PHILIPP LEHMAN

THE FONT INSTALLATION GUIDE

USING POSTSCRIPT FONTS TO THEIR FULL POTENTIAL WITH LATEX

1.10

MARCH 2003

CONTENTS

INTRODUCTION

This guide to setting up Type 1 Postscript 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. Because of its concept, you will need the following additional manuals when working with it:

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 most files required for proper Postscript font support are generated by 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, you can download it from 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 canonical 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 it 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) under Latex 2ε. This scheme 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 for-

1. `http://www.ctan.org/tex-archive/fonts/utilities/fontinst/doc/fontinst.dvi`
2. `http://www.ctan.org/tex-archive/info/fontname/`
3. `ftp://tug.ctan.org/tex-archive/info/fontname.tar.gz`

mat from CTAN.[4] Feel free to skip the chapter about math fonts as we are only going to deal with text fonts. Setting up math fonts is a science in its own right.

Please note that this guide is written with version 1.8 of fontinst in mind. All recipes proposed here should still work with the current beta version, which will eventually be released as fontinst 1.9, but they do not exploit the new features of the upcoming release. Once fontinst 1.9 is released und proves to be stable, I will update this guide as my time permits.

Acknowledgments

I am indebted to Timothy Eyre for taking the time to proofread and comment on revision 0.68 of this guide. I would also like to thank William Adams and Adrian Heathcote for pointing out spelling mistakes.

4. http://www.ctan.org/tex-archive/macros/latex/doc/fntguide.pdf

TUTORIAL I

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. Before we begin, let's 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 (extension `pfb`) and the corresponding ASCII metric files (`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 ASCII format (`afm`), to generate metric files and virtual fonts. Fontinst is normally not used interactively but controlled by a TEX file. Since the fontinst file is specific to a given font family, we need to write a suitable file for our fonts first and run it through TEX afterwards.

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`).

STEP 4: INSTALLING THE FILES – We install all font metrics (`afm`), Type 1 font outlines (`pfb`), TEX font metrics (`tfm`), virtual fonts (`vf`), and font definition files (`fd`) into the local TEX tree. The remaining files are not required anymore and may be deleted.

STEP 5: CREATING 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, if a version of xdvi with native support for Postscript fonts is available, for 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.

1.1 Renaming the 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 the DOS filesystem as well as the ISO-9660 filesystem used for data CD-ROMs. 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 but foundry and typeface codes more liberally.

Note that 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 complementing the former. The advanced package will usually contain one of the following additional font sets: a set of optical small caps[1] and hanging figures[2], a set of expert fonts[3], additional weights, or a combination of these sets. This package has to be purchased

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 better than 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.
2. While hanging or 'old style' figures have ascenders and descenders to blend in with lowercase and mixed case text, lining figures are aligned with the height of the capital letters (compare 1369 to 1369). 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.
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 v for further information.

separately and can normally not be used independently in a sensible way. 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:

```
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 scratch 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 scratch 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 Using fontinst

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 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:

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.

```
1 \input fontinst.sty
2 \latinfamily{psb}{}
3 \bye
```

After loading fontinst (1) we simply call the `\latinfamily` macro with the base of the file names (the foundry code plus the typeface code) as the first argument (2). The second argument is code to be executed whenever this typeface is used. This is often employed to suppress hyphenation of fixed-width typefaces by setting the hyphenation character to a non-existing encoding position. If we wanted to suppress hyphenation for this font family, we would call the macro like this:

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

We save the file as `drv-psb.tex`, for example, and run it through `tex`:

```
tex drv-psb.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 for 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 *à* 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 European languages. 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.

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. Although some foundries, 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
```

After fontinst is finished, we run all property list files (`pl`) through `pltotf` to create Tex font metric files (`tfm`) and all virtual property list files (`vpl`) files through `vptovf` to create virtual fonts (`vf`). When using the Bash shell, this can be done 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.

1.3 Installing the files

The Tetex distribution supports a total of three Tex trees: a global one, a local one, and a user tree. The global tree is usually maintained by package management software. The local tree is for everything that is not part of the Tetex

distribution but should be available system-wide. The user tree is 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/
/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),[5] 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 sub-

5. `http://www.tug.org/tds/`

directories 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 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 files left in the working directory will not be used any more and may be deleted.

1.4 Creating map files

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. pdftex is a special case because it combines the roles of Tex and of 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. We will deal with the three most popular applications, the Postscript driver dvips, the DVI viewer xdvi, and pdftex. All of them need to be provided with a suitable map file. For dvips, the syntax of this file is explained in detail in the dvips manual.[6] For pdftex, it is is explained in the pdftex manual, and for xdvi in the documentation that comes with the source distribution. 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. This section will explain how to do that.

6. `http://www.radicaleye.com/dvipsman/`

Let's take a look at the first line of what will become `psb.map`, our map file for Sabon. The first column indicates the name of the raw Tex font without any file extension:

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

Since the `\latinfamily` macro reencodes all regular text fonts from Adobe Standard encoding (Fontname code `8a`) to Tex Base 1 (`8r`) when creating metric files for Tex, it corresponds to the name of the `pfb` file with encoding `8r` instead of `8a`. In this case, `psbr8a.pfb` becomes `psbr8r`.

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

The second column is the Postscript name of the font. 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 `psb.map`.

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

The third column of our map file is a reencoding instruction. As mentioned above, the `\latinfamily` macro reencodes 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   "TeXBase1Encoding ReEncodeFont"   <8r.enc <psbr8a.pfb
```

Finally, the last column contains 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` file, since we want the fonts to be embedded in the Postscript file. Now the map file for our basic Sabon set looks like this:

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

```
psbb8r  Sabon-Bold       "TeXBase1Encoding ReEncodeFont"           <8r.enc <psbb8a.pfb
psbbi8r Sabon-BoldItalic "TeXBase1Encoding ReEncodeFont"           <8r.enc <psbbi8a.pfb
```

In addition to that, we need to tell dvips about the slanted versions of all upright fonts which `\latinfamily` creates by default. 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; `psbr8r` becomes `psbro8r` and `psbb8r` is changed to `psbbo8r`:

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

Note that the name of the `pfb` file does *not* change. We also add a `SlantFont` instruction to the third column. By default, `\latinfamily` uses a slant factor of 0.167 when creating the modified metrics and our map file has to indicate this accordingly. Our complete map file looks like this:

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

The format suggested here is suitable for xdvi, dvips, and pdftex. We copy `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:

```
% Standard map file provided by default
p +psfonts.map
% New map file for Sabon
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:

```
% Standard map file provided by default
map +pdftex.map
% New map file for Sabon
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 provided by default
dvipsmap ps2pk.map
```

```
dvipsmap ...
% New map file for Sabon
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. Do not forget to update the file hash tables by running `texhash` or an equivalent command!

