# What is Font Production?

Font Production (also: Font Mastering) is an umbrella term for all the steps taken from a finished typeface design to a marketed product. Depending on the foundry, this workflow is different, and may involve different views, tools, and solutions.

The steps taken into account depend on the typeface designer in question: An artistic designer like Hermann Zapf would deliver a sheet with an Alphabet to Linotype. Everything after that was *Font Production.*

Any other designer will probably deliver digital files to Linotype (or another foundry). Depending on how comfortable the designer is with the production workflow (and how much of a control freak they are), they might take over one or more of them:

- interpolating different weights
- kerning
- creating OpenType features (beyond kerning)
- hinting (PS & perhaps TrueType)
- filling in font metadata
- generating binaries in various formats (likely OTF & TTF)
- testing in different environments
  (Mac & Win, even in little-liked environments like MS Word)
- proofing fonts, feedback for the designer
- specimen design & marketing

Independent designers who don’t want to distribute their fonts through a foundry (many of which are selling on MyFonts), have to complete the steps above by themselves, or have to pay someone to do it for them (if they are responsible enough). If corners are cut (for instance, dropping testing), it is likely they will have to deal with customer complaints.

# Kerning

Kerning is the process of creating spacing exceptions. No matter how well a font is spaced, some combinations (e.g. AVATAR) just need some manual adjustment. Kerning pairs should not be the norm for every combination, rather something like a “last resort”.

Unless you design a monospaced typeface, your fonts will need kerning. The bigger fonts get, the more kerning is needed. However, please start kerning only *after* the design is completely finished.

A word of warning - kerning is very subjective. The definition of a “well-kerned” typeface may change from one designer to another - and it even changes over time (look at fonts from the ’70s). Kerning may be a little bit overrated these days - generations grew up reading books set on Linotype machines, which had zero possibility for kerning.

The kerning workflow is tedious, and needs a lot of patience  because of the amount of pairs to check - number of glyphs by the power of 2.

## Kerning Tools

- Metrics Machine
- Various Robofont Extensions
- Glyphs, *KernKraft* extension
- FontLab
- DTL KernMaster
- iKern

## Kerning Strategies

Depending on the application, the kerning workflow may be different. Metrics Machine relies on *pair lists*, which are lists of letter combinations to be created create within the application. This is convenient because it allows for flexible adaptation to a given font file.

FontLab will rely on creating your own kerning proofs as .txt files - which is good for consistency across families, but it is easy to miss things.

iKern is a paid service provided by Igino Marini. He claims to have found a way to do automated kerning algorithmically. Some designers trust it, others don’t. You have to get in touch with him through iKern.com.

## Group Kerning

Group Kerning (or Class Kerning) is a way to facilitate the kerning of related pairs. Glyphs (such as A Ä Â À Á) are grouped togeter; when they are kerned against other grouped glyphs (such as o ö ô ò ó), all possible combinations will have the same kerning value. When a group is kerned against an ungrouped glyph, the same rule applies.

## Kerning Exceptions

A glyph can be a member of only a single kerning group for either side. A group of related glyphs will kern well in most situations, but there may be cases where an exception is needed. An example for that would be T kerned against the group (i í ì ï) - while most combinations may work, Tï migt need an exception.

## Kerning Workflow

This list describes the workflow I would use when working with Metrics Machine:

- Organize glyphs into kerning groups.
  Try to be comprehensive, but don’t over-generalize shapes.
- Create reference groups
  A Metrics Machine invention - useful to search for glyphs with certain properties, such as all uppercase, all lowercase, all figures, all Cyrillic, etc.
- Build pair lists
  Depending on how far you want to go, you can create pair lists for: UC–UC, LC–LC, UC–LC, LC–UC, punctuation–UC–punctuation, punctuation–figures–punctuation, etc. Many possibilities.
- Go through pair lists
  Depending on the size of the project, this may take from a couple of days to more than a week (or your whole life)
- Proof fonts (In InDesign, printed, etc)
- Repeat the last steps until one font is fully kerned
- Move the kerning from one master to another master and repeat the last three steps for that master (kerning interpolates!)

# Font Formats

Today, there are two “big” font formats for use on the desktop, the web & elsewhere - CFF & TTF. There is a variety of file formats associated with those (OTF, TTF, WOFF, etc), but the internal structure is very similar.

- both formats describe vector–based outline fonts
- both formats can contain up to 65,000 glyphs per font
- both may contain typographic features (OpenType features)

## CFF – Compact Font Format

- Invented by Adobe, based on PostScript.
- Possible filename suffixes: .otf .pfa .pfb .woff .woff2 .otc (collection of multiple fonts)
- Curves are described with 3rd–order beziers ($x^3$), known from Adobe Illustrator
- CFF fonts often are based on a grid of 1000 UPM
- Hinting is generalized and relies on a smart rasterizer
- Subroutinization (re–using of shapes) makes those files very compact

## TTF – True Type Font

- Co–Invented by Microsoft & Apple
- possible filename suffixes: .ttf .otf .woff .woff2 .eot .ttc (collection of multiple fonts)
- curves are described with 2nd–order beziers ($x^2$)
- often are based on a grid of 2048 UPM
- hinting is a more involved process, rasterization may therefore be better under certain circumstances (e.g. Windows XP).

