Preventing tofu with pdfTeX and Unicode engines

Frank Mittelbach

Abstract

Discussion of input encodings vs. font encodings, missing characters, Unicode, and TeX history.

1 With tofu through the years

Tofu is not just an essential ingredient for many Asian dishes, it is also the nickname for the little squares produced by many browsers when they are asked to render a character for which they do not have a glyph available.

Especially in the early days of the World Wide Web, websites in foreign languages (from the perspective of your computer) got often littered with such squares, making text comprehension quite difficult if not impossible in some cases. So instead of getting

> ¿But aren't Kafka's Schloß & Æsop's Œuvres often naïve vis-à-vis the dæmonic *phœnix's official rôle* in fluffy soufflés?

you might have seen something like

> □But aren't Kafka's Schlo□ □ □sop's □uvres often na□ve vis-□-vis the d□monic *ph□nix's official r□le* in fluffy souffl□s?

Over the years the situation with browsers improved (partly because using inferior fonts was deemed acceptable as long as they could render the needed glyphs), but even nowadays you may find tofu-littered sites, or perhaps worse, those where your browser thinks it can show you the glyphs but renders the wrong ones.

While with browsers you may accept a certain imperfection in the rendering, tofu in printed material is quite unacceptable. Typesetting systems should always use the correct glyphs or at least tell you very explicitly if they are unable to do so for some reason, to allow you to apply some corrective actions. In the remainder of this article we will discuss how TeX and in particular LaTeX is doing in this respect and what a user can or must do to avoid such a capital blunder.

Early vegetarian dishes with TeX

In the early days of TeX the use of fonts was easy because you could use any font you wanted as long as it was called Computer Modern.

In other words there was essentially only one set of fonts available for use with TeX and the glyphs it contained and how to address them was described in *The TeXbook* [3]. Furthermore, all fonts only contained 128 glyphs, i.e., essentially the base Latin characters, a few accents to construct diacritical characters using the `\accent` primitive and a few other symbols such as †, $ and so forth to be accessed through command names.

Thus, once you learned the construction methods and memorized the control sequences for accessing the existing symbols you could be sure that the characters you used would faithfully appear in the printed result. Of course, part of the reason for this was the limited glyph set; already any Latin-based language other than English posed serious issues, namely that necessary glyphs were missing entirely, or only available as constructed characters (whenever accents where involved) — which prevented TeX from applying hyphenation.

So as TeX got more popular outside the English-speaking world there was considerable pressure on Don Knuth (largely by European users, the author among them) to extend TeX so that it could better handle languages with larger character sets. At the 1989 TeX conference in Stanford we finally managed to convince Don to reopen (in a limited way) TeX development and produce TeX 3. This version of TeX was then able to deal with more than one language within a document (e.g., use multiple hyphenation patterns) and support 8-bit input and output (that is, 256 characters in a font).

While this enabled the use of different input code pages for different character sets, as was standard in those days, and also solved the problem of hyphenating words containing accented characters (by using fonts with precomposed glyphs), it also posed new challenges.

Depending on the active code page when writing a document, a given keyboard character might be associated with a different number (between 0 and 255) and that number had to be mapped to the right slot in a font to produce the glyph that was originally intended. So the days of input number equals font glyph position were definitely over, and the TeX world had to come up with a more elaborate scheme to translate one into the other to avoid missing or wrong characters in the output.

The LaTeX 2$_\varepsilon$ solution

For LaTeX the solution came in the form of the New Font Selection Scheme [4], and in particular with the packages `inputenc` (for managing input in different code pages and mapping it to a standard internal representation) and `fontenc` (for translating this internal representation to the correct glyph positions in different fonts).

Introducing font encodings

LaTeX classified the font encodings and gave them names such as `OT1`, `T1`, `TS1`, `T2A`, `T2B`, etc. Each such font encoding defined which glyphs are in a font using that encoding and to which character code (again, 0–255) each glyph was assigned in the font. Thus, if you had two different fonts with the same encoding you could exchange one for the other and still be one hundred

doi.org/10.47397/tb/42-3/tb132mitt-tofu

percent sure[1] that your document typeset correctly, with no missing or incorrect glyphs in the output.

