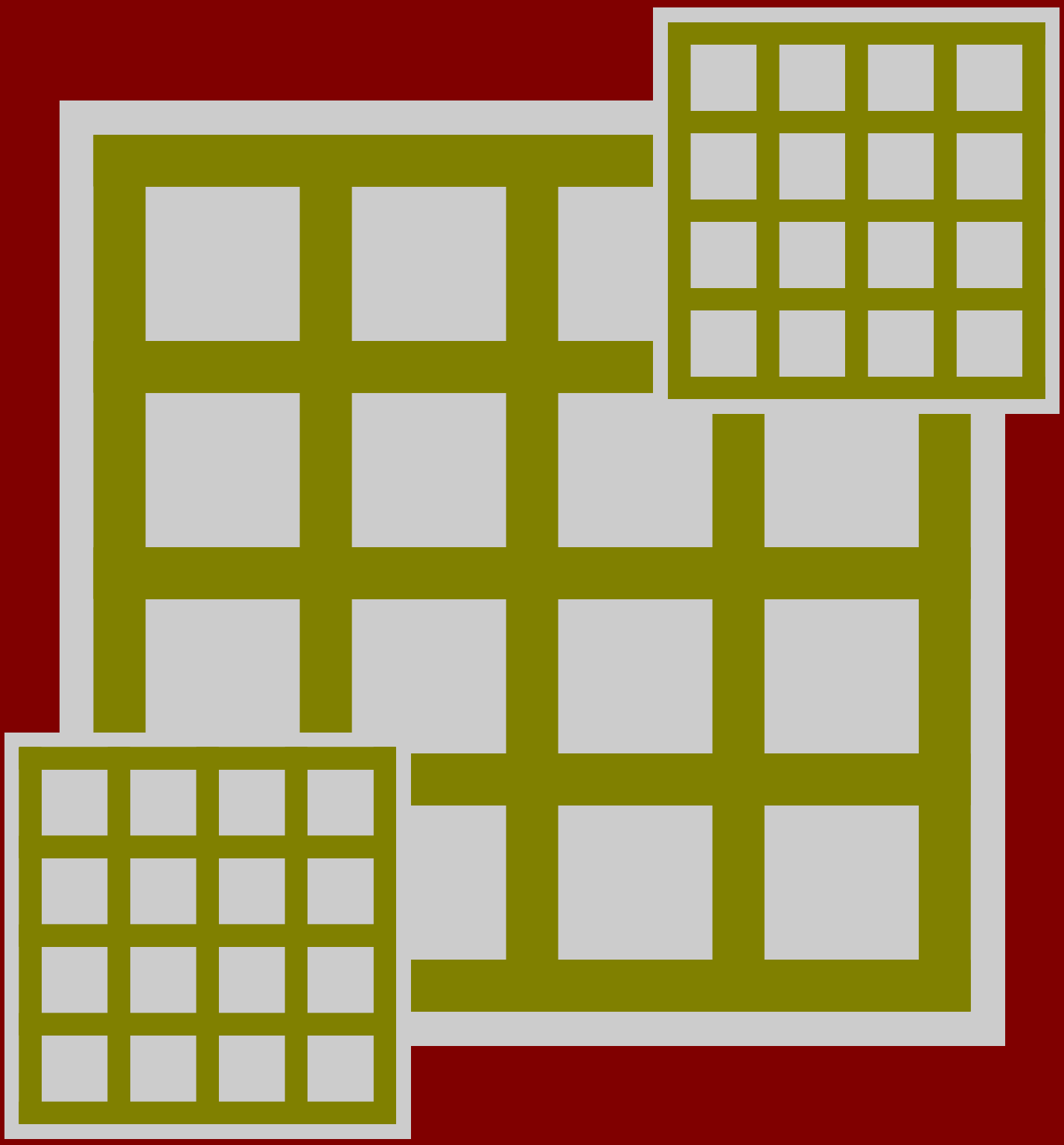


TEXFONT explained

Introduction

It is probably a known fact that TEX can handle fonts of any kind, as well as satisfy the needs of those who do not live in english speaking countries. Nevertheless, the majority of documents typeset by TEX, use the more or less standard Computer Modern Typefaces in an encoding not that well suited for non english usage.

The (subtle) relations between font encodings, hyphenation patterns, input encodings, operating systems, font metrics, and alike makes font handling the most complicated (and fuzzy) part of macro packages.

