# The Font Installation Guide

Using Postscript fonts to their full potential with Latex

Originally written by

Philipp Lehman

December 2004 · Revision 2.14

# Contents

# Introduction

This guide to setting up PostScript Type 1 fonts for use with TeX and LaTeX is not systematic but task-oriented. It will discuss the most common scenarios you are likely to encounter when installing PostScript fonts. The individual tutorials collected here are not self-contained, though: the second tutorial will presuppose that you have read the first one and so on. All the tools employed in the installation process are documented well, the actual difficulty most users are facing when trying to install new fonts is understanding how to put all the pieces together. This applies to fontinst, the TeX font installation tool, in particular. Controlled by TeX commands, fontinst is a powerful and extremely flexible tool. While its manual documents all available commands individually, you will most likely wonder how to actually employ them after reading the manual. This is what this guide is about.

**Please note that the original author of this guide has resigned as maintainer. As of this writing (August 2010), almost all of the information in this document is still applicable. Section 1.5, however, is in need of a partial update and the Fontname scheme has been gradually losing its attraction.**

## Additional documentation

You will need the following additional manuals while working with this guide:

**The fontinst manual** Shipping as `fontinst.dvi`, the fontinst manual is the most important piece of documentation you will need when working with this guide since all files required for proper PostScript font support can be generated with fontinst. You do not need to work through the sections explaining all low-level commands in detail, but make sure that you have read the more general parts and that you have a basic understanding of what fontinst is and what it does. If this manual is not included in your distribution, get it from the Comprehensive TeX Archive Network (CTAN).[1]

**The Fontname scheme** Fonts used with TeX are usually renamed according to a dedicated naming standard, the Fontname scheme by Karl Berry. Take a look at the outline of the scheme as given in `fontname.dvi` and make sure you have copies of the individual map files at hand. These lists define names for a large number of commercial PostScript fonts. You will need them while working with this guide. If the documentation of the Fontname scheme is not part of your distribution, you can read it online[2] or download the complete package from a CTAN FTP server.[3]

**The LaTeX font selection guide** It might be a good idea to read the LaTeX font selection guide as well before proceeding with the first tutorial. It provides an overview of the New Font Selection Scheme (NFSS), the part of LaTeX which controls font selection. This system is not used during font installation, but it will help you to understand certain aspects of the installation process. This guide ships with most TeX distributions as `fntguide.dvi` and is also available in PDF format from CTAN.[4] Feel free to skip the chapter about math fonts as we are only going to deal with text fonts here. Setting up math fonts is a science in its own right.

## Software requirements

Fontinst and the tools required to compile font metrics and virtual fonts are part of all major TeX distributions. The installation recipes discussed in this guide should therefore work on virtually every platform supported by TeX.

[1] http://www.ctan.org/tex-archive/fonts/utilities/fontinst/doc/manual/
[2] http://www.ctan.org/tex-archive/info/fontname/
[3] ftp://tug.ctan.org/tex-archive/info/fontname.tar.gz
[4] http://www.ctan.org/tex-archive/macros/latex/doc/fntguide.pdf

The installation procedure as a whole, however, includes certain aspects which are specific to a given platform and TeX distribution. Such aspects include the final location of fonts and support files, the way additional tools such as dvips are configured, and the command used to update the file hash tables. In such cases the guide will be referring to the teTeX distribution. Adapting these instructions to other platforms and distributions, however, should not pose a major obstacle. Note that the installation recipes in this guide require fontinst version 1.9 or later. As of this writing (December 2004), version 1.926 is the latest stable release available from CTAN.[5] If you are not sure whether your TeX installation includes an up-to-date version of fontinst, locate the file `fontinst.sty` on your system and inspect its header. Alternatively, you can use the following plain TeX file:

```
\input fontinst.sty
\fontinstversion
\bye
```

Run this file through plain `tex` and inspect the newly created DVI file. It should bear the version number 1.926 or higher. If you need to update fontinst, note that you most probably require a newer version of the PostScript encoding vector `8r.enc` for dvips and pdfTeX as well. This file is distributed separately and is not included in the fontinst release.[6] The latest release of the guide you are just reading can always be found at CTAN.[7] You might want to check for an update before you continue.

## Further assistance

While this guide tries to address the most common questions concerning the integration of PostScript Type 1 fonts with TeX, it cannot cover all possible cases, nor will it introduce all of fontinst's features. If you have any further questions concerning fontinst after reading this guide, you might want to post them on the fontinst mailing list.[8] If you are in need of further assistance with respect to TeX and fonts in general, you might also want to consider

[5] http://www.ctan.org/tex-archive/fonts/utilities/fontinst/
[6] http://www.ctan.org/tex-archive/info/fontname/8r.enc
[7] http://www.ctan.org/tex-archive/info/Type1fonts/fontinstallationguide/
[8] http://www.tug.org/mailman/listinfo/fontinst/

posting them on the TeX-fonts mailing list.[9] Both lists are friendly forums enjoying competent contributors. They are not busy at all times, so allow a few days for replies before you turn to a different forum. Posting questions on the `comp.text.tex` newsgroup is worthwhile as well, especially with more general questions. This newsgroup is very busy at all times, but the replies might be less focused when it comes to specific questions about fontinst or PostScript Type 1 fonts. Please refrain from sending any questions to the author of this guide.

[9] http://www.math.utah.edu/mailman/listinfo/tex-fonts/

# Tutorial 1

# The basics

This introductory tutorial serves two purposes. It covers the most basic installation scenario by explaining how to use fontinst's `\latinfamily` macro to integrate a small font family into a TeX system. By providing step-by-step installation instructions, it will also discuss the installation procedure as a whole. The later tutorials will focus on the more advanced capabilities of fontinst without going over all of the installation steps again. Before we begin, let us take a look at an overview of the installation procedure:

**Step 1: renaming the font files** First of all, we copy all Type 1 fonts (suffix `pfb`) and the corresponding metric files in text format (`afm`) to a temporary directory and rename them according to the Fontname scheme.

**Step 2: creating metrics and virtual fonts** We will use fontinst, a font installer that works with Adobe font metric files in text format (`afm`), to generate TeX metric files and virtual fonts. Fontinst is normally not used interactively but controlled by a TeX file. Since the driver file is specific to a given font family, we need to write a suitable file for our fonts.

**Step 3: compiling metrics and virtual fonts** Fontinst will generate font metrics and virtual fonts in a human-

readable format which need to be converted to a machine-readable form afterwards. Hence we run all property list files (`pl`) created by fontinst through `pltotf` to create TeX font metrics (`tfm`) and all virtual property list files (`vpl`) through `vptovf` to create virtual fonts (`vf`) and the corresponding TeX font metrics for them.

**Step 4: installing fonts and support files** Calling fontinst a font installer is slightly misleading in that it does not move any files around. We still need to move all font metrics (`afm`), Type 1 font outlines (`pfb`), TeX font metrics (`tfm`), virtual fonts (`vf`), and font definition files (`fd`) to the local TeX tree manually.

**Step 5: creating and installing map files** The fonts are now set up for TeX and LaTeX, but not for DVI and PDF drivers, which are configured separately. We create map files for dvips, pdfTeX, and xdvi. We install the map files and add them to the applications' configuration files.

**Step 6: updating the hash tables** Finally, we run `texhash` to update the file hash tables used by the kpathsea search library. The remaining files in our working directory are not required anymore and may be deleted.

## 1.1 Renaming the font files

Users unfamiliar with fontinst tend to moan when introduced to the Fontname scheme for the first time. This file naming standard, which is also known by the name of its creator as the Karl Berry scheme, is often regarded as overly complicated, cumbersome, unclear, and unmanageable. And indeed, it will appear somewhat cumbersome to anyone working with an operating system that does not impose silly limits on the lengths of file names. All of that is not the fault of its creator, however, but an inevitable result of the historical need to encode a complete font designation in a string of eight characters in order to cope with the limitations of filesystems which are not capable of handling longer file names. The most important asset of the Fontname scheme is that it is the only formalized naming system widely used within the TeX community. Given the large number of files required to integrate a given typeface into a TeX system, installations without formal file naming would quickly get out of control. So, if the next couple of paragraphs should sound a bit cumbersome to you, you are in good company. Rest assured that after installing a few font families and watching your installation grow, you will understand the benefits of this scheme.

In order to understand the basic principles of the Fontname scheme, see the file `fontname.dvi` for an overview as well as excerpts from various map files. Browse the map files of individual vendors for the complete listings. When using the `\latinfamily` macro, strict adherence to the scheme is required. If you write a custom fontinst file using lower-level commands, the naming is technically up to you. It is still a good idea to stick to the naming system where possible. If a given typeface is not included in the map file for the respective foundry, take the foundry code from `supplier.map` and the code of the typeface from `typeface.map`. If the typeface is not listed at all, you will need to create a new code. This should be an unused one if possible. Try handling weight, variant, and encoding codes as strictly as possible. Foundry and typeface codes may be handled more liberally.

For large text font families, most font vendors do not put all fonts in a single package. They usually offer a base package containing upright and italic/oblique fonts plus an advanced package which has to be purchased separately and can normally not be used independently in a sensible way. Typical examples of this kind are so-called SC & OSF packages including a set of optical small caps[1] and hanging figures[2] which complement the fonts in the base package. The advanced package might also contain a set of expert[3] fonts, additional weights such as light, semibold, or black, additional widths such as condensed or extended, or symbols fonts providing text ornaments.

We will use Sabon as an example in this tutorial. The Sabon family offered by Adobe is split up into two packages. The base package contains upright and italic fonts (with lining figures) in regular and bold weights, while the so-called SC & OSF package provides optical small caps and hanging figures. Hanging figures are also known as 'old style' figures, hence the name SC & OSF. In the first and the second tutorial we will deal with the base package only. Adding the SC & OSF package to the base install will be discussed in the third tutorial. As we receive the package from Adobe or from a vendor, it contains the following files:

[1] Optical or real small caps, as opposed to mechanical or 'faked' ones, are special glyphs found in a dedicated small caps font. They are preferable to mechanical small caps since they were actually drawn by the font designer. Mechanical small caps are generated by taking the tall caps of the font and scaling them down. The installation of optical small caps will be discussed in tutorial 3.

[2] While lining figures are aligned with the height of the capital letters, hanging or 'old style' figures have ascenders and descenders to blend in with lowercase and mixed case text. Hanging figures are designed for use within mixed case text whereas lining figures are suitable for all uppercase text only. The latter also work well for applications like numbered lists and, since they are usually monospaced, for tabular settings. The installation of hanging figures will be discussed in tutorial 3 as well.

[3] Expert fonts are complements to be used in conjunction with regular text fonts. They usually contain optical small caps, additional sets of figures, ligatures as well as some other symbols. Please refer to tutorial 5 for further information.

```
sar_____.afm    sai_____.afm    sab_____.afm    sabi_____.afm
sar_____.inf    sai_____.inf    sab_____.inf    sabi_____.inf
sar_____.pfb    sai_____.pfb    sab_____.pfb    sabi_____.pfb
sar_____.pfm    sai_____.pfm    sab_____.pfm    sabi_____.pfm
```

Of those files, we only need two types: the font metrics in Ascii format (`afm`) and the binary font outlines (`pfb`). We copy these to our working directory to rename them. In this case, finding the proper names is simple because the typeface is listed explicitly in `adobe.map`:

```
psbr8a    Sabon-Roman                  A    088    sar_____
psbri8a   Sabon-Italic                 A    088    sai_____
psbb8a    Sabon-Bold                   A    088    sab_____
psbbi8a   Sabon-BoldItalic             A    088    sabi____
```

The first column indicates the Fontname name and the last column the original name of the files as shipped by the vendor.[4] After renaming, we find the following files in the working directory:

```
psbr8a.afm      psbri8a.afm      psbb8a.afm      psbbi8a.afm
psbr8a.pfb      psbri8a.pfb      psbb8a.pfb      psbbi8a.pfb
```

We can now begin with the installation process.

## 1.2 Creating metrics and virtual fonts

Since writing a fontinst file can be quite a time-consuming thing to do, fontinst provides a special macro which is able to deal with standard scenarios like this one. You can look up the `\latinfamily` command in the fontinst manual to understand what it does in detail. For our situation, it will suffice to say that it is able to recognize the standard fonts we provide by their file name – hence the need for strict adherence to the Fontname scheme in this

[4] The fourth column may also prove helpful: it indicates the number of the Adobe font package to which this font belongs. This number will save you a lot of time if you are trying to locate updated metric files for a font on Adobe's FTP server since the files are sorted by package number there.

case. Fontinst will create all metric and auxiliary files required by LaTeX without further directions in the form of lower-level commands. Therefore our fontinst file is as simple as it can get:

```
1 \input fontinst.sty
2 \needsfontinstversion{1.926}
3 \recordtransforms{psb-rec.tex}
4 \latinfamily{psb}{}
5 \endrecordtransforms
6 \bye
```

First of all, we load fontinst and add line 2 to verify that we are using a suitably recent version of fontinst. After that, we basically call the `\latinfamily` macro in line 4, using the base of the file names (the foundry code plus the typeface code) as its first argument. The second argument of this macro is code to be expanded whenever this typeface is used. It is frequently employed to suppress hyphenation of fixed-width fonts by setting the hyphenation character to a non-existing encoding slot. If we wanted to suppress hyphenation for this font family, we would call the macro like this:

```
\latinfamily{psb}{\hyphenchar\font=-1}
```

When installing fixed-width fonts we would also define the integer variable `monowidth`. This variable is used internally by fontinst's encoding vectors. Initializing it to any value will disable ligatures and adjust certain spacing values. The exact value does not matter since fontinst uses the existence of this variable as a boolean indicator – do not set it to zero for proportional fonts. For fixed-width fonts the following line should be given before calling `\latinfamily`:

```
\setint{monowidth}{1}
```

Lines 3 and 5 enclose the `\latinfamily` macro in an environment that records all transformations applied to the fonts during installation and writes them to the external file `psb-rec.tex`. We will need this file later in order to create a map file for Sabon. This process will be discussed in section 1.5. We save our fontinst driver file as, for example, `psb-drv.tex` and run it through `tex`:

```
tex psb-drv.tex
```

The `\latinfamily` macro will create metric files, virtual fonts, and auxiliary files for four different encodings: TeX Base 1, OT1, T1, and TS1. While TeX Base 1 serves as the basis of virtual fonts using other encodings, it is usually not employed as such on the LaTeX level, although `\latinfamily` provides font definition files for the TeX Base 1 encoded fonts as well.

The OT1 encoding is a 7-bit legacy encoding solely suitable for text using the English alphabet only because it requires the use of composite glyphs when typesetting accented letters. These glyphs are inferior to the native glyphs provided by PostScript fonts. When using OT1 encoding and typesetting the letter *a* with a grave accent, for example, TeX does not use the real glyph *ïżœ* as provided by the font because OT1 discards all accented letters (this amounts to almost half of the glyphs found in common PostScript fonts). Instead, TeX will use the stand-alone grave accent and move it over the lowercase letter *a* to form a composite glyph. Apart from their inferior typographic quality, composite letters break TeX's hyphenation algorithm so that words containing an accented letter are not hyphenated beyond this letter. Another problem with them is that they break searching for words containing accented letters in PDF files. In short, OT1 should be considered obsolete unless you need the letters of the English alphabet only. But even in this case, T1 encoding would be a sound choice.

T1, also known as Cork encoding, is a more recent text encoding suitable for a wide range of languages using the Latin script. Also known as Text Companion encoding, TS1 complements T1 by providing additional glyphs such as currency signs and other frequently used symbols like 'copyright' or 'registered'. TS1 is never used as the main text encoding because it merely contains symbols. A user interface to the glyphs found in TS1 is provided by the `textcomp` package. Refer to appendix B for a list of all symbols provided by `textcomp`.

When running the fontinst driver file, fontinst will write a lot of messages to the terminal. These will include warnings about glyphs not being found, since a few glyphs defined in OT1 and T1 encoding are missing from the glyph set of our fonts:

```
(/usr/share/texmf/tex/fontinst/base/ot1.etx
Warning: missing glyph ‘dotlessj’.
Warning: missing glyph ‘lslashslash’.

(/usr/share/texmf/tex/fontinst/base/t1.etx
Warning: missing glyph ‘perthousandzero’.
Warning: missing glyph ‘dotlessj’.
```

```
Warning: missing glyph ‘Eng’.
Warning: missing glyph ‘eng’.
```

These warnings are normal, the missing glyphs are simply not provided by most PostScript fonts. In addition to that, you will most likely be lacking the ligatures ‘ff’, ‘ffi’, and ‘ffl’. This means that they will not be typeset as a single glyph but as a sequence of characters. There is no warning message in this case as fontinst will construct the ligatures using the single-letter glyphs at hand. You will usually find these ligatures in so-called expert fonts which complement the base fonts. Some foundries however, like FontFont, include them in the base fonts. Standard PostScript fonts should always provide the ligatures ‘fi’ and ‘fl’. The situation is worse for TS1 encoding since parts of it are rather exotic, defining glyphs not found in industry-standard fonts such as a ‘copyleft’ symbol, or glyphs which should rather go in a dedicated symbol font such as arrow symbols:

```
(/usr/share/texmf/tex/fontinst/base/ts1.etx
Warning: missing glyph ‘arrowleft’.
Warning: missing glyph ‘arrowright’.
Warning: missing glyph ‘tieaccentlowercase’.
Warning: missing glyph ‘tieaccentcapital’.
Warning: missing glyph ‘newtieaccentlowercase’.
Warning: missing glyph ‘newtieaccentcapital’.
Warning: missing glyph ‘blank’.
Warning: missing glyph ‘hyphendbl’.
Warning: missing glyph ‘zerooldstyle’.
Warning: missing glyph ‘oneoldstyle’.
Warning: missing glyph ‘twooldstyle’.
Warning: missing glyph ‘threeoldstyle’.
Warning: missing glyph ‘fouroldstyle’.
Warning: missing glyph ‘fiveoldstyle’.
Warning: missing glyph ‘sixoldstyle’.
Warning: missing glyph ‘sevenoldstyle’.
Warning: missing glyph ‘eightoldstyle’.
Warning: missing glyph ‘nineoldstyle’.
Warning: missing glyph ‘angbracketleft’.
Warning: missing glyph ‘angbracketright’.
```

```
Warning: missing glyph ‘Omegainv’.
Warning: missing glyph ‘bigcircle’.
Warning: missing glyph ‘Omega’.
Warning: missing glyph ‘arrowup’.
Warning: missing glyph ‘arrowdown’.
Warning: missing glyph ‘born’.
Warning: missing glyph ‘divorced’.
Warning: missing glyph ‘died’.
Warning: missing glyph ‘leaf’.
Warning: missing glyph ‘married’.
Warning: missing glyph ‘musicalnote’.
Warning: missing glyph ‘hyphendblchar’.
Warning: missing glyph ‘dollaroldstyle’.
Warning: missing glyph ‘centoldstyle’.
Warning: missing glyph ‘colonmonetary’.
Warning: missing glyph ‘won’.
Warning: missing glyph ‘naira’.
Warning: missing glyph ‘guarani’.
Warning: missing glyph ‘peso’.
Warning: missing glyph ‘lira’.
Warning: missing glyph ‘recipe’.
Warning: missing glyph ‘interrobang’.
Warning: missing glyph ‘interrobangdown’.
Warning: missing glyph ‘dong’.
Warning: missing glyph ‘pertenthousand’.
Warning: missing glyph ‘pilcrow’.
Warning: missing glyph ‘baht’.
Warning: missing glyph ‘numero’.
Warning: missing glyph ‘discount’.
Warning: missing glyph ‘estimated’.
Warning: missing glyph ‘openbullet’.
Warning: missing glyph ‘servicemark’.
Warning: missing glyph ‘quillbracketleft’.
Warning: missing glyph ‘quillbracketright’.
```

```
Warning: missing glyph ‘copyleft’.
Warning: missing glyph ‘circledP’.
Warning: missing glyph ‘referencemark’.
Warning: missing glyph ‘radical’.
Warning: missing glyph ‘euro’.
```

While this may seem like a long list, it is not unusual when installing fonts not specifically designed for TeX. You will get the most common symbols such as currency signs and other frequently used symbols, and chances are that you are not going to miss the lacking ones. If you want to learn more about these encodings, simply run fontinst's encoding vectors through `latex` to get a DVI file containing a commented listing of all the glyphs:

```
latex 8r.etx
latex ot1.etx
latex t1.etx
latex ts1.etx
```

When fontinst is finished, all TeX font metrics and virtual fonts are available in a human-readable format which still requires some post-processing before we can install these files.

## 1.3 Compiling metrics and virtual fonts

In order to convert the TeX metrics into a binary format that TeX can read directly, we run the property list files (`pl`) created by fontinst through `pltotf` to generate TeX font metric files (`tfm`). We also run the virtual property list files (`vpl`) files through `vptovf` to create virtual fonts (`vf`). When using the Bash shell, this can be accomplished as follows:

```
for file in  *.pl; do pltotf $file; done
for file in *.vpl; do vptovf $file; done
```

The generation of TeX font metrics, virtual fonts, and font definition files is now complete.

## 1.4 Installing fonts and support files

The teTeX distribution supports a total of three TeX directory trees by default: a global one, a local one, and a user tree. The global tree is usually maintained by package management software. It contains all files provided by the teTeX distribution. The local tree is for everything that is not part of teTeX but should be available system-wide.[5] The user tree is intended for private files of individual users on the system.

Fonts and everything related to them should go in the local tree if you have administrative access on the system. Putting them in the global tree is a bad idea because they might get overwritten when you update teTeX; putting them in a private one will restrict access to them to a single user which is probably not what you want if you have administrative access. It is a good idea to define the variable `$TEXMF` (all trees) in a way that references `$TEXMFLOCAL` (the local tree) before `$TEXMFMAIN` (the global tree). This will allow you to install newer versions of selected packages in the local tree without updating the whole install. I recommend defining `$TEXMF` as follows in `texmf.cnf`:

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

This will give you two levels on top of the global install: your local extensions will be preferred over files in the global tree and can in turn be overridden by individual users who put files in their private tree (`$HOMETEXMF`). These settings should go into the global configuration file for the kpathsea search library, `texmf.cfg`. For the rest of this section we will assume that we are installing the fonts in the local tree and that its top directory is `/usr/local/share/texmf`. The relevant branches of the local tree are as follows:

```
/usr/local/share/texmf/
/usr/local/share/texmf/dvips/
/usr/local/share/texmf/dvips/config/
/usr/local/share/texmf/fonts/
/usr/local/share/texmf/fonts/afm/
/usr/local/share/texmf/fonts/afm/adobe/
/usr/local/share/texmf/fonts/afm/adobe/sabon/
```

[5] Other TeX distributions such as MiKTeX do not cater for three separate trees but they also support a local tree which is separate from the global system tree.

```
/usr/local/share/texmf/fonts/tfm/
/usr/local/share/texmf/fonts/tfm/adobe/
/usr/local/share/texmf/fonts/tfm/adobe/sabon/
/usr/local/share/texmf/fonts/type1/
/usr/local/share/texmf/fonts/type1/adobe/
/usr/local/share/texmf/fonts/type1/adobe/sabon/
/usr/local/share/texmf/fonts/vf/
/usr/local/share/texmf/fonts/vf/adobe/
/usr/local/share/texmf/fonts/vf/adobe/sabon/
/usr/local/share/texmf/pdftex/
/usr/local/share/texmf/pdftex/config/
/usr/local/share/texmf/tex/
/usr/local/share/texmf/tex/latex/
/usr/local/share/texmf/tex/latex/adobe/
/usr/local/share/texmf/tex/latex/adobe/sabon/
/usr/local/share/texmf/xdvi/
/usr/local/share/texmf/xdvi/config/
```

The main components of this directory structure are defined by the TeX Directory Structure (TDS),[6] another standard introduced to cope with the large number of files that make up a typical TeX system. The appropriate locations for the different file types should be more or less obvious. The `fonts/` branch has subdirectories for Ascii font metrics (`afm/`), TeX font metrics (`tfm/`), Type 1 fonts (`type1/`), and virtual fonts (`vf/`). It is customary to create subdirectories for the foundry and for each font family. You can take the names of these subdirectories from the Fontname scheme as well, although this is not a requirement. The standard directory name for the foundry is given in the file `supplier.map`, the standard name for the typeface in `typeface.map`. Here are the relevant lines from both files for Sabon:

```
p  adobe        @r{Adobe (@samp{p} for PostScript)}
sb sabon        Sabon b:ClassicalGaramondBT
```

The font description files (`fd`) for LaTeX go in a subdirectory of `tex/latex/`. The exact location is up to you but

[6] http://www.tug.org/tds/

I recommend using the `foundry/typeface` scheme as well. We do not need the directories `dvips/`, `pdftex/`, and `xdvi/` at this point, but we are going to use them later. Now we create all directories and copy the files into the local tree as follows:

```
cp *.afm /usr/local/share/texmf/fonts/afm/adobe/sabon/
cp *.tfm /usr/local/share/texmf/fonts/tfm/adobe/sabon/
cp *.pfb /usr/local/share/texmf/fonts/type1/adobe/sabon/
cp *.vf  /usr/local/share/texmf/fonts/vf/adobe/sabon/
cp *.fd  /usr/local/share/texmf/tex/latex/adobe/sabon/
```

All the files that TeX and LaTeX need in order to use Sabon are now available. At this point we could create a perfectly valid DVI file with the right amount of blank space for every glyph – but we would not see a single glyph when looking at a DVI preview. Note that TeX itself is completely indifferent to the actual font files. It will only use the metrics in the `tfm` files without accessing the glyph outlines. Rendering or embedding fonts is at the responsibility of the application which displays the DVI file or processes it further in order to generate PostScript. A DVI file merely contains high-level references to fonts and glyphs, it does not contain the actual font files or any low-level instructions concerning reencoding or rendering. pdfTeX is a special case because it combines the roles of TeX and a PDF driver. All of these applications need to know which fonts to use. This information is provided in 'map' files which map font metrics to font outlines.