Many other font formats preceded OTF & TTF. Most notably T1 or “PostScript” fonts, which held a maximum of 256 glyphs, were not cross–platform, split into multiple files, and did not contain Unicode values.

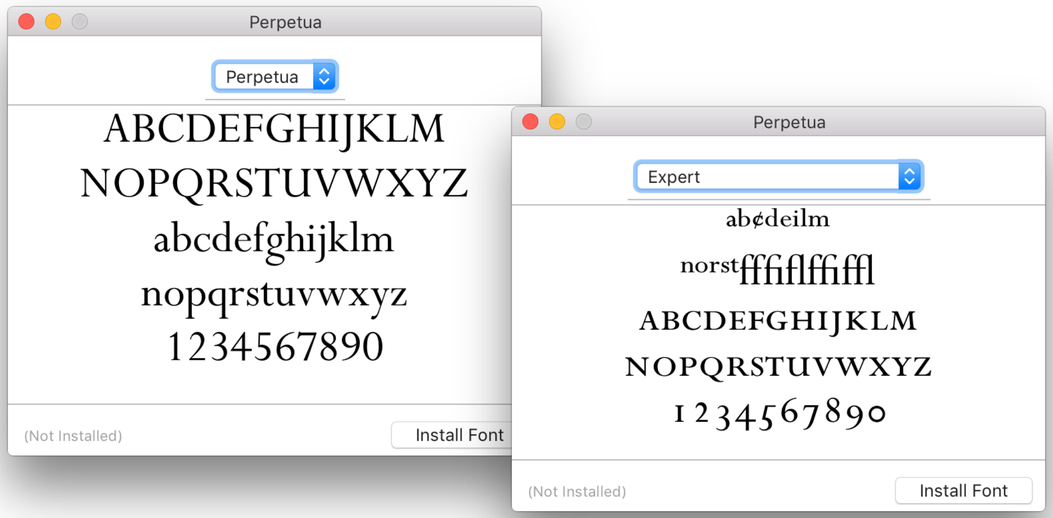
## What are Pro, Std & Expert fonts?

Those are mostly marketing terms, their meaning changes from foundry to foundry.

Adobe has come up with the “Pro” suffix. When the library of Type 1 fonts (e.g. *Trajan*) was converted to OpenType, functionality was added. Since it was likely that users would have both old & new versions of a given typeface on their systems, fonts needed a new family name – they were renamed to (for instancea) *Trajan Pro*. When Trajan was extended to Cyrillic and Greek in 2012, it was named *Trajan Pro 3*, because it represented the official 3rd itera-tion of the (digital) Trajan.

FontShop’s FontFont label distinguishes Std from Pro fonts by their language support. Languages such as Polish are not supported in the *Standard* fonts, so designers who want to use Polish have to go *Pro* (Not sure what is so professional about that).
I assume this artificial distinction is a way to offer cheaper purchase options for the same design.

## Expert Fonts



A single design, split into two files. For typography *Experts*. To enter an ffi ligature, the designer had to choose the Expert Font, and type Z – a semantic nightmare.

## Web Fonts

The biggest issue fonts on the web have created is a licensing problem. Browsers are perfectly happy dealing with a “raw” .otf or .ttf in the assets of a web site, but the creator of these fonts might not be too happy about it, because anyone can just download them. To solve that problem, different font formats have been developed exclusively for the web:

### WOFF

WOFF is the most common font for web fonts, and essentially is just a container format. It may be compressed and contain some additional metadata, but otherwise it is just the same as the original font file. WOFF can “wrap” both OTF and TTF fonts, and does nothng else but create a barrier for the casual user - a WOFF file cannot be used as a desktop font.
Today, WOFF is supported in all browsers.

### WOFF2

WOFF2 is an intended improvement of WOFF, proposed by Google. The main difference is shrinking its file size by using various kinds of compression algorithms for various kinds of tables, based on the data they are most likely to contain.
Currently not supported in Safari, IE.

### EOT

Invention by Microsoft - “Embedded OpenType” fonts are TrueType, not OpenType fonts. They are exclusively supported to Internet Explorer, and have lost significance.

### SVG fonts

An old format for web font delivery, bloated and bulky. For a while, they were the only kinds of fonts supported in iOS and Safari, that is why vendors distributed them. SVG web fonts had major shortcomings (such as not supporting hinting). History.

## How are webfonts created?

Since all webfont formats are just modified desktop fonts, the starting point always is a .ttf or an .otf.