Although TEX distributions come with many fonts, in often obscure names, made even more obscure by encoding specific instances and/or exceptions. Browsing the font sub–tree of the `texmf` directory can drive you crazy.

As a result, installing a font, or at least making sure that the fonts installed already can be used, is not trivial. It is not enough to make your macro package aware of their existence, you also should inform the post processor (which in the case of PDFTEX is build in) of (re)encoding issues of shape manipulations. And of course each back–end has its own demands.

In CONTEXT, font support is implemented in layers. One can use an individual font, define a relationship between them, and collect such relationships in typeface collections. These issues are covered in the font manual that comes with CONTEXT.

In this manual we will describe TEXFONT, a PERL script that can help you to manage the files that are needed for TEX and the back–ends. We hope that this tool is of help, although we cannot guarantee that life will be easy from now on. The script only covers 'normal' fonts. For setting up math fonts, vendors often provide the necessary files and CONTEXT is aware of these files. Special fonts, like expert fonts, assume a more in depth knowledge of font handling. We may deal with them in the future.

The more demanding user can of course fall back on more complicated tools like `fontinst`. Although I never used this tool, my impression is that it does a good job on manipulating fonts, but most users will not need this kind of fine tuning.

Currently this manual is directed towards CONTEXT *users who also use* PDFTEX*. Future versions may also cover* DVI *(to* PDF*) drivers.*

Organizing fonts

In the TEX community, much comes for free. This is definitely not true for fonts, although a decent free collection is available to get you started. There are thousands of interesting fonts out there, and occasionally you may want to buy one.

So, what to do when this floppy or CDROM arrives. A quick look at the contents will show you that there are at least `pfb` and `afm` files on it. If not, then you have a problem. But, given that these files are there, where should they go?

Regular TEX distributions are organized conforming the TEX Directory Structure. There you will find the commonly used and stable components under the main tree. In this tree you will find a font sub–tree:

```
tex/texmf/fonts
```

Any deviation from the default components, being your own extensions or updates, should go into a local tree:

```
tex/texmf-local/fonts
```

The main tree is preferably just a copy of for instance the one that comes on TEX Live. The `tex/texmf-local` path is where you normally will unzip a ConTEXt update. This is also a natural place to put your fonts, given that you have access to this tree. That way, you can easily replace the main tree without spoiling your local settings.

When you buy a font for your own usage only, the local tree is a good place for them. But when you want to share them you need to be aware of licensing issues. Licences may permit installation on 5 machines, printing on 1 printer, placement on a server with restricted access for 3 simultaneous users, and alike.

So, this is why it may make sense to introduce another tree, which we can maintain separately. In this tree we put those files that are needed to use the fresh fonts. So we may have at least the following paths:

```
tex/texmf-fonts/fonts
tex/texmf-fonts/pdftex/config
```

Below the `/fonts` subpath the font specific files are organized according to vendor and collection.

```
texmf / fonts  / tfm    / vendor   / name / *.tfm
               / afm    / vendor   / name / *.afm
               / pfm    / vendor   / name / *.pfm
               / vf     / vendor   / name / *.vf
               / type1  / vendor   / name / *.pfb
```

The PDFTEX directory is organized as follows

```
texmf / pdftex / config /                  *.cfg
               / config /                  *.map
               / config / encoding /       *.enc
```

Of course you need to set up a couple of environment variables in order to let your TEX system understand this all. Here we assume that you use WEB2C.

```
TEXMFMAIN=/tex/texmf
TEXMFLOCAL=/tex/texmf-local
TEXMFFONTS=/tex/texmf-fonts
```

The order of searching these trees is determined by the following variable:

```
TEXMF={$TEXMFFONTS,$TEXMFLOCAL,!!$TEXMFMAIN}
```

Specific settings can be taken care of in the `texmf.cnf` file. The previous definition could have gone there.

```
TEXMFCNF=/tex/texmf-local/web2c
```

Since TEXFONT looks for both the local and dedicated font tree, you're free in your choice, although from the perspective of managing fonts it may be handy to use the dedicated tree.

Back to the question "Where should my font files go?" we can now answer: "In the font tree.". Say that you've bought Officina from ITC, you can put the files in:

```
/tex/texmf-fonts/fonts/source/itc/officina
```

The files that TEX needs for using these fonts, will later be moved to and/or put in:

```
/tex/texmf-fonts/fonts/afm/itc/officina
/tex/texmf-fonts/fonts/tfm/itc/officina
/tex/texmf-fonts/fonts/vf/itc/officina
/tex/texmf-fonts/fonts/type1/itc/officina
```

Here, type1 does not denote a suffix, but a more verbose naming of pfb files. The tfm files contain the metric information that TEX needs during the typesetting process, while the virtual font files vf are needed in the postprocessing stage to sort out how glyphs are composed (if they are composed at all).