## 1.5 Creating and installing map files

In this guide, we will deal with the three most popular applications supporting PostScript fonts, the PostScript driver dvips, the DVI viewer xdvi, and pdfTeX. All of them need to be provided with a suitable map file. Creating map files is a rather laborious task when done manually as used to be custom in the past. As of version 1.9, fontinst is capable of creating map files almost automatically. We still need to provide it with some instructions and the required data, but this data can be collected by fontinst itself during the installation process. For this reason, we had fontinst write the records to the file `psb-rec.tex` in section 1.2. In order to transform these records into a proper map file, fontinst still needs some guidance by means of another fontinst driver file. We will call it `psb-map.tex`:

```
1 \input finstmsc.sty
2 \reststr{PSfontsuffix}{.pfb}
3 \adddriver{dvips}{psb.map}
4 \input psb-rec.tex
5 \donedrivers
6 \bye
```

We start off by loading `finstmsc.sty`, the component of the fontinst package which provides the map file writer we need. Line 2 resets the string used as the suffix of all font files to `.pfb` (the default is `.pfa`) and line 3 activates the map fragment writer for dvips, instructing it to write the properly formated records to `psb.map`, our final map file. We load the data from `psb-rec.tex` and start the transformation process in line 4. After running `psb-map.tex` through `tex`, we get a valid map file for dvips. But what about xdvi and pdfTeX? As of this writing (December 2004), map file support in fontinst is restricted to dvips and dvipdfm. Fortunately, xdvi and pdfTeX are capable of reading dvips's map files to a certain extent. If written with a little bit of care, dvips, pdfTeX, and xdvi can share the same map file. The map files created by fontinst's map file writer for dvips should work fine in this respect. For details on the format of map files, please refer to tutorial 7.

We install the map file by copying `psb.map` to the branch `dvips/config/` in the local TeX tree. In order to configure dvips, we locate the default configuration file of dvips (`config.ps`) in the main TeX tree and copy it to the same location. If the search order for all TeX trees is set up as suggested above, this local copy will now be picked up instead of the global one. We open this file in a text editor, locate the section for map files (lines defining map files begin with a lowercase `p`), and add the new map file so that the updated section looks as follows:

```
% Map files
p +psfonts.map
p ...
p +psb.map
```

The procedure for pdfTeX is similar: the configuration file is called `pdftex.cfg` and map files are marked with the string `map` at the beginning of the line. After copying the file to the branch `pdftex/config` of the local tree and updating it, the relevant section should look similar to the following example:

```
% Map files
```

```
map +pdftex.map
map ...
map +psb.map
```

We repeat this step one more time for xdvi. The configuration file for xdvi is called `xdvi.cfg`, the local branch is `xdvi/config` and lines indicating a map file begin with `dvipsmap`:

```
% Map files
dvipsmap ps2pk.map
dvipsmap ...
dvipsmap psb.map
```

In addition to that, we have to make sure that an encoding definition for TeX Base 1 encoding is provided as well. The configuration file for xdvi should contain the following line:

```
% Tag    Suffix  Encoding name           Encoding file
enc      8r      TeXBase1Encoding        8r.enc
```

The installation is now finished. All files left in the working directory will not be used any more and may be deleted. Do not forget to update the file hash tables by running `texhash` or an equivalent command!

## 1.6 Using the new fonts

Everything you need to know about using the fonts can be found in the LaTeX font selection guide.[7] The second chapter of this guide documents the standard NFSS commands used to switch fonts under LaTeX. Let us take a look at some examples. To select Sabon at any point in a LaTeX file, we use a command like:

```
\fontfamily{psb}\selectfont
```

Sabon provides two weights which are readily available using compact font selection macros like `\textbf` and `\bfseries`. Larger font families may offer more than two weights. To select a particular weight, we use the

[7] http://www.ctan.org/tex-archive/macros/latex/doc/fntguide.pdf

`\fontseries` command in conjunction with the NFSS series codes defined during the installation of the font family. Please refer to the code tables in appendix A for a list of the most common NFSS codes. To select the semibold (`sb`) weight for example, we would use the following commands:

```
\fontseries{sb}\selectfont
```

Compact font switching macros such as `\mdseries` and `\bfseries` do not switch to a fixed NFSS font series, they use `\mddefault` and `\bfdefault` for the regular and bold weight respectively. If we want to use semibold as the default bold weight, for example, we simply redefine `\bfdefault` accordingly:

```
\renewcommand*{\bfdefault}{sb}
```

In order to use Sabon as the default roman typeface for the whole document, we redefine `\rmdefault` in the preamble:

```
\renewcommand*{\rmdefault}{psb}
```

It is much more convenient to put the initialization of the font family into a dedicated style file (`sty`), though. Our file `sabon.sty` might look like this:

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{sabon}[2002/04/17 v1.0 Adobe Sabon]
3 \RequirePackage[T1]{fontenc}
4 \RequirePackage{textcomp}
5 \renewcommand*{\rmdefault}{psb}
6 \endinput
```

Essentially, we redefine `\rmdefault` in order to use Sabon as the default roman typeface for the whole document. In addition to that, we load the `fontenc` package and switch to T1 encoding, which is more appropriate for PostScript fonts than the OT1 encoding used by default. We also load the `textcomp` package which provides a user interface for the symbols found in TS1 encoding. This will allow us to access symbols such as ‘copyright’ or ‘registered’ (appendix B provides a list of all symbols supported by `textcomp`). If the `textcomp` package is used in conjunction with `inputenc`, it is even possible to enter most of these symbols directly in a LaTeX file.

There is one thing we have to keep in mind when switching to T1 encoding. The default encoding is a global setting that applies to all text fonts used in a LaTeX file, unless the encoding is reset explicitly using the NFSS macro `\fontencoding`. It will affect the font family defined as `\rmdefault`, but also the families set up as `\sfdefault` and `\ttdefault`. By default, these are Computer Modern Sans Serif (`cmss`) and Computer Modern Typewriter (`cmtt`). Using these fonts in conjunction with T1 encoding will pose some problems most European TeX users are already well familiar with. It is perfectly possible, provided that we use a suitable version of the Computer Modern fonts. Choosing a suitable version, however, can be quite difficult. We will discuss some typical issues related to that in the following section.

Alternatively, we could use some other T1 encoded sans serif and typewriter typefaces available in PostScript format. For example, here is an enhanced version of `sabon.sty` using Helvetica (`phv`) and Courier (`pcr`):

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{sabon}[2002/04/17 v1.0 Adobe Sabon with PS fonts]
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\renewcommand*{\rmdefault}{psb}
\renewcommand*{\sfdefault}{phv}
\renewcommand*{\ttdefault}{pcr}
\endinput
```

This setup is certainly not the most fortunate one in terms of typography, but it should be safe from a technical perspective. Helvetica and Courier are part of the PostScript base fonts built into every Level 2 PostScript device. TeX distributions usually do not ship with the original versions of these fonts but they provide suitable replacements for them. Viewing DVI files on screen or creating PDF files should not pose any problem in this case.

For our setup of Sabon, the next section is only relevant if you want to use Computer Modern Sans Serif and Computer Modern Typewriter in conjunction with Sabon. If you deploy different T1 encoded sans serif and typewriter typefaces, which are available in PostScript format, all you need to do is redefine `\sffamily` and `\ttfamily` in `sabon.sty` or in the preamble of the respective LaTeX file as shown above for Helvetica and Courier. You might still want to read the next section in this case because it discusses one of the most frequently asked questions concerning fonts under TeX and LaTeX.

## 1.7 Computer Modern and T1 encoding

The Computer Modern fonts designed for T1 and TS1 encoding are called EC and TC fonts respectively, together known as European Computer Modern. When switching to T1 encoding, we implicitly switch to these fonts. Note that European Computer Modern, while being derived from Donald Knuth's original Computer Modern typefaces, is not simply a T1 encoded drop-in replacement. Over the years it has evolved into an independent typeface. The additional fonts created for the European Computer Modern family have been subject to debate based on their design. Some of them are considered to be typographically inferior to the original designs. From a technical perspective, the problem with the European Computer Modern fonts is that, historically, they have been available in Metafont format only. This implies that PostScript and PDF files will contain bitmap representations of these fonts when we switch to T1 encoding. Bitmap fonts, however, have a fixed resolution and so are not independent of the output device. They are not suitable for on-screen display and pose a major obstacle in the usually PostScript-based workflow of professional print shops, if they are tolerated at all.

Donald Knuth had designed the Computer Modern fonts in Metafont format and with OT1 encoding in mind. Blue Sky Research and Y&Y developed PostScript versions of these fonts later, which were donated to the public in 1997 and have been shipping with most TeX distributions ever since. While these fonts work fine for PostScript and PDF files, they are not suitable for tasks requiring letters not found in the English alphabet because their glyph base is still restricted to OT1 encoding. Jï¿œrg Knappen's European Computer Modern fonts address this issue by providing a more comprehensive set of glyphs, but they have in turn been subject to the limitations of Metafont. In the following, I will briefly introduce several solutions which try to address these problems. Most of them are trade-offs in one way or another. Tables 1.1 and 1.2 attempt to provide an overview of the major design variations over the Computer Modern theme along with their implementations. These tables are by no means exhaustive, there are even more fonts derived from the original Computer Modern typefaces.

To work around the hyphenation problem of OT1 encoding while sticking to the original Computer Modern fonts, there is a choice of two packages on CTAN which provide T1 encoded virtual fonts based on the original Computer Modern family of fonts: the AE[8] and the ZE[9] fonts. The AE fonts are built on top of Computer Modern

[8] http://www.ctan.org/tex-archive/fonts/ae/
[9] http://www.ctan.org/tex-archive/fonts/zefonts/

| TYPEFACE | FONTS | |
|---|---|---|
| | NAME | FORMAT |
| Computer Modern | CM | Metafont |
| | CM, Blue Sky | PostScript |
| | CM, Bakoma | PostScript, TrueType |
| | AE | virtual fonts |
| | ZE | virtual fonts |
| European Computer Modern | EC & TC | Metafont |
| | EC & TC, MicroPress | PostScript |
| | Tt2001 | PostScript |
| | CM-super | PostScript |
| Latin Modern | LM | PostScript |
| European Modern | EM | PostScript |

Table 1.1: Computer Modern fonts and formats

| FONTS | ENCODINGS | |
|---|---|---|
| | NATIVE | SUPPORTED |
| CM | OT1 | OT1 |
| CM, Blue Sky | OT1 | OT1 |
| AE | OT1 | T1, with composite glyphs |
| ZE | OT1 | T1, with composite glyphs |
| EC, TC | T1, TS1 | T1, TS1 |
| CM-super | Adobe Standard | T1, TS1, T2A, T2B, T2C, X2 |
| LM | font specific | T1, TS1, LY1, QX1 |

Table 1.2: Computer Modern fonts and encodings

exclusively, but unfortunately they lack almost a dozen T1 characters including double and single guillemets, which makes their default setup unsuitable for all French and a lot of German texts. For Computer Modern Typewriter, the situation is even worse. There is a supplemental package called `aecompl` which adds Metafont versions of the missing characters, but that again brings up the problem we were trying to avoid in the first place. A different complement called `aeguill`[10] at least adds PostScript versions of the guillemets. An enhanced version of `sabon.sty` might then look like this:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{sabon}[2002/04/17 v1.0 Adobe Sabon with AE]
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\RequirePackage{aeguill}
\renewcommand*{\rmdefault}{psb}
\endinput
```

The ZE fonts take a different approach to work around this problem: the missing characters are taken from standard

[10] http://www.ctan.org/tex-archive/macros/latex/contrib/supported/aeguill/

PostScript fonts such as Times and Helvetica. This means that there will be some typographical inconsistencies, but we are safe from a technical point of view. While the AE fonts and the corresponding supplemental packages ship with most TeX distributions, you might need to download the ZE fonts from CTAN. When using the ZE fonts, our enhanced version of `sabon.sty` would look like this:

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{sabon}[2002/04/17 v1.0 Adobe Sabon with ZE]
3 \RequirePackage[T1]{fontenc}
4 \RequirePackage{textcomp}
5 \RequirePackage{zefonts}
6 \renewcommand*{\rmdefault}{psb}
7 \endinput
```

There is a more robust solution you might be interested in if you require T1 encoded Computer Modern fonts. Free PostScript versions of the European Computer Modern fonts have been made available, although they might not have made their way into every TeX distribution yet. As mentioned before, one problem with OT1 encoded fonts is that they rely on composite glyphs which break searching for words containing accented letters in PDF files. Both the AE and the ZE fonts, although they enable TeX to hyphenate words containing accented letters properly, still suffer from this particular problem as they are based on OT1 encoded fonts internally. It is highly advisable to switch to a real T1 version of the Computer Modern fonts in PostScript format. Such fonts are included in two independent packages: Pï¿œter Szabï¿œ's Tt2001[11] as well as Vladimir Volovich's more recent CM-super[12] package. Both packages include PostScript fonts which are traced and post-processed conversions of their Metafont counterparts.

Unless you know that a specific font you need is provided by the Tt2001 package only, go with the more advanced CM-super package which will bring you as close to a real solution as you can possibly get when using the European Computer Modern fonts. Note, however, that it is a rather large download. Since it includes a huge number of fonts, the compressed package is about 64 MB in size. The CM-super fonts use Adobe Standard as their native encoding, but the glyph set provided by these fonts includes Cyrillic letters as well. In addition to T1

[11] http://www.ctan.org/tex-archive/fonts/ps-type1/ec/
[12] http://www.ctan.org/tex-archive/fonts/ps-type1/cm-super/

and TS1, CM-super supports the Cyrillic encodings T2A, T2B, T2C, and X2. See the package documentation for installation instructions and answers to the most frequently asked questions. Here is a version of our style file for use in conjunction with CM-super:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{sabon}[2002/04/17 v1.0 Adobe Sabon with CM-Super]
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\RequirePackage{type1ec}
\renewcommand*{\rmdefault}{psb}
\endinput
```

Recently, yet another new implementation of Computer Modern has been released to the public, the promising Latin Modern fonts created by Bogusław Jackowski and Janusz M. Nowacki. Unlike Tt2001 and CM-super, Latin Modern is derived from the original Computer Modern designs and has been augmented with accented letters as well as other glyphs missing from the very small glyph base of the original fonts. While the Latin Modern fonts are younger than European Computer Modern, they are a parallel development from a systematic perspective. Consequently, they are not affected by the controversial design decisions underlying certain parts of the European Computer Modern family of fonts. They use a font specific encoding by default and feature a glyph base suitable for T1, TS1, LY1 as well as the Polish encoding QX1. Even though these fonts are still under development they are already quite usable as of this writing (December 2004). Here is yet another iteration of our style file for Sabon, combined with Latin Modern for the sans serif and typewriter families:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{sabon}[2003/07/27 v1.0 Adobe Sabon with LM]
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\RequirePackage{lmodern}
\renewcommand*{\rmdefault}{psb}
\endinput
```

Apart from these free fonts, there are also commercial offerings from Y&Y[13] and MicroPress.[14] Judging by the vendors' websites, MicroPress offers PostScript versions of European Computer Modern while the European Modern fonts by Y&Y are augmented PostScript versions of the original Computer Modern typeface. Please refer to the respective website for details and pricing.

[13] http://www.yandy.com/em.htm
[14] http://www.micropress-inc.com/fonts/ecfonts/ecmain.htm

# Tutorial 2

# Standard font sets

The `\latinfamily` shorthand is very convenient, but it is not designed to cope with complex installation scenarios. Sooner or later you will probably have more specific requirements or simply desire more control over the basics. This will require using lower-level fontinst commands in most cases.

## 2.1 A verbose fontinst file

In this tutorial, we will essentially repeat the scenario discussed in the previous one. This time, however, we will employ lower-level commands. The verbose driver file introduced here will also serve as a template for subsequent tutorials.

```
1 \input fontinst.sty
2 \needsfontinstversion{1.926}
3 \substitutesilent{bx}{b}
```

After loading fontinst and verifying the version we set up an alias that will suppress a warning when the respective font is substituted. Why would we want to set up this particular alias? Note that `bx` is the NFSS code of the 'bold extended' series. The LaTeX macros `\textbf` and `\bfseries` do not switch to a fixed series, they use `\bfdefault` instead which defaults to `bx`. As long as we are using the Computer Modern fonts this is fine since they actually include bold extended fonts. For font families which do not, however, using these macros would result in a warning. To avoid that, we would need to redefine `\bfdefault` to a suitable weight. The problem here is that `\bfdefault` is a global setting applying to all of LaTeX's font families (`\rmdefault`, `\sfdefault`, and `\ttdefault`), but it is not safe to assume that all of them will offer the same weights. To avoid any need to redefine `\bfdefault` unless we really want to, we set up an alias so that every request for 'bold extended' (`bx`) is substituted by 'bold' (`b`). Unless bold extended fonts are available, simply think of `bx` as the default bold weight.[1]

The standard weight is selected by LaTeX in a similar way. The relevant macro is called `\mddefault` and defaults to `m`. Make sure that the NFSS series `m` is always defined, either mapped to actual fonts or as a substitution. In this case our font family provides regular-weight fonts so we will simply use them for the `m` series. Some font families, however, are based on the main weights 'light' and 'demibold' instead of 'regular' and 'bold'. In this case, we would either just map these weights to the `m` and `b` series directly or use the proper NFSS series codes (`l` and `db`) plus the following substitutions:

```
\substitutesilent{m}{l}
\substitutesilent{bx}{db}
```

Again, think of `m` as the default weight if regular-weight fonts are not available. Every font family should provide mappings for the NFSS series `m` and `bx` in the font definition file. If fonts matching these series exactly are not available, use substitutions to ensure that the defaults for `\mddefault` and `\bfdefault` will work without user intervention. Since `\mddefault` and `\bfdefault` are overall settings applying to all of LaTeX's families, redefining them explicitly may cause problems. Doing so should be an option, not a requirement.

```
4 \setint{smallcapsscale}{800}
```

[1] This substitution is a default setting that fontinst will always silently include. We could omit the respective line in this case, but if semibold fonts are available you might prefer using those as the default bold weight.

The basic Sabon set we are dealing with offers upright and italic fonts but no optical small caps. As a substitute, fontinst is capable of transparently generating so-called mechanical or 'faked' small caps – as opposed to optical or 'real' small caps which are actual glyphs found in a dedicated small caps font. Mechanical small caps are generated by taking the tall caps of the font and scaling them by a certain factor: 1000 means full size, 800 means 0.8. Since Type 1 fonts scale linearly, scaling down tall caps implies that they will appear lighter than the corresponding lowercase glyphs, thus disturbing the color of the page. However, if they are too tall they do not mix well with the lowercase alphabet.

Optical small caps usually match the x-height of the font. This is the height of the lowercase alphabet without ascenders and descenders. They blend in seamlessly with lowercase and mixed case text. Depending on the typeface, this usually corresponds to a value in the range of 650–750. If you scale down tall caps so that they match the x-height of the font, they will appear too light in running text. Finding a suitable value for this is obviously a trade-off. We are going to use fontinst's default setting of 800 here but you might want to experiment with a value in the range of 750–800. For serious applications of small caps we would need optical small caps, provided in a dedicated small caps or in an expert font. For details on small caps and expert sets, please refer to tutorial 3 and 5 respectively.

```
5 \setint{slant}{167}
```

The integer variable `smallcapsscale` is a known variable used by fontinst's encoding vectors. We could use it in conjunction with `\latinfamily` as well. The variable `slant` is specific to our fontinst file. We define it for convenience so that we can set the slant factor for all subsequent font transformations globally. The slant factor defines how much the glyphs slope to the right. It is a real number equivalent to the tangent of the slant angle. Fontinst represents this number as an integer though, so we have to multiply the tangent by 1000. The value 167 (∼9.5ï¿œ) is a reasonable default. Any value significantly greater that 176 (∼10ï¿œ) is usually too much.

```
6 \recordtransforms{psb-rec.tex}
```

We start off with some basic font transformations. All reencoding and transformation steps will be recorded in the file `psb-rec.tex` so that we can automatically build a matching map file later. The environment recording font transformations, which we open in line 6, will be closed at the very end of the fontinst file, in line 35.

```
7  \transformfont{psbr8r}{\reencodefont{8r}{\fromafm{psbr8a}}}
8  \transformfont{psbri8r}{\reencodefont{8r}{\fromafm{psbri8a}}}
9  \transformfont{psbb8r}{\reencodefont{8r}{\fromafm{psbb8a}}}
10 \transformfont{psbbi8r}{\reencodefont{8r}{\fromafm{psbbi8a}}}
```

The fonts are reencoded from Adobe Standard (Fontname code `8a`) to TeX Base 1 encoding (`8r`). Please refer to the fontinst manual for an explanation of the syntax of the individual commands used here and in the following.

```
11 \transformfont{psbro8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbr8a}}}
12 \transformfont{psbbo8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbb8a}}}
```

Like the `\latinfamily` shorthand, our fontinst file should create slanted fonts as well. These need to be reencoded and, well, slanted. We are using the `slant` variable defined in line 5 to set the slant factor. The raw, TeX Base 1 encoded fonts are now prepared for the generation of virtual fonts.

```
13 \installfonts
14 \installfamily{T1}{psb}{}
```

The installation of a font family is enclosed in an environment which we open in line 13 and close later in line 23. First of all, the font family we are about to install has to be declared: we have Adobe Sabon and we are going to install it in T1 encoding (Fontname code `8t`). The third argument to `\installfamily` corresponds to the second one of the `\latinfamily` command: it is used to include code in the font definition file that will be expanded by LaTeX whenever the font is selected. T1 will serve as our base encoding in LaTeX's text mode later. The `\latinfamily` command also provides OT1 (`7t`) and TeX Base 1 encoded fonts. We will omit both encodings here as we do not need them. While raw TeX Base 1 encoded fonts (`8r`) form the basis of all virtual fonts, they are usually not deployed as such on the TeX level, and the OT1 encoding is not suitable for PostScript fonts anyway. We will therefore deliberately ignore it and focus on T1 and TS1 exclusively.

```
15 \installfont{psbr8t}{psbr8r,newlatin}{t1}{T1}{psb}{m}{n}{}
```

To create the individual virtual fonts, we use fontinst's `\installfont` command. The first argument to `\installfont` is the virtual font we are going to create, the second one is a list of files used to build this font. These can be `afm`, `mtx`, or `pl` files, their suffix is omitted. If multiple fonts are provided, `\installfont` does not overwrite any

encoding slots when reading in additional files, it simply fills vacant slots if it finds suitable glyphs in the next font. The metric file `newlatin.mtx` is an auxiliary file provided by fontinst which should always be read when creating OT1 or T1 encoded text fonts. The third argument is the file name of an encoding vector without the file suffix, in this case `t1.etx`. The remaining arguments are written verbatim to the font definition file and declare the respective font in a format that the LaTeX font selection scheme (NFSS) can process: T1 encoding, Adobe Sabon[2], medium[3], normal (that is, upright or roman). The last argument is only relevant if fonts with different design sizes are available. It is empty for linearly scaled fonts.

```
16 \installfont{psbrc8t}{psbr8r,newlatin}{t1c}{T1}{psb}{m}{sc}{}
```

The small caps font is slightly different. Since we do not have any Type 1 font containing optical small caps we need to 'fake' them by scaling the uppercase alphabet and putting the scaled glyphs in the encoding slots of the lowercase alphabet. Fortunately, we do not have to deal with the actual low-level glyph scaling. We simply load `t1c.etx`, a special encoding vector which will take care of that, using the value of `smallcapsscale` as the scaling factor.

```
17 \installfont{psbro8t}{psbro8r,newlatin}{t1}{T1}{psb}{m}{sl}{}
18 \installfont{psbri8t}{psbri8r,newlatin}{t1}{T1}{psb}{m}{it}{}
```

Since the slanting was already performed on the raw fonts, the virtual slanted and the italic fonts are handled just like the upright ones. Now all regular fonts are done and we can repeat this part (15–18) for the bold fonts:

```
19 \installfont{psbb8t}{psbb8r,newlatin}{t1}{T1}{psb}{b}{n}{}
20 \installfont{psbbc8t}{psbb8r,newlatin}{t1c}{T1}{psb}{b}{sc}{}
21 \installfont{psbbo8t}{psbbo8r,newlatin}{t1}{T1}{psb}{b}{sl}{}
22 \installfont{psbbi8t}{psbbi8r,newlatin}{t1}{T1}{psb}{b}{it}{}
23 \endinstallfonts
```

After that, we start adding virtual fonts for TS1 encoding. The TS1 encoding complements T1 by providing additional glyphs such as currency signs and other frequently used symbols. This encoding will not be used as

[2] LaTeX does not really care about the name of the font or the foundry. This argument simply defines the code that identifies the font within the NFSS.