In practice only a small number of font encodings ever got used and new fonts usually were made available in these "popular" encodings by providing the necessary font re-encodings through the virtual font mechanism, or through re-encodings done by device drivers (such as `dvips`) or directly in the engine (in the case of `pdftex`).

As an overall result, life for LaTeX users was again fairly easy after 1994 and remained this way well into this century, because by simply specifying which font encoding to use, documents would normally be typeset without defects, regardless of the font family that got used. Further, due to the fact that for users writing in Latin-based languages essentially every interesting font available was provided in the `T1` encoding, it was also clear which glyphs were available and those were available almost universally.

Pitfalls with missing input encodings

There was still the need to specify the input encoding — at least if one wanted to input accented characters directly from the keyboard instead of using TeX constructs like `\"a`. One problem in this respect was that, depending on the language you were writing in, it sometimes worked even without specifying the input encoding. This was possible because the `T1` font encoding was nearly identical to the quite common `latin1` input encoding.[2] Years later, omitting the input encoding declaration even when it worked initially finally backfired: once LaTeX moved on to make UTF-8 the default encoding, documents stored in legacy encodings failed if they didn't contain an input declaration.[3]

Pitfalls with the `TS1` encoding

When 8-bit fonts became more common, the TeX community defined two font encodings during a conference at Cork in 1990. The first is `T1`, which holds common Latin text glyphs that play a role in hyphenation and therefore have to be present in the same font when seen by TeX. The second is `TS1`, which contains other symbols, such as oldstyle numerals or currency symbols; these can be fetched from a secondary font without harm to the hyphenation algorithm, because they do not appear as part of words to be hyphenated.

[1] Well, more like 99% since sometimes fonts claimed to be in one encoding but didn't faithfully follow its specification, e.g., didn't provide all glyphs or sometimes even placed wrong glyphs at some slots.

[2] For example, with French texts it worked throughout. However, with German only the "umlauts" worked, but the sharp s "ß" generated a different character.

[3] The remedy for such old documents is to either add the missing declaration or re-encode the old source and store it in UTF-8.

On the whole, the glyphs in the `T1` encoding were well-chosen and it is usually possible to arrange any commercial or free font to be presented in this encoding to TeX.[4] As a result, substituting `T1`-encoded fonts means that you can be fairly sure that there will be no tofu in your output afterwards.

Unfortunately, this is not at all true for the `TS1` encoding. Here the community made a big mistake by going overboard in adding several "supposedly" useful glyphs to the encoding that could be produced in theory (and for Computer Modern and similar TeX fonts were in fact produced), but that simply did not exist in any font that had its origin outside the TeX world.

As a result, to use such glyphs from the `TS1` encoding meant that you had to stay with a very limited number of font families. Alternatively, you had to be very careful not to use any of the problematic symbols to avoid tofu.

To ease this situation, the `TS1` encoding was subdivided into five sub-encodings and a LaTeX interface was established to identify that a font family with a certain NFSS name belonged to one of the sub-encodings. This way LaTeX was enabled to make "reasonable" adjustments when a requested symbol was not available in the current font, either by substituting it from a different font or by giving you an error message that the symbol is not there — not perfect but better than tofu in the end. This was implemented in the `textcomp` package which provided the LaTeX commands to access the symbols from `TS1`.

In one of the recent LaTeX releases the code from `textcomp` was moved to the LaTeX format, so that these extra symbols are now available out of the box without the need to load an additional package. At the same time, the classification of fonts into `TS1` sub-encodings was reworked. We now support nine sub-encodings and the LaTeX format contains close to 200 declarations that sort the commonly available font families into the right sub-encodings. Thus these days the situation is fairly well under control again — at least with pdfTeX.

2 Unicode

One of the goals of Unicode is to uniquely identify each and every character used in different languages and scripts around the world, thereby avoiding some of the possible translation problems that occurred because a text was written under the assumption of one (8-bit) encoding, but interpreted later under a different encoding.

While this was a huge step forward for correctly interpreting any source document (because it eliminated all of the the different input encodings — all is now

[4] There are a few exceptions where some seldom used glyphs are missing, e.g., `\textpertenthousand` or `\textcompwordmark`.

Unicode), it unfortunately reintroduced a new helping of tofu through the back door.