Generating metrics

Before you can set up CONTEXT to use fonts, you need to generate the metric files needed.

```
texfont --ve=itc --co=officina --ma --in
```

This command means as much as: we're going to handle the Officina collection from vendor ITC, make all directories needed, and install the files there. Installation here comes down to copying the afm and pfb files, as well as generating the tfm and vf files.

We started in the installation path with files like:

```
ovb_____.afm
ovb_____.pfb
```

and end up with files spread over the font tree named:

```
/tex/fonts/afm/itc/officina/ovb_____.afm
/tex/fonts/type1/itc/officina/ovb_____.pfb
/tex/fonts/tfm/itc/officina/raw-ovb.tfm
/tex/fonts/tfm/itc/officina/texnansi-ovb.tfm
/tex/fonts/vf/itc/officina/texnansi-raw-ovb.tfm
```

From this you can deduce that we clean up names, as well as handle an encoding vector. By default we use the texnansi vector, but you can specify another one if needed:

```
texfont --en=ec --ve=itc --co=officina --ma --in
```

TEXFONT also generates a TEX file that demonstrates the usage of these fonts. Normally you will define typescripts to take care of this, or use the predefined ones that come with CONTEXT.

```
\definefontsynonym[OfficinaSerif-Bold][ec-ovb][encoding=ec]
```

We also need to instruct PDFTEX on how to embed the font. For this purpose, a map file is generated:

```
\loadmapfile[ec-itc-officina.map]
```

In this map file, you will find lines like:

```
ec-raw-ovb OfficinaSerif-Bold 4 < ovb_____.pfb ec.enc
```

In practice it means that TEX will base its typesetting decisions on the metric file, but for inclusion will fall back on the associated (to be reencoded) file. This file has references to the raw file and these references are resolved in the map file.

The default encoding is `texnansi`. For western languages, `8r` and `ec` are also a good choice. But, in any case, make sure that all the characters that you need are there by processing the generated `tex` file.

```
texexec ec-itc-officina
```

If this file processes all right, then at least you know that you have the correct files on your system. When processing this file, you can enable compact mode.

```
texexec ec-itc-officina --mode=compact
```

Example

Normally you need more then one run of TEXFONT to create the files needed. The next sequence removes old instances and installs the new fonts. We assume that you have put the `afm` and `pfb` files on the sourcepath relative to the current location.

```
texfont --ve=itc --co=officina --re
texfont --ve=itc --co=officina --so=itc/officina --ma --in

texfont --ve=itc --co=officina --so=itc/officina --ca=* ovbk_*
texfont --ve=itc --co=officina --so=itc/officina --sl=* ovbk_*
texfont --ve=itc --co=officina --so=itc/officina --sl=* ovb_*

texfont --ve=itc --co=officina --so=itc/officina --ca=* owbk_*
texfont --ve=itc --co=officina --so=itc/officina --sl=* owbk_*
texfont --ve=itc --co=officina --so=itc/officina --sl=* owb_*
```

For setting up CONTEXT to actually use these fonts, you need to take a look at the `tex` file that is generated and/or take a look at the font manual. Of course you can