[3] In fact, the more appropriate name would be *regular* because *medium* is a moderate bold weight with the NFSS code `mb`.

the main output encoding of our LaTeX files later. It is accessed exclusively by way of macros provided by the `textcomp` package.

```
24 \installfonts
25 \installfamily{TS1}{psb}{}
```

Like `newlatin.mtx`, the file `textcomp.mtx` is an auxiliary metric file provided by fontinst. It should always be added when creating TS1 encoded fonts. The third argument in line 26, the encoding vector, refers to `ts1.etx` in this case:

```
26 \installfont{psbr8c}{psbr8r,textcomp}{ts1}{TS1}{psb}{m}{n}{}
```

Since the TS1 encoding contains symbols and figures only, the TS1 encoded regular and small caps fonts are identical. Hence we do not need a TS1 encoded virtual small caps font, but we still have to instruct LaTeX where to get the respective glyphs from. Otherwise the macros of the `textcomp` package might not work properly whenever the active NFSS shape is `sc`. To do so, we use the `\installfontas` macro and 'install' the virtual font built in line 26 once more, this time as the small caps shape. This will merely add a line to the font definition file without creating an additional virtual font:

```
27 \installfontas{psbr8c}{TS1}{psb}{m}{sc}{}
```

The slanted and italic fonts are handled like the upright one:

```
28 \installfont{psbro8c}{psbro8r,textcomp}{ts1}{TS1}{psb}{m}{sl}{}
29 \installfont{psbri8c}{psbri8r,textcomp}{ts1}{TS1}{psb}{m}{it}{}
```

We repeat 26–29 for the bold fonts:

```
30 \installfont{psbb8c}{psbb8r,textcomp}{ts1}{TS1}{psb}{b}{n}{}
31 \installfontas{psbb8c}{TS1}{psb}{b}{sc}{}
32 \installfont{psbbo8c}{psbbo8r,textcomp}{ts1}{TS1}{psb}{b}{sl}{}
33 \installfont{psbbi8c}{psbbi8r,textcomp}{ts1}{TS1}{psb}{b}{it}{}
```

Finally, we close all environments and terminate:

```
34 \endinstallfonts
```

```
\endrecordtransforms
\bye
```

## 2.2 The *latinfamily* macro revisited

Note that our fontinst file is not strictly equivalent to the `\latinfamily` macro but rather stripped down to the most useful parts with respect to typical PostScript fonts. Essentially, we did not create any font description files for the raw TeX Base 1 encoded fonts and we dropped OT1 encoding. If you are curious, you should be able to reconstruct all the steps taken by `\latinfamily` when looking at the log file created by fontinst while keeping our file in mind. Here are the relevant lines from the log file after running `\latinfamily` on the basic Sabon set. Only lines beginning with '`INFO> run`' are relevant in this context as they indicate lower-level macros used by `\latinfamily`:

```
INFO> run \transformfont  <psbr8r> from <psbr8a>
INFO> run \installrawfont <psbr8r><psbr8r,8r><8r><8r><psb><m><n>
INFO> run \installfont <psbr7t><psbr8r,newlatin><OT1><OT1><psb><m><n>
INFO> run \installfont <psbr8t><psbr8r,newlatin><T1><T1><psb><m><n>
INFO> run \installfont <psbr8c><psbr8r,textcomp><TS1><TS1><psb><m><n>
INFO> run \installfont <psbrc7t><psbr8r,newlatin><OT1c><OT1><psb><m><sc>
INFO> run \installfont <psbrc8t><psbr8r,newlatin><T1c><T1><psb><m><sc>
INFO> run \transformfont  <psbro8r> from <psbr8r> (faking oblique)
INFO> run \installrawfont <psbro8r><psbro8r,8r><8r><8r><psb><m><sl>
INFO> run \installfont <psbro7t><psbro8r,newlatin><OT1><OT1><psb><m><sl>
INFO> run \installfont <psbro8t><psbro8r,newlatin><T1><T1><psb><m><sl>
INFO> run \installfont <psbro8c><psbro8r,textcomp><TS1><TS1><psb><m><sl>
INFO> run \transformfont  <psbri8r> from <psbri8a>
INFO> run \installrawfont <psbri8r><psbri8r,8r><8r><8r><psb><m><it>
INFO> run \installfont <psbri7t><psbri8r,newlatin><OT1i><OT1><psb><m><it>
INFO> run \installfont <psbri8t><psbri8r,newlatin><T1i><T1><psb><m><it>
INFO> run \installfont <psbri8c><psbri8r,textcomp><TS1i><TS1><psb><m><it>
INFO> run \transformfont  <psbb8r> from <psbb8a>
```

```
INFO> run \installrawfont <psbb8r><psbb8r,8r><8r><8r><psb><b><n>
INFO> run \installfont <psbb7t><psbb8r,newlatin><OT1><OT1><psb><b><n>
INFO> run \installfont <psbb8t><psbb8r,newlatin><T1><T1><psb><b><n>
INFO> run \installfont <psbb8c><psbb8r,textcomp><TS1><TS1><psb><b><n>
INFO> run \installfont <psbbc7t><psbb8r,newlatin><OT1c><OT1><psb><b><sc>
INFO> run \installfont <psbbc8t><psbb8r,newlatin><T1c><T1><psb><b><sc>
INFO> run \transformfont  <psbbo8r> from <psbb8r> (faking oblique)
INFO> run \installrawfont <psbbo8r><psbbo8r,8r><8r><8r><psb><b><sl>
INFO> run \installfont <psbbo7t><psbbo8r,newlatin><OT1><OT1><psb><b><sl>
INFO> run \installfont <psbbo8t><psbbo8r,newlatin><T1><T1><psb><b><sl>
INFO> run \installfont <psbbo8c><psbbo8r,textcomp><TS1><TS1><psb><b><sl>
INFO> run \transformfont  <psbbi8r> from <psbbi8a>
INFO> run \installrawfont <psbbi8r><psbbi8r,8r><8r><8r><psb><b><it>
INFO> run \installfont <psbbi7t><psbbi8r,newlatin><OT1i><OT1><psb><b><it>
INFO> run \installfont <psbbi8t><psbbi8r,newlatin><T1i><T1><psb><b><it>
INFO> run \installfont <psbbi8c><psbbi8r,textcomp><TS1i><TS1><psb><b><it>
```

This listing is a complete summary of what the `\latinfamily` macro does in this case, broken down into lower-level commands. The order of the commands differs slightly from our file, because the `\transformfont` calls are not grouped at the beginning but rather used 'on demand' for each shape. This difference is irrelevant from a technical point of view. `\transformfont` must obviously be called before `\installfont` or `\installrawfont` tries to use the transformed fonts, but the exact location does not matter. Since we did not create any font description files for TeX Base 1 encoding, we did not use the `\installrawfont` macro in our fontinst file. This macro does not build a virtual font but rather sets up a raw, TeX Base 1 encoded font for use under LaTeX.

Here are some crucial points we would have to keep in mind when writing a fontinst file that does exactly what `\latinfamily` would do: the macro `\installrawfont` is used in conjunction with `8r.mtx` instead of `newlatin.mtx`, the encoding file is obviously `8r.etx` in this case. Creating OT1 encoded virtual fonts requires `newlatin.mtx` and `ot1.etx`. You will also notice that, in addition to `ot1c.etx` and `t1c.etx`, fontinst used encoding files like `ot1i.etx` and `t1i.etx` when creating italic virtual fonts. For T1 encoding, `t1.etx` and `t1i.etx` are equivalent (`t1i.etx` simply reads `t1.etx` internally), hence we did not bother using `t1i.etx` in our fontinst file. The situation is the same with `ts1.etx` and `ts1i.etx`. For OT1 encoding, however, the difference is crucial because this encoding

differs depending on the shape: the upright shape features a dollar symbol while the italic shape puts an italic pound symbol in the slot of the dollar. This is yet another idiosyncrasy of OT1.

# Tutorial 3

# Optical small caps and hanging figures

When choosing a new typeface, bear in mind that optical small caps and hanging figures are not available for all commercial PostScript fonts. If they are available for a certain typeface, they are usually provided separately, either in a SC & OSF or in an expert font package. We will deal with the former case in this tutorial, the latter will be discussed in tutorial 5. Suppose we have acquired the Sabon SC & OSF package to complement our base install of Sabon. This package provides four additional fonts: a regular SC & OSF, an italic OSF, a bold OSF, and a bold italic OSF font. These fonts will provide us with hanging figures for all shapes in both weights. Small caps are available for the regular weight only; we will still have to make do with mechanical small caps for the bold weight. Note that Adobe does not include a separate regular-weight upright OSF font. The respective figures are to be found in the small caps font instead. Our original file set looks like this:

```
sar_____.afm    sai_____.afm    sab_____.afm    sabi_____.afm
sar_____.inf    sai_____.inf    sab_____.inf    sabi_____.inf
sar_____.pfb    sai_____.pfb    sab_____.pfb    sabi_____.pfb
sar_____.pfm    sai_____.pfm    sab_____.pfm    sabi_____.pfm
```

```
sarsc___.afm    saiof___.afm    sabof___.afm    sabio___.afm
sarsc___.inf    saiof___.inf    sabof___.inf    sabio___.inf
sarsc___.pfb    saiof___.pfb    sabof___.pfb    sabio___.pfb
sarsc___.pfm    saiof___.pfm    sabof___.pfm    sabio___.pfm
```

After renaming and choosing the required files, we could start off with the following set of files:

```
psbr8a.afm      psbri8a.afm     psbb8a.afm      psbbi8a.afm
psbr8a.pfb      psbri8a.pfb     psbb8a.pfb      psbbi8a.pfb

psbrc8a.afm     psbrij8a.afm    psbbj8a.afm     psbbij8a.afm
psbrc8a.pfb     psbrij8a.pfb    psbbj8a.pfb     psbbij8a.pfb
```

But before we begin, let us take a closer look at the encoding of the fonts. We will have to deal with some peculiarities characteristic for typical SC & OSF sets. Taking a look at `psbr8a.afm`, you will see that in Adobe Standard encoding, which is the native encoding of all fonts of the Sabon family, the figures are encoded as 'zero', 'one', 'two' and so on:

```
C 48 ; WX 556 ; N zero ; B 52 -15 504 705 ;
C 49 ; WX 556 ; N one  ; B 91   0 449 705 ;
C 50 ; WX 556 ; N two  ; B 23   0 507 705 ;
```

Compare that to the glyph names of figures in an expert font:

```
C 48 ; WX 511 ; N zerooldstyle ; B 40 -14 480 436 ;
C 49 ; WX 328 ; N oneoldstyle  ; B 35  -3 294 425 ;
C 50 ; WX 440 ; N twooldstyle  ; B 44  -3 427 436 ;
```

The different glyph names are appropriate because regular PostScript fonts usually come with lining figures by default while expert fonts feature hanging ('old style') figures. Now let us take a look at `psbrc8a.afm`:

```
C 48 ; WX 556 ; N zero ; B  41 -15 515 457 ;
C 49 ; WX 556 ; N one  ; B 108   0 448 442 ;
C 50 ; WX 556 ; N two  ; B  72   0 512 457 ;
```

When comparing these glyph names to the actual outlines in `psbrc8a.pfb`,[1] we can see that this font in fact comes with hanging ('old style') figures even though the figures are labeled using the standard names. This is the case with all OSF fonts included in the SC & OSF package. The reason why this complicates the installation procedure will become clear when we take a look at the TeX side. In T1 encoding, for example, the figures are (essentially) encoded like this by default:

```
\setslot{zero}\endsetslot
\setslot{one}\endsetslot
\setslot{two}\endsetslot
```

While TS1 encoding (essentially) references them as follows:

```
\setslot{zerooldstyle}\endsetslot
\setslot{oneoldstyle}\endsetslot
\setslot{twooldstyle}\endsetslot
```

We face a similar problem with small caps. The lowercase letters in `psbr8a.afm` are labeled like this:

```
C 97 ; WX 500 ; N a ; B 42 -15 465 457 ;
C 98 ; WX 556 ; N b ; B 46 -15 514 764 ;
C 99 ; WX 444 ; N c ; B 25 -15 419 457 ;
```

Expert fonts, which provide small caps as well but do not need to follow Adobe Standard encoding, encode small caps as follows:

```
C 97 ; WX 457 ; N Asmall ; B -15  -3 467 446 ;
C 98 ; WX 481 ; N Bsmall ; B  34  -3 437 437 ;
C 99 ; WX 501 ; N Csmall ; B  38 -14 477 448 ;
```

Our font `psbrc8a` features small caps in place of lowercase letters but it has to follow Adobe Standard encoding:

```
C 97 ; WX 556 ; N a ; B 10   0 546 509 ;
C 98 ; WX 556 ; N b ; B 49   0 497 490 ;
C 99 ; WX 556 ; N c ; B 49 -12 512 502 ;
```

[1] The correct name of this font is `psbrcj8a`, but we will stick to the naming proposed in Fontname's `adobe.map` here.

This is one of the tricky parts when installing typical SC & OSF sets. Fontinst's encoding vectors expect distinct names for distinct glyphs while the metric files of SC & OSF fonts do not provide unique names for optical small caps and hanging figures. The other idiosyncrasy of SC & OSF sets is specific to a few font foundries including Adobe: there is no upright OSF font so we have to take the upright hanging figures from the small caps font when building virtual fonts. If you are installing a font family featuring an upright OSF font you obviously do not need to exchange any glyphs to get upright hanging figures.

## 3.1 The fontinst file

In the following, we will use the fontinst file introduced in the last tutorial as a template and add the features we need. We will create two LaTeX font families: `psb` and `psbj`. The former will provide lining figures while the latter will use the hanging figures of the OSF fonts instead. Both families will incorporate optical small caps where available. In the following, all comments concerning the fontinst file will be restricted to those aspects diverging from our template. Please refer to the previous tutorial for a commentary on the original template.

```
\input fontinst.sty
\needsfontinstversion{1.926}
\substitutesilent{bx}{b}
\setint{smallcapsscale}{800}
\setint{slant}{167}
\recordtransforms{psb-rec.tex}
\transformfont{psbr8r}{\reencodefont{8r}{\fromafm{psbr8a}}}
\transformfont{psbri8r}{\reencodefont{8r}{\fromafm{psbri8a}}}
\transformfont{psbb8r}{\reencodefont{8r}{\fromafm{psbb8a}}}
\transformfont{psbbi8r}{\reencodefont{8r}{\fromafm{psbbi8a}}}
```

The first couple of lines of our template remain unchanged.

```
\transformfont{psbrc8r}{\reencodefont{8r}{\fromafm{psbrc8a}}}
\transformfont{psbrij8r}{\reencodefont{8r}{\fromafm{psbrij8a}}}
\transformfont{psbbj8r}{\reencodefont{8r}{\fromafm{psbbj8a}}}
\transformfont{psbbij8r}{\reencodefont{8r}{\fromafm{psbbij8a}}}
```

After the reencodings inherited form our template (7–10), we insert the new fonts which need to be reencoded as well (11–14).

```
15 \transformfont{psbro8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbr8a}}}
16 \transformfont{psbbo8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbb8a}}}
17 \transformfont{psbrco8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbrc8a}}}
18 \transformfont{psbboj8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbbj8a}}}
```

In addition to that, we need slanted versions of the new fonts. Slanting the small caps font (17) may seem a bit unusual given that we do not want to create a slanted small caps shape. But since the regular-weight hanging figures are found in the small caps font, we need a slanted version of that as well to provide matching figures for the slanted shape of the `psbj` family later.

```
19 \installfonts
20 \installfamily{T1}{psb}{}
21 \installfont{psbr8t}{psbr8r,newlatin}{t1}{T1}{psb}{m}{n}{}
```

Building the virtual font for the upright shape (21) is straightforward and in line with our driver template. We depart from our template as we approach the tricky part mentioned above, the font `psbrc8r` which contains both small caps and hanging figures. Before we get to the problem of hanging versus lining figures, let us consider a basic case first. To install a text font featuring optical small caps, such as `psbrc8r`, we would use the following line:

```
\installfont{psbrc8t}{psbrc8r,newlatin}{t1}{T1}{psb}{m}{sc}{}
```

We need the encoding file `t1.etx` in this case since the small caps are labeled like ordinary lowercase glyphs in `psbrc8a`. The encoding file `t1c.etx` would be inappropriate because it expects the small caps to be labeled as 'Asmall', 'Bsmall', and 'Csmall'. If it does not find any matching glyphs, it 'fakes' them by scaling down the tall caps of the respective font – which is why we used `t1c.etx` in tutorial 2. The problem is that `psbrc8r` also contains hanging figures whereas we want `psb` to be a consistent family featuring lining figures throughout. Fortunately, we can leverage virtual fonts to combine the glyphs of several raw fonts. In this case, we take the lining figures from `psbr8r`:

```
22 \installfont{psbrc8t}{psbrc8r,psbr8r suffix lining,newlatin}{lining,t1}{T1}{psb}{m}{sc}{}
```

We read `psbrc8r` first and `psbr8r` after that. Note that `\installfont` does not overwrite any encoding slots when processing additional metric files, it simply fills vacant slots if it finds suitable glyphs in the next font. If the glyphs had unique names, we could simply take the (lining) figures from `psbr8r` while the rest of the glyphs including the small caps would be provided by `psbrc8r`. But the figures in `psbrc8r` are labeled just like those in `psbr8r`. How is fontinst supposed to distinguish between the two sets? This is where fontinst's `suffix` option comes into play. This option will add the suffix 'lining' to the names of all glyphs in `psbr8r`. Our encoding vector `t1.etx`, however, does not contain any glyph names ending in 'lining'. The figures are encoded as 'zero', 'one', 'two'. Hence we need to create an additional file to get the figures right – the encoding file `lining.etx`:

```
\relax
\encoding
\setcommand\digit#1{#1lining}
\endencoding
\endinput
```

So how does all of this fit together? To understand this approach, we need to take another look at how `t1.etx` defines the encoding slots for all figures:

```
\setslot{\digit{one}}\endsetslot
\setslot{\digit{two}}\endsetslot
\setslot{\digit{three}}\endsetslot
```

The glyph names are not given verbatim in `t1.etx`, they are passed to the `\digit` macro as an argument. The default definition of this macro as given in `t1.etx` looks like this:

```
\setcommand\digit#1{#1}
```

This means that the glyph labeled 'one' in the `afm` file will end up in the encoding slot for the numeral one in the virtual font. Our encoding file `lining.etx`, which is read before `t1.etx`, predefines the `\digit` macro as follows:

```
\setcommand\digit#1{#1lining}
```

Since fontinst's `\setcommand` macro will only define a command if it has not been defined yet, this is the definition which will be used when `t1.etx` is processed. With all of that in mind, let us go over line 22 again:

```
\installfont{psbrc8t}{psbrc8r,psbr8r suffix lining,newlatin}{lining,t1}{T1}{psb}{m}{sc}{}
```

Because of `lining.etx`, fontinst expects all figures to be labeled in the form 'zerolining', 'onelining', 'twolining' and so on. All the other glyph names defined by the encoding file `t1.etx` remain unchanged. When processing the metric files, fontinst will consider the glyphs found in `psbrc8r` first, but it will skip the figures. When processing `psbr8r` after that, it will skip almost all of the glyphs it finds in this font since appending the string `lining` to their names has effectively rendered them invalid. The only exception are the figures which now match the format defined in `lining.etx`. Hence the virtual font `psbrc8t` will be based on `psbrc8r` coupled with the figures of `psbr8r`.

```
23 \installfont{psbri8t}{psbri8r,newlatin}{t1}{T1}{psb}{m}{it}{}
24 \installfont{psbro8t}{psbro8r,newlatin}{t1}{T1}{psb}{m}{sl}{}
```

The installation of the remaining fonts does not differ from our template. We continue with the bold fonts:

```
25 \installfont{psbb8t}{psbb8r,newlatin}{t1}{T1}{psb}{b}{n}{}
26 \installfont{psbbc8t}{psbb8r,newlatin}{t1c}{T1}{psb}{b}{sc}{}
```

Optical small caps are available for the regular weight only. For the bold series we have to make do with 'faked' small caps, hence we use the encoding file `t1c.etx` in line 26. The remaining lines for T1 encoding do not require any adjustments either:

```
27 \installfont{psbbi8t}{psbbi8r,newlatin}{t1}{T1}{psb}{b}{it}{}
28 \installfont{psbbo8t}{psbbo8r,newlatin}{t1}{T1}{psb}{b}{sl}{}
29 \endinstallfonts
```

That's it for T1 encoding. We continue with TS1:

```
30 \installfonts
31 \installfamily{TS1}{psb}{}
```

While TS1 is primarily intended for symbols complementing T1, it includes hanging figures as well. As the only way to use them is loading the `textcomp` package and typing cumbersome text commands like `\textzerooldstyle` (see appendix B.2), it is not very useful to have them in TS1. Our `psbj` family will make them the default figures anyway so that they are readily available. But we are being picky. Let us see how we can put hanging figures in `TS1/psb` as well. As mentioned above, the problem is that the OSF fonts use regular glyph names for the hanging

figures while fontinst's TS1 encoding vector references them by 'oldstyle' names. Hence we have to turn regular figures – which are in fact hanging figures not encoded as such – into hanging figures. For the upright fonts, the hanging figures are in fact in the small caps font which complicates the installation even more. But we have dealt with this problem before and the approach should look familiar:

```
\installfont{psbr8c}{psbr8r,psbrc8r suffix oldstyle,textcomp}{ts1}{TS1}{psb}{m}{n}{}
```

This time, we do not need an additional encoding file because `ts1.etx` uses 'oldstyle' names by default. All we need to do in order to ensure unique glyph names is adding the string `oldstyle` to the glyphs in `psbrc8r` when building the virtual font.

```
\installfontas{psbr8c}{TS1}{psb}{m}{sc}{}
\installfont{psbro8c}{psbro8r,psbrco8r suffix oldstyle,textcomp}{ts1}{TS1}{psb}{m}{sl}{}
```

The slanted shape is handled in a similar way because it relies on the figures in the small caps font as well. For the remaining virtual fonts the installation is simpler. Since we have raw OSF fonts with hanging figures, all we need to do is rename these figures for TS1 encoding:

```
\installfont{psbri8c}{psbri8r,psbrij8r suffix oldstyle,textcomp}{ts1}{TS1}{psb}{m}{it}{}
\installfont{psbb8c}{psbb8r,psbbj8r suffix oldstyle,textcomp}{ts1}{TS1}{psb}{b}{n}{}
\installfontas{psbb8c}{TS1}{psb}{b}{sc}{}
\installfont{psbbo8c}{psbbo8r,psbboj8r suffix oldstyle,textcomp}{ts1}{TS1}{psb}{b}{sl}{}
\installfont{psbbi8c}{psbbi8r,psbbij8r suffix oldstyle,textcomp}{ts1}{TS1}{psb}{b}{it}{}
\endinstallfonts
```

This is the first half of our fontinst file. Compared to the template introduced in the previous tutorial it adds optical small caps to T1 and hanging figures to TS1 encoding. We will now create an additional font family (`psbj`) featuring hanging figures by default.

```
\installfonts
\installfamily{T1}{psbj}{}
\installfont{psbrj8t}{psbr8r,psbrc8r suffix oldstyle,newlatin}{t1j}{T1}{psbj}{m}{n}{}
```

In order to incorporate hanging figures, we need to exchange the figure sets of some fonts again. There is no need to create an additional encoding file this time. Fontinst ships with an encoding vector called `t1j.etx` which uses 'oldstyle' names by default. We use `t1j.etx` and add an 'oldstyle' suffix to the glyph names of the small caps font to combine `psbr8r` with the figures in `psbrc8r`. If you are installing a font package which includes an upright OSF font, simply use that and build the virtual font as shown for the italic OSF font in line 46.

```
44 \installfont{psbrcj8t}{psbrc8r,newlatin}{t1}{T1}{psbj}{m}{sc}{}
```

The small caps font does not require any modifications this time. The raw font already contains hanging figures so we can use it as-is. Since `psbrc8r` uses standard glyph names for small caps and hanging figures, we use the regular encoding vector `t1.etx`.

```
45 \installfont{psbroj8t}{psbro8r,psbrco8r suffix oldstyle,newlatin}{t1j}{T1}{psbj}{m}{sl}{}
```

The slanted shape is handled like the upright one: we combine `psbro8r` with the slanted hanging figures provided by `psbrco8r`.

```
46 \installfont{psbrij8t}{psbrij8r,newlatin}{t1}{T1}{psbj}{m}{it}{}
```

Building the italic virtual font is trivial because we have an italic OSF font with easily accessible hanging figures in the standard slots (note the regular encoding vector `t1.etx`). Since there are OSF fonts for all bold shapes as well, they do not require any special modifications either. We simply use the appropriate OSF fonts instead of the fonts from the basic Sabon package:

```
47 \installfont{psbbj8t}{psbbj8r,newlatin}{t1}{T1}{psbj}{b}{n}{}
48 \installfont{psbbcj8t}{psbbj8r,newlatin}{t1c}{T1}{psbj}{b}{sc}{}
```

We create 'faked' bold small caps using the encoding file `t1c.etx` because there is no bold small caps font.

```
49 \installfont{psbboj8t}{psbboj8r,newlatin}{t1}{T1}{psbj}{b}{sl}{}
50 \installfont{psbbij8t}{psbbij8r,newlatin}{t1}{T1}{psbj}{b}{it}{}
51 \endinstallfonts
```

Since TS1 is not a regular text encoding, we do not need to create TS1 encoded fonts for the `psbj` family. Any `TS1/psbj` fonts would be identical to `TS1/psb` anyway. To ensure that the `textcomp` package works with the `psbj`

family as well, however, we still have to create a suitable font definition file for `TS1/psbj`:

```
\installfonts
\installfamily{TS1}{psbj}{}
```

We use `\installfontas` to 'install' the TS1 encoded virtual fonts of the `psb` family as `TS1/psbj`. As mentioned before, this will merely add some lines to the font definition file without creating any additional virtual fonts:

```
\installfontas{psbr8c}{TS1}{psbj}{m}{n}{}
\installfontas{psbr8c}{TS1}{psbj}{m}{sc}{}
\installfontas{psbro8c}{TS1}{psbj}{m}{sl}{}
\installfontas{psbri8c}{TS1}{psbj}{m}{it}{}
\installfontas{psbb8c}{TS1}{psbj}{b}{n}{}
\installfontas{psbb8c}{TS1}{psbj}{b}{sc}{}
\installfontas{psbbo8c}{TS1}{psbj}{b}{sl}{}
\installfontas{psbbi8c}{TS1}{psbj}{b}{it}{}
\endinstallfonts
\endrecordtransforms
\bye
```

Now we have a fully functional setup of `psb` and `psbj` in T1 and TS1 encoding.

