Behdad Esfahbod. State of Text Rendering 2024. `behdad.org/text2024/`

[2] Eddie Kohler. LCDF Typetools. `www.lcdf.org/type/`

◇ Ken Moffat
`https://typosetting.co.uk`

My website exploring language support and more in libre TrueType and OpenType fonts