The reason is simple: with Unicode as the means to reliably address a glyph to be typeset in a font, such a font has to contain glyphs for *all* characters available in Unicode, because TEX just takes the Unicode number and tells the current font "typeset this glyph". While this is in theory possible in the TrueType or OpenType font formats (using font collections), there is no single font (or collection) that offers anything close to this.[5] LATEX has no way to identify if glyphs are missing, because the typesetting of paragraph text is a very low-level process in TEX and in contrast to the pdfTEX engine where LATEX can reliably assume that a font in T1 encoding implements the whole encoding, in Unicode engines all fonts are in the TU encoding (the whole of Unicode), which no font provides.

In theory it would have been possible to devise sub-encodings of TU and assign each and every font to the appropriate sub-encoding, as was done with TS1, but in practice this would be a hopeless undertaking, because each and every font implements its own set of glyphs, so no useful classification is possible.

Thus when you typeset in XƎTEX or LuaTEX and you request using a certain font family with something like

```
\setmainfont{Alegreya}
```

you just have to hope your chosen family contains glyphs for all characters that you intend to use in your document; if not, you will end up with tofu.

To give you some figures: Latin Modern Roman (the default font in LATEX on Unicode engines) implements 794 characters, the ParaType font used for this article 717, the Optima font on the Mac just 264, the free Alegreya font 1251 and Noto Serif 2840. Regardless, there can be no guarantee that the characters contained in your document are covered.

Letting TEX tell you about your tofu

The TEX program offers one tracing parameter, called \tracinglostchars, that, if set to a positive value, reports missing glyphs (a.k.a. tofu) in the log file, e.g.,

```
Missing character: There is no
                È (U+00C8) in font cmr10!
```

[5] The font I know that comes closest is Code2000 [2], which provides around 90K characters in its latest incarnation — but even that is only a fraction of the Unicode universe (over 140K characters). Google's Noto project [1], which stands for "**no to**fu", was established to develop fonts for typesetting text in any of the world's languages and scripts. It currently has almost 64K characters, which are split across nearly two hundred font families, e.g., if you want to typeset in Latin you can use Noto Sans, but for Japanese you need Noto Sans Japanese and so forth.

Interestingly enough, this information is not even given by default, but only when you explicitly ask for it — obviously, Don Knuth did not foresee that TEX is used with fonts other than those carefully crafted for TEX and containing all the characters you may want.

Recently all TEX engines were enhanced to make tofu reporting a little better: you can now set this parameter to 2, after which it reports its finding also on the terminal (the new default value in LATEX), or you can set it to 3, after which it will throw an error rather than the easy-to-miss warning. With Unicode engines we strongly recommend to always set

```
\tracinglostchars=3
```

in the preamble of your document — it is much better to get errors when writing your documents instead of getting reports by others about tofu in your published work. As explained before, when typesetting with pdfTEX there is little danger of ending up with tofu, so there it is less important to change the parameter, though it obviously doesn't hurt.

3 Typesetting Unicode font tables

When I worked on the font chapter for the new edition of *The LATEX Companion*, third edition [5], I wanted to produce glyph tables for various fonts to examine which characters they encode and how they looked. To my surprise I could not find any TEX tool to do this for me. There is, of course, the old nfssfont which I had adapted from work by Don Knuth, but that is of no help with Unicode fonts as it can only display tables of the first 256 characters, i.e., 8-bit fonts. So during my last stay at Bachotek (before the pandemic) I sketched out some code, the result of which is now available as the unicodefonttable package (see companion article).

References

[1] Google Fonts. Noto: A typeface for the world. fonts.google.com/noto

[2] J. Kass. Code2000. Font resource implementing much of Unicode. en.wikipedia.org/wiki/Code2000

[3] D.E. Knuth. *The TEXbook*, vol. A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[4] F. Mittelbach, R. Schöpf. The new font family selection — User interface to standard LATEX. *TUGboat* 11(1):91–97, Apr. 1990. tug.org/TUGboat/tb11-1/tb27mitt.pdf

[5] F. Mittelbach with U. Fischer. *The LATEX Companion*. Pearson Education, Boston, MA, USA, third ed., to appear in 2022.

◇ Frank Mittelbach
Mainz, Germany
https://www.latex-project.org
https://ctan.org/pkg/unicodefonttable