## 3.2 An extended style file

With two Sabon families at hand, we might want to update `sabon.sty` to make them easily accessible. We add the two options `lining` and `oldstyle` for the respective font families (6, 7) and make hanging figures the default (8). Loading the package with the option `oldstyle` or without any option will set up `psbj` as the default roman family while using the `lining` option will make it select `psb` instead:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{sabon}[2002/05/12 v1.1 Adobe Sabon]
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
```

```
\RequirePackage{nfssext}
\DeclareOption{lining}{\renewcommand*{\rmdefault}{psb}}
\DeclareOption{oldstyle}{\renewcommand*{\rmdefault}{psbj}}
\ExecuteOptions{oldstyle}
\ProcessOptions*
\endinput
```

It might also be handy to have dedicated text commands to switch between the two figure sets. Since such commands will need to work with all font families anyway, let us put them in a stand-alone style file and load it in `sabon.sty` (5). The style file `nfssext.sty` might look like this:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{nfssext}[2003/03/14 v1.2 Experimental NFSS Extensions]
\newcommand*{\exfs@tempa}{}
\newcommand*{\exfs@tempb}{}
\newcommand*{\exfs@try@family}[1]{%
  \let\exfs@tempa\relax
  \begingroup
    \fontfamily{#1}\try@load@fontshape
    \expandafter\ifx\csname\curr@fontshape\endcsname\relax
      \PackageWarning{nfssext}{%
        Font family '\f@encoding/#1' not available\MessageBreak
        Ignoring font switch}%
    \else
      \gdef\exfs@tempa{\fontfamily{#1}\selectfont}%
    \fi
  \endgroup
  \exfs@tempa}
```

This is an outline for a command that makes use of a few NFSS internals to switch to a specific family if and only if it is available. Essentially, we try to load the requested family in the current encoding (8). If this succeeds, we set up a macro (14) to be expanded later that will actually switch font families; if not, we print a warning message (10–12) and do nothing.

```
\def\exfs@get@base#1#2#3#4\@nil{#1#2#3}%
\DeclareRobustCommand{\lnstyle}{%
  \not@math@alphabet\lnstyle\relax
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil}}
\DeclareRobustCommand{\osstyle}{%
  \not@math@alphabet\osstyle\relax
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil j}}
```

The macros `\lnstyle` and `\osstyle` switch to lining and hanging ('old style') figures respectively. They are employed like `\bfseries` or `\itshape`. Internally, they will take the first three letters of the current NFSS font family name (18), append a letter to it where appropriate (none for lining figures, `j` for hanging figures), and call `\exfs@try@family`. Even though this mechanism is rather simple-minded, it should work just fine for all fonts set up according to the Fontname scheme.

```
\DeclareTextFontCommand{\textln}{\lnstyle}
\DeclareTextFontCommand{\textos}{\osstyle}
\endinput
```

The corresponding text commands, `\textln` and `\textos`, take one mandatory argument and can be employed like `\textbf` or `\textit`.

## 3.3 The fonts supplied with TeX

The standard PostScript fonts supplied with the most common TeX distributions do not include optical small caps, nor do they include hanging figures. The default typeface of both plain TeX and LaTeX however, Computer Modern Roman, does include such glyphs. Unfortunately, the design of the small caps is flawed. Their height corresponds to what you usually end up with when creating mechanical small caps. Being too tall, these small caps hardly blend in with lowercase text at all, even though their color matches that of the lowercase alphabet.

Hanging figures are included in Computer Modern as well, but they are hidden in some of the math fonts. The only way to use them with the default setup is rather cumbersome: the macro `\oldstylenums` will take the numbers to be typeset as hanging figures as an argument. There is a set of virtual fonts for the European Computer

Modern fonts which make these hanging figures the default in TeX's text mode so that they are readily available. These fonts are provided in the ECO package available from CTAN.[2] Please refer to the package documentation for installation and usage instructions. Since this package essentially consists of a set of virtual fonts, it should also work in conjunction with the CM-super fonts mentioned in section 1.7. Note that all of this applies to TeX's text mode only. In math mode, TeX will use an independant set of fonts.

[2] http://www.ctan.org/tex-archive/fonts/eco/

# Tutorial 4

# The euro currency symbol

While the euro symbol has been supported by LaTeX for quite some time – it is included in TS1 encoding and the `textcomp` package provides the corresponding text command `\texteuro` – the real problem is getting fonts that provide this glyph and setting them up accordingly. You might want to read this tutorial even if you are not affected by this particular issue, because it deals with some generic encoding problems that you may encounter in a different context as well. There is a bit more to updating a font than drawing a euro symbol and putting it in the font. It has to be properly encoded as well. Since the euro symbol is not defined in Adobe Standard encoding, it can normally only be included as an uncoded glyph in regular PostScript text fonts. An uncoded glyph is only accessible after reencoding and assigning it to a valid encoding slot. Some font foundries decided to follow this path in order to conform to Adobe Standard encoding. Others prefer to drop some supposedly rarely used glyph and put the euro symbol in its encoding slot instead. While this violates the encoding standard, it can be more convenient under certain circumstances. In the following, we will explore ways to handle both situations cleanly. Finally, we will learn how to take the euro symbol from an external font if none is provided by the text font itself.

## 4.1 Uncoded euro symbol

While Adobe used to be rather inattentive to the problem at first, the foundry is finally updating their typeface portfolio by gradually adding matching euro symbols to their fonts – a process that has been promoted by the introduction of the OpenType font format. Recent releases of Adobe Garamond, for example, already ship with matching euro symbols. A quick look at the `afm` file shows that in this case, the foundry decided to handle the encoding problem in a strict manner. The new symbol is correctly labeled as 'Euro' but it is not encoded by default as that would violate Adobe Standard encoding. An encoding slot number of -1 means that the glyph is uncoded:

```
C -1 ; WX 572 ; N Euro ; B -13 -14 542 640 ;
```

In order to access it, we need to reencode the font and assign the glyph 'Euro' to a valid encoding slot. The standard procedure we have been pursuing in this guide involves reencoding all fonts to TeX Base 1 encoding anyway precisely because of cases like this one. By reencoding all fonts to TeX Base 1 encoding we ensure that all glyphs our virtual fonts rely on are properly encoded in the raw fonts we use as their basis. As of fontinst 1.9, the encoding file `8r.etx` includes the euro symbol. Note that you also need a matching version of `8r.enc` for dvips and pdfTeX. This file is distributed separately and not included in the fontinst release.[1] Since reencoding all text fonts to TeX Base 1 encoding is part of our regular installation routine anyway, the fontinst file does not need any adjustments:

```
\transformfont{padr8r}{\reencodefont{8r}{\fromafm{padr8a}}}
```

The reencoding step will ensure that the euro symbol is available in all TeX Base 1 encoded raw fonts. We can use them to build TS1 encoded virtual fonts as usual:

```
\installfont{padr8c}{padr8r,textcomp}{ts1}{TS1}{pad}{m}{n}{}
```

After installing the fonts and all auxiliary files, the euro symbol will be available as `\texteuro` when loading the `textcomp` package.

[1] http://www.ctan.org/tex-archive/info/fontname/8r.enc

## 4.2 Euro symbol encoded as currency symbol

Bitstream was one of the first type foundries to update their font collection and add a matching euro symbol to all fonts. When looking at the fonts, the first thing we notice is that the foundry decided to encode the euro symbol as the generic currency symbol '¤'. The reason for this is that you can access the symbol without reencoding the font, which can be very difficult on some systems. Since the generic currency symbol is hardly ever used anyway, it is no surprise that this particular glyph was dropped. We could install Bitstream fonts as usual and use `\textcurrency` instead of `\texteuro` to access the euro symbol, but that would imply keeping the idiosyncrasies of a given font in mind while writing, and modifying the text when changing the typeface – not quite what one would expect when working with a high-level markup language like LaTeX.

When taking a closer look at the `pfb` and `afm` files, we can see that the fonts in fact contain two euro symbols. The first one is found in encoding slot 168, that is, it is encoded as the currency symbol and labeled accordingly. To verify that, we have to take a look at the `pfb` files in a font editor. Since the euro symbol is both encoded and labeled just like a currency symbol, there is no way to tell the difference by looking at the `afm` file only:

```
C 168 ; WX 556 ; N currency ; B 6 -12 513 697 ;
```

The other one is uncoded (slot -1) and labeled as 'Euro':

```
C -1 ; WX 556 ; N Euro ; B 6 -12 513 697 ;
```

If we want a readily available euro symbol (and one that is available *as such*), we have two options in this case. Either we reencode the font and assign the uncoded euro symbol to a valid encoding slot or we use the already encoded euro symbol found in the slot of the currency symbol and move it to the proper encoding slot. The former case was already discussed above, let us now investigate the latter.

The easiest way to move the glyph to a different slot is resetting it when creating the TS1 encoded virtual fonts. This requires that the glyph is already encoded (in any slot) in the raw fonts serving as their basis. Since the TeX Base 1 encoding we use for all base fonts includes both the euro and the generic currency symbol this should not pose any problems. The low-level commands which reset the glyph go in a dedicated metric file, `reseteur.mtx`, which we have to create ourselves:

```
1 \relax
2 \metrics
3 \resetglyph{euro}
4   \glyph{currency}{1000}
5 \endsetglyph
6 \setleftrightkerning{euro}{currency}{1000}
7 \unsetglyph{currency}
8 \endmetrics
```

We reset the glyph ‘euro’ based on the glyph ‘currency’ scaled to its full size in line 3–5, adjust the kerning on either side of ‘euro’ to match that of ‘currency’ and finally unset the glyph ‘currency’ in line 7 because there is no such thing as a generic currency symbol in this font. In the fontinst file, we include the metric file `reseteur.mtx` in the file list of the respective `\installfont` command right after the metrics for this font have been read. This might look as follows:

```
\installfont{bsbr8c}{bsbr8r,reseteur,textcomp}{ts1}{TS1}{bsb}{m}{n}{}
```

We only need to do this for the TS1 encoded virtual fonts as T1 does not include the euro symbol. Apart from that, the fontinst file does not need any adjustments.

## 4.3 Euro symbol taken from symbol font

Let us go back to our install of Sabon to see if we can get euro support for Sabon as well. The font itself does not include any euro symbol at all. If we do not provide a euro symbol, fontinst will automatically try to fake it by overstriking the capital letter ‘C’ with two horizontal bars as a last resort. This procedure might yield acceptable results in some cases, but the quality varies significantly from typeface to typeface. The result can be anything from reasonable to completely unsuable. Still, it is by all means worth a try if no matching euro symbol is available.

We could also try to take the euro symbol from an external symbol font. While some font foundries at least provide special symbol fonts containing a collection of matching euro glyphs for all typefaces that have not been updated yet, Adobe merely offers a set of generic euro fonts containing glyphs that do not really match any typeface at all. From a typographical perspective, this is a desperate workaround. However, lacking a matching euro symbol,

we do not have a choice. The Adobe Euro fonts[2] come in three flavors: serif (Euro Serif), sans serif (Euro Sans), and condensed sans serif (Euro Mono, intended for use with monospaced fonts). Each family consists of regular, regular italic, bold, and bold italic fonts.

Instead of using a serif euro that does not match our typeface we will use the sans serif design which has a more generic look that adheres to the reference design of the European Commission. Granted, this one does not match our typeface either – but at least it does not pretend to do so. Now that we are aware of the most common encoding pitfalls, we inspect the `afm` files first before proceeding with the installation. The Euro fonts put the euro symbol in all encoding slots. When looking at the `afm` file, we can see that the fonts use a font specific encoding and that the glyphs are labeled as 'Euro' with a consecutive number appended to the name:

```
C 33 ; WX 750 ; N Euro.001 ; B 10 -12 709 685 ;
C 34 ; WX 750 ; N Euro.002 ; B 10 -12 709 685 ;
C 35 ; WX 750 ; N Euro.003 ; B 10 -12 709 685 ;
C 36 ; WX 750 ; N Euro.004 ; B 10 -12 709 685 ;
C 37 ; WX 750 ; N Euro.005 ; B 10 -12 709 685 ;
```

On further inspection, we find two additional glyphs. There is a glyph labeled 'Euro' in slot 128 as well as an uncoded glyph labeled 'uni20AC':

```
C 128 ; WX 750 ; N Euro    ; B 10 -12 709 685 ;
C  -1 ; WX 750 ; N uni20AC ; B 10 -12 709 685 ;
```

The number 20AC is 8364 in hexadecimal and slot 8364 is the encoding slot of the euro symbol in Unicode encoding, hence the string 'uni20AC'. Obviously someone was trying to make sure that every application out there would be able to access that euro symbol. Fortunately, this covers our situation as well. We need a glyph that is encoded and labeled as 'Euro'. The encoding slot number does not matter since we will include it in a virtual font using a different encoding anyway. The one in slot 128 fits our needs perfectly. In practice, this means that we can simply add the file name to the input file list of an `\installfont` command when creating TS1 encoded virtual fonts with fontinst:

```
\installfont{psbr8c}{psbr8r,zpeurs,textcomp}{ts1}{TS1}{psb}{m}{n}{}
```

[2] http://www.adobe.com/type/eurofont.html

Since the Adobe Euro fonts are non-standard, their naming is non-standard as well. We will discuss that in more detail below. Before running this file, we need to copy the properly named `afm` files of the Adobe Euro fonts to the working directory so that fontinst will find them. For the euro glyph to be available later, the Euro fonts need to be installed in the usual way so that TeX as well as pdfTeX, dvips, and xdvi are able to use them.

## 4.4 Installing symbol fonts

From a technical perspective, symbol fonts differ from text fonts in that they are not based on any standardized encoding. They use a font specific encoding instead. Essentially, this means that the order of the glyphs in the font is arbitrary. When installing symbol fonts, we will usually not reencode them. This implies that we have to provide some kind of user interface tailored for the font if we want to access the glyphs directly. We will discuss that in detail below. Adobe's Euro font package[3] provides us with the following set of files:

```
_1______.afm    _1i_____.afm    _1b_____.afm    _1bi____.afm
_1______.inf    _1i_____.inf    _1b_____.inf    _1bi____.inf
_1______.pfb    _1i_____.pfb    _1b_____.pfb    _1bi____.pfb
_1______.pfm    _1i_____.pfm    _1b_____.pfm    _1bi____.pfm
_2______.afm    _2i_____.afm    _2b_____.afm    _2bi____.afm
_2______.inf    _2i_____.inf    _2b_____.inf    _2bi____.inf
_2______.pfb    _2i_____.pfb    _2b_____.pfb    _2bi____.pfb
_2______.pfm    _2i_____.pfm    _2b_____.pfm    _2bi____.pfm
_3______.afm    _3i_____.afm    _3b_____.afm    _3bi____.afm
_3______.inf    _3i_____.inf    _3b_____.inf    _3bi____.inf
_3______.pfb    _3i_____.pfb    _3b_____.pfb    _3bi____.pfb
_3______.pfm    _3i_____.pfm    _3b_____.pfm    _3bi____.pfm
```

The Fontname map file `adobe.map` defines the following names for these fonts:

```
zpeur     EuroSerif-Regular                A    916    _3______
zpeub     EuroSerif-Bold                   A    916    _3b_____
```

[3] http://www.adobe.com/type/eurofont.html

```
zpeubi    EuroSerif-BoldItalic            A    916    _3bi____
zpeuri    EuroSerif-Italic                A    916    _3i_____
zpeurs    EuroSans-Regular                A    916    _1______
zpeubs    EuroSans-Bold                   A    916    _1b_____
zpeubis   EuroSans-BoldItalic             A    916    _1bi____
zpeuris   EuroSans-Italic                 A    916    _1i_____
zpeurt    EuroMono-Regular                A    916    _2______
zpeubt    EuroMono-Bold                   A    916    _2b_____
zpeubit   EuroMono-BoldItalic             A    916    _2bi____
zpeurit   EuroMono-Italic                 A    916    _2i_____
```

We select all `afm` and all `pfb` files, rename them, and start off with the following file set:

```
zpeur.afm       zpeuri.afm       zpeub.afm       zpeubi.afm
zpeur.pfb       zpeuri.pfb       zpeub.pfb       zpeubi.pfb
zpeurs.afm      zpeuris.afm      zpeubs.afm      zpeubis.afm
zpeurs.pfb      zpeuris.pfb      zpeubs.pfb      zpeubis.pfb
zpeurt.afm      zpeurit.afm      zpeubt.afm      zpeubit.afm
zpeurt.pfb      zpeurit.pfb      zpeubt.pfb      zpeubit.pfb
```

The installation of symbol fonts does not really require fontinst as far as creating the metrics is concerned because we do not need to reencode the fonts or create virtual fonts based on them. Simply running `afm2tfm` on each `afm` file to create the corresponding `tfm` file for TeX would do the job:

```
afm2tfm zpeur.afm  zpeur.tfm
afm2tfm zpeuri.afm zpeuri.tfm
afm2tfm zpeub.afm  zpeub.tfm
afm2tfm zpeubi.afm zpeubi.tfm
```

`afm2tfm` is able to create slanted fonts as well:

```
afm2tfm zpeur.afm -s 0.167 zpeuro.tfm
afm2tfm zpeub.afm -s 0.167 zpeubo.tfm
```

The downside of using `afm2tfm` is that we have to create font definition files and map files manually. Font definition files are not required if the fonts are only referenced by other virtual fonts, but they will allow us the access the fonts directly in any LaTeX file. We will use fontinst to take advantage of its map file writer and the fact that it generates font definition files automatically.

```
\input fontinst.sty
\needsfontinstversion{1.926}
\setint{slant}{167}
\recordtransforms{peu-rec.tex}
```

Our fontinst file starts with a familiar header.

```
\transformfont{zpeuro}{\slantfont{\int{slant}}{\fromafm{zpeur}}}
\transformfont{zpeubo}{\slantfont{\int{slant}}{\fromafm{zpeub}}}
\transformfont{zpeuros}{\slantfont{\int{slant}}{\fromafm{zpeurs}}}
\transformfont{zpeubos}{\slantfont{\int{slant}}{\fromafm{zpeubs}}}
\transformfont{zpeurot}{\slantfont{\int{slant}}{\fromafm{zpeurt}}}
\transformfont{zpeubot}{\slantfont{\int{slant}}{\fromafm{zpeubt}}}
```

Symbol fonts are not reencoded but we still need slanted versions of the upright euro fonts to go with our slanted text fonts.

```
\installfonts
\installfamily{U}{peu}{}
```

When creating font definition files for symbol fonts, we use the encoding code `U` to indicate an unknown (font specific) encoding.

```
\installrawfont{zpeur}{zpeur}{txtfdmns,zpeur mtxasetx}{U}{peu}{m}{n}{}
```

We want to convert the font metrics given in the `afm` file to TeX font metrics directly, without using virtual fonts as a mediating layer. For this kind of task, we need fontinst's `\installrawfont` macro. In case of a straight `afm` to `tfm` conversion, the name of the TeX font metric file (first argument) is identical to the name of the `afm` file (second argument). Like `\installfont`, the `\installrawfont` macro requires an encoding file as well. So how do we deal with a font specific encoding? We load `zpeur` followed by the option `mtxasetx`. This option instructs fontinst to

create an ad-hoc encoding vector based on the order of the glyphs in the font. We add the file `txtfdmns.etx` to ensure that TeX's `\fontdimen` parameters are set for this font as well (they are normally set by encoding files like `t1.etx`).

```
\installfontas{zpeur}{U}{peu}{m}{sc}{}
```

The fonts do not include a small caps shape so we reuse the upright one. The remaining fonts are installed in a similar way:

```
\installrawfont{zpeuro}{zpeuro}{txtfdmns,zpeuro mtxasetx}{U}{peu}{m}{sl}{}
\installrawfont{zpeuri}{zpeuri}{txtfdmns,zpeuri mtxasetx}{U}{peu}{m}{it}{}
\installrawfont{zpeub}{zpeub}{txtfdmns,zpeub mtxasetx}{U}{peu}{b}{n}{}
\installfontas{zpeub}{U}{peu}{b}{sc}{}
\installrawfont{zpeubo}{zpeubo}{txtfdmns,zpeubo mtxasetx}{U}{peu}{b}{sl}{}
\installrawfont{zpeubi}{zpeubi}{txtfdmns,zpeubi mtxasetx}{U}{peu}{b}{it}{}
\endinstallfonts
```

We add the Euro Sans fonts:

```
\installfonts
\installfamily{U}{peus}{}
\installrawfont{zpeurs}{zpeurs}{txtfdmns,zpeurs mtxasetx}{U}{peus}{m}{n}{}
\installfontas{zpeurs}{U}{peus}{m}{sc}{}
\installrawfont{zpeuros}{zpeuros}{txtfdmns,zpeuros mtxasetx}{U}{peus}{m}{sl}{}
\installrawfont{zpeuris}{zpeuris}{txtfdmns,zpeuris mtxasetx}{U}{peus}{m}{it}{}
\installrawfont{zpeubs}{zpeubs}{txtfdmns,zpeubs mtxasetx}{U}{peus}{b}{n}{}
\installfontas{zpeubs}{U}{peus}{b}{sc}{}
\installrawfont{zpeubos}{zpeubos}{txtfdmns,zpeubos mtxasetx}{U}{peus}{b}{sl}{}
\installrawfont{zpeubis}{zpeubis}{txtfdmns,zpeubis mtxasetx}{U}{peus}{b}{it}{}
\endinstallfonts
```

And the Euro Mono fonts:

```
\installfonts
\installfamily{U}{peut}{}
\installrawfont{zpeurt}{zpeurt}{txtfdmns,zpeurt mtxasetx}{U}{peut}{m}{n}{}
```

```
\installfontas{zpeurt}{U}{peut}{m}{sc}{}
\installrawfont{zpeurot}{zpeurot}{txtfdmns,zpeurot mtxasetx}{U}{peut}{m}{sl}{}
\installrawfont{zpeurit}{zpeurit}{txtfdmns,zpeurit mtxasetx}{U}{peut}{m}{it}{}
\installrawfont{zpeubt}{zpeubt}{txtfdmns,zpeubt mtxasetx}{U}{peut}{b}{n}{}
\installfontas{zpeubt}{U}{peut}{b}{sc}{}
\installrawfont{zpeubot}{zpeubot}{txtfdmns,zpeubot mtxasetx}{U}{peut}{b}{sl}{}
\installrawfont{zpeubit}{zpeubit}{txtfdmns,zpeubit mtxasetx}{U}{peut}{b}{it}{}
\endinstallfonts
\endrecordtransforms
\bye
```

Fontinst will record the mapping data in `peu-rec.tex`, but we still need an additional driver file which converts these records to the final map file:

```
\input finstmsc.sty
\resetstr{PSfontsuffix}{.pfb}
\adddriver{dvips}{peu.map}
\input peu-rec.tex
\donedrivers
\bye
```

After running both fontinst files through `tex`, we process the property list files (`pl`) created by fontinst with `pltotf` in order to generate TeX font metric files (`tfm`). We install the map file `peu.map` as well as all `afm`, `tfm`, `pfb`, and `fd` files in the local TeX tree as explained section 1.4 and add `peu.map` to the configuration files for pdfTeX, dvips, and xdvi. Finally, we run `texhash`. The euro symbol can now be used in virtual fonts. Since we have font definition files for LaTeX as well, we could also access it in any LaTeX file with a command sequence like this one:

```
{\fontencoding{U}\fontfamily{peu}\selectfont\char 128}
```

So let us make that a generic euro package, `peufonts.sty`, for use with all fonts that do not provide a native euro symbol:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{peufonts}[2002/10/25 v1.0 Adobe Euro Fonts]
\RequirePackage{textcomp}
```

```
4  \DeclareRobustCommand{\eurrm}{%
5    {\fontencoding{U}\fontfamily{peu}\selectfont\char 128}}
6  \DeclareRobustCommand{\eursf}{%
7    {\fontencoding{U}\fontfamily{peus}\selectfont\char 128}}
8  \DeclareRobustCommand{\eurtt}{%
9    {\fontencoding{U}\fontfamily{peut}\selectfont\char 128}}
```

We define three macros, `\eurrm`, `\eursf`, and `\eurtt`, which typeset a serif, sans serif, and monospaced euro symbol respectively. Note the additional set of braces. They form a group which keeps the font change local.