ask around to see if someone already has the typescripts made. Without comment we show how such a definition looks. We assume that you put these definitions in a file called `typeface.tex`.

```
\starttypescript [map] [8r]

\loadmapfile [8r-itc-officina]

\stoptypescript

\starttypescript [serif] [officina] [name]

\definefontsynonym[Serif]           [OfficinaSerif-Book]
\definefontsynonym[SerifItalic]     [OfficinaSerif-BookItalic]
\definefontsynonym[SerifSlanted]    [OfficinaSerif-BookSlanted]
\definefontsynonym[SerifBold]       [OfficinaSerif-Bold]
\definefontsynonym[SerifBoldItalic] [OfficinaSerif-BoldItalic]
\definefontsynonym[SerifBoldSlanted][OfficinaSerif-BoldSlanted]
\definefontsynonym[SerifCaps]       [OfficinaSerif-Caps]

\stoptypescript

\starttypescript [sans] [officina] [name]

\definefontsynonym[Sans]           [OfficinaSans-Book]
\definefontsynonym[SansItalic]     [OfficinaSans-BookItalic]
\definefontsynonym[SansSlanted]    [OfficinaSans-BookSlanted]
\definefontsynonym[SansBold]       [OfficinaSans-Bold]
\definefontsynonym[SansBoldItalic] [OfficinaSans-BoldItalic]
\definefontsynonym[SansBoldSlanted][OfficinaSans-BoldSlanted]
\definefontsynonym[SansCaps]       [OfficinaSans-Caps]

\stoptypescript

\starttypescript [serif] [officina] [8r]

\definefontsynonym[OfficinaSerif-Book]       [8r-ovbk][encoding=8r]
\definefontsynonym[OfficinaSerif-BookItalic] [8r-ovwi][encoding=8r]
\definefontsynonym[OfficinaSerif-Bold]       [8r-ovb] [encoding=8r]
\definefontsynonym[OfficinaSerif-BoldItalic] [8r-ovbi][encoding=8r]

\definefontsynonym[OfficinaSerif-BookSlanted][8r-ovbk-slanted-167]    [encoding=8r]
\definefontsynonym[OfficinaSerif-BoldSlanted][8r-ovb-slanted-167]     [encoding=8r]
\definefontsynonym[OfficinaSerif-Caps]       [8r-ovbk-capitalized-800][encoding=8r]

\stoptypescript

\starttypescript [sans] [officina] [8r]

\definefontsynonym[OfficinaSans-Book]       [8r-owbk]                [encoding=8r]
\definefontsynonym[OfficinaSans-BookItalic] [8r-owwi]                [encoding=8r]
\definefontsynonym[OfficinaSans-Bold]       [8r-owb]                 [encoding=8r]
\definefontsynonym[OfficinaSans-BoldItalic] [8r-owbi]                [encoding=8r]
\definefontsynonym[OfficinaSans-BookSlanted][8r-owbk-slanted-167]    [encoding=8r]
\definefontsynonym[OfficinaSans-BoldSlanted][8r-owb-slanted-167]     [encoding=8r]
\definefontsynonym[OfficinaSans-Caps]       [8r-owbk-capitalized-800][encoding=8r]

\stoptypescript
```

In your document style, you can now stick to simple definitions like:

```
\usetypescriptfile[typeface]

\definetypeface[officina][rm][serif][officina][default][encoding=8r]
\definetypeface[officina][ss][sans] [officina][default][encoding=8r]

\setupbodyfont[officina,rm,10pt]
```

Of course you can mix these fonts with other ones, in which case you may want to apply relative scaling first. The fonts manual also explains how you can set up this font to use protruding characters. Since we use symbolic names, you can also use these to access the fonts in for instance special situations, like when you construct a title page:

```
\definefont[TitleFont][Sans at 72pt]
\definefont[TitleFont][OfficinaSans-Book at 72pt]
```

Although you can also access the file name directly, these methods are more descriptive.

Tweaking shapes

A font is seldom just one shape and file. Apart from the upright shape, there can be an italic, bold and bold–italic alternative, or, oblique, bold and bold–oblique. Italic and oblique share their forward slanted look. By applying appropriate transformations, you can slant any font or make it wider or narrower.

So, we have at our hands, either or not by manipulation, normal, italic or oblique, slanted, as well as their bold variants. The following command creates a slanted bold officina font.

```
texfont --ve=itc --co=officina --sl=* ovb
```

Instead of a `*`, you can provide a number. The default slant is .167. Another manipulation is to widen a font, using the extend key:

```
texfont --ve=itc --co=officina --ex=1.2 ovb
```

These commands lead to font metric files with names as:

```
texnansi-ovb-slanted-167.tfm
texnansi-ovb-extended-1200.tfm
```

Combinations are also possible. In the default TEX distributions, where the 8 character file name limit is still honoured, a less verbose naming scheme is used. Our alternative is less efficient, but opens the possibility to have more then one slanted alternative.