Various command line tools can help you with web font conversion. My esteemed colleague Bram Stein has compiled a handy list of tools here:

https://github.com/bramstein/homebrew-webfonttools

WOFF:

```
sfnt2woff inputFont.otf
sfnt2woff inputFont.ttf
```

WOFF with better compression:

```
sfnt2woff-zopfli inputFont.otf
sfnt2woff-zopfli inputFont.ttf
```

WOFF 2:

```
woff2_compress inputFont.otf
woff2_compress inputFont.ttf
```

Some websites will convert your fonts to webfonts, too (such as the FontSquirrel Webfont Generator) - however, if you would like control over your files, doing it locally is perhaps the better option.

## Tables

Fonts are binary files which we cannot easily open in a text editor. However, there are tools at our disposal which make inspecting font tables easy:

- ttx - a command-line tool by Just van Rossum
- spot - a command-line tool, part of the Adobe FDK
- DTL FontMaster - a UI tool, which is free in a basic version.

The internals of font files (TTF & OTF) are based on the same standard, and therefore their structure is very similar. Fonts could be looked at as a collection of Excel Spreadsheets. Tables usually have a four-letter name and each very specific functionality. Examples:

| | |
|---|---|
| **cmap** | Glyph to Unicode mapping |
| **head** | General information about the font, such as the version number, creation date, the bounding box of all glyphs, etc. |
| **hhea** | Some general horizontal metrics, such as the largest left & right sidebearings, some vertical metrics. |
| **hmtx** | Advance width & left sidebearing for each glyph |
| **maxp** | The number of glyphs in the font (used to calculate memory requirements) |
| **name** | Naming table for font name, style name, designer information, license information, foundry URL etc. |
| **OS/2** | OS/2 and Windows specific metrics, such as average character width, classified width & weight for the font, Panose number, Vendor ID, etc. |
| **glyf** | The actual outlines in a TTF font |
| **CFF** | The actual outlines in an OTF font (Compact Font Format) |

This table-based format is called *sfnt* and can easily be expanded with new tables. One recent expansion is the **sbix** table, which includes PNGs - Apple Color Emoji.

Two tables include OpenType features:

| | |
|---|---|
| **GSUB** | All OT features that deal with glyph substitution, such as Small Caps, Ligatures, Stylistic Alternates, etc. |
| **GPOS** | OpenType features that deal with glyph positioning, such as Kerning, Capital Spacing, Mark feature. |

To have a look at the OpenType feature code contained within a font for which we don’t have the source files, we can use **spot:**

```
spot -t GSUB=7 Testfont.otf
spot -t GPOS=7 Testfont.otf
```

Spot also has a way to build a visual proof for OpenType features:

```
spot -P liga Testfont.otf > liga.ps
spot -P smcp,kern Testfont.otf > smcp_kern.ps
spot -Proof Testfont.otf > proof.ps
```

The .ps (Postscript) file can just be opened in Preview or Acrobat and “distilled” into a PDF.

## OpenType Features

OpenType features are delightful, and something many users expect in fonts today. One only hopes that InDesign and other layout applications will do a better job at exposing them.

Some OpenType features are optional, others essential. Oftentimes, OT features are used as a selling point for a typeface – in my opinion, this is misguided. OT features always depend on the design, and should be informed by it.

A fairly standard set of features to be expected from a text face today:

- Small Caps
- various styles of figures
  (old style, lining, tabular & proportional)
- ligatures (in case the design calls for them)
- superior and inferior figures, and perhaps letters
- fractions

Script typefaces might need other features:

- contextual alternates
- swashes
- discretionary ligatures
- stylistic sets
- ornaments

It makes sense to think about the reason for writing a feature or anoter - Sans Serif designs might not need all the same features as a Serif for Text; not all script typefaces are equal, etc.

## How to write OpenType features

OpenType Features can be written in Robofont’s “Feature” tab. In Glyphs, the syntax is the same, but the opening and closing brackets (feature xxxx { ) are not necessary.

The syntax for writing OT features is quite simple. There are two general kinds of features, GSUB and GPOS. SUB stands for substitution, POS is short for position. An example for a typical GSUB feature would be Small Caps:

```
feature smcp {
    sub a by A.sc;
} smcp;
```

A typical GPOS feature is kerning:

```
feature kern {
    pos A V -80;
} kern;
```

Common Feature Tags:

| | |
|---|---|
| **liga** | ligatures (for solving spacing problems) |
| **dlig** | discretionary ligatures (just for decoration) |
| **pnum** | proportional figures |
| **onum** | old style (text) figures |
| **tnum** | tabular figures |
| **lnum** | lining figures |
| **smcp** | small caps |
| **c2sc** | caps to small caps |

The best possible (and up-to-date) guide to writing OT features was written by Tal Leming - found at http://opentypecookbook.com

## Hinting

To be Best covered in FDK presentation

## Character Sets

Best explained in Andy’s PDF

## Vertical Metrics

Best explained in Andy’s PDF