1.5 Using the 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's 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` or `\bfseries`. Larger font families may offer more than two weights. To select a particular weight, we use the `\fontseries` command in conjunction with the NFSS series codes defined during the installation of the font family. Please refer to the code table on page 78 of this guide for a list of the most common NFSS codes. To select the semibold (`sb`) weight for example, we would use the following construct:

```
\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}
3 \RequirePackage[T1]{fontenc}
4 \RequirePackage{textcomp}
```

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

```
\renewcommand{\rmdefault}{psb}
\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 preload 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'. If the `textcomp` package is used in conjunction with `inputenc`, it is even possible to enter some of these symbols directly in a Latex file.

There is one thing we have to keep in mind when switching to T1 encoding, though: the default encoding is an overall setting that applies to all text fonts used in the 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 `cmss` and `cmtt`: Computer Modern Sans Serif and Computer Modern Typewriter. Using Computer Modern in T1 encoding will pose some problems most European Tex users are already well familiar with. For our setup of Sabon, the following paragraphs are only relevant if you want to use Computer Modern Sans Serif and Computer Modern Typewriter as `\sffamily` and `\ttfamily` respectively. 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.

1.6 European Computer Modern

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 based on 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 means that PDF files created with pdftex or converted from Postscript would contain bitmap versions of these fonts when using T1 encoding. Bitmap fonts, however, are not independent of the output device and so are not suitable for on-screen display.

Let me summarize the technical implications: the original Computer Modern fonts have been available in Postscript format, but they are not suitable for applications that require letters not found in the English alphabet because they

are based on OT1 encoding. The European Computer Modern fonts address this issue by providing a more comprehensive set of glyphs, but they are not suitable for PDF files or device-independent Postscript files as required in professional printing because they were designed using Metafont. There are several solutions at our disposal to escape from this tricky situation. All of them are trade-offs in one way or another. The last one, the CM-super package, will bring us as close to a real solution as we can possibly get when using free versions of the Computer Modern fonts. This does not imply that the commercial offerings mentioned briefly at the end of this tutorial are perfect. Since I have never used any of them, I simply cannot comment on their quality.

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 Postscript versions of the original Computer Modern fonts: the AE fonts[8] and the ZE fonts[9]. The AE fonts are built on top of Computer Modern exclusively, but unfortunately they lack almost a dozen T1 characters including the French 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}
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\RequirePackage{ae}
\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 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:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{sabon}
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
```

8. http://www.ctan.org/tex-archive/fonts/ae/
9. http://www.ctan.org/tex-archive/fonts/zefonts/
10. http://www.ctan.org/tex-archive/macros/latex/contrib/supported/aeguill/

```
\RequirePackage{zefonts}
\renewcommand{\rmdefault}{psb}
\endinput
```

There is, however, a more robust solution you might be interested in if you are using Computer Modern a lot. Free Postscript versions of the European Computer Modern fonts have been made available. As mentioned before, one problem with OT1 encoded fonts is that they rely on composite glyphs which break searching in PDF files for all words containing accented letters. Both the ZE fonts and the AE fonts, although they enable Tex to hyphenate words containing accented letters properly, still suffer from this particular problem as they are based on the OT1 encoded Postscript versions of the Computer Modern fonts internally. If you work with PDF output, this might be another reason to switch to a Postscript version of the European Computer Modern fonts. Such fonts are included in two independent packages: Péter Szabó's Tt2001[11] as well as Vladimir Volovich's more recent CM-super[12] package. Both packages provide Postscript fonts which are essentially 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. Note, however, that it is rather large: the compressed package is about 64 MB in size. See the package documentation for installation instructions and answers to the most frequently asked questions. Here is a version of `sabon.sty` for use in conjunction with the CM-super package:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{sabon}
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\RequirePackage{typelec}
\renewcommand{\rmdefault}{psb}
\endinput
```

Apart from these free fonts, commercial Postscript versions of the European Computer Modern fonts have been offered by MicroPress[13] and Y&Y[14]. Please refer to the respective website for details and pricing. Since these fonts are explicitly provided for usage with Latex, their documentation should explain how to install and use them.

11. http://www.ctan.org/tex-archive/fonts/ps-type1/ec/
12. http://www.ctan.org/tex-archive/fonts/ps-type1/cm-super/
13. http://www.micropress-inc.com/fonts/ecfonts/ecmain.htm
14. http://www.yandy.com/em.htm

TUTORIAL II

STANDARD FONT SETS

While the `\latinfamily` shorthand is very convenient, it is not capable of coping 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.

11.1 The 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 file introduced here will also serve as a template for subsequent tutorials.

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

After loading fontinst 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 is set to `bx` by default. As long as you 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, you 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`).[1] Unless bold extended fonts are available, simply think of `bx` as the default bold weight.

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}
```

1. This is a default substitution that fontinst will always silently include. We could omit line 2 here, but if semibold fonts are available you might prefer using those as a substitute for `bx`.

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.

```
3 \substitutesilent{sc}{n}
```

We also add a substitution for the `sc` shape, which will in fact be used by the TS1 encoded families only. Since TS1 contains symbols and figures, we do not need an additional small caps font for this encoding as it would be identical to the upright variant anyway. However, to ensure that all text commands of the `textcomp` package will always work, even if the active NFSS shape is `sc`, we set up this shape substitution.

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

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 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 III and V respectively.

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

The integer variable `smallcapsscale` is a predefined 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 (~ 9.5°) is a reasonable default. Any value significantly greater that 176 (~ 10°) is usually too much.[2]

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

We start off with some basic font transformations: all 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.

```
10 \transformfont{psbro8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbr8a}}}
11 \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.

```
12 \installfonts
13 \installfamily{T1}{psb}{}
14 \installfamily{TS1}{psb}{}
```

The installation of a font family is enclosed in an environment which we open in line 12 and close later in line 29. 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`) as well as in TS1 (`8c`). 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 read by Latex whenever the font is selected. T1 will serve as our base encoding in Latex's text mode later. It is complemented by TS1 which provides additional glyphs such as currency signs and other frequently used symbols. 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 or Truetype fonts anyway. We will therefore deliberately ignore it and focus on T1 and TS1 exclusively.

```
15 \installfont{psbr8t}{psbr8r,latin}{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

2. I suggest you do not bother trying to match the slope of the italic fonts when creating a slanted variant of a roman font. This will usually not work for typefaces with true italics because the latter are an independent design.

to create, the second one is a list of files used to build this font. These can be `afm`, `mtx`, or `pl` files, their extension is omitted. If multiple fonts are provided, `\installfont` does not overwrite any encoding positions when reading in additional files, it simply fills vacant slots if it finds suitable glyphs in the next font. The metric file `latin.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 extension, 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[3], medium[4], 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,latin}{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 positions of the lowercase alphabet. Fortunately, we don't 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 scale factor.

```
17 \installfont{psbro8t}{psbro8r,latin}{t1}{T1}{psb}{m}{sl}{}
18 \installfont{psbri8t}{psbri8r,latin}{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,latin}{t1}{T1}{psb}{b}{n}{}
20 \installfont{psbbc8t}{psbb8r,latin}{t1c}{T1}{psb}{b}{sc}{}
21 \installfont{psbbo8t}{psbbo8r,latin}{t1}{T1}{psb}{b}{sl}{}
22 \installfont{psbbi8t}{psbbi8r,latin}{t1}{T1}{psb}{b}{it}{}
```

After that, we add virtual fonts for TS1 encoding:

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

Like `latin.mtx`, `textcomp.mtx` is an auxiliary metric file provided by fontinst. It should always be added when creating TS1 encoded fonts for the `textcomp` package. The third argument, the encoding vector, refers to `ts1.etx` in this case. As TS1 encoding is for symbols only and we did set up a shape substitution, we do not need a TS1 encoded small caps font. Slanted and italic fonts are handled like the upright one:

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

3. Latex doesn't really care about the name of the font or the foundry. This argument simply defines the code that identifies the font within the NFSS.
4. In fact, the more appropriate name would be *regular* because *medium* is a moderate bold weight with the NFSS code `mb`.