Since TEXFONT is mainly a wrapper around `afm2tfm`, we also provide a third manipulation: creating small caps fonts.

```
texfont --ve=itc --co=officina --ca=* ovb
```

We now get:

```
texnansi-ovb-capitalized-800.tfm
```

Generating instances

The Computer Modern Roman typefaces were created by Donald Knuth and their shapes are described as programs. There are quite some design axis and parameters that can be tweaked in order to get different instances. This is why they qualify as meta-fonts. In its own way, Adobe has created the Multiple Master Fonts format.[1]

If you decide (or are forced) to use a multiple master font, you need to make sure that you get all the files needed to tweak them. For instance, for using the Myriad fonts, you need files like:

```
MyriadMM-LightCn.afm          MyriadMM-LightCnIt.afm
MyriadMM-BlackCn.afm          MyriadMM-BlackCnIt.afm
MyriadMM-LightSemiEx.afm      MyriadMM-LightSemiExIt.afm
MyriadMM-BlackSemiEx.afm      MyriadMM-BlackSemiExIt.afm
MyriadMM.amfm                 MyriadMM-It.amfm
MyriadMM.pfb                  MyriadMM-It.pfb
```

When the author bought this font, it came as an install binary, that needed the Adobe Type Manager. After a couple of failures, he finally found some `pfb` and `mmm` files on my system.

[1] Probably due to lack of interest, Adobe is no longer advocating this format, although it will survive in Open Type fonts.

You need the programs `mmafm` and `mmpfb` to create the instances.[2] These programs need an `amfm` file, which did not came on the floppy, but a bit of emailing and renaming finally lead to the files mentioned previously.

The following commands will create an acceptable collection of normal and italic, bold and bold italic shapes.

```
texfont --we=400 --wi=600 MyriadMM
texfont --we=700 --wi=600 MyriadMM
texfont --we=400 --wi=600 MyriadMM-It
texfont --we=700 --wi=600 MyriadMM-It
```

After this you will have files like:

```
MyriadMM-weight-400-width-600.afm
MyriadMM-weight-400-width-600.pfb
```

These alternatives can now be made TEX–ready by saying:

```
texfont --ve=adobe --co=myriad --ma --in MyriadMM-we*
texfont --ve=adobe --co=myriad --ma --in MyriadMM-It-we*
```

Slanted, boldslanted and capitalized variants can be created with:

```
texfont --ve=adobe --co=myriad --sl=* MyriadMM-we*
texfont --ve=adobe --co=myriad --ca=* MyriadMM-we*
```

Map files

There are several ways to tell PDFTEX which map files to use. One way is to add an entry to the file `pdftex.cfg`, like:

```
map +texnansi-urw-palatino.map
```

You can also add an entry in the TEX file, by saying:

```
\pdfmapfile{+texnansi-urw-palatino.map}
```

or in CONTEXT with:

```
\loadmapfile[texnansi-urw-palatino.map]
```

[2] Currently these are only available for the Unix operating system.

This command prevents duplicate loading of files. If you want ConTeXt to load the files automatically, you can add an entry to your `cont-sys.tex` file:

```
\autoloadmapfilestrue
```

Beware: currently pdfTeX only loads map files before the first page is shipped out. If you define fonts halfway the document, you must make sure that the associated map file is loaded at the top of your document. A rather rough way out is to say:

```
\usetypescript[map][texnansi,ec,8r]
```

or whatever combination of encodings you want. If you provide the keyword `all`, ConTeXt will load all known map files.

Switches

You can control TeXfont's behaviour with command line switches. You can get an overview of these by saying

```
texfont --help
```

Some switches expect a number or string. You can abbreviate switches as long as you make sure that they can be distinguished.

`fontroot=path` This is the place where the files that are generated will go to. You can either set the font root manually, or let TeXfont consult your path settings.

`sourcepath=path` When you install new fonts, by default the current path is taken. This switch can be used to specify an absolute or relative path. If you provide `auto` as path, TeXfont will try to locate the source path by means of the vendor and collection specification.

`vendor=name` When generating metrics, this key is mandatory. It is used as a directory specifier under the `/fonts` path and ends up as part of the map file name.

`collection=name` Like the `vendor` key, when generating metrics, this key is mandatory. It is used as a directory specifier under the `/vendor` path and ends up as part of the map file name.

`encoding=name` Here you specify the font encoding vector. You need to make sure that you have a correct file on your system (like `ec.enc` or `8r.enc`). The default encoding is `texnansi` but any decent alternative will do.

`slant=number` The number specified here is normally not that large. An often used value is `.167` which is also the default. If you provide a `*`, the default value is used. Values between 0.0 and 1.5 are accepted. The number, multiplied by 1000 and rounded ends up in the name of the font instance.

`extend=number` This specifier determines how much a glyph will be stretched in horizontal direction. The default value is 1.2. See also `slant`.