```
10 \DeclareOption{serif}{\def\eur{\eurrm}}
11 \DeclareOption{sans}{\def\eur{\eursf}}
12 \DeclareOption{mono}{\def\eur{\eurtt}}
13 \DeclareOption{textcomp}{%
14   \PackageInfo{peufonts}{Hijacking '\string\texteuro'}%
15   \def\texteuro{\eur}}
16 \ExecuteOptions{sans}
17 \ProcessOptions*
18 \endinput
```

We also provide `\eur` along with three options controlling whether it uses the serif, sans serif, or monospaced euro symbol. Sans is set up as the default in line 16. The option `textcomp` will hijack the text command `\texteuro` as provided by the `textcomp` package. This is very handy when using the `inputenc` package with Latin 9 (ISO-8859-15) as input encoding and entering the euro symbol directly, as `inputenc` uses `\texteuro` internally. With this option, we may also type `\texteuro` or simply `€` in the input file to typeset a euro symbol. For this to work, `inputenc` has to be loaded before this package. Please keep in mind that this is a global redefinition affecting all text fonts. We do not activate it by default as some fonts may provide a native euro symbol. We also write a message to the log when redefining `\texteuro` and request the `textcomp` package in line 3.

# Tutorial 5

# Expert font sets, regular setup

Expert fonts are complements to be used in conjunction with regular text fonts. They usually contain optical small caps, additional sets of figures – hanging, inferior, superior –, the f-ligatures ff, fi, fl, ffi, and ffl, plus a few text fractions and some other symbols. Since they are companion fonts only, which do not contain the regular uppercase and lowercase alphabet, they are not useful on their own. To employ them in a sensible way we need the basic text fonts as well. In this tutorial, we will install the complete Monotype Janson font set as provided by the base and the expert package offered by Agfa Monotype. The base package contains four text fonts (regular, regular italic, bold, bold italic):

```
jan_____.afm    jani____.afm    janb____.afm    janbi___.afm
jan_____.inf    jani____.inf    janb____.inf    janbi___.inf
jan_____.pfb    jani____.pfb    janb____.pfb    janbi___.pfb
jan_____.pfm    jani____.pfm    janb____.pfm    janbi___.pfm
```

The expert package adds the corresponding expert fonts:

```
jny_____.afm    jnyi____.afm    jnyb____.afm    jnybi___.afm
```

```
jny_____.inf    jnyi____.inf    jnyb____.inf    jnybi___.inf
jny_____.pfb    jnyi____.pfb    jnyb____.pfb    jnybi___.pfb
jny_____.pfm    jnyi____.pfm    jnyb____.pfm    jnybi___.pfm
```

When talking about 'expert font sets' in this tutorial, we are referring to all of the above (base plus expert package). The proper file names for Monotype Janson are given in `monotype.map`. Expert fonts have essentially the same file name as the corresponding text fonts, but their encoding code is `8x` instead of `8a` for Adobe Standard encoding. After renaming the files, we start off with the following file set:

```
mjnr8a.afm      mjnri8a.afm      mjnb8a.afm      mjnbi8a.afm
mjnr8a.pfb      mjnri8a.pfb      mjnb8a.pfb      mjnbi8a.pfb

mjnr8x.afm      mjnri8x.afm      mjnb8x.afm      mjnbi8x.afm
mjnr8x.pfb      mjnri8x.pfb      mjnb8x.pfb      mjnbi8x.pfb
```

There are two ways to install an expert font set. Apart from writing a verbose fontinst file using low-level commands we may also use the `\latinfamily` macro. We will take a look at the latter case first and proceed with a verbose fontinst file afterwards.

## 5.1 A basic fontinst file

As usual, our driver file starts with a typical header (1–4). The Janson expert package provides optical small caps for the regular weight but the bold expert fonts do not contain any small caps. For the bold series, we have to make do with mechanical small caps. The `\latinfamily` macro will take care of that automatically. All we need to do is define a scaling factor of 0.72 on line 4:

```
1 \input fontinst.sty
2 \needsfontinstversion{1.926}
3 \substitutesilent{bx}{b}
4 \setint{smallcapsscale}{720}
5 \recordtransforms{mjn-rec.tex}
```

In the third tutorial we have incorporated lining and hanging figures by creating two font families: a family with the basic, three-character font family name (lining figures) and a second family featuring hanging figures, with the letter `j` appended to the font family name. The character `j` is the Fontname code for hanging figures. In this tutorial, we need an additional code: the letter `x`, indicating a font featuring expert glyphs. When installing expert sets with the `\latinfamily` macro we use these family names to instruct fontinst that we have an expert set at hand and that we want it to create a font family featuring expert glyphs with lining figures (6) plus a second family featuring expert glyphs with hanging figures (7):

```
\latinfamily{mjnx}{}
\latinfamily{mjnj}{}
\endrecordtransforms
\bye
```

Please note that appending `x` and `j` to the font family name works for expert font sets only. The `\latinfamily` macro is not capable of dealing with SC & OSF font sets in the same way. Such font sets always require a fontinst file using low-level commands such as the one discussed in tutorial 3.

## 5.2 A verbose fontinst file

While the `\latinfamily` macro incorporates the most fundamental features of expert sets, such as optical small caps and additional f-ligatures, it does not exploit all the glyphs found in expert fonts. To take advantage of them, we need to use low-level fontinst commands, at least for parts of the fontinst file. But before we start with our verbose fontinst file, let us first take a look at some encoding issues specific to expert fonts. When dealing with SC & OSF fonts in the third tutorial, we had to rename some glyphs or move them around because in SC & OSF fonts, hanging figures and small caps are found in the standard slots for figures and the lowercase alphabet. With small caps and hanging figures provided by expert fonts the installation is in fact simpler since all glyph names are unique. To understand the difference, we will take a brief look at the glyph names in the respective `afm` files. Compare the names of the lowercase glyphs as found in `mjnr8a.afm` to the small caps glyph names in `mjnr8x.afm`:

```
C  97 ; WX 427 ; N a ; B 59 -13 409 426 ;
```

```
C  98 ; WX 479 ; N b ; B 18 -13 442 692 ;
C  99 ; WX 427 ; N c ; B 44 -13 403 426 ;

C  97 ; WX 479 ; N Asmall ; B 19  -4 460 451 ;
C  98 ; WX 438 ; N Bsmall ; B 31  -4 395 434 ;
C  99 ; WX 500 ; N Csmall ; B 37 -12 459 443 ;
```

The situation is similar for lining and hanging (‘old style’) figures. The following lines are taken from `mjnr8a.afm` and `mjnr8x.afm` respectively:

```
C 48 ; WX 469 ; N zero ; B  37 -12 432 627 ;
C 49 ; WX 469 ; N one  ; B 109  -5 356 625 ;
C 50 ; WX 469 ; N two  ; B  44   0 397 627 ;

C 48 ; WX 469 ; N zerooldstyle ; B 39  0 431 387 ;
C 49 ; WX 271 ; N oneoldstyle  ; B 44 -5 229 405 ;
C 50 ; WX 396 ; N twooldstyle  ; B 37  0 356 415 ;
```

In practice, this means that adding expert fonts to the basic font set amounts to little more than adding them to the input file list of `\installfont` in most cases. Still, some additional steps are required. Fortunately, all we need to do in order to make optical small caps and hanging figures readily available is using dedicated encoding vectors provided by fontinst. These encoding vectors reference the glyphs by names corresponding to those found in expert fonts, thus allowing us to pick optical small caps and hanging figures at will. With that in mind, we can get down to business. Our fontinst file begins with a typical header:

```
1 \input fontinst.sty
2 \needsfontinstversion{1.926}
3 \substitutesilent{bx}{b}
4 \setint{smallcapsscale}{720}
5 \setint{slant}{167}
6 \recordtransforms{mjn-rec.tex}
```

Unfortunately, Monotype Janson provides small caps for the regular weight only. Hence we have to make do with mechanical small caps for the bold series. We set a scaling factor of 0.72 for that in line 4.

```
\transformfont{mjnr8r}{\reencodefont{8r}{\fromafm{mjnr8a}}}
\transformfont{mjnri8r}{\reencodefont{8r}{\fromafm{mjnri8a}}}
\transformfont{mjnb8r}{\reencodefont{8r}{\fromafm{mjnb8a}}}
\transformfont{mjnbi8r}{\reencodefont{8r}{\fromafm{mjnbi8a}}}
\transformfont{mjnro8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{mjnr8a}}}
\transformfont{mjnbo8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{mjnb8a}}}
```

We reencode (7–10) and slant (11–12) the basic fonts as usual. Expert fonts do not require any reencoding, but we need slanted versions of them as well:

```
\transformfont{mjnro8x}{\slantfont{\int{slant}}{\fromafm{mjnr8x}}}
\transformfont{mjnbo8x}{\slantfont{\int{slant}}{\fromafm{mjnb8x}}}
```

We will create two font families: `mjnx`, featuring expert glyphs, optical small caps, and lining figures, plus `mjnj` incorporating hanging instead of lining figures. TS1 encoded virtual fonts will be generated for the `mjnx` family only.

```
\installfonts
\installfamily{T1}{mjnx}{}
\installfont{mjnr9e}{mjnr8r,mjnr8x,newlatin}{t1}{T1}{mjnx}{m}{n}{}
```

As mentioned above, incorporating expert glyphs boils down to adding an additional file to the arguments of the `\installfont` command, in this case the file `mjnr8x.afm`. Note that we use the encoding suffix `9e` instead of `8t` for all T1 encoded virtual fonts of the `mjnx` family to indicate that they feature expert glyphs. While the code `8t`, as defined by the Fontname scheme, is for T1 (Cork) encoding, `9e` indicates T1 plus expert glyphs. Please refer to section 2.4 of the Fontname scheme for a comprehensive list of these codes and the code tables on page 109 of this guide for additional hints.

```
\installfont{mjnrc9e}{mjnr8r,mjnr8x,newlatin}{t1c}{T1}{mjnx}{m}{sc}{}
```

For the small caps font we use the encoding vector `t1c.etx` which will map the small caps in `mjnr8x.afm` to the encoding slots of the lowercase alphabet in our T1 encoded virtual font. The remaining virtual fonts of the `mjnx` family are built as expected:

```
\installfont{mjnri9e}{mjnri8r,mjnri8x,newlatin}{t1}{T1}{mjnx}{m}{it}{}
\installfont{mjnro9e}{mjnro8r,mjnro8x,newlatin}{t1}{T1}{mjnx}{m}{sl}{}
\installfont{mjnb9e}{mjnb8r,mjnb8x,newlatin}{t1}{T1}{mjnx}{b}{n}{}
\installfont{mjnbc9e}{mjnb8r,mjnb8x,newlatin}{t1c}{T1}{mjnx}{b}{sc}{}
```

Since the bold expert font does not include small caps, we have to create mechanical ones. The `t1c.etx` encoding vector will deal with that transparently if it does not find optical small caps in any of the raw fonts, using the value of `smallcapsscale` as the scaling factor.

```
\installfont{mjnbi9e}{mjnbi8r,mjnbi8x,newlatin}{t1}{T1}{mjnx}{b}{it}{}
\installfont{mjnbo9e}{mjnbo8r,mjnbo8x,newlatin}{t1}{T1}{mjnx}{b}{sl}{}
\endinstallfonts
```

That's it for T1 encoding. Creating TS1 encoded virtual fonts featuring expert glyphs is pretty straightforward. In order to take advantage of the additional glyphs provided by expert fonts, we simply add them to the input file list:

```
\installfonts
\installfamily{TS1}{mjnx}{}
\installfont{mjnr9c}{mjnr8r,mjnr8x,textcomp}{ts1}{TS1}{mjnx}{m}{n}{}
```

Note the encoding suffix of the virtual fonts. We use `9c` instead of `8c` to indicate that the virtual fonts feature expert glyphs.

```
\installfontas{mjnr9c}{TS1}{mjnx}{m}{sc}{}
\installfont{mjnri9c}{mjnri8r,mjnri8x,textcomp}{ts1}{TS1}{mjnx}{m}{it}{}
\installfont{mjnro9c}{mjnro8r,mjnro8x,textcomp}{ts1}{TS1}{mjnx}{m}{sl}{}
\installfont{mjnb9c}{mjnb8r,mjnb8x,textcomp}{ts1}{TS1}{mjnx}{b}{n}{}
\installfontas{mjnb9c}{TS1}{mjnx}{b}{sc}{}
\installfont{mjnbi9c}{mjnbi8r,mjnbi8x,textcomp}{ts1}{TS1}{mjnx}{b}{it}{}
\installfont{mjnbo9c}{mjnbo8r,mjnbo8x,textcomp}{ts1}{TS1}{mjnx}{b}{sl}{}
\endinstallfonts
```

The `mjnx` family including T1 and TS1 encoded fonts is now complete. We continue with the `mjnj` family which we want to feature hanging figures by default:

```
\installfonts
\installfamily{T1}{mjnj}{}
\installfont{mjnr9d}{mjnr8r,mjnr8x,newlatin}{t1j}{T1}{mjnj}{m}{n}{}
```

The encoding code `9d` indicates a T1 encoded font with expert glyphs and hanging figures. We will use this code for all T1 encoded virtual fonts of the `mjnj` family. This family is supposed to feature hanging figures in the standard encoding slots for figures. We have to keep in mind that the regular encoding vector for T1 encoding (`t1.etx`) references the figures as 'zero' and 'one' while the hanging ('old style') figures in the expert font (which we want to be available by default) are labeled 'zerooldstyle' and 'oneoldstyle'. In order to arrange the glyphs according to our wishes, we use the special encoding vector `t1j.etx`. This file is essentially equivalent to `t1.etx`, but it will automatically append the suffix 'oldstyle' to the names of figures referenced by the encoding vector.

```
\installfont{mjnrc9d}{mjnr8r,mjnr8x,newlatin}{t1cj}{T1}{mjnj}{m}{sc}{}
```

For the small caps shape, we use the encoding file `t1cj.etx` instead of `t1c.etx` to make hanging figures the default. The other virtual fonts are built like the upright shape:

```
\installfont{mjnri9d}{mjnri8r,mjnri8x,newlatin}{t1j}{T1}{mjnj}{m}{it}{}
\installfont{mjnro9d}{mjnro8r,mjnro8x,newlatin}{t1j}{T1}{mjnj}{m}{sl}{}
\installfont{mjnb9d}{mjnb8r,mjnb8x,newlatin}{t1j}{T1}{mjnj}{b}{n}{}
\installfont{mjnbc9d}{mjnb8r,mjnb8x,newlatin}{t1cj}{T1}{mjnj}{b}{sc}{}
```

The bold expert fonts do not include small caps but the encoding file `t1cj.etx` is capable of creating mechanical small caps transparently, hence we use it for the bold small caps font as well.

```
\installfont{mjnbi9d}{mjnbi8r,mjnbi8x,newlatin}{t1j}{T1}{mjnj}{b}{it}{}
\installfont{mjnbo9d}{mjnbo8r,mjnbo8x,newlatin}{t1j}{T1}{mjnj}{b}{sl}{}
\endinstallfonts
```

Finally, we use `\installfontas` to 'install' the TS1 encoded virtual fonts of the `mjnx` family as `TS1/mjnj`. This will merely add some lines to the font definition file without creating any additional virtual fonts:

```
\installfonts
\installfamily{TS1}{mjnj}{}
\installfontas{mjnr9c}{TS1}{mjnj}{m}{n}{}
```

```
\installfontas{mjnr9c}{TS1}{mjnj}{m}{sc}{}
\installfontas{mjnri9c}{TS1}{mjnj}{m}{it}{}
\installfontas{mjnro9c}{TS1}{mjnj}{m}{sl}{}
\installfontas{mjnb9c}{TS1}{mjnj}{b}{n}{}
\installfontas{mjnb9c}{TS1}{mjnj}{b}{sc}{}
\installfontas{mjnbi9c}{TS1}{mjnj}{b}{it}{}
\installfontas{mjnbo9c}{TS1}{mjnj}{b}{sl}{}
\endinstallfonts
```

At this point, we have a comprehensive text setup featuring expert f-ligatures, optical small caps as well as a choice of readily available lining and hanging figures. However, there are some glyphs in expert fonts that we have not considered yet.

## 5.3 Inferior and superior figures

Expert fonts usually provide superior and inferior figures which can be combined with a dedicated fraction slash called 'solidus' to typeset arbitrary text fractions like $^{1}/_{2}$ or even $^{31}/_{127}$. Please note that these figures are not suitable for TeX's math mode but they can be useful in text mode even if there is no need to typeset text fractions. For example, in this guide the footnote marks in the body text are typeset using superior figures and inferior figures are used for the line numbers of the code listings. Like hanging figures, we want inferior and superior figures to be readily available. Therefore, we will create two additional font families, `mjn0` and `mjn1`, which put inferior and superior figures in the standard encoding slots for figures just like our `mjnj` family does for hanging figures. We have been using the encoding vector `t1j.etx` to make hanging figures the default in this tutorial so let us find out what `t1j.etx` does in detail and try to modify this approach according to our needs. The regular T1 encoding vector `t1.etx` defines the encoding slots for all figures as follows:

```
\setslot{\digit{one}}\endsetslot
\setslot{\digit{two}}\endsetslot
\setslot{\digit{three}}\endsetslot
```

The glyph names of figures are not given verbatim, they are used as an argument to the `\digit` macro. This is the default definition of said macro as given in `t1.etx`:

```
\setcommand\digit#1{#1}
```

This means that the glyph labeled 'one' in the `afm` file will end up in the encoding slot for the numeral one in the virtual font – and so on. `t1j.etx` defines the `\digit` macro as follows:

```
\setcommand\digit#1{#1oldstyle}
```

In this case the glyph labeled 'oneoldstyle' in the `afm` file will end up in the encoding slot for the numeral one in the T1 encoded virtual font. When comparing the glyph names of hanging, inferior, and superior figures in the `afm` files of our expert fonts now, the approach we need to take in order to access them should be obvious:

```
C  48 ; WX 469 ; N zerooldstyle ; B 39  0 431 387 ;
C  49 ; WX 271 ; N oneoldstyle  ; B 44 -5 229 405 ;
C  50 ; WX 396 ; N twooldstyle  ; B 37  0 356 415 ;

C 210 ; WX 323 ; N zeroinferior ; B 27 -13 296 355 ;
C 211 ; WX 323 ; N oneinferior  ; B 84  -5 240 357 ;
C 212 ; WX 323 ; N twoinferior  ; B 27   0 288 358 ;

C 200 ; WX 323 ; N zerosuperior ; B 27 293 296 661 ;
C 201 ; WX 323 ; N onesuperior  ; B 84 298 240 661 ;
C 202 ; WX 323 ; N twosuperior  ; B 27 303 288 661 ;
```

Just like 'old style' figures, inferior and superior figures use suffixes to the respective glyph names in (properly encoded) expert fonts. This means that we can simply load an additional encoding file before loading `t1.etx` and predefine the `\digit` macro accordingly. For inferior figures, we create the file `inferior.etx`:

```
\relax
\encoding
\setcommand\digit#1{#1inferior}
\endencoding
\endinput
```

All we need to do is use `\setcommand` toï¿œpredefine the `\digit` macro like this:

```
\setcommand\digit#1{#1inferior}
```

This will add the suffix 'inferior' to all digits. Since fontinst's `\setcommand` macro works like LaTeX's `\providecommand`, the encoding file `t1.etx` will not overwrite our definition if we load it after `inferior.etx`. The approach is similar for superior figures. We create another encoding file called `superior.etx`:

```
\relax
\encoding
\setcommand\digit#1{#1superior}
\endencoding
\endinput
```

With `inferior.etx` and `superior.etx` at hand, we can now easily create the font families `mjn0` and `mjn1`. Let us put the new encoding vectors in our working directory and go back to the fontinst file:

```
59 \installfonts
60 \installfamily{T1}{mjn0}{}
61 \installfont{mjnr09e}{mjnr8r,mjnr8x,newlatin}{inferior,t1}{T1}{mjn0}{m}{n}{}
```

We add the Fontname code `0` to the names of the virtual fonts in order to indicate inferior figures, load our newly created encoding file `inferior.etx` before `t1.etx`, and adapt the NFSS font declarations accordingly. Other than that, the virtual fonts of the `mjn0` family are generated in the usual way:

```
62 \installfontas{mjnr09e}{T1}{mjn0}{m}{sc}{}
63 \installfont{mjnri09e}{mjnri8r,mjnri8x,newlatin}{inferior,t1}{T1}{mjn0}{m}{it}{}
64 \installfont{mjnro09e}{mjnro8r,mjnro8x,newlatin}{inferior,t1}{T1}{mjn0}{m}{sl}{}
65 \installfont{mjnb09e}{mjnb8r,mjnb8x,newlatin}{inferior,t1}{T1}{mjn0}{b}{n}{}
66 \installfontas{mjnb09e}{T1}{mjn0}{b}{sc}{}
67 \installfont{mjnbi09e}{mjnbi8r,mjnbi8x,newlatin}{inferior,t1}{T1}{mjn0}{b}{it}{}
68 \installfont{mjnbo09e}{mjnbo8r,mjnbo8x,newlatin}{inferior,t1}{T1}{mjn0}{b}{sl}{}
69 \endinstallfonts
```

Any TS1 encoded virtual fonts of the `mjn0` family would not differ from those of `mjnx`, so we create a font definition file which points LaTeX to the TS1 encoded virtual fonts we created for the `mjnx` family before:

```
\installfonts
\installfamily{TS1}{mjn0}{}
\installfontas{mjnr9c}{TS1}{mjn0}{m}{n}{}
\installfontas{mjnr9c}{TS1}{mjn0}{m}{sc}{}
\installfontas{mjnri9c}{TS1}{mjn0}{m}{it}{}
\installfontas{mjnro9c}{TS1}{mjn0}{m}{sl}{}
\installfontas{mjnb9c}{TS1}{mjn0}{b}{n}{}
\installfontas{mjnb9c}{TS1}{mjn0}{b}{sc}{}
\installfontas{mjnbi9c}{TS1}{mjn0}{b}{it}{}
\installfontas{mjnbo9c}{TS1}{mjn0}{b}{sl}{}
\endinstallfonts
```

For the `mjn1` family, we adapt the names of the virtual fonts (adding the Fontname code 1 to indicate superior figures), the encoding files (`superior.etx` and `t1.etx`), and the NFSS declarations:

```
\installfonts
\installfamily{T1}{mjn1}{}
\installfont{mjnr19e}{mjnr8r,mjnr8x,newlatin}{superior,t1}{T1}{mjn1}{m}{n}{}
```

We create the remaining virtual fonts in a similar way:

```
\installfontas{mjnr19e}{T1}{mjn1}{m}{sc}{}
\installfont{mjnri19e}{mjnri8r,mjnri8x,newlatin}{superior,t1}{T1}{mjn1}{m}{it}{}
\installfont{mjnro19e}{mjnro8r,mjnro8x,newlatin}{superior,t1}{T1}{mjn1}{m}{sl}{}
\installfont{mjnb19e}{mjnb8r,mjnb8x,newlatin}{superior,t1}{T1}{mjn1}{b}{n}{}
\installfontas{mjnb19e}{T1}{mjn1}{b}{sc}{}
\installfont{mjnbi19e}{mjnbi8r,mjnbi8x,newlatin}{superior,t1}{T1}{mjn1}{b}{it}{}
\installfont{mjnbo19e}{mjnbo8r,mjnbo8x,newlatin}{superior,t1}{T1}{mjn1}{b}{sl}{}
\endinstallfonts
```

Finally, we create a font definition file for `TS1/mjn1` and terminate our fontinst file:

```
\installfonts
```

```
\installfamily{TS1}{mjn1}{}
\installfontas{mjnr9c}{TS1}{mjn1}{m}{n}{}
\installfontas{mjnr9c}{TS1}{mjn1}{m}{sc}{}
\installfontas{mjnri9c}{TS1}{mjn1}{m}{it}{}
\installfontas{mjnro9c}{TS1}{mjn1}{m}{sl}{}
\installfontas{mjnb9c}{TS1}{mjn1}{b}{n}{}
\installfontas{mjnb9c}{TS1}{mjn1}{b}{sc}{}
\installfontas{mjnbi9c}{TS1}{mjn1}{b}{it}{}
\installfontas{mjnbo9c}{TS1}{mjn1}{b}{sl}{}
\endinstallfonts
\endrecordtransforms
\bye
```

Our setup is now complete as far as LaTeX is concerned. We still need to create another fontinst file that will read the data recorded in `mjn-rec.tex` and convert it to a map file suitable for dvips. The format of this driver file is discussed in section 1.5 of this guide.

## 5.4 An extended style file

Our style file for Janson, `janson.sty`, is based on the one suggested in section 3.2. We simply adjust the package name and the names of the font families:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{janson}[2002/12/30 v1.0 Monotype Janson]
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\RequirePackage{nfssext}
\DeclareOption{lining}{\renewcommand*{\rmdefault}{mjnx}}
\DeclareOption{oldstyle}{\renewcommand*{\rmdefault}{mjnj}}
\ExecuteOptions{oldstyle}
\ProcessOptions*
\endinput
```

With an expert font set at hand, however, we have to extend `nfssext.sty` to support expert families:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{nfssext}[2003/03/14 v1.2 Experimental NFSS Extensions]
\newcommand*{\exfs@tempa}{}
\newcommand*{\exfs@tempb}{}
\newcommand*{\exfs@try@family}[2][]{%
  \let\exfs@tempa\relax
  \begingroup
    \fontfamily{#2}\try@load@fontshape
    \expandafter\ifx\csname\curr@fontshape\endcsname\relax
      \edef\exfs@tempa{#1}%
      \ifx\exfs@tempa\@empty
        \PackageWarning{nfssext}{%
          Font family '\f@encoding/#2' not available\MessageBreak
          Ignoring font switch}%
      \else
        \PackageInfo{nfssext}{%
          Font family '\f@encoding/#2' not available\MessageBreak
          Font family '\f@encoding/#1' tried instead}%
        \exfs@try@family{#1}%
      \fi
    \else
      \gdef\exfs@tempa{\fontfamily{#2}\selectfont}%
    \fi
  \endgroup
  \exfs@tempa}
```

As soon as expert fonts come into play, the `\lnstyle` macro has to cater for two font families which, depending on the font, may contain lining figures: a basic font family with a three-character code or an expert family with a four-character code ending with the letter `x`. To make sure that `nfssext.sty` will work for fonts like Janson as well as fonts without an expert set, the first thing we need to do is extend our main font switching macro, enabling it to cope with both cases. To do so, we will introduce an optional argument. Essentially, we try to load the font

family given by the mandatory argument first (8). If this family is not available, we do not quit with a warning but add a note to the log file (16–18) and try the family given by the optional argument next (19). If loading the alternative family fails as well, we finally print a warning message (12–14). If the optional argument is not used, the second step will be omitted.

```
\def\exfs@get@base#1#2#3#4\@nil{#1#2#3}
\DeclareRobustCommand{\lnstyle}{%
  \not@math@alphabet\lnstyle\relax
  \exfs@try@family[\expandafter\exfs@get@base\f@family\@nil]%
                  {\expandafter\exfs@get@base\f@family\@nil x}}
```

After that, the `\lnstyle` macro needs to be adjusted in order to exploit the optional argument. It will try the expert family with a four-character code first (30) and make `\exfs@try@family` fall back to the basic font family with a three-character code (29) if the former is not available.