We repeat 23–25 for the bold fonts:

```
\installfont{psbb8c}{psbb8r,textcomp}{ts1}{TS1}{psb}{b}{n}{}
\installfont{psbbo8c}{psbbo8r,textcomp}{ts1}{TS1}{psb}{b}{sl}{}
\installfont{psbbi8c}{psbbi8r,textcomp}{ts1}{TS1}{psb}{b}{it}{}
```

Finally, we close the install environment and terminate:

```
\endinstallfonts
\bye
```

11.2 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,latin><OT1><OT1><psb><m><n>
INFO> run \installfont <psbr8t><psbr8r,latin><T1><T1><psb><m><n>
INFO> run \installfont <psbr8c><psbr8r,textcomp><TS1><TS1><psb><m><n>
INFO> run \installfont <psbrc7t><psbr8r,latin><OT1c><OT1><psb><m><sc>
INFO> run \installfont <psbrc8t><psbr8r,latin><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,latin><OT1><OT1><psb><m><sl>
INFO> run \installfont <psbro8t><psbro8r,latin><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,latin><OT1i><OT1><psb><m><it>
INFO> run \installfont <psbri8t><psbri8r,latin><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,latin><OT1><OT1><psb><b><n>
INFO> run \installfont <psbb8t><psbb8r,latin><T1><T1><psb><b><n>
INFO> run \installfont <psbb8c><psbb8r,textcomp><TS1><TS1><psb><b><n>
INFO> run \installfont <psbbc7t><psbb8r,latin><OT1c><OT1><psb><b><sc>
INFO> run \installfont <psbbc8t><psbb8r,latin><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,latin><OT1><OT1><psb><b><sl>
INFO> run \installfont <psbbo8t><psbbo8r,latin><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,latin><OT1i><OT1><psb><b><it>
INFO> run \installfont <psbbi8t><psbbi8r,latin><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 `latin.mtx`, the encoding file is obviously `8r.etx` in this case. Creating OT1 encoded virtual fonts requires `latin.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 because `t1i.etx` reads `t1.etx` internally, hence we did not use `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.

11.3 Map files revisited

With all of that in mind, let's now go back to the dvips map file from the first tutorial and take another look at it. The meaning of the reencoding and slanting instructions should be much clearer now:

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

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 `ReEncodeFont` 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 fontinst file to the first line of the map file:

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

The situation is similar for the slanted fonts. The font files embedded in the Postscript file are not slanted, they are upright. Fontinst has performed 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 `psb.map` and embedding them along with the `SlantFont` 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 finally performing the rendering. The value of the `SlantFont` 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. That's why its precision is fixed to three decimal places. Let's 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 xdvi deals 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.

TUTORIAL III

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 V. 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. Adobe do 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's 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` etc.:

```
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 a so-called 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 so-called expert fonts feature hanging ('old style') figures amongst other things. Now let's 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`, we would see that this font in fact comes with hanging ('old style') figures even though the figures are labeled using the standard names.[1] 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 ;
```

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

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

(including Adobe) only: there is no separate upright OSF font so we have to take the upright hanging figures from the small caps font when building virtual fonts.

III.1 The fontinst file

For fontinst, we use the 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.

```
1  \input fontinst.sty
2  \substitutesilent{bx}{b}
3  \substitutesilent{sc}{n}
4  \setint{smallcapsscale}{800}
5  \setint{slant}{167}
6  \transformfont{psbr8r}{\reencodefont{8r}{\fromafm{psbr8a}}}
7  \transformfont{psbri8r}{\reencodefont{8r}{\fromafm{psbri8a}}}
8  \transformfont{psbb8r}{\reencodefont{8r}{\fromafm{psbb8a}}}
9  \transformfont{psbbi8r}{\reencodefont{8r}{\fromafm{psbbi8a}}}
10 \transformfont{psbrc8r}{\reencodefont{8r}{\fromafm{psbrc8a}}}
11 \transformfont{psbrij8r}{\reencodefont{8r}{\fromafm{psbrij8a}}}
12 \transformfont{psbbj8r}{\reencodefont{8r}{\fromafm{psbbj8a}}}
13 \transformfont{psbbij8r}{\reencodefont{8r}{\fromafm{psbbij8a}}}
```

The first couple of lines of our template remain unchanged (1–9). After the reencodings inherited form our template, we insert the additional fonts since they need to be reencoded as well (10–13).

```
14 \transformfont{psbro8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbr8a}}}
15 \transformfont{psbbo8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbb8a}}}
16 \transformfont{psbrco8r}{\slantfont{\int{slant}}\reencodefont{8r}{\fromafm{psbrc8a}}}
17 \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 (16) may seem like a strange thing to do at first since we do not really want to create a slanted small caps shape. But since 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.

```
18 \installfonts
19 \installfamily{T1}{psb}{}
20 \installfamily{TS1}{psb}{}
21 \installfont{psbr8t}{psbr8r,latin}{t1}{T1}{psb}{m}{n}{}
22 \installfont{psbrc8t}{psbrc8r,unsetnum,kernoff,psbr8r,kernon,latin}{t1}{T1}{psb}{m}{sc}{}
```

The file `psbrc8r` provides small caps and hanging figures, but we want `psb` to be a consistent family using lining figures throughout. Therefore, we read `psbrc8r` first and clear the encoding positions of all figures using commands from a separate metric file, `unsetnum.mtx`, right after that. This file is listed

further down; all it does is clear all figure slots. When adding psbr8r afterwards, the figure slots of the virtual font psbrc8t will be filled using the lining figures found in psbr8r. 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. This allows us to insert the lining figures of psbr8r in the virtual font while the rest of the glyphs including the small caps is taken from psbrc8r. As to the encoding vector, we use the regular encoding file t1.etx in this case since psbrc8r uses standard glyph names for the small caps so that t1c.etx would be inappropriate.

There is one more thing we have to take into account: adding a metric file to the \installfont command also adds kerning information provided by that file. The problem here is that some of the glyph names in our raw fonts are not unique since the small caps in psbrc8r are encoded and labeled just like the lowercase letters in psbr8r. The kerning data in psbr8r, however, refers to ordinary lowercase letters. Under certain circumstances, misleading kerning data might thus be included in the virtual small caps font psbrc8t. To avoid that, we add two auxiliary files provided by fontinst, kernon.mtx and kernoff.mtx, which enable and disable fontinst's \setkern command. When added to the input file list as shown above, this will effectively ignore the kerning data in psbr8r.