`caps=number` This specifier determines how much lowercase glyphs will be scaled in vertical direction. The default value is 0.8. See also `slant`.

`weight=number` When generating a multiple master instance, this number determines how bold a glyph will look. Values between 200 and 700 give acceptable results. The exact specification depends on the font. The number becomes part of the filename.

`width=number` This number defines how wide a glyph will be. See also `weight`.

`install` This switch instructs TEXFONT to copy files from the source path into the right locations of the font tree.

`makepath` If you provide this switch, TEXFONT will create the paths needed for installing the fonts. Otherwise, you have to do so yourself, otherwise TEXFONT will quit with a warning.

`listing` Given that your sourcepath is set to `auto`, this switch will result in a list of the metric files of the installed fonts.

`remove` Given that your sourcepath is set to `auto`, this switch will result in deletion of the metric files of the installed fonts.

`test` If you are just playing a bit with TEXFONT, you may not want to clobber your font path with files that you will never use. This switch sets the vendor and collection equal to `test`.

`show` If you want, TEXFONT can process the test file it generates during the installation.

`batch` In the next section we will discuss batch files. This switch instructs TEXFONT to treat the filename given as a batch file.

If you peek into the source of TEXFONT, you will notice a couple of more switches. These are not documented here, and may disappear in future versions.

`virtual` Create a virtual font (`vf` and `tfm` file) instead of a normal one (`tfm` only).

Batch files

Once a decent conversion is sorted out, keying in commands like the ones mentioned in previous sections will become an annoyance. This is why TEXFONT supports a crude but effective way of processing batch files. One of the batch files that comes with TEXFONT, `type-tmf.dat`, has entries like:

```
--en=? --ve=urw --co=palatino --so=auto
--en=? --ve=urw --co=palatino --so=auto --ca=* uplr8a
--en=? --ve=urw --co=palatino --so=auto --sl=* uplr8a
--en=? --ve=urw --co=palatino --so=auto --sl=* uplb8a
```

The `?` will be replaced by the encoding passed to TEXFONT when processing this batch file:

```
texfont --encoding=8r --batch type-tmf.dat
```

If you have TEX Live installed, you can play with this file and the examples shown in this batch file, since it only uses the free fonts that are on the system. If you run the whole file, for which you can best set up a font tree first, you get a nice collection of fonts. This way you can build your own (stable) font tree. Another batch file is `type-buy.dat`, a file used at PRAGMA ADE. This file has a companion `type-buy.tex` file that contains typescripts. You need to load this file explicitly in order to get access to these scripts.

```
\usetypescriptfile[type-buy]
```

If you run a batch file with `vendor` and `collection` keys, only the section marked as such will be processed. A section is marked with:[3]

```
# urw palatino
```