```
\DeclareRobustCommand{\osstyle}{%
  \not@math@alphabet\osstyle\relax
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil j}}
```

The availability of hanging figures is expressed by appending the letter `j` to the font family code for both basic and expert font sets, so `\osstyle` does not need any modification.

```
\DeclareRobustCommand{\instyle}{%
  \not@math@alphabet\instyle\relax
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil 0}}
\DeclareRobustCommand{\sustyle}{%
  \not@math@alphabet\sustyle\relax
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil 1}}
```

With inferior and superior figures implemented as two additional font families, `mjn0` and `mjn1`, we add two macros activating these families by adding `0` and `1` to the family name respectively.

```
\DeclareTextFontCommand{\textln}{\lnstyle}
\DeclareTextFontCommand{\textos}{\osstyle}
\DeclareTextFontCommand{\textin}{\instyle}
```

```
\DeclareTextFontCommand{\textsu}{\sustyle}
\endinput
```

We also add two text commands, `\textin` and `\textsu`, which activate these figures locally, similar to `\textit` or `\textbf`.

## 5.5 Using the features of expert fonts

Most features of expert font sets such as additional f-ligatures and optical small caps will be available automatically when selecting the new font families. Using them does not require any additional macros. Lining and hanging figures can be conveniently selected by activating the respective font family, in this case `mjnx` and `mjnj`, or by using the style file `janson.sty` suggested above. Since inferior and superior figures are not used as regular figures, they are treated differently. We will take a look at some possible applications. The inferior and superior figures found in expert fonts were originally intended for typesetting text fractions so let us write a simple macro for that. To typeset a fraction, we combine inferior and superior figures with the `\textfractionsolidus` macro provided by the `textcomp` package. Accessing the figures implies switching font families locally. Note the additional set of braces which will keep the font change local:

```
\newcommand*{\textfrac}[2]{%
  {\fontfamily{mjn1}\selectfont #1}%
  \textfractionsolidus
  {\fontfamily{mjn0}\selectfont #2}}
```

Writing `\textfrac{1}{2}` in the input file will typeset the fraction $^{1}/_{2}$. When looking at regular and expert fonts in a font editor, you will see that they contain a fixed number of text fractions. Some of them are included in TS1 encoding and supported by the `textcomp` package (see appendix B), but typing rather long commands such as `\textthreequarters` is not exactly convenient. Since there are only nine of them they are not very useful anyway. With a complete set of inferior and superior figures at our disposal, our macro will work for arbitrary fractions like $^{3}/_{7}$ or $^{13}/_{17}$. Instead of using 'hard-wired' fonts as shown above, it is even better to use the font switching macros provided by `nfssext.sty` instead since they will dynamically adjust to the active text font:

```
\newcommand*{\textfrac}[2]{%
  \textsu{#1}%
  \textfractionsolidus
  \textin{#2}}
```

What about using superior figures as footnote numbers? To do so, we need to redefine `\@makefnmark`. This is LaTeX's default definition:

```
\def\@makefnmark{\hbox{\@textsuperscript{\normalfont\@thefnmark}}}
```

In order to use optical superior figures instead of mechanical ones, we drop `\@textsuperscript` and switch font families instead:

```
\def\@makefnmark{\hbox{\fontfamily{mjn1}\selectfont\@thefnmark}}
```

We do not need to add additional braces in this case since `\hbox` will keep the font change local. Using our new font switching macros, this may also be accomplished like this:

```
\def\@makefnmark{\hbox{\sustyle\@thefnmark}}
```

Keep in mind that, if you want to put a definition of `\@makefnmark` in the preamble of a regular LaTeX input file (as opposed to a class or a style file), it has to be enclosed in `\makeatletter`ï¿œand `\makeatother`:

```
\makeatletter
\def\@makefnmark{\hbox{\sustyle\@thefnmark}}
\makeatother
```

# Tutorial 6

# Expert font sets, extended setup

In this tutorial we will combine what we have learned in tutorials 3 and 5 to install a very complete font set featuring expert fonts, small caps, and hanging figures. This tutorial will also add multiple weights, italic small caps, italic swashes and text ornaments to that. Our example is Adobe Minion, base plus expert packages:

```
pmnr8a    Minion-Regular                A    143    morg____
pmnrc8a   Minion-RegularSC              A    144    mosc____
pmnri8a   Minion-Italic                 A    143    moi_____
pmnric8a  Minion-ItalicSC               A    144    moisc___
pmnriw7a  Minion-SwashItalic            A    144    moswi___
pmns8a    Minion-Semibold               A    143    mosb____
pmnsc8a   Minion-SemiboldSC             A    144    mosbs___
pmnsi8a   Minion-SemiboldItalic         A    143    mosbi___
pmnsic8a  Minion-SemiboldItalicSC       A    144    mosic___
pmnsiw7a  Minion-SwashSemiboldItalic    A    144    mossb___
pmnb8a    Minion-Bold                   A    143    mob_____
pmnbj8a   Minion-BoldOsF                A    144    mobos___
```

```
pmnbi8a   Minion-BoldItalic               A    143    mobi____
pmnbij8a  Minion-BoldItalicOsF            A    144    mobio___
pmnc8a    Minion-Black                    A    143    mobl____
pmncj8a   Minion-BlackOsF                 A    144    mozof___
pmnr8x    MinionExp-Regular               A    144    mjrg____
pmnri8x   MinionExp-Italic                A    144    mji_____
pmns8x    MinionExp-Semibold              A    144    mjsb____
pmnsi8x   MinionExp-SemiboldItalic        A    144    mjsbi___
pmnb8x    MinionExp-Bold                  A    144    mjb_____
pmnbi8x   MinionExp-BoldItalic            A    144    mjbi____
pmnc8x    MinionExp-Black                 A    144    mjbl____
pmnrp     Minion-Ornaments                A    144    moor____
```

Note that the bold and black fonts do not feature optical small caps. There are expert fonts for these weights, but they do not contain any small caps glyphs. When looking at the list of available shapes in each weight class it should be obvious that the semibold fonts are the intended default bold weight of this typeface. The bold fonts are merely intended for applications requiring a stronger contrast, for example to highlight the keywords in a dictionary. We will omit the black fonts in this tutorial as they are only of limited use. If required, they are easily added to the fontinst file. In addition to these text fonts, the expert package includes a set of regular-weight display fonts intended for titling and display work at very large sizes. Generated from the same master sources by interpolation, the display fonts share the lettershapes of the text fonts while being based on a design size of 72 pt. Since they form a complete set including small caps and expert fonts, they are handled just like the Minion text set and we will not explicitly consider them here.

## 6.1 The fontinst file

With a very comprehensive set of fonts at our disposal, we will be fastidious. We will not create any computed glyph shapes (no mechanical small caps and no slanted fonts), making this setup suitable for professional typesetting. Without further ado, we start off as usual:

```
1 \nonstopmode
```

```
2  \input fontinst.sty
3  \needsfontinstversion{1.926}
4  \substitutesilent{bx}{sb}
5  \recordtransforms{pmn-rec.tex}
```

We make semibold the default bold weight by substituting `sb` for `bx` in line 4. First of all, we reencode all base fonts which are based on Adobe Standard encoding. Even though the swash fonts are also based on Adobe Standard, they are handled like expert fonts because they contain a special set of glyphs, all of which are encoded by default:

```
6  \transformfont{pmnr8r}{\reencodefont{8r}{\fromafm{pmnr8a}}}
7  \transformfont{pmnrc8r}{\reencodefont{8r}{\fromafm{pmnrc8a}}}
8  \transformfont{pmnri8r}{\reencodefont{8r}{\fromafm{pmnri8a}}}
9  \transformfont{pmnric8r}{\reencodefont{8r}{\fromafm{pmnric8a}}}
10 \transformfont{pmns8r}{\reencodefont{8r}{\fromafm{pmns8a}}}
11 \transformfont{pmnsc8r}{\reencodefont{8r}{\fromafm{pmnsc8a}}}
12 \transformfont{pmnsi8r}{\reencodefont{8r}{\fromafm{pmnsi8a}}}
13 \transformfont{pmnsic8r}{\reencodefont{8r}{\fromafm{pmnsic8a}}}
14 \transformfont{pmnb8r}{\reencodefont{8r}{\fromafm{pmnb8a}}}
15 \transformfont{pmnbi8r}{\reencodefont{8r}{\fromafm{pmnbi8a}}}
```

In this tutorial we are dealing with a typeface featuring both SC & OSF and expert sets. When building virtual fonts, we could take small caps and hanging figures from either set of fonts. We will use the expert fonts anyway in order to take advantage of the extra f-ligatures exclusively found in expert fonts, so why not simply take the small caps from the same source as demonstrated in tutorial 5? Note that there is one problem with taking optical small caps from an expert font: there will be no kerning between the uppercase alphabet and the small caps replacing the lowercase letters because the glyphs are found in separate fonts. Without dedicated small caps fonts there is nothing we can do about that short of adding kerning pairs manually. Now that we have both expert and small caps fonts, however, we could take an approach similar to the one outlined in tutorial 3, adding the expert font on top of that to get the additional ligatures. We will use a different technique though, which extracts the more comprehensive kerning data from the small caps fonts while taking the glyphs from the base and the expert fonts only:

```
16 \reglyphfonts
```

```
17 \input csckrn2x.tex
18 \reglyphfont{pmnrc8x}{pmnrc8r}
19 \reglyphfont{pmnric8x}{pmnric8r}
20 \reglyphfont{pmnsc8x}{pmnsc8r}
21 \reglyphfont{pmnsic8x}{pmnsic8r}
22 \endreglyphfonts
```

To do so, we use fontinst's `reglyphfonts` environment and the `\reglyphfont` macro. This macro will essentially apply a batch job to the metric file given as the second argument and save the result to a new file specified by the first argument. The actual transformation commands are read from `csckrn2x.tex`, which is provided by fontinst. This file will discard all glyph metrics, keeping only the kerning data we need. It will also rename all glyphs so that they conform to the naming conventions of expert fonts (hence we use the encoding code `8x`, which indicates an expert font, when saving the transformed data). This way we ensure that we get unique glyph names. Apart from being conceptually cleaner, this approach has the additional benefit of not requiring the small caps fonts after the metrics and the virtual fonts have been generated, resulting in slightly smaller PDF and PostScript files if the fonts are embedded.

```
23 \installfonts
24 \installfamily{T1}{pmnx}{}
25 \installfont{pmnr9e}{pmnr8r,pmnr8x,newlatin}{t1}{T1}{pmnx}{m}{n}{}
26 \installfont{pmnri9e}{pmnri8r,pmnri8x,newlatin}{t1}{T1}{pmnx}{m}{it}{}
27 \installfontas{pmnri9e}{T1}{pmnx}{m}{sl}{}
```

The setup of the upright and italic shapes does not differ from tutorial 5 at all. We do not create slanted fonts but install the italic font as both italic and slanted shape.

```
28 \installfont{pmnrc9e}{pmnr8r,pmnr8x,pmnrc8x,newlatin}{t1c}{T1}{pmnx}{m}{sc}{}
```

When creating the small caps font, we use the encoding file `t1c.etx` since the small caps in the expert font bear unique names suitable for this encoding vector. We also want to add the kerning data found in `pmnrc8a.afm` to our virtual font. This data was already extracted and saved to `pmnrc8x` in line 18, so we simply add this file to the input file list. After that, we have a fully kerned small caps font.

```
\installfont{pmnric9e}{pmnri8r,pmnri8x,pmnric8x,newlatin}{t1c}{T1}{pmnx}{m}{si}{}
```

Minion also features an italic small caps font which we install just like its upright counterpart, using `si` as the NFSS shape code.

```
\installfont{pmns9e}{pmns8r,pmns8x,newlatin}{t1}{T1}{pmnx}{sb}{n}{}
\installfont{pmnsi9e}{pmnsi8r,pmnsi8x,newlatin}{t1}{T1}{pmnx}{sb}{it}{}
\installfontas{pmnsi9e}{T1}{pmnx}{sb}{sl}{}
\installfont{pmnsc9e}{pmns8r,pmns8x,pmnsc8x,newlatin}{t1c}{T1}{pmnx}{sb}{sc}{}
\installfont{pmnsic9e}{pmnsi8r,pmnsi8x,pmnsic8x,newlatin}{t1c}{T1}{pmnx}{sb}{si}{}
```

We repeat these steps for the semibold and the bold weight. The bold weight is slightly different because there are no optical small caps:

```
\installfont{pmnb9e}{pmnb8r,pmnb8x,newlatin}{t1}{T1}{pmnx}{b}{n}{}
\installfont{pmnbi9e}{pmnbi8r,pmnbi8x,newlatin}{t1}{T1}{pmnx}{b}{it}{}
\installfontas{pmnbi9e}{T1}{pmnx}{b}{sl}{}
\installfontas{pmnb9e}{T1}{pmnx}{b}{sc}{}
\installfontas{pmnbi9e}{T1}{pmnx}{b}{si}{}
\endinstallfonts
```

After finishing T1 encoding we continue with TS1. Our approach to TS1 encoding does not differ substantially from tutorial 5:

```
\installfonts
\installfamily{TS1}{pmnx}{}
\installfont{pmnr9c}{pmnr8r,pmnr8x,textcomp}{ts1}{TS1}{pmnx}{m}{n}{}
\installfont{pmnri9c}{pmnri8r,pmnri8x,textcomp}{ts1}{TS1}{pmnx}{m}{it}{}
\installfontas{pmnri9c}{TS1}{pmnx}{m}{sl}{}
\installfontas{pmnr9c}{TS1}{pmnx}{m}{sc}{}
\installfontas{pmnri9c}{TS1}{pmnx}{m}{si}{}
\installfont{pmns9c}{pmns8r,pmns8x,textcomp}{ts1}{TS1}{pmnx}{sb}{n}{}
\installfont{pmnsi9c}{pmnsi8r,pmnsi8x,textcomp}{ts1}{TS1}{pmnx}{sb}{it}{}
\installfontas{pmnsi9c}{TS1}{pmnx}{sb}{sl}{}
\installfontas{pmns9c}{TS1}{pmnx}{sb}{sc}{}
```

```
\installfontas{pmnsi9c}{TS1}{pmnx}{sb}{si}{}
\installfont{pmnb9c}{pmnb8r,pmnb8x,textcomp}{ts1}{TS1}{pmnx}{b}{n}{}
\installfont{pmnbi9c}{pmnbi8r,pmnbi8x,textcomp}{ts1}{TS1}{pmnx}{b}{it}{}
\installfontas{pmnbi9c}{TS1}{pmnx}{b}{sl}{}
\installfontas{pmnb9c}{TS1}{pmnx}{b}{sc}{}
\installfontas{pmnbi9c}{TS1}{pmnx}{b}{si}{}
\endinstallfonts
```

The `pmnx` family is now complete. We continue with `pmnj` which will feature hanging figures by default:

```
\installfonts
\installfamily{T1}{pmnj}{}
\installfont{pmnr9d}{pmnr8r,pmnr8x,newlatin}{t1j}{T1}{pmnj}{m}{n}{}
\installfont{pmnri9d}{pmnri8r,pmnri8x,newlatin}{t1j}{T1}{pmnj}{m}{it}{}
\installfontas{pmnri9d}{T1}{pmnj}{m}{sl}{}
```

To make hanging figures the default throughout the `pmnj` family we employ the encoding file `t1j.etx`. Other than that, the setup of the upright and italic shapes does not differ from `pmnx`.

```
\installfont{pmnrc9d}{pmnr8r,pmnr8x,pmnrc8x,newlatin}{t1cj}{T1}{pmnj}{m}{sc}{}
\installfont{pmnric9d}{pmnri8r,pmnri8x,pmnric8x,newlatin}{t1cj}{T1}{pmnj}{m}{si}{}
```

For the small caps shape of the `pmnj` family we essentially use the technique introduced above. Since this font family will feature hanging figures, however, we load the encoding file `t1cj.etx`.

```
\installfont{pmns9d}{pmns8r,pmns8x,newlatin}{t1j}{T1}{pmnj}{sb}{n}{}
\installfont{pmnsi9d}{pmnsi8r,pmnsi8x,newlatin}{t1j}{T1}{pmnj}{sb}{it}{}
\installfontas{pmnsi9d}{T1}{pmnj}{sb}{sl}{}
\installfont{pmnsc9d}{pmns8r,pmns8x,pmnsc8x,newlatin}{t1cj}{T1}{pmnj}{sb}{sc}{}
\installfont{pmnsic9d}{pmnsi8r,pmnsi8x,pmnsic8x,newlatin}{t1cj}{T1}{pmnj}{sb}{si}{}
```

Again, we repeat these steps for the semibold weight. The bold fonts are handled like those of the `pmnx` family, only differing in the choice of the encoding file:

```
\installfont{pmnb9d}{pmnb8r,pmnb8x,newlatin}{t1j}{T1}{pmnj}{b}{n}{}
\installfont{pmnbi9d}{pmnbi8r,pmnbi8x,newlatin}{t1j}{T1}{pmnj}{b}{it}{}
```

```
\installfontas{pmnbi9d}{T1}{pmnj}{b}{sl}{}
\installfontas{pmnb9d}{T1}{pmnj}{b}{sc}{}
\installfontas{pmnbi9d}{T1}{pmnj}{b}{si}{}
\endinstallfonts
```

We employ fontinst's `\installfontas` macro to provide a complete font definition file for TS1 encoding, using the TS1 encoded fonts of the `pmnx` family:

```
\installfonts
\installfamily{TS1}{pmnj}{}
\installfontas{pmnr9c}{TS1}{pmnj}{m}{n}{}
\installfontas{pmnr9c}{TS1}{pmnj}{m}{sc}{}
\installfontas{pmnri9c}{TS1}{pmnj}{m}{it}{}
\installfontas{pmnri9c}{TS1}{pmnj}{m}{sl}{}
\installfontas{pmnri9c}{TS1}{pmnj}{m}{si}{}
\installfontas{pmns9c}{TS1}{pmnj}{sb}{n}{}
\installfontas{pmns9c}{TS1}{pmnj}{sb}{sc}{}
\installfontas{pmnsi9c}{TS1}{pmnj}{sb}{it}{}
\installfontas{pmnsi9c}{TS1}{pmnj}{sb}{sl}{}
\installfontas{pmnsi9c}{TS1}{pmnj}{sb}{si}{}
\installfontas{pmnb9c}{TS1}{pmnj}{b}{n}{}
\installfontas{pmnb9c}{TS1}{pmnj}{b}{sc}{}
\installfontas{pmnbi9c}{TS1}{pmnj}{b}{it}{}
\installfontas{pmnbi9c}{TS1}{pmnj}{b}{sl}{}
\installfontas{pmnbi9c}{TS1}{pmnj}{b}{si}{}
\endinstallfonts
```

In addition to `pmnx` and `pmnj`, we add dedicated font families incorporating inferior and superior figures:

```
\installfonts
\installfamily{T1}{pmn0}{}
\installfont{pmnr09e}{pmnr8r,pmnr8x,newlatin}{inferior,t1}{T1}{pmn0}{m}{n}{}
\installfont{pmnri09e}{pmnri8r,pmnri8x,newlatin}{inferior,t1}{T1}{pmn0}{m}{it}{}
\installfontas{pmnr09e}{T1}{pmn0}{m}{sc}{}
\installfontas{pmnri09e}{T1}{pmn0}{m}{sl}{}
```

```
\installfontas{pmnri09e}{T1}{pmn0}{m}{si}{}
\installfont{pmns09e}{pmns8r,pmns8x,newlatin}{inferior,t1}{T1}{pmn0}{sb}{n}{}
\installfont{pmnsi09e}{pmnsi8r,pmnsi8x,newlatin}{inferior,t1}{T1}{pmn0}{sb}{it}{}
\installfontas{pmns09e}{T1}{pmn0}{sb}{sc}{}
\installfontas{pmnsi09e}{T1}{pmn0}{sb}{sl}{}
\installfontas{pmnsi09e}{T1}{pmn0}{sb}{si}{}
\installfont{pmnb09e}{pmnb8r,pmnb8x,newlatin}{inferior,t1}{T1}{pmn0}{b}{n}{}
\installfont{pmnbi09e}{pmnbi8r,pmnbi8x,newlatin}{inferior,t1}{T1}{pmn0}{b}{it}{}
\installfontas{pmnb09e}{T1}{pmn0}{b}{sc}{}
\installfontas{pmnbi09e}{T1}{pmn0}{b}{sl}{}
\installfontas{pmnbi09e}{T1}{pmn0}{b}{si}{}
\endinstallfonts
```

Since inferior figures are found in the expert fonts, our approach here does not differ from the one introduced in section 5.3. We also provide a font definition file for TS1 encoding:

```
\installfonts
\installfamily{TS1}{pmn0}{}
\installfontas{pmnr9c}{TS1}{pmn0}{m}{n}{}
\installfontas{pmnr9c}{TS1}{pmn0}{m}{sc}{}
\installfontas{pmnri9c}{TS1}{pmn0}{m}{it}{}
\installfontas{pmnri9c}{TS1}{pmn0}{m}{sl}{}
\installfontas{pmnri9c}{TS1}{pmn0}{m}{si}{}
\installfontas{pmns9c}{TS1}{pmn0}{sb}{n}{}
\installfontas{pmns9c}{TS1}{pmn0}{sb}{sc}{}
\installfontas{pmnsi9c}{TS1}{pmn0}{sb}{it}{}
\installfontas{pmnsi9c}{TS1}{pmn0}{sb}{sl}{}
\installfontas{pmnsi9c}{TS1}{pmn0}{sb}{si}{}
\installfontas{pmnb9c}{TS1}{pmn0}{b}{n}{}
\installfontas{pmnb9c}{TS1}{pmn0}{b}{sc}{}
\installfontas{pmnbi9c}{TS1}{pmn0}{b}{it}{}
\installfontas{pmnbi9c}{TS1}{pmn0}{b}{sl}{}
\installfontas{pmnbi9c}{TS1}{pmn0}{b}{si}{}
\endinstallfonts
```

The same holds true for superior figures:

```
\installfonts
\installfamily{T1}{pmn1}{}
\installfont{pmnr19e}{pmnr8r,pmnr8x,newlatin}{superior,t1}{T1}{pmn1}{m}{n}{}
\installfont{pmnri19e}{pmnri8r,pmnri8x,newlatin}{superior,t1}{T1}{pmn1}{m}{it}{}
\installfontas{pmnr19e}{T1}{pmn1}{m}{sc}{}
\installfontas{pmnri19e}{T1}{pmn1}{m}{sl}{}
\installfontas{pmnri19e}{T1}{pmn1}{m}{si}{}
\installfont{pmns19e}{pmns8r,pmns8x,newlatin}{superior,t1}{T1}{pmn1}{sb}{n}{}
\installfont{pmnsi19e}{pmnsi8r,pmnsi8x,newlatin}{superior,t1}{T1}{pmn1}{sb}{it}{}
\installfontas{pmns19e}{T1}{pmn1}{sb}{sc}{}
\installfontas{pmnsi19e}{T1}{pmn1}{sb}{sl}{}
\installfontas{pmnsi19e}{T1}{pmn1}{sb}{si}{}
\installfont{pmnb19e}{pmnb8r,pmnb8x,newlatin}{superior,t1}{T1}{pmn1}{b}{n}{}
\installfont{pmnbi19e}{pmnbi8r,pmnbi8x,newlatin}{superior,t1}{T1}{pmn1}{b}{it}{}
\installfontas{pmnb19e}{T1}{pmn1}{b}{sc}{}
\installfontas{pmnbi19e}{T1}{pmn1}{b}{sl}{}
\installfontas{pmnbi19e}{T1}{pmn1}{b}{si}{}
\endinstallfonts
```

And TS1 encoding:

```
\installfonts
\installfamily{TS1}{pmn1}{}
\installfontas{pmnr9c}{TS1}{pmn1}{m}{n}{}
\installfontas{pmnr9c}{TS1}{pmn1}{m}{sc}{}
\installfontas{pmnri9c}{TS1}{pmn1}{m}{it}{}
\installfontas{pmnri9c}{TS1}{pmn1}{m}{sl}{}
\installfontas{pmnri9c}{TS1}{pmn1}{m}{si}{}
\installfontas{pmns9c}{TS1}{pmn1}{sb}{n}{}
\installfontas{pmns9c}{TS1}{pmn1}{sb}{sc}{}
\installfontas{pmnsi9c}{TS1}{pmn1}{sb}{it}{}
\installfontas{pmnsi9c}{TS1}{pmn1}{sb}{sl}{}
\installfontas{pmnsi9c}{TS1}{pmn1}{sb}{si}{}
```

```
\installfontas{pmnb9c}{TS1}{pmn1}{b}{n}{}
\installfontas{pmnb9c}{TS1}{pmn1}{b}{sc}{}
\installfontas{pmnbi9c}{TS1}{pmn1}{b}{it}{}
\installfontas{pmnbi9c}{TS1}{pmn1}{b}{sl}{}
\installfontas{pmnbi9c}{TS1}{pmn1}{b}{si}{}
\endinstallfonts
```

In order to incorporate the italic swashes we will create an additional font family called `pmnw`.