```
23 \installfont{psbro8t}{psbro8r,latin}{t1}{T1}{psb}{m}{sl}{}
24 \installfont{psbri8t}{psbri8r,latin}{t1}{T1}{psb}{m}{it}{}
25 \installfont{psbb8t}{psbb8r,latin}{t1}{T1}{psb}{b}{n}{}
26 \installfont{psbbc8t}{psbb8r,latin}{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, so we use the encoding file t1c.etx here (26). The remaining lines for T1 encoding do not require any adjustments:

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

That's it for T1 encoding. While TS1 is primarily intended for symbols complementing T1, it includes hanging figures as well. Since the only way to use them is loading the textcomp package and typing some rather cumbersome text commands like \textzerooldstyle and \textoneoldstyle it is not terribly 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. We have put down some hard, cold cash for the Sabon SC & OSF package and we want to make the most of it. Let's see how we can put hanging figures in TS1/psb as well. As mentioned above, the problem here 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. To do that, we need an additional resource provided by fontinst, the metric file resetosf.mtx. With this in mind, let's add a section for TS1 encoding to our fontinst file:

```
\installfont{psbr8c}{psbr8r,unsetnum,kernoff,psbrc8r,kernon,resetosf,textcomp}{ts1}%
  {TS1}{psb}{m}{n}{}
```

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 first steps should therefore look familiar: we read `psbr8r`, clear the standard figures using `unsetnum`, and read `psbrc8r`. Since we are dealing with TS1 here, one additional step is required. We add `resetosf.mtx` to the input file list of this `\installfont` command to rename the figures found in `psbrc8r` (the figures in `psbr8r` have already been discarded by `unsetnum`). `resetosf` will rename the figures to `zerooldstyle` and so on. We also add `kernon.mtx` and `kernoff.mtx` to protect the kerning data. Typing `\textthreeoldstyle` in a Latex file when the `textcomp` package has been loaded will now typeset a proper hanging three.

```
\installfont{psbro8c}{psbro8r,unsetnum,kernoff,psbrco8r,kernon,resetosf,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 the OSF fonts already provide hanging figures, all we need to do is rename them for TS1 encoding by adding `resetosf.mtx`:

```
\installfont{psbri8c}{psbrij8r,resetosf,textcomp}{ts1}{TS1}{psb}{m}{it}{}
\installfont{psbb8c}{psbbj8r,resetosf,textcomp}{ts1}{TS1}{psb}{b}{n}{}
\installfont{psbbo8c}{psbboj8r,resetosf,textcomp}{ts1}{TS1}{psb}{b}{sl}{}
\installfont{psbbi8c}{psbbij8r,resetosf,textcomp}{ts1}{TS1}{psb}{b}{it}{}
\endinstallfonts
```

This is the first half of our fontinst file which is dealing with the `psb` family. Compared to the template introduced in the previous tutorial it adds optical small caps to T1 and hanging figures to TS1 encoding. We will create an additional font family called `psbj` which we want to use hanging figures throughout.

```
\installfonts
\installfamily{T1}{psbj}{}
\installfont{psbrj8t}{psbr8r,unsetnum,kernoff,psbrc8r,kernon,latin}{t1}{T1}{psbj}{m}{n}{}
```

If we want the `psbj` family to incorporate hanging figures, we need to exchange the figure set of the virtual font like we did when creating the regular-weight small caps font above. But this time, we do it the other way around: we read `psbr8r` first, clear the encoding slots of all figures, and add `psbrc8r` afterwards to fill the figure slots using the hanging figures found in `psbrc8r`. Only the figures found in `psbrc8r` will be included in the virtual font as all other encoding slots were already filled by `psbr8r`. Again, care needs to be taken with the kerning data here. The kerning information in `psbrc8r` refers to small caps although the glyphs are encoded as ordinary lowercase letters. Hence we need to add `kernon.mtx` and `kernoff.mtx` to discard the kerning data in `psbrc8r`.

```
41 \installfont{psbrcj8t}{psbrc8r,latin}{t1}{T1}{psbj}{m}{sc}{}
```

The small caps font does not require any modifications this time. `psbrc8r` 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`.

```
42 \installfont{psbroj8t}{psbro8r,unsetnum,kernoff,psbrco8r,kernon,latin}{t1}{T1}{psbj}{m}{sl}{}
```

The slanted shape is straightforward to the upright one: we read `psbro8r`, clear the figures, and add the slanted hanging figures provided by `psbrco8r`. We also toggle fontinst's `\setkern` macro by adding `kernon` and `kernoff`.

```
43 \installfont{psbrij8t}{psbrij8r,latin}{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. 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:

```
44 \installfont{psbbj8t}{psbbj8r,latin}{t1}{T1}{psbj}{b}{n}{}
45 \installfont{psbbcj8t}{psbbj8r,latin}{t1c}{T1}{psbj}{b}{sc}{}
```

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

```
46 \installfont{psbboj8t}{psbboj8r,latin}{t1}{T1}{psbj}{b}{sl}{}
47 \installfont{psbbij8t}{psbbij8r,latin}{t1}{T1}{psbj}{b}{it}{}
48 \endinstallfonts
49 \bye
```

This is the complete fontinst file for the NFSS font families `psb` and `psbj`. It requires the metric file `unsetnum.mtx` which is part of the fontinst package. Metric files always begin with `\relax` and enclose all commands in a `metrics` environment. Essentially, `unsetnum.mtx` consists of several `\unsetglyph` commands which clear all figure slots:

```
\relax
\metrics
\unsetglyph{zero}
\unsetglyph{one}
\unsetglyph{two}
\unsetglyph{three}
\unsetglyph{four}
\unsetglyph{five}
\unsetglyph{six}
\unsetglyph{seven}
\unsetglyph{eight}
\unsetglyph{nine}
\endmetrics
```

You probably will have noticed that we did not create TS1 encoded fonts for the `psbj` family. The reason is quite simple: since TS1 is not a regular text encoding `TS1/psbj` would be identical to `TS1/psb` anyway. To ensure that the `textcomp`

package works for the psbj family nonetheless, we need to set up some substitutions. Since fontinst does not support family substitutions we cannot create them automatically. We have to write a font definition file manually. The file ts1psbj.fd should like this:

```
\ProvidesFile{ts1psbj.fd}
\DeclareFontFamily{TS1}{psbj}{}
\DeclareFontShape{TS1}{psbj}{m} {n} {<-> ssub * psb/m/n} {}
\DeclareFontShape{TS1}{psbj}{m} {sc}{<-> ssub * psb/m/sc}{}
\DeclareFontShape{TS1}{psbj}{m} {sl}{<-> ssub * psb/m/sl}{}
\DeclareFontShape{TS1}{psbj}{m} {it}{<-> ssub * psb/m/it}{}
\DeclareFontShape{TS1}{psbj}{b} {n} {<-> ssub * psb/b/n} {}
\DeclareFontShape{TS1}{psbj}{b} {sc}{<-> ssub * psb/b/sc}{}
\DeclareFontShape{TS1}{psbj}{b} {sl}{<-> ssub * psb/b/sl}{}
\DeclareFontShape{TS1}{psbj}{b} {it}{<-> ssub * psb/b/it}{}
\DeclareFontShape{TS1}{psbj}{bx}{n} {<-> ssub * psb/b/n} {}
\DeclareFontShape{TS1}{psbj}{bx}{sc}{<-> ssub * psb/b/sc}{}
\DeclareFontShape{TS1}{psbj}{bx}{sl}{<-> ssub * psb/b/sl}{}
\DeclareFontShape{TS1}{psbj}{bx}{it}{<-> ssub * psb/b/it}{}
\endinput
```

The syntax of font definition files is explained in the Latex font selection guide and will not be discussed in detail here.[2] The main point of this file should be evident: for all series and shapes, we substitute TS1/psb for TS1/psbj because we did not create virtual fonts for TS1/psbj. The ssub directive is a silent substitution. For details, see chapter 4 of the font selection guide, section 4.4 in particular. With this additional font definition file we now have a fully functional setup for psb and psbj in T1 and TS1 encoding.

III.2 The map file

After running the fontinst file through Tex and installing the new fonts, we still need to update the map file psb.map. We add the following lines for the additional fonts found in the SC & OSF package:

```
psbrc8r  Sabon-RomanSC       "TeXBase1Encoding ReEncodeFont"          <8r.enc <psbrc8a.pfb
psbrij8r Sabon-ItalicOsF     "TeXBase1Encoding ReEncodeFont"          <8r.enc <psbrij8a.pfb
psbbj8r  Sabon-BoldOsF       "TeXBase1Encoding ReEncodeFont"          <8r.enc <psbbj8a.pfb
psbbij8r Sabon-BoldItalicOsF "TeXBase1Encoding ReEncodeFont"          <8r.enc <psbbij8a.pfb
```

In addition to this, we need slanted versions of the new fonts. For the bold OSF font this is obvious. Since regular-weight hanging figures are found in the small caps font, we need a slanted version of this font as well to provide matching figures for the slanted shape of the psbj family. This leads us to the slanted small caps variant:

```
psbrco8r Sabon-RomanSC "0.167 SlantFont TeXBase1Encoding ReEncodeFont" <8r.enc <psbrc8a.pfb
psbboj8r Sabon-BoldOsF "0.167 SlantFont TeXBase1Encoding ReEncodeFont" <8r.enc <psbbj8a.pfb
```

This is the complete map file:

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

2. http://www.ctan.org/tex-archive/macros/latex/doc/fntguide.pdf

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

III.3 The style file

With two Sabon families at hand, we might want to update `sabon.sty` to make them readily available. We add the two options `oldstyle` and `lining` 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. 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's put them in a stand-alone style file, `nfssext.sty`, and load that in `sabon.sty` (5):

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{sabon}
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\RequirePackage{nfssext}
\DeclareOption{oldstyle}{\renewcommand{\rmdefault}{psbj}}
\DeclareOption{lining}{\renewcommand{\rmdefault}{psb}}
\ExecuteOptions{oldstyle}
\ProcessOptions
\endinput
```

The style file `nfssext.sty` might look like this:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{nfssext}
\newcommand{\exfs@try@family}[1]{%
  \expandafter\ifx\csname\f@encoding+#1\endcsname\relax
    \let\exfs@tempa\relax
    \begingroup
      \fontfamily{#1}\try@load@fontshape
      \expandafter\ifx\csname\f@encoding+#1\endcsname\relax
        \PackageWarning{nfssext}{%
          Font family \f@encoding/#1 unavailable,\MessageBreak
          ignoring font switch}%
      \else
        \gdef\exfs@tempa{\fontfamily{#1}\selectfont}%
      \fi
    \endgroup
    \exfs@tempa
  \else
    \fontfamily{#1}\selectfont
  \fi}
```

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 check if the requested family in the current encoding has been defined already (4). If so, we simply switch font families (18); if not, we attempt to load the font definitions (7). If this succeeds, we set up a macro (13) to be expanded after the group that will actually switch font families; if not, we print a warning message (9–11) 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 can be employed just like `\bfseries` or `\itshape`. Internally, they will take the first three letters of the current NFSS font family name (20), 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 properly 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`.

III.4 Fonts supplied with Tex

The standard Postscript fonts supplied with most 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.

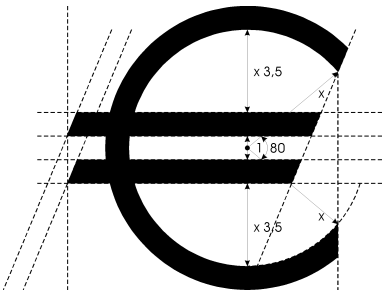
Hanging figures are included in Computer Modern as well, but they are hidden in some of the math fonts. The only way to use them with the default setup is rather cumbersome: the command `\oldstylenums{}` will take the numbers to be typeset as hanging figures as an argument. There is a set of virtual fonts for European Computer Modern (EC) 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.[3] Please refer to the package

3. http://www.ctan.org/tex-archive/fonts/eco/

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 1.6.

TUTORIAL IV

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 an 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 position. Some font foundries decided to follow this path in order to conform to Adobe Standard encoding. Others preferred to drop some supposedly rarely used glyph and put the euro symbol in its encoding position 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.

IV.1 Uncoded euro symbol

While rather inattentive to the problem at first, Adobe are finally updating their typeface portfolio by gradually adding matching euro symbols to their fonts. 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 which does not define this symbol at all. An encoding slot number of `-1` tells us that the glyph was not assigned to any encoding position:

```
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 position. 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. The trouble is that older versions of Tex Base 1

encoding do not include the euro symbol either. The current stable release of fontinst, version 1.8, ships with an encoding vector that is not suitable for our situation for this very reason. You can verify that by running the file `8r.etx` through Latex to create a documented listing of the encoding vector:

```
latex 8r.etx
```

If the version of this file is 1.801 (dated June 29, 1998), it does not include the euro symbol. You can solve this problem by updating fontinst to the current beta release[1] which ships with an updated encoding vector. In this case, no modifications will be required and you can install the font as usual. Note, however, that you will need a matching version of `8r.enc` as well, so that dvips and pdftex can use the symbol.[2]

In the following, we will create our own updated version of `8r.etx` and `8r.enc`. Apart from illustrating how to deal with a typical encoding problem this might also serve as a guide if updating fontinst is not an option for some reason. First, we create a copy of the file `8r.etx` as provided by fontinst. The updated encoding vector in the fontinst beta release puts the euro symbol in slot 128. We will do the same to ensure that our vector remains compatible with the official distribution. Otherwise, we could not call it a Tex Base 1 encoding vector any more. Let's take a look at the relevant part of `8r.etx`:

```
624 \setslot{asciitilde}
625    \comment{The ASCII tilde '\textasciitilde'.
626       This is included for compatibility with typewriter fonts used
627       for computer listings.}
628 \endsetslot
629
630 \comment{The following 32 slots, 128--159, are based on Windows ANSI.}
631
632 \nextslot{130}
633 \setslot{quotesinglbase}
634    \comment{A German single quote mark '\quotesinglbase' similar to a comma,
635       but with different sidebearings.}
636 \endsetslot
```

Slot 126 defines `asciitilde`, slots 127–129 are empty, and slot 130 defines the lower single quotation mark `quotesinglbase`. The slot number is automatically incremented by 1 for each `\setslot` command, but if some slots are left empty, the slot has to be set explicitly with a `\nextslot` command. This is done for `quotesinglbase` above. We want to add the euro symbol in slot 128, so we add the following:

```
630 \comment{The following 32 slots, 128--159, are based on Windows ANSI.}

    \nextslot{128}
    \setslot{Euro}
       \comment{The euro currency symbol '\texteuro'.}
    \endsetslot
```

1. http://www.ctan.org/tex-archive/fonts/utilities/fontinst-prerelease/
2. http://www.ctan.org/tex-archive/info/fontname/8r.enc

```
\nextslot{130}
\setslot{quotesinglbase}
   \comment{A German single quote mark '\quotesinglbase' similar to a comma,
      but with different sidebearings.}
\endsetslot
```

Since slot 127 is empty and the last slot defined was 126 we need to set the slot explicitly by adding `\nextslot` before actually defining the encoding position. When defining the slot, keep in mind that the glyph names are case sensitive; `Euro` is not equivalent to `euro`. We also add an explanation so that the commented version of this file provides a meaningful explanation. This is all we need. It might be a good idea to update `\title` and `\date` at the beginning of the file to avoid any confusion. Finally, we install this file in the branch `tex/fontinst/base/` of the local Tex tree. If our system has been set up as recommended in the first tutorial, fontinst will now pick up our updated encoding vector. Now we need a version of `8r.enc` that matches our `8r.etx`. This is what the relevant part of `8r.enc` looks like:

```
% 0x70
 /p /q /r /s /t /u /v /w
 /x /y /z /braceleft /bar /braceright /asciitilde
 /.notdef
% 0x80
 /.notdef /.notdef /quotesinglbase /florin
 /quotedblbase /ellipsis /dagger /daggerdbl
 /circumflex /perthousand /Scaron /guilsinglleft
 /OE /.notdef /.notdef /.notdef
```

Note that in Postscript encoding vectors empty slots are marked `/.notdef`. We can spot the same pattern: `/asciitilde` in slot 126 is followed by three empty slots (127–129) and finally `/quotesinglbase` in slot 130. We count the slots and add `/Euro` in slot 128 (given in hexadecimal notation as `0x80` here):

```
% 0x70
 /p /q /r /s /t /u /v /w
 /x /y /z /braceleft /bar /braceright /asciitilde
 /.notdef
% 0x80
 /Euro /.notdef /quotesinglbase /florin
 /quotedblbase /ellipsis /dagger /daggerdbl
 /circumflex /perthousand /Scaron /guilsinglleft
 /OE /.notdef /.notdef /.notdef
```

After that, we move our modified `8r.enc` to `dvips/base/` in the local Tex tree and update the kpathsea file databases by running `texhash`. Our system is now ready for the euro. 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. The reencoding is performed as usual:

```
\transformfont{padr8r}{\reencodefont{8r}{\fromafm{padr8a}}}
\transformfont{padri8r}{\reencodefont{8r}{\fromafm{padri8a}}}
\transformfont{padb8r}{\reencodefont{8r}{\fromafm{padb8a}}}
\transformfont{padbi8r}{\reencodefont{8r}{\fromafm{padbi8a}}}
```

The new `8r` encoding vector will ensure that the euro symbol is available in all Tex Base 1 encoded raw fonts, so we can simply use them to build TS1 encoded virtual fonts:

```
\installfont{padr8c}{padr8r,textcomp}{ts1}{TS1}{pad}{m}{n}{}
\installfont{padri8c}{padri8r,textcomp}{ts1}{TS1}{pad}{m}{it}{}
\installfont{padb8c}{padb8r,textcomp}{ts1}{TS1}{pad}{b}{n}{}
\installfont{padbi8c}{padbi8r,textcomp}{ts1}{TS1}{pad}{b}{it}{}
```

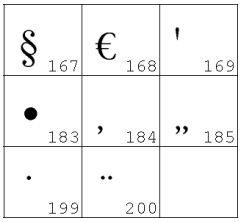
After installing the fonts and creating a map file as usual, the euro symbol will be available as `\texteuro` when loading the `textcomp` package.

IV.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 reasoning behind this is that you can access the symbol without reencoding the font. 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. One of them is uncoded (slot -1) and labeled as `Euro`:

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

The other one is found in encoding slot 168, that is, it is encoded as the currency symbol and named accordingly. To verify that, we have to take a look at the `pfb` files in a font viewer or 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 ;
```

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 position or we use the already encoded euro symbol found in the slot of the currency symbol and move it to the proper encoding position. The former case was already discussed above, let's now investigate the latter.

The best way to move the glyph to a different slot is resetting it when creating the TS1 encoded virtual font. We use an approach that is functionally equivalent to the way we have reset the hanging figures in the previous tutorial. The appropriate low-level commands that set the glyph go in a dedicated metric file, `reseteur.mtx`, which we have to create ourselves:

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

We set the glyph `euro` based on the glyph `currency` scaled to its full size (3–5), adjust the kerning on either side of `euro` to match that of `currency` (6) and finally unset the glyph `currency` (7) because there is no such thing as a 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.

IV.3 Euro symbol taken from external symbol font

Let's 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 an euro symbol at all so all we can do is take it from an external font. While some other 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.[3] In typographical respects, this is merely a desperate workaround. However, lacking a matching euro symbol, we do not have a choice. The Adobe Euro fonts 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 shape of 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 positions. 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 ;
```

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

```
C 36 ; WX 750 ; N Euro.004 ; B 10 -12 709 685 ;
C 37 ; WX 750 ; N Euro.005 ; B 10 -12 709 685 ;
C 38 ; WX 750 ; N Euro.006 ; B 10 -12 709 685 ;
C 39 ; WX 750 ; N Euro.007 ; B 10 -12 709 685 ;
C 40 ; WX 750 ; N Euro.008 ; B 10 -12 709 685 ;
C 41 ; WX 750 ; N Euro.009 ; B 10 -12 709 685 ;
```

On further inspection, we find two additional glyphs. There is a glyph labeled as `Euro` in slot 128 as well as an uncoded glyph labeled as `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. This is the encoding position of the euro symbol in Unicode encoding, hence the string `uni20ac`. If nothing else, one thing is for sure: 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 both properly encoded and labeled as `Euro`; the encoding position 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. This time no reencoding or renaming is required. The relevant section of our fontinst file for Sabon would look as follows:

```
\installfont{psbr8c}{psbr8r,zpeurs,textcomp}{ts1}{TS1}{psb}{m}{n}{}
\installfont{psbro8c}{psbro8r,zpeuros,textcomp}{ts1}{TS1}{psb}{m}{sl}{}
\installfont{psbri8c}{psbri8r,zpeuris,textcomp}{ts1}{TS1}{psb}{m}{it}{}
\installfont{psbb8c}{psbb8r,zpeubs,textcomp}{ts1}{TS1}{psb}{b}{n}{}
\installfont{psbbo8c}{psbbo8r,zpeubos,textcomp}{ts1}{TS1}{psb}{b}{sl}{}
\installfont{psbbi8c}{psbbi8r,zpeubis,textcomp}{ts1}{TS1}{psb}{b}{it}{}
```

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. Because the above fontinst file depends on this installation, it makes sense to do it first. Since the installation of symbol fonts differs from that of regular text fonts, we will take a look at the required steps. The Euro font package[4] will provide 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
```

4. `http://www.adobe.com/type/eurofont.html`

```
_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 canonical names for these fonts:

```
zpeur     EuroSerif-Regular               A    916    _3_____
zpeub     EuroSerif-Bold                  A    916    _3b____
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
```

As we do not really need fontinst when dealing with symbol fonts, we simply run `afm2tfm` on each `afm` file to create a corresponding `tfm` file for Tex:

```
afm2tfm zpeur.afm   zpeur.tfm
afm2tfm zpeuri.afm  zpeuri.tfm
afm2tfm zpeub.afm   zpeub.tfm
afm2tfm zpeubi.afm  zpeubi.tfm
afm2tfm zpeurs.afm  zpeurs.tfm
afm2tfm zpeuris.afm zpeuris.tfm
afm2tfm zpeubs.afm  zpeubs.tfm
afm2tfm zpeubis.afm zpeubis.tfm
afm2tfm zpeurt.afm  zpeurt.tfm
afm2tfm zpeurit.afm zpeurit.tfm
afm2tfm zpeubt.afm  zpeubt.tfm
afm2tfm zpeubit.afm zpeubit.tfm
```

We also need slanted versions of all upright fonts. As slant factor, we use the generic value 0.167:

```
afm2tfm zpeur.afm  -s 0.167 zpeuro.tfm
afm2tfm zpeub.afm  -s 0.167 zpeubo.tfm
afm2tfm zpeurs.afm -s 0.167 zpeuros.tfm
afm2tfm zpeubs.afm -s 0.167 zpeubos.tfm
afm2tfm zpeurt.afm -s 0.167 zpeurot.tfm
afm2tfm zpeubt.afm -s 0.167 zpeubot.tfm
```

In addition to that, we need a map file for dvips. Map files for symbol fonts are simpler than those for text fonts because the fonts are not reencoded. Therefore,

there will be no `ReEncodeFont` instruction and no encoding vector. The first lines of `peu.map` look like this:

```
zpeur   EuroSerif-Regular                <zpeur.pfb
zpeuri  EuroSerif-Italic                 <zpeuri.pfb
zpeub   EuroSerif-Bold                   <zpeub.pfb
zpeubi  EuroSerif-BoldItalic             <zpeubi.pfb
zpeurs  EuroSans-Regular                 <zpeurs.pfb
zpeuris EuroSans-Italic                  <zpeuris.pfb
zpeubs  EuroSans-Bold                    <zpeubs.pfb
zpeubis EuroSans-BoldItalic              <zpeubis.pfb
zpeurt  EuroMono-Regular                 <zpeurt.pfb
zpeurit EuroMono-Italic                  <zpeurit.pfb
zpeubt  EuroMono-Bold                    <zpeubt.pfb
zpeubit EuroMono-BoldItalic              <zpeubit.pfb
```

We also need to add `SlantFont` instructions for all slanted shapes:

```
zpeuro  EuroSerif-Regular "0.167 SlantFont" <zpeur.pfb
zpeubo  EuroSerif-Bold    "0.167 SlantFont" <zpeub.pfb
zpeuros EuroSans-Regular  "0.167 SlantFont" <zpeurs.pfb
zpeubos EuroSans-Bold     "0.167 SlantFont" <zpeubs.pfb
zpeurot EuroMono-Regular  "0.167 SlantFont" <zpeurt.pfb
zpeubot EuroMono-Bold     "0.167 SlantFont" <zpeubt.pfb
```

While we are at it, let's also write some font definition files for Latex. These are not required if the fonts are only referenced by other virtual fonts, but they will allow us the access the Euro fonts directly in any Latex file. The syntax of the commands used in font definition files is explained in the Latex font selection guide mentioned in the introduction. Our font definition file for Euro Serif, `upeu.fd`, should look like this:

```
\ProvidesFile{upeu.fd}
\DeclareFontFamily{U}{peu}{}
\DeclareFontShape{U}{peu}{m} {n} {<-> zpeur} {}
\DeclareFontShape{U}{peu}{m} {sc}{<-> ssub * peu/m/n} {}
\DeclareFontShape{U}{peu}{m} {sl}{<-> zpeuro}{}
\DeclareFontShape{U}{peu}{m} {it}{<-> zpeuri}{}
\DeclareFontShape{U}{peu}{b} {n} {<-> zpeub} {}
\DeclareFontShape{U}{peu}{b} {sc}{<-> ssub * peu/b/n} {}
\DeclareFontShape{U}{peu}{b} {sl}{<-> zpeubo}{}
\DeclareFontShape{U}{peu}{b} {it}{<-> zpeubi}{}
\DeclareFontShape{U}{peu}{bx}{n} {<-> ssub * peu/b/n} {}
\DeclareFontShape{U}{peu}{bx}{sc}{<-> ssub * peu/b/n} {}
\DeclareFontShape{U}{peu}{bx}{sl}{<-> ssub * peu/b/sl}{}
\DeclareFontShape{U}{peu}{bx}{it}{<-> ssub * peu/b/it}{}
\endinput
```

For Euro Sans, `upeus.fd`:

```
\ProvidesFile{upeus.fd}
\DeclareFontFamily{U}{peus}{}
\DeclareFontShape{U}{peus}{m} {n} {<-> zpeurs} {}
\DeclareFontShape{U}{peus}{m} {sc}{<-> ssub * peus/m/n} {}
\DeclareFontShape{U}{peus}{m} {sl}{<-> zpeuros}{}
\DeclareFontShape{U}{peus}{m} {it}{<-> zpeuris}{}
\DeclareFontShape{U}{peus}{b} {n} {<-> zpeubs} {}
\DeclareFontShape{U}{peus}{b} {sc}{<-> ssub * peus/b/n} {}
```

```
\DeclareFontShape{U}{peus}{b} {sl}{<-> zpeubos}{}
\DeclareFontShape{U}{peus}{b} {it}{<-> zpeubis}{}
\DeclareFontShape{U}{peus}{bx}{n} {<-> ssub * peus/b/n} {}
\DeclareFontShape{U}{peus}{bx}{sc}{<-> ssub * peus/b/n} {}
\DeclareFontShape{U}{peus}{bx}{sl}{<-> ssub * peus/b/sl}{}
\DeclareFontShape{U}{peus}{bx}{it}{<-> ssub * peus/b/it}{}
\endinput
```

And for Euro Mono, `upeut.fd`:

```
\ProvidesFile{upeut.fd}
\DeclareFontFamily{U}{peut}{}
\DeclareFontShape{U}{peut}{m} {n} {<-> zpeurt} {}
\DeclareFontShape{U}{peut}{m} {sc}{<-> ssub * peut/m/n} {}
\DeclareFontShape{U}{peut}{m} {sl}{<-> zpeurot}{}
\DeclareFontShape{U}{peut}{m} {it}{<-> zpeurit}{}
\DeclareFontShape{U}{peut}{b} {n} {<-> zpeubt} {}
\DeclareFontShape{U}{peut}{b} {sc}{<-> ssub * peut/b/n} {}
\DeclareFontShape{U}{peut}{b} {sl}{<-> zpeubot}{}
\DeclareFontShape{U}{peut}{b} {it}{<-> zpeubit}{}
\DeclareFontShape{U}{peut}{bx}{n} {<-> ssub * peut/b/n} {}
\DeclareFontShape{U}{peut}{bx}{sc}{<-> ssub * peut/b/n} {}
\DeclareFontShape{U}{peut}{bx}{sl}{<-> ssub * peut/b/sl}{}
\DeclareFontShape{U}{peut}{bx}{it}{<-> ssub * peut/b/it}{}
\endinput
```

We install the map file `peu.map` as well as all `afm`, `tfm`, `pfb`, and `fd` files in the local Tex tree as explained in the first tutorial 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 construct like this one:

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

So let's 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}
\RequirePackage{textcomp}
\DeclareRobustCommand{\eurrm}{{%
  \fontencoding{U}\fontfamily{peu}\selectfont\char 128}}
\DeclareRobustCommand{\eursf}{{%
  \fontencoding{U}\fontfamily{peus}\selectfont\char 128}}
\DeclareRobustCommand{\eurtt}{{%
  \fontencoding{U}\fontfamily{peut}\selectfont\char 128}}
```

We define three commands, `\eurrm`, `\eursf`, and `\eurtt`, which typeset a serif, sans serif, and monospaced euro symbol respectively. Note the additional braces to keep the font change local.

```
\DeclareOption{sans}{\def\eur\eursf}
\DeclareOption{serif}{\def\eur\eurrm}
\DeclareOption{mono}{\def\eur\eurtt}
\DeclareOption{textcomp}{%
  \PackageInfo{peufonts}{Hijacking '\string\texteuro'}%
  \def\texteuro{\eur}}
```

```
\ExecuteOptions{sans}
\ProcessOptions
\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 19. 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 (ISO8859-15) 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 preload `textcomp` in line 3 to ensure that it is always loaded before `peufonts`.

IV.4 Euro symbol taken from external text font

There is yet another way to get the euro symbol for a font that does not provide one by default. Suppose we have an external text font including a euro symbol that would go reasonably well with our copy of Sabon. If this euro symbol is uncoded but labeled correctly, we could simply add the text font to the input file list of the respective `\installfont` commands as shown in section IV.3 and then proceed as outlined in section IV.1. What if it is encoded as the currency symbol in the external text font? In this case, we take an approach that is based on section IV.2 with some minor adjustments. Let's assume we have a copy of Bitstream Classical Garamond. Since Classical Garamond is Bitstream's take on Sabon, the euro symbol of this typeface will obviously go quite well with our install of Sabon. The syntax of the `\installfont` commands will look like this:

```
\installfont{psbr8c}{psbr8r,unsetcur,bsbr8r,reseteur,psbr8r,textcomp}{ts1}{TS1}{bsb}{m}{n}{}
```

`psb` is Adobe Sabon, `bsb` is Bitstream Classical Garamond, and `reseteur.mtx` has been discussed in section IV.2. In this case, we need an additional metric file, called `unsetcur.mtx` here, that clears the currency slot before `bsbr8r.afm` is read. Without this additional step, the euro symbol found in the currency slot of `bsbr8r.afm` would be discarded as `psbr8r.afm` has already provided this symbol. `reseteur.mtx` would then move the currency symbol found in `psbr8r.afm` to the euro slot, which is obviously not what we want. We need to clear the currency slot using `unsetcur.mtx`, which is quite simple:

```
\relax
\metrics
\unsetglyph{currency}
\endmetrics
```

With this additional resource, what happens is this: `psbr8r.afm` is read and processed, the currency slot is cleared by `unsetcur.mtx`, then `bsbr8r.afm` is

read, filling the currency slot with its euro glyph (which is encoded as the currency symbol in `bsbr8r.afm`). Our metric file `reseteur.mtx` will then move the euro symbol found in `bsbr8r.afm` to the euro slot and clear the currency slot. After that, we read `psbr8r.afm` again to get the original Adobe Sabon currency symbol of back. Our virtual font will now contain all glyphs found in Adobe Sabon plus the euro symbol of Bitstream Classical Garamond, all properly encoded. Note that, for this to work, we need a complete install of Bitstream Classical Garamond, including map files for dvips and pdftex, in addition to the steps outlined above.

TUTORIAL V

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 canonical 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.

v.1 Simple fontinst file

As usual, our file begins with a typical header setting up some common font substitutions (2–3). While the Janson expert package provides optical small caps for the regular weight, the bold expert fonts do not contain optical small

caps. For the bold series, we have to make do with mechanical small caps. We use a scale factor of 0.72 (4):

```
\input fontinst.sty
\substitutesilent{bx}{b}
\substitutesilent{sc}{n}
\setint{smallcapsscale}{720}
```

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 (5) plus a second family featuring expert glyphs with hanging figures (6):

```
\latinfamily{mjnx}{}
\latinfamily{mjnj}{}
\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. These sets always require a fontinst file using low-level commands such as the one discussed in tutorial III.

v.2 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 use them, you will need to use low-level fontinst commands, at least for parts of the file. But before we start with our verbose fontinst file, let's 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 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 (1–5):

```
1  \input fontinst.sty
2  \substitutesilent{bx}{b}
3  \substitutesilent{sc}{n}
4  \setint{smallcapsscale}{720}
5  \setint{slant}{167}
```

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 scale factor of 0.72 for that (4).

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

We reencode (6–9) and slant (10–11) the basic fonts as usual. Expert fonts do not require any reencoding, but we do need slanted variants of them as well:

```
12 \transformfont{mjnro8x}{\slantfont{\int{slant}}{\fromafm{mjnr8x}}}
13 \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.

```
14 \installfonts
15 \installfamily{T1}{mjnx}{}
16 \installfamily{TS1}{mjnx}{}
17 \installfont{mjnr9e}{mjnr8r,mjnr8x,latin}{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 77 of this guide for additional hints.

```
\installfont{mjnrc9e}{mjnr8r,mjnr8x,latinsc}{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 positions of the lowercase alphabet in our T1 encoded virtual font. Instead of `latin.mtx` we use the special metric file `latinsc.mtx` in this case. The remaining virtual fonts of the `mjnx` family are built as expected:

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

Since the bold expert font does not provide small caps, we create mechanical ones. The `t1c.etx` encoding vector will deal with that transparently, but we have to make sure that the regular `latin.mtx` metric file is read here since there are no optical small caps in the raw font.

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

That's it for T1 encoding. Creating TS1 encoded virtual fonts featuring expert glyphs is pretty straightforward. We simply add the expert fonts to the input file list. Note the encoding suffix of the virtual fonts. We use `9c` instead of `8c` to indicate that the virtual fonts feature expert glyphs:

```
\installfont{mjnr9c}{mjnr8r,mjnr8x,textcomp}{ts1}{TS1}{mjnx}{m}{n}{}
\installfont{mjnro9c}{mjnro8r,mjnro8x,textcomp}{ts1}{TS1}{mjnx}{m}{sl}{}
\installfont{mjnri9c}{mjnri8r,mjnri8x,textcomp}{ts1}{TS1}{mjnx}{m}{it}{}
\installfont{mjnb9c}{mjnb8r,mjnb8x,textcomp}{ts1}{TS1}{mjnx}{b}{n}{}
\installfont{mjnbo9c}{mjnbo8r,mjnbo8x,textcomp}{ts1}{TS1}{mjnx}{