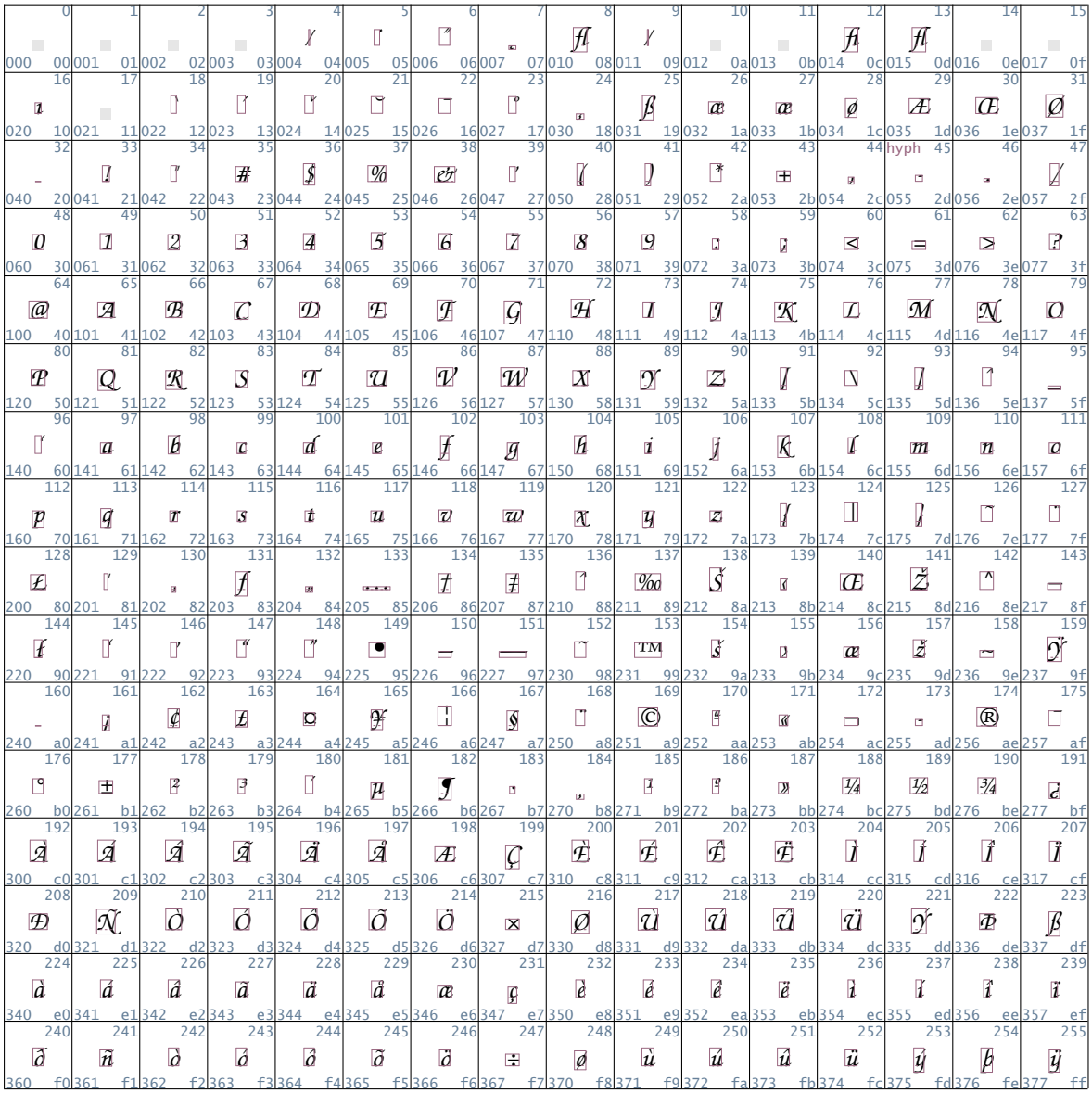
If you make batch files (or sections) for each font collection that you install, you can always regenerate the files quickly. That way you can occasionally clean up your disk and make a fresh start.

On the next pages we will show the public Zapf Chancery font in some rather common encodings. Watch out: the current release of PDFTEX only reads map files that are defined before the first page of the document is shipped out.

---

[3] Instead of `#`, a `%` may be used.

```
texfont --encoding=texnansi --vendor=urw --collection=zapfchan

\loadmapfile[texnansi-urw-zapfchan]

\definefontsynonym[ZapfChancery][texnansi-uzcmi8a][encoding=texnansi]

\showfont[ZapfChancery] \showligatures[ZapfChancery]
```