```
\installfonts
\installfamily{T1}{pmnw}{}
\installfontas{pmnr9d}{T1}{pmnw}{m}{n}{}
\installfont{pmnriw9d}{pmnri8r,unsetcaps,pmnriw7a,pmnri8x,newlatin}{t1j}{T1}{pmnw}{m}{it}{}
\installfontas{pmnri9d}{T1}{pmnw}{m}{sl}{}
\installfontas{pmnrc9d}{T1}{pmnw}{m}{sc}{}
\installfontas{pmnric9d}{T1}{pmnw}{m}{si}{}
\installfontas{pmns9d}{T1}{pmnw}{sb}{n}{}
\installfont{pmnsiw9d}{pmnsi8r,unsetcaps,pmnsiw7a,pmnsi8x,newlatin}{t1j}{T1}{pmnw}{sb}{it}{}
\installfontas{pmnsi9d}{T1}{pmnw}{sb}{sl}{}
\installfontas{pmnsc9d}{T1}{pmnw}{sb}{sc}{}
\installfontas{pmnsic9d}{T1}{pmnw}{sb}{si}{}
\installfontas{pmnb9d}{T1}{pmnw}{b}{n}{}
\installfontas{pmnbi9d}{T1}{pmnw}{b}{it}{}
\installfontas{pmnbi9d}{T1}{pmnw}{b}{sl}{}
\installfontas{pmnb9d}{T1}{pmnw}{b}{sc}{}
\installfontas{pmnbi9d}{T1}{pmnw}{b}{si}{}
\endinstallfonts
```

We read the respective base font and clear the slots of the capital letters using the metric file `unsetcaps.mtx`. After that we add the corresponding swash font and finally the expert font as usual. We employ `t1j.etx` to get hanging figures by default. Since there are only two swash fonts, the remaining shapes of the `pmnw` family are taken from `pmnj`. Our self-made metric file `unsetcaps.mtx` uses the `\unsetglyph` command as follows:

```
\relax
```

```
\metrics
\unsetglyph{A}
\unsetglyph{B}
\unsetglyph{C}
...
\unsetglyph{X}
\unsetglyph{Y}
\unsetglyph{Z}
\endmetrics
```

We are merely clearing the slots of capital letters found in the English alphabet here. Capital letters with an accent are not removed because the Minion swash set does not provide accented swash capitals anyway. This means that all accented capital letters will be taken from the ordinary italic font.

```
\installfonts
\installfamily{TS1}{pmnw}{}
\installfontas{pmnr9c}{TS1}{pmnw}{m}{n}{}
\installfontas{pmnr9c}{TS1}{pmnw}{m}{sc}{}
\installfontas{pmnri9c}{TS1}{pmnw}{m}{it}{}
\installfontas{pmnri9c}{TS1}{pmnw}{m}{sl}{}
\installfontas{pmnri9c}{TS1}{pmnw}{m}{si}{}
\installfontas{pmns9c}{TS1}{pmnw}{sb}{n}{}
\installfontas{pmns9c}{TS1}{pmnw}{sb}{sc}{}
\installfontas{pmnsi9c}{TS1}{pmnw}{sb}{it}{}
\installfontas{pmnsi9c}{TS1}{pmnw}{sb}{sl}{}
\installfontas{pmnsi9c}{TS1}{pmnw}{sb}{si}{}
\installfontas{pmnb9c}{TS1}{pmnw}{b}{n}{}
\installfontas{pmnb9c}{TS1}{pmnw}{b}{sc}{}
\installfontas{pmnbi9c}{TS1}{pmnw}{b}{it}{}
\installfontas{pmnbi9c}{TS1}{pmnw}{b}{sl}{}
\installfontas{pmnbi9c}{TS1}{pmnw}{b}{si}{}
\endinstallfonts
```

After creating a font definition file for TS1 encoding, our setup for the text fonts is complete.

## 6.2 Installing text ornaments

The Minion expert package also includes a dedicated ornament font, `pmnrp.pfb`. As discussed before in section 4.4, we do not really need fontinst when installing symbol fonts. But since we require a map file and a font definition file as well, using fontinst is in fact easier than running `afm2tfm` and creating the auxiliary files manually:

```
\installfonts
\installfamily{U}{pmnp}{}
\installrawfont{pmnrp}{pmnrp}{txtfdmns,pmnrp mtxasetx}{U}{pmnp}{m}{n}{}
\endinstallfonts
```

And finally, we terminate our driver properly:

```
\endrecordtransforms
\bye
```

## 6.3 Extending the user interface

Before creating a style file for Minion, we will update `nfssext.sty` one more time to support its additional features. Support for swashes is easily added since the framework is already in place. Therefore, the first part of the file does not require any changes, we simply add support for swashes by defining `\swstyle` in a similar vein (40–42):

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{nfssext}[2003/03/14 v1.2 Experimental NFSS Extensions]
\newcommand*{\exfs@tempa}{}
\newcommand*{\exfs@tempb}{}
\newcommand*{\exfs@try@family}[2][]{%
  \let\exfs@tempa\relax
  \begingroup
    \fontfamily{#2}\try@load@fontshape
    \expandafter\ifx\csname\curr@fontshape\endcsname\relax
      \edef\exfs@tempa{#1}%
```

```
      \ifx\exfs@tempa\@empty
        \PackageWarning{nfssext}{%
          Font family '\f@encoding/#2' not available\MessageBreak
          Ignoring font switch}%
      \else
        \PackageInfo{nfssext}{%
          Font family '\f@encoding/#2' not available\MessageBreak
          Font family '\f@encoding/#1' tried instead}%
        \exfs@try@family{#1}%
      \fi
    \else
      \gdef\exfs@tempa{\fontfamily{#2}\selectfont}%
    \fi
  \endgroup
  \exfs@tempa}
\def\exfs@get@base#1#2#3#4\@nil{#1#2#3}
\DeclareRobustCommand{\lnstyle}{%
  \not@math@alphabet\lnstyle\relax
  \exfs@try@family[\expandafter\exfs@get@base\f@family\@nil]%
                  {\expandafter\exfs@get@base\f@family\@nil x}}
\DeclareRobustCommand{\osstyle}{%
  \not@math@alphabet\osstyle\relax
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil j}}
\DeclareRobustCommand{\instyle}{%
  \not@math@alphabet\instyle\relax
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil 0}}
\DeclareRobustCommand{\sustyle}{%
  \not@math@alphabet\sustyle\relax
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil 1}}
\DeclareRobustCommand{\swstyle}{%
  \not@math@alphabet\swstyle\relax
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil w}}
```

Adding thorough support for italic small caps is not quite as easy. The problem is that the creators of the NFSS apparently did not think of italic small caps when putting italics and small caps in the same category. Since both variants are on the shape axis of the NFSS they are mutually exclusive. While this will not keep us from using `\fontshape` to select italic small caps explicitly, nesting `\scshape` and `\itshape` does not have the desired effect. When nested, these macros simply override each other instead of switching to italic small caps. This problem is not as exotic as it may seem because italic small caps are hardly ever used explicitly. Typically, they come into play when small caps and italics are mixed on the same line. For example, think of a page header which is set in small caps, containing a highlighted word set in italics; or an italic section heading with an acronym set in small caps. To work around this problem, we will have to redefine a few NFSS macros. But first of all, we will add a macro for explicit switching to italic small caps.

```
\newcommand*{\sidefault}{si}
```

Note that the NFSS does not use fixed shape codes like `it` and `sc` for the italic and the small caps shape, but rather macros like `\itdefault` and `\scdefault`. We will handle italic small caps in a similar way by defining `\sidefault`, which defaults to `si`. Now let us define `\sishape` for explicit switching to italic small caps:

```
\DeclareRobustCommand{\sishape}{%
  \not@math@alphabet\sishape\relax
  \fontshape\sidefault\selectfont}
```

While we are able to typeset italic small caps by selecting them explicitly, macros like `\itshape` and `\scshape` will simply ignore the new shape. Hence we redefine these macros to make them take advantage of italic small caps transparently. In order to do so, we need a macro that will merge properties of the shape axis, thereby allowing us to treat italics and small caps as if they were not on the same axis:

```
\newcommand*{\exfs@merge@shape}[3]{%
  \edef\exfs@tempa{#1}%
  \edef\exfs@tempb{#2}%
  \ifx\f@shape\exfs@tempb
    \expandafter\ifx\csname\f@encoding/\f@family/\f@series/#3\endcsname\relax
    \else
      \edef\exfs@tempa{#3}%
```

```
    \fi
  \fi
  \fontshape{\exfs@tempa}\selectfont}
```

This macro will switch to the font shape given as the first argument unless the current shape is identical to the one indicated by the second argument. In this case it will switch to the shape designated by the third argument instead, provided that it is available for the current font family. With this macro at hand we redefine `\itshape`:

```
\DeclareRobustCommand{\itshape}{%
  \not@math@alphabet\itshape\mathit
  \exfs@merge@shape{\itdefault}{\scdefault}{\sidefault}}
```

Essentially, `\itshape` will switch to the font shape `it` unless the current shape is `sc`, in which case it will switch to `si` instead, provided that `si` is available. `\scshape` does it the other way around:

```
\DeclareRobustCommand{\scshape}{%
  \not@math@alphabet\scshape\relax
  \exfs@merge@shape{\scdefault}{\itdefault}{\sidefault}}
```

We also redefine `\upshape` to make it switch to `sc` instead of `n` if the current shape is `si`:

```
\DeclareRobustCommand{\upshape}{%
  \not@math@alphabet\upshape\relax
  \exfs@merge@shape{\updefault}{\sidefault}{\scdefault}}
```

If no italic small caps are available, all of these macros will behave like they did before, making them suitable for global use. While we are at it, we also define a new macro, `\dfshape`, that will reset the current shape to the default (`n` unless `\shapedefault` has been redefined) regardless of the current shape:

```
\DeclareRobustCommand{\dfshape}{%
  \not@math@alphabet\dfshape\relax
  \fontshape\shapedefault\selectfont}
```

Before we add text commands for our new font switches, there is still one thing left to do. The macro `\swstyle`, which we have defined above (40–42), will switch to the font family providing italic swashes (for example, `pmnw`).

However, it will not activate the italic shape. It would be convenient to have a macro which takes care of all of that. We first create an auxiliary macro holding the shape which provides the actual swashes:

```
\newcommand*{\swshapedefault}{\itdefault}
```

Then we create a macro which will call `\swstyle` and select the shape providing the italic swashes in one shot:

```
\DeclareRobustCommand{\swshape}{%
  \not@math@alphabet\swshape\relax
  \swstyle\fontshape\swshapedefault\selectfont}
```

Finally, we add text commands for our new font switches:

```
\DeclareTextFontCommand{\textln}{\lnstyle}
\DeclareTextFontCommand{\textos}{\osstyle}
\DeclareTextFontCommand{\textin}{\instyle}
\DeclareTextFontCommand{\textsu}{\sustyle}
\DeclareTextFontCommand{\textsi}{\sishape}
\DeclareTextFontCommand{\textdf}{\dfshape}
\DeclareTextFontCommand{\textsw}{\swshape}
```

As far as text is concerned, all features of Minion are readily available at this point. Using the ornaments would still require low-level commands, though.

## 6.4 A high-level interface for ornaments

Technically, ornament fonts are comparable to the euro fonts discussed in section 4.3. To typeset the first ornament of Minion, for example, we could use the following command:

```
{\usefont{U}{pmnp}{m}{n}\char 97}
```

As this is rather awkward and requires looking at the `afm` file to find out the encoding slot of each ornament, we will implement a higher-level solution. The problem is that ornament fonts do not conform to any encoding, so there is no standard we could rely on as far as the order of the glyphs in the font is concerned. We have to provide

this information explicitly in `minion.sty`. To facilitate this, we define the following macro:

```
\newcommand*{\DeclareTextOrnament}[7]{%
  \expandafter\def\csname#1@orn\@roman#2\endcsname{#3/#4/#5/#6/#7}}
```

To declare the first ornament of Minion, this macro would be employed as follows:

```
\DeclareTextOrnament{pmn}{1}{U}{pmnp}{m}{n}{97}
```

We use the first three letters of the font family name as an identifier (`pmn`) and assign a number (1 in this case) to the ornament defined by the remaining arguments. These arguments form a complete font declaration with a syntax similar to that of the NFSS macro `\DeclareFontShape`. The last argument is the encoding slot of the ornament (97 here) as given in the `afm` file. You might wonder why we use a complete font declaration here. Since all ornaments are located in the same font, using the same encoding, series, and shape, this seems to be redundant. In this case, this is actually true. The problem is that ornaments are not necessarily provided in dedicated fonts. Adobe Garamond, for example, comes with ornaments which are included in some of the alternate text fonts so we use a complete declaration for maximum flexibility. Internally, the ornaments are saved in a format modeled after the way the NFSS handles font shapes. When typesetting an ornament later, we need a macro to parse this font declaration:

```
\begingroup
  \catcode`\/=12
  \gdef\exfs@split@orndef#1/#2/#3/#4/#5\@nil{%
    \def\f@encoding{#1}%
    \def\f@family{#2}%
    \def\f@series{#3}%
    \def\f@shape{#4}%
    \def\exfs@tempa{#5}}
\endgroup
```

Since we use the base of the font family name as an identifier, we also need a macro that expands to the first three letters of the current font family:

```
\def\exfs@base@family{\expandafter\exfs@get@base\f@family\@nil}
```

Now we can finally implement a user macro that actually typesets the ornament. We will simply call it `\ornament`:

```
92 \DeclareRobustCommand{\ornament}[1]{%
93   \expandafter\ifx\csname\exfs@base@family @orn\@roman#1\endcsname\relax
94     \PackageWarning{nfssext}{%
95       Ornament #1 undefined for font family '\exfs@base@family'\MessageBreak
96       Setting debug mark}%
97     \rule{1ex}{1ex}%
98   \else
99     \begingroup
100      \edef\exfs@tempb{\csname\exfs@base@family @orn\@roman#1\endcsname}%
101      \expandafter\expandafter\expandafter\exfs@split@orndef
102        \expandafter\string\exfs@tempb\@nil
103      \selectfont\char\exfs@tempa
104    \endgroup
105  \fi}
106 \endinput
```

First of all, we check if the desired ornament has been declared (93) and issue a warning if not (94–96). We also typeset a mark (97) to facilitate debugging in this case. If it has been declared, we expand and parse the declaration (100–102), switch fonts, and typeset the ornament (103). We use grouping to keep the font change local.

## 6.5 An extended style file

The style file for Minion is similar to the ones suggested in section 3.2 and 5.4. The only difference is the declaration of the text ornaments. This is the first part of `minion.sty`:

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{minion}[2003/03/25 v1.0 Adobe Minion]
3 \RequirePackage[T1]{fontenc}
4 \RequirePackage{textcomp}
5 \RequirePackage{nfssext}
6 \DeclareOption{oldstyle}{\renewcommand*{\rmdefault}{pmnj}}
```

```
\DeclareOption{lining}{\renewcommand*{\rmdefault}{pmnx}}
\ExecuteOptions{oldstyle}
\ProcessOptions*
```

When declaring the text ornaments, we take the encoding slot numbers from the respective `afm` file:

```
C  97 ; WX  885 ; N ornament1 ; B 50  -65  835 744 ;
C  98 ; WX 1036 ; N ornament2 ; B 50    4  986 672 ;
C  99 ; WX 1066 ; N ornament3 ; B 50 -106 1016 745 ;
C 100 ; WX  866 ; N ornament4 ; B 50   98  816 534 ;
C 101 ; WX  390 ; N ornament5 ; B 50   86  341 550 ;
```

We add a declaration for each ornament:

```
\DeclareTextOrnament{pmn}{1}{U}{pmnp}{m}{n}{97}
\DeclareTextOrnament{pmn}{2}{U}{pmnp}{m}{n}{98}
\DeclareTextOrnament{pmn}{3}{U}{pmnp}{m}{n}{99}
\DeclareTextOrnament{pmn}{4}{U}{pmnp}{m}{n}{100}
\DeclareTextOrnament{pmn}{5}{U}{pmnp}{m}{n}{101}
\DeclareTextOrnament{pmn}{6}{U}{pmnp}{m}{n}{102}
\DeclareTextOrnament{pmn}{7}{U}{pmnp}{m}{n}{103}
\DeclareTextOrnament{pmn}{8}{U}{pmnp}{m}{n}{104}
\DeclareTextOrnament{pmn}{9}{U}{pmnp}{m}{n}{105}
\DeclareTextOrnament{pmn}{10}{U}{pmnp}{m}{n}{106}
\DeclareTextOrnament{pmn}{11}{U}{pmnp}{m}{n}{107}
\DeclareTextOrnament{pmn}{12}{U}{pmnp}{m}{n}{108}
\DeclareTextOrnament{pmn}{13}{U}{pmnp}{m}{n}{109}
\DeclareTextOrnament{pmn}{14}{U}{pmnp}{m}{n}{110}
\DeclareTextOrnament{pmn}{15}{U}{pmnp}{m}{n}{111}
\DeclareTextOrnament{pmn}{16}{U}{pmnp}{m}{n}{112}
\DeclareTextOrnament{pmn}{17}{U}{pmnp}{m}{n}{113}
\DeclareTextOrnament{pmn}{18}{U}{pmnp}{m}{n}{114}
\DeclareTextOrnament{pmn}{19}{U}{pmnp}{m}{n}{115}
\DeclareTextOrnament{pmn}{20}{U}{pmnp}{m}{n}{116}
\DeclareTextOrnament{pmn}{21}{U}{pmnp}{m}{n}{117}
```

```
\DeclareTextOrnament{pmn}{22}{U}{pmnp}{m}{n}{118}
\DeclareTextOrnament{pmn}{23}{U}{pmnp}{m}{n}{119}
\endinput
```

As mentioned before, Adobe Garamond features ornaments in the alternate text fonts, requiring a complete font declaration. In this case, the definitions would look as follows:

```
\DeclareTextOrnament{pad}{1}{U}{pada}{m}{n}{49}
\DeclareTextOrnament{pad}{2}{U}{pada}{m}{n}{50}
\DeclareTextOrnament{pad}{3}{U}{pada}{m}{it}{49}
```

Note that the ornament macro is deliberately designed to be sensitive to the active font family. When using Minion, `\ornament{1}` will typeset the first ornament of Minion. When using Adobe Garamond, the same command sequence will typeset an ornament taken from Adobe Garamond. If you would like to use the ornaments in a font independent manner, you can always go back to lower-level commands which merely depend on a font definition file (`upmnp.fd` and `upada.fd` here) for the respective ornament font:

```
{\usefont{U}{pmnp}{m}{n}\char 97}
{\usefont{U}{pada}{m}{n}\char 49}
```

# Tutorial 7

# Creating map files

With the advent of fontinst 1.9, the tedious and error-prone task of creating map files manually should finally be a thing of the past. As outlined in section 1.5, fontinst can now do most of the work for us. Generally speaking, DVI and PDF drivers do not share a common map file format. For dvips, the syntax of map files is explained in detail in the dvips manual.[1] For pdfTeX, it is is explained in the pdfTeX manual, and for xdvi in the documentation that comes with the source distribution of xdvi. As of this writing (December 2004), map file support in fontinst is restricted to dvips and dvipdfm. Unless the map files contain unusual PostScript instructions, however, xdvi and pdfTeX are capable of using dvips' files, hence fontinst's map file generator for dvips covers more than just dvips.

Since map files are a crucial part in the process of transforming a LaTeX source file into a screen image or a printed document, understanding their format is still a must when debugging a malfunctioning TeX installation or dealing with very unusual installation scenarios. While the other tutorials in this guide rely on fontinst's map file generator, this tutorial will focus on building a map file manually. The format presented here is a subset of the syntax supported by dvips which will also work with pdfTeX and xdvi.

[1] http://www.radicaleye.com/dvipsman/

## 7.1 The syntax of map files

In tutorials 1 and 2 we have dealt with a typical installation scenario, the base package of Adobe Sabon. The map file was generated by fontinst's built-in map file writer. In the following, we will go over all the steps required to build this map file manually. The Sabon base package provides four fonts: regular, regular italic, bold, and bold italic. Both the `\latinfamily` macro employed in tutorial 1 and the lower-level fontinst file introduced in tutorial 2 will create slanted versions of the upright regular and bold fonts in addition to that. Therefore, we need to provide mapping records for a total of six fonts. Let us take a look at the complete map file before going over the individual lines:

```
psbr8r  Sabon-Roman      <8r.enc <psbr8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbri8r Sabon-Italic     <8r.enc <psbri8a.pfb "TeXBase1Encoding ReEncodeFont"
psbb8r  Sabon-Bold       <8r.enc <psbb8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbbi8r Sabon-BoldItalic <8r.enc <psbbi8a.pfb "TeXBase1Encoding ReEncodeFont"
psbro8r Sabon-Roman      <8r.enc <psbr8a.pfb  "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
psbbo8r Sabon-Bold       <8r.enc <psbb8a.pfb  "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
```

Map files for dvips consist of up to three segments. The first segment is the name of the raw TeX font without any file suffix:

```
psbr8r Sabon-Roman <8r.enc <psbr8a.pfb "TeXBase1Encoding ReEncodeFont"
```

Note that both fontinst's built-in `\latinfamily` macro and all lower-level installation recipes suggested in this guide share a fundamental step: all regular text fonts are reencoded from Adobe Standard encoding (Fontname code `8a`) to TeX Base 1 (`8r`) before any virtual fonts are built. Therefore, the name of the raw TeX font corresponds to the one of the `pfb` file with encoding code `8r` instead of `8a`:

```
psbr8r  Sabon-Roman      <8r.enc <psbr8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbri8r Sabon-Italic     <8r.enc <psbri8a.pfb "TeXBase1Encoding ReEncodeFont"
psbb8r  Sabon-Bold       <8r.enc <psbb8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbbi8r Sabon-BoldItalic <8r.enc <psbbi8a.pfb "TeXBase1Encoding ReEncodeFont"
```

The second segment is the PostScript name of the font:

```
psbr8r Sabon-Roman <8r.enc <psbr8a.pfb "TeXBase1Encoding ReEncodeFont"
```

Do not try to guess the right name or copy it from some map file you found somewhere on the web some time ago. If your font is included in one of the foundry-specific lists of the Fontname scheme, the PostScript name is given in the second column of the respective table. If it is not or if you are in doubt, the PostScript name should be taken from the header of the `afm` file for every font. Here are a few lines from `psbr8a.afm`:

```
StartFontMetrics 2.0
Comment Copyright (c) 1989 Adobe Systems Incorporated. All Rights Reserved.
Comment Creation Date:Fri Mar 10 16:47:51 PST 1989
FontName Sabon-Roman
FullName 12 Sabon* Roman 05232
FamilyName Sabon
EncodingScheme AdobeStandardEncoding
```

The relevant part is the line starting with 'FontName'. The PostScript name of this font is 'Sabon-Roman'. For each font, we copy this name verbatim to the map file:

```
psbr8r  Sabon-Roman      <8r.enc <psbr8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbri8r Sabon-Italic     <8r.enc <psbri8a.pfb "TeXBase1Encoding ReEncodeFont"
psbb8r  Sabon-Bold       <8r.enc <psbb8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbbi8r Sabon-BoldItalic <8r.enc <psbbi8a.pfb "TeXBase1Encoding ReEncodeFont"
```

The third segment of our map file consists of a list of options. Options are either files to be embedded in the PostScript code or PostScript instructions enclosed in double quotes:

```
psbr8r Sabon-Roman <8r.enc <psbr8a.pfb "TeXBase1Encoding ReEncodeFont"
```

First of all, we need to include a list of files that dvips will embed in the PostScript file. In this case, we need the PostScript encoding vector `8r.enc` for TeX Base 1 encoding and the `pfb` files, since we want the fonts to be embedded in the PostScript file:

```
psbr8r  Sabon-Roman      <8r.enc <psbr8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbri8r Sabon-Italic     <8r.enc <psbri8a.pfb "TeXBase1Encoding ReEncodeFont"
psbb8r  Sabon-Bold       <8r.enc <psbb8a.pfb  "TeXBase1Encoding ReEncodeFont"
```

```
psbbi8r Sabon-BoldItalic <8r.enc <psbbi8a.pfb "TeXBase1Encoding ReEncodeFont"
```

Our last option is a PostScript reencoding instruction. As mentioned above, the `\latinfamily` macro and our lower-level fontinst driver files reencode all fonts from Adobe Standard encoding to TeX Base 1 when creating metric files for TeX. This affects the metrics only, which are defined in the `tfm` files generated by fontinst, while the glyph outlines as defined in the `pfb` file still use the font's native encoding. Therefore, we add a reencoding directive to the map file that will instruct all applications dealing with the actual glyph outlines to reencode them accordingly:

```
psbr8r Sabon-Roman <8r.enc <psbr8a.pfb "TeXBase1Encoding ReEncodeFont"
```

Where do we get the string `TeXBase1Encoding` from? It is the PostScript name of the encoding vector, as defined in `8r.enc`. If you open the file `8r.enc` in a text editor, you will see a header consisting of comments concerning TeX Base 1 encoding. After that, the definition of the encoding vector begins as follows:

```
% Comments
/TeXBase1Encoding [
 /.notdef /dotaccent /fi /fl
 /fraction /hungarumlaut /Lslash /lslash
```

Note that the T1 and TS1 encodings are used for the virtual fonts only, they are what TeX will work with. A PostScript file created by dvips, however, does not contain any virtual fonts. They will have been resolved into the raw fonts they are based on by dvips. The raw fonts used to build virtual ones were reencoded to TeX Base 1 encoding during the installation. But this reencoding step affects the font metrics only while the `pfb` files embedded in the PostScript code still use Adobe Standard as their native encoding. Therefore every application reading the final file has to repeat the reencoding step for the font outlines before rendering the fonts. This is what the PostScript reencoding instruction is all about. Since we cannot expect every application to know about TeX Base 1 encoding, we embed the respective encoding vector (`8r.enc`) along with the fonts. Compare the first `\transformfont` command in the verbose fontinst file discussed in section 2.1 to the corresponding line of the map file:

```
\transformfont{psbr8r}            {\reencodefont{8r}                    {\fromafm{psbr8a}}}
              psbr8r Sabon-Roman "TeXBase1Encoding ReEncodeFont" <8r.enc         <psbr8a.pfb
```

In addition to the base fonts provided by the Sabon package, we need to tell dvips about any slanted versions created by fontinst. We copy the lines for Sabon-Roman and Sabon-Bold and insert `o`, the Fontname code for slanted fonts, after the weight code of the TeX font name. Note that the name of the `pfb` file does not change:

```
psbro8r Sabon-Roman <8r.enc <psbr8a.pfb "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
psbbo8r Sabon-Bold  <8r.enc <psbb8a.pfb "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
```

We also add a 'SlantFont' instruction. By default, `\latinfamily` uses a slant factor of 0.167 when creating the modified metrics:

```
psbro8r Sabon-Roman <8r.enc <psbr8a.pfb "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
psbbo8r Sabon-Bold  <8r.enc <psbb8a.pfb "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
```

Note that the font files embedded in the PostScript file are not slanted, they are upright. Fontinst performs the slanting for the font metrics only, it does not touch the font outlines at all. The slanting of the glyph outlines will be performed by a PostScript printer or an interpreter like Ghostscript. After resolving the virtual fonts, all that dvips does as far as the raw fonts are concerned is reading the files listed in the map file and embedding them along with the slanting instruction. The transformation of the glyph outlines takes place when the PostScript code is rendered on screen or on paper. Both 'ReEncodeFont' and 'SlantFont' are instructions for the application performing the final rendering. The value of the slanting instruction has to correspond to the slant factor used in the fontinst file. As mentioned above, fontinst's representation of the slant factor is slightly different. The value used in the map file is a real number corresponding to fontinst's (integer) slant factor divided by 1000. Its precision is therefore limited to three decimal places. Going back to the fontinst file discussed in section 2.1 once again, compare a line of the map file to the corresponding line of the fontinst file:

```
\transformfont{psbro8r}{\slantfont{167}\reencodefont{8r}              {\fromafm{psbr8a}}}
psbro8r Sabon-Roman    "0.167 SlantFont TeXBase1Encoding ReEncodeFont" <8r.enc <psbr8a.pfb
```

Essentially, think of map files as a way of recording all encoding and shape modifications applied to the font metrics during the installation, so that they can be repeated for the font outlines when the final PostScript file is displayed or printed. This information is required for the raw fonts only because all the information concerning the virtual fonts is contained in the virtual font files. When using DVI or PDF as the final output format, the division of labor

between the various tools involved differs since pdfTeX combines the roles of TeX and dvips, while DVI viewers deal with both the virtual fonts and the rendering of the font outlines on screen. The principle, however, remains the same. Therefore pdfTeX and xdvi require map files as well.

Technically, so-called SC & OSF fonts like those discussed in tutorial 3 are regular text fonts as well. While they are a little bit more difficult to install than other text fonts, their map file records do not differ from other fonts at all. Here is a map file for the extended Sabon set in tutorial 3:

```
psbr8r   Sabon-Roman         <8r.enc <psbr8a.pfb   "TeXBase1Encoding ReEncodeFont"
psbri8r  Sabon-Italic        <8r.enc <psbri8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbb8r   Sabon-Bold          <8r.enc <psbb8a.pfb   "TeXBase1Encoding ReEncodeFont"
psbbi8r  Sabon-BoldItalic    <8r.enc <psbbi8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbrc8r  Sabon-RomanSC       <8r.enc <psbrc8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbrij8r Sabon-ItalicOsF     <8r.enc <psbrij8a.pfb "TeXBase1Encoding ReEncodeFont"
psbbj8r  Sabon-BoldOsF       <8r.enc <psbbj8a.pfb  "TeXBase1Encoding ReEncodeFont"
psbbij8r Sabon-BoldItalicOsF <8r.enc <psbbij8a.pfb "TeXBase1Encoding ReEncodeFont"
psbro8r  Sabon-Roman   <8r.enc <psbr8a.pfb  "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
psbbo8r  Sabon-Bold    <8r.enc <psbb8a.pfb  "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
psbrco8r Sabon-RomanSC <8r.enc <psbrc8a.pfb "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
psbboj8r Sabon-BoldOsF <8r.enc <psbbj8a.pfb "TeXBase1Encoding ReEncodeFont 0.167 SlantFont"
```

Note that there is a slanted version of the small caps font even though we did not make the slanted small caps available under LaTeX in tutorial 3. We need it to provide matching figures for the slanted shape of the `psbj` family, which has to take regular-weight hanging figures from the small caps font.

## 7.2 Expert and symbol fonts

Map file records for expert and symbol fonts are simpler than those of text fonts. There is no 'ReEncodeFont' instruction and no encoding vector because such fonts are usually not reencoded. A map file for the Euro fonts discussed in section 4.4, for example, would include the following lines:

```
zpeur   EuroSerif-Regular    <zpeur.pfb
zpeuri  EuroSerif-Italic     <zpeuri.pfb
```

```
zpeub   EuroSerif-Bold       <zpeub.pfb
zpeubi  EuroSerif-BoldItalic <zpeubi.pfb
```

We still need to add 'SlantFont' instructions for the slanted shapes:

```
zpeuro  EuroSerif-Regular    <zpeur.pfb  "0.167 SlantFont"
zpeubo  EuroSerif-Bold       <zpeub.pfb  "0.167 SlantFont"
```

The Janson expert fonts discussed in tutorial 5 would be mapped in a similar way:

```
mjnr8x  JansonExpertMT            <mjnr8x.pfb
mjnri8x JansonExpertMT-Italic     <mjnri8x.pfb
mjnb8x  JansonExpertMT-Bold       <mjnb8x.pfb
mjnbi8x JansonExpertMT-BoldItalic <mjnbi8x.pfb
mjnro8x JansonExpertMT            <mjnr8x.pfb  "0.167 SlantFont"
mjnbo8x JansonExpertMT-Bold       <mjnb8x.pfb  "0.167 SlantFont"
```

In addition to expert fonts, the Minion fonts discussed in tutorial 6 include swash and symbol fonts which are also mapped like that because they were not reencoded:

```
pmnriw7a Minion-SwashItalic         <pmnriw7a.pfb
pmnsiw7a Minion-SwashSemiboldItalic <pmnsiw7a.pfb
pmnrp    Minion-Ornaments           <pmnrp.pfb
```

# Appendix

# Appendix A

# Code tables

The tables on the following pages are intended to give an idea of how the codes of the Fontname scheme relate to those used by LaTeX's font selection scheme (NFSS). The Fontname codes are what we use when renaming the font files during the installation while the NFSS codes are what we need when selecting a certain font under LaTeX later. Sticking to the NFSS codes listed below is not a technical requirement for a functional font installation. When using the `\latinfamily` macro, fontinst will indeed use these NFSS codes. When employing low-level fontinst commands, however, the NFSS font declaration is controlled by the last five arguments of the `\installfont` command. In theory, we could use an arbitrary code and the NFSS would handle that just fine. It is still highly recommended to stick to these codes to avoid confusion and incompatibility. Two dashes in one of the table cells indicate that there is no customary code for this font property in the respective scheme whereas a blank cell means that the code is omitted. Properties which are not catered for by the `\latinfamily` macro are marked with an asterisk in the last column.

Please note that Fontname codes and NFSS codes cannot be mapped on a one-to-one basis in all cases since the two schemes are rather different in concept. Weights and widths, which are treated separately by the Fontname

scheme, need to be concatenated and handled as a 'series' when using the NFSS since the latter does not have independent categories ('axes') for weight and width. The 'variant' category of the Fontname scheme on the other hand, which embraces several different properties including shapes like italics as well as special glyph sets such as small caps or alternative figures, does not correspond to a single NFSS axis. Some variants, like italics and small caps for example, are mapped to the 'shape' axis of the NFSS. Others, such as alternative figures, are handled in completely different ways. Table A.3 lists variants corresponding to the most common NFSS shapes only. When looking at the documentation of the Fontname scheme, you will find a lot more variant codes not mentioned here. Although they are used for file naming, they do not, or, at least do not necessarily correspond to a customary NFSS shape. Hanging, inferior, and superior numbers (Fontname codes `j`, `0`, and `1`), for example, are treated as 'variants' by the Fontname scheme but they are usually implemented as independent font families on the level of the NFSS. For the encodings listed in table A.4 the situation is similar. For example, a virtual font in T1 encoding featuring expert glyphs is indicated by adding `9e` to the file name. However, on the level of the NFSS the encoding code is `T1` for all T1 encoded fonts and the fact that the font provides expert glyphs is expressed by adding the letter `x` to the font family name.

| WEIGHT | FONTNAME CODE | NFSS SERIES |
|---|---|---|
| ultra light, thin, hairline | a | ul* |
| extra light | j | el* |
| light | l | l |
| book | k | m |
| regular | r | m |
| medium | m | mb |
| demibold | d | db |
| semibold | s | sb |
| bold | b | b |
| heavy | h | eb |
| black | c | eb |
| extra bold, extra black | x | eb |
| ultra bold, ultra black | u | ub |
| poster | p | --* |

Table A.1: Codes for font weights

| WIDTH | FONTNAME CODE | NFSS SERIES |
|---|---|---|
| ultra compressed | u | uc* |
| ultra condensed | o | uc* |
| extra compressed, extra condensed | q | ec* |
| compressed | p | c* |
| condensed | c | c* |
| narrow | n | c |
| regular | | |
| extended | x | x* |
| expanded | e | x* |
| extra expanded | v | ex* |
| ultra expanded | -- | ux* |
| wide | w | --* |

Table A.2: Codes for font widths

| VARIANT | FONTNAME CODE | NFSS SHAPE |
|---|---|---|
| normal, upright, roman | | n |
| italic | i | it |
| oblique, slanted | o | sl |
| small caps | c | sc |
| italic small caps | ic | si* |
| upright italic | -- | ui* |
| outline | l | ol* |

Table A.3: Codes for font variants

| ENCODING | FONTNAME CODE | NFSS ENCODING |
|---|---|---|
| Adobe Standard | 8a | 8a |
| Expert | 8x | 8x |
| TeX Base 1 | 8r | 8r |
| TeX Text | 7t | OT1 |
| TeX Tex with expert set | 9t | OT1 |
| TeX Text with expert set and OSF | 9o | OT1 |
| Cork | 8t | T1 |
| Cork with expert set | 9e | T1 |
| Cork with expert set and OSF | 9d | T1 |
| Text Companion | 8c | TS1 |
| Text Companion with expert set | 9c | TS1 |

Table A.4: Codes for font encodings

# Appendix B

# Text companion symbols

The symbol lists in this appendix give an overview of the symbols defined in TS1 encoding and the corresponding text commands provided by the `textcomp` package. Please note that regular PostScript fonts will not provide the full range of text companion symbols. Some parts of the TS1 encoding are rather exotic and specific to fonts which were created with TeX and TS1 encoding in mind.

## B.1 Symbols in text fonts

The text companion symbols in the following list will usually be available when installing common PostScript fonts based on Adobe Standard encoding. Some symbols are marked with an asterisk. They are not found as such in PostScript text fonts but fontinst will try to fake them, using glyphs which are almost always included in such fonts. In order to create a centered asterisk, for example, fontinst uses the regular asterisk and tries to center it vertically. The quality of the result depends on the particular glyph and the given font. For example, creating a composite centigrade symbol by combining the degree sign with the capital letter 'C' should always work. On the

other hand, faking something like double brackets depends on certain assumptions concerning the dimensions of the brackets provided by the font. These assumptions may or may not apply to a given font. The euro symbol is a special case since it is not included in Adobe Standard encoding. If a font provides a native euro symbol, however, it will usually be included in the base fonts. If there is no euro symbol at all, fontinst tries to fake it by overstriking the capital letter 'C' with two horizontal bars as a last resort. In this case, the quality of the result will vary significantly from typeface to typeface. Please refer to tutorial 4 for a more detailed discussion of issues related to the euro symbol.

| | | | |
|---|---|---|---|
| \textacutedbl | ˝ | \textasciiacute | ´ |
| \textasciibreve | ˘ | \textasciicaron | ˇ |
| \textasciidieresis | ¨ | \textasciigrave | ` |
| \textasciimacron | ¯ | \textasteriskcentered* | * |
| \textbardbl* | ‖ | \textbrokenbar | ¦ |
| \textbullet | • | \textcelsius* | ℃ |
| \textcent | ¢ | \textcopyright | © |
| \textcurrency | ¤ | \textdaggerdbl | ‡ |
| \textdagger | † | \textdegree | ° |
| \textdiv | ÷ | \textdollar | $ |
| \texteuro* | € | \textflorin | f |
| \textfractionsolidus | ⁄ | \textgravedbl | ‟ |
| \textinterrobangdown* | ⸘ | \textinterrobang* | ‽ |
| \textlbrackdbl* | ⟦ | \textlnot | ¬ |
| \textminus | − | \textmu | µ |
| \textonehalf | ½ | \textonequarter | ¼ |
| \textonesuperior | ¹ | \textordfeminine | ª |
| \textordmasculine | º | \textparagraph | ¶ |
| \textperiodcentered | · | \textperthousand | ‰ |
| \textpm | ± | \textquotesingle | ' |
| \textrbrackdbl* | ⟧ | \textregistered | ® |
| \textsection | § | \textsterling | £ |

| | | | |
|---|---|---|---|
| \textthreequartersemdash* | — | \textthreequarters | ¾ |
| \textthreesuperior | ³ | \texttildelow* | ~ |
| \texttimes | × | \texttrademark | ™ |
| \texttwelveudash* | – | \texttwosuperior | ² |
| \textyen | ¥ | | |

## B.2 Symbols specific to expert fonts

When installing PostScript fonts complemented by an expert package the text companion symbols in this list will usually be available in addition to the ones mentioned above. Apart from the dash these glyphs cannot be faked.

| | | | |
|---|---|---|---|
| \textzerooldstyle | 0 | \textoneoldstyle | 1 |
| \texttwooldstyle | 2 | \textthreeoldstyle | 3 |
| \textfouroldstyle | 4 | \textfiveoldstyle | 5 |
| \textsixoldstyle | 6 | \textsevenoldstyle | 7 |
| \texteightoldstyle | 8 | \textnineoldstyle | 9 |
| \textcentoldstyle | ¢ | \textcolonmonetary | ₡ |
| \textdollaroldstyle | $ | \textthreequartersemdash | — |

## B.3 Symbols specific to TeX fonts

The text companion symbols in this list mostly belong to the more exotic parts of the TS1 encoding. They are only provided by fonts which were created with TeX and TS1 encoding in mind, for example European Computer Modern and Latin Modern.

| | | | |
|---|---|---|---|
| \textbaht | ฿ | \textbigcircle | ◯ |
| \textblank | ␢ | \textborn | ⋆ |
| \textcircledP | Ⓟ | \textcopyleft | 🄯 |
| \textdblhyphenchar | = | \textdblhyphen | = |

| | | | |
|---|---|---|---|
| \textdied | † | \textdiscount | ⁒ |
| \textdivorced | o\|o | \textdong | ₫ |
| \textdownarrow | ↓ | \textestimated | ℮ |
| \textguarani | ₲ | \textlangle | ⟨ |
| \textleaf | 🍃 | \textleftarrow | ← |
| \textlira | ₤ | \textlquill | ⁅ |
| \textmarried | ⚭ | \textmho | ℧ |
| \textmusicalnote | ♪ | \textnaira | ₦ |
| \textnumero | № | \textohm | Ω |
| \textopenbullet | ○ | \textpertenthousand | ‱ |
| \textpeso | ₱ | \textpilcrow | ¶ |
| \textquotestraightbase | ‚ | \textquotestraightdblbase | „ |
| \textrangle | ⟩ | \textrecipe | ℞ |
| \textreferencemark | ※ | \textrightarrow | → |
| \textrquill | ⁆ | \textservicemark | ℠ |
| \textsurd | √ | \textuparrow | ↑ |
| \textwon | ₩ | | |

# Appendix C

# The GNU Free Documentation License

**Version 1.2, November 2002**



## C.0 Preamble

The purpose of this license is to make a manual, textbook, or other functional and useful document 'free' in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this license preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This license is a kind of 'copyleft', which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free

software.

We have designed this license in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this license is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this license principally for works whose purpose is instruction or reference.

## C.1 Applicability and definitions

This license applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this license. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The *document*, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as *you*. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A *modified version* of the document means any work containing the document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A *secondary section* is a named appendix or a front-matter section of the document that deals exclusively with the relationship of the publishers or authors of the document to the document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the document is in part a textbook of mathematics, a secondary section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The *invariant sections* are certain secondary sections whose titles are designated, as being those of invariant sections, in the notice that says that the document is released under this license. If a section does not fit the above definition of secondary then it is not allowed to be designated as invariant. The document may contain zero invariant sections. If the document does not identify any invariant sections then there are none.

The *cover texts* are certain short passages of text that are listed, as front-cover texts or back-cover texts, in the notice that says that the document is released under this license. A front-cover text may be at most five words,

and a back-cover text may be at most 25 words.

A *transparent* copy of the document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not transparent. An image format is not transparent if used for any substantial amount of text. A copy that is not 'transparent' is called 'opaque'.

Examples of suitable formats for transparent copies include plain Ascii without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The *title page* means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this license requires to appear in the title page. For works in formats which do not have any title page as such, 'title page' means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section *entitled* XYZ means a named subunit of the document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as 'Acknowledgements', 'Dedications', 'Endorsements', or 'History'.) To "preserve the title" of such a section when you modify the document means that it remains a section "entitled XYZ" according to this definition.

The document may include warranty disclaimers next to the notice which states that this license applies to the document. These warranty disclaimers are considered to be included by reference in this license, but only as regards disclaiming warranties: any other implication that these warranty disclaimers may have is void and has no effect on the meaning of this license.

## C.2 Verbatim copying

You may copy and distribute the document in any medium, either commercially or noncommercially, provided that this license, the copyright notices, and the license notice saying this license applies to the document are reproduced in all copies, and that you add no other conditions whatsoever to those of this license. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section C.3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## C.3 Copying in quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the document, numbering more than 100, and the document's license notice requires cover texts, you must enclose the copies in covers that carry, clearly and legibly, all these cover texts: front-cover texts on the front cover, and back-cover texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute opaque copies of the document numbering more than 100, you must either include a machine-readable transparent copy along with each opaque copy, or state in or with each opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete transparent copy of the document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of opaque copies in quantity, to ensure that this transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the document.

## C.4 Modifications

You may copy and distribute a modified version of the document under the conditions of sections C.2 and C.3 above, provided that you release the modified version under precisely this license, with the modified version filling the role of the document, thus licensing distribution and modification of the modified version to whoever possesses a copy of it. In addition, you must do these things in the modified version:

A. Use in the title page (and on the covers, if any) a title distinct from that of the document, and from those of previous versions (which should, if there were any, be listed in the history section of the document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the title page, as authors, one or more persons or entities responsible for authorship of the modifications in the modified version, together with at least five of the principal authors of the document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the title page the name of the publisher of the modified version, as the publisher.

D. Preserve all the copyright notices of the document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the modified version under the terms of this license.

G. Preserve in that license notice the full lists of invariant sections and required cover texts given in the document's license notice.

H. Include an unaltered copy of this license.

I. Preserve the section entitled 'History', preserve its title, and add to it an item stating at least the title, year, new authors, and publisher of the modified version as given on the title page. If there is no section entitled 'History' in the document, create one stating the title, year, authors, and publisher of the document as given on its title page, then add an item describing the modified version as stated in the previous sentence.

J. Preserve the network location, if any, given in the document for public access to a transparent copy of the document, and likewise the network locations given in the document for previous versions it was based on. These may be placed in the 'History' section. You may omit a network location for a work that was published at least four years before the document itself, or if the original publisher of the version it refers to gives permission.

K. For any section entitled 'Acknowledgements' or 'Dedications', preserve the title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the invariant sections of the document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled 'Endorsements'. Such a section may not be included in the modified version.

N. Do not retitle any existing section to be entitled 'Endorsements' or to conflict in title with any invariant section.

O. Preserve any warranty disclaimers.

If the modified version includes new front-matter sections or appendices that qualify as secondary sections and contain no material copied from the document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of invariant sections in the modified version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled 'Endorsements', provided it contains nothing but endorsements of your modified version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a front-cover text, and a passage of up to 25 words as a back-cover text, to the end of the list of cover texts in the modified version. Only one passage of front-cover text and one of back-cover text may be added by (or through arrangements made by) any one entity. If the document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the document do not by this license give permission to use their names for publicity for or to assert or imply endorsement of any modified version.

## C.5 Combining documents

You may combine the document with other documents released under this license, under the terms defined in section C.4 above for modified versions, provided that you include in the combination all of the invariant sections of all of the original documents, unmodified, and list them all as invariant sections of your combined work in its license notice, and that you preserve all their warranty disclaimers.

The combined work need only contain one copy of this license, and multiple identical invariant sections may be replaced with a single copy. If there are multiple invariant sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of invariant sections in the license notice of the combined work.

In the combination, you must combine any sections entitled 'History' in the various original documents, forming one section entitled 'History'; likewise combine any sections entitled 'Acknowledgements', and any sections entitled 'Dedications'. You must delete all sections entitled 'Endorsements.'

## C.6 Collections of documents

You may make a collection consisting of the document and other documents released under this license, and replace the individual copies of this license in the various documents with a single copy that is included in the collection, provided that you follow the rules of this license for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this license, provided you insert a copy of this license into the extracted document, and follow this license in all other respects regarding verbatim copying of that document.

## C.7 Aggregation with independent works

A compilation of the document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an 'aggregate' if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the document is included in an aggregate, this license does not apply to the other works in the aggregate which are not themselves derivative works of the document.

If the cover text requirement of section C.3 is applicable to these copies of the document, then if the document is less than one half of the entire aggregate, the document's cover texts may be placed on covers that bracket the document within the aggregate, or the electronic equivalent of covers if the document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## C.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the document under the terms of section C.4. Replacing invariant sections with translations requires special permission from their copyright holders, but you may include translations of some or all invariant sections in addition to the original versions of these invariant sections. You may include a translation of this license, and all the license notices in the document,

and any warranty disclaimers, provided that you also include the original English version of this license and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this license or a notice or disclaimer, the original version will prevail.

If a section in the document is entitled 'Acknowledgements', 'Dedications', or 'History', the requirement (section C.4) to preserve its title (section C.1) will typically require changing the actual title.

## C.9 Termination

You may not copy, modify, sublicense, or distribute the document except as expressly provided for under this license. Any other attempt to copy, modify, sublicense or distribute the document is void, and will automatically terminate your rights under this license. However, parties who have received copies, or rights, from you under this license will not have their licenses terminated so long as such parties remain in full compliance.

## C.10 Future revisions of this license

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.[1]

Each version of the license is given a distinguishing version number. If the document specifies that a particular numbered version of this license "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the document does not specify a version number of this license, you may choose any version ever published (not as a draft) by the Free Software Foundation.

[1] http://www.gnu.org/copyleft/

# Appendix D

# Revision history

**2.14 2010-08-29**
Original author resigned as maintainer

**2.14 2004-12-05**
Added appendix B
Extended section 1.2, adding note about monowidth fonts
Extended section 4.3, adding note about faked euro symbol
Extended introduction, adding section on getting further help

**2.00 2004-04-18**
Final release for fontinst 1.9
Completed revision process for fontinst 1.9

**1.90 2004-03-14**
Revised tutorial 6
Updated tutorial 7

**1.80 2004-02-14**
Revised tutorial 5
Updated tutorial 7

**1.70 2004-01-31**
Revised tutorial 4, adding section 4.4
Updated tutorial 7, adding section 7.2

**1.60 2004-01-31**
Revised tutorial 3
Updated tutorial 7

**1.50 2004-01-25**
Revised tutorial 2
Updated tutorial 7

**1.40 2004-01-24**
Started revision process for fontinst 1.9
Revised tutorial 1
Added tutorial 7
Revised appendix

**1.23 2003-08-31**
Updated section 1.7, adding tables 1.1 and 1.2
Updated section 1.7, adding note about Latin Modern
Revised section 1.6

**1.20 2003-07-17**
Updated section 4.1, adding preliminary hints for fontinst 1.9
Updated sections 3.2, 5.4, and 6.3, fixing bug in `nfssext.sty`
Updated section 6.3, improving swash support in `nfssext.sty`
Revised discussion of ornaments in section 6.5
Revised discussion of swashes in section 6.1
Added spelling corrections by William Adams

Added spelling corrections by Adrian Burd
Removed note soliciting contributions

**1.10 2003-03-27**
Added GNU Free Documentation License as appendix C
Added explicit licensing clause

**1.00 2003-03-25**
Final release for fontinst 1.8
Added tutorial 6
Updated notes on contributions

**0.80 2003-03-23**
Added spelling corrections and suggestions by Timothy Eyre
Revised section 1.6, splitting off section 1.7
Added section 3.3

**0.68 2003-02-09**
Revised section 4.2
Updated notes on contributions

**0.66 2003-01-26**
Added highlighting to code listings

**0.65 2003-01-19**
Added spelling corrections by Adrian Heathcote
Added spelling corrections by William Adams
Added section 2.2
Revised introduction

**0.60 2003-01-11**
Revised tutorial 3
Added discussion of kerning issues to section 3.1

**0.54 2003-01-04**
Revised discussion of OT1 encoding in tutorial 1

Added minor changes to appendix A

**0.52 2003-01-02**

Added table A.2 to appendix A

Revised introduction to tables in appendix A

**0.50 2002-12-30**

Added tutorial 5

Added appendix A featuring tables A.1, A.3, and A.4

Added revision history as appendix D

**0.43 2002-10-25**

First public release featuring tutorials 1–4

Added installation instructions to section 4.3

**0.40 2002-08-11**

Added tutorial 4

**0.30 2002-05-12**

Added tutorial 3

**0.20 2002-04-17**

Unreleased draft featuring tutorials 1 and 2