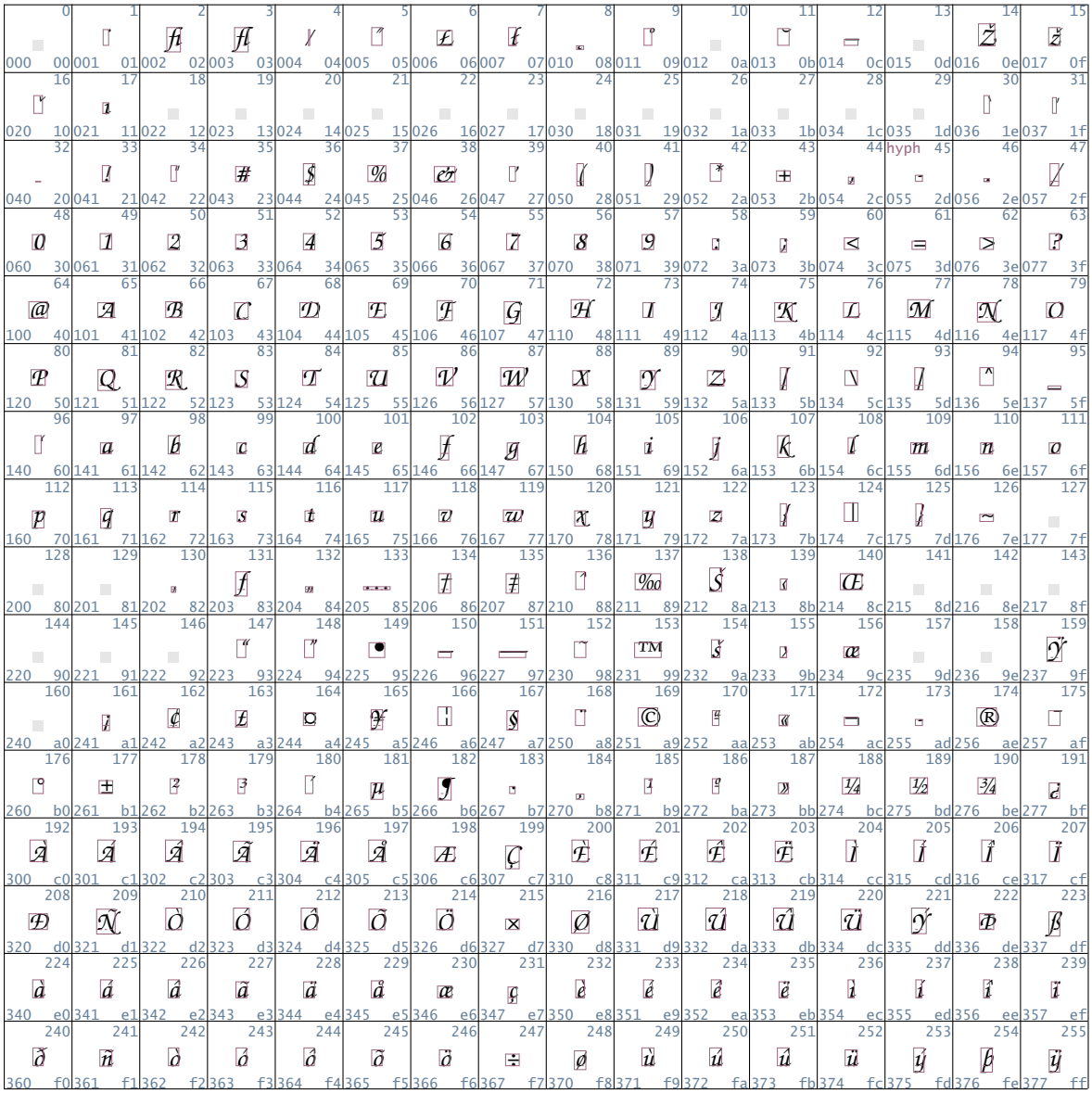


name: texnansi-uzcmi8a at 11.0pt encoding: texnansi mapping: texnansi handling: default

```
ff  fi  fl  ffi  ffl  ‘‘  ’’  --  ---
```

ff fi fl ffi ffl “ ” – —

```
texfont --encoding=8r --vendor=urw --collection=zapfchan

\loadmapfile[8r-urw-zapfchan]

\definefontsynonym[ZapfChancery][8r-uzcmi8a][encoding=8r]

\showfont[ZapfChancery] \showligatures[ZapfChancery]
```



name: 8r-uzcmi8a at 11.0pt encoding: 8r mapping: 8r handling: default

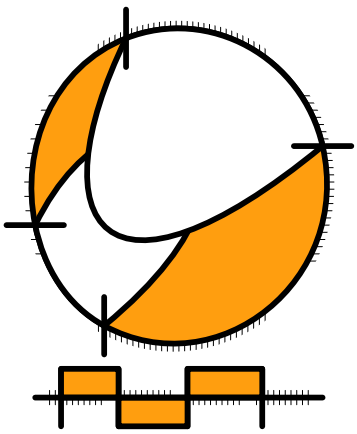
```
ff  fi  fl  ffi  ffl  ‘‘  ’’  --  ---
```

```
texfont --encoding=ec --vendor=urw --collection=zapfchan

\loadmapfile[ec-urw-zapfchan]

\definefontsynonym[ZapfChancery][ec-uzcmi8a][encoding=ec]

\showfont[ZapfChancery] \showligatures[ZapfChancery]
```

name: ec-uzcmi8a at 11.0pt encoding: ec mapping: ec handling: default

```
ff  fi  fl  ffi  ffl  ‘‘  ’’  --  ---
```

ff fi fl ffi ffl “ ” – —



PRAGMA

Advanced Document Engineering | Ridderstraat 27 | 8061GH Hasselt NL
tel: +31 (0)38 477 53 69 | email: pragma@wxs.nl | internet: www.pragma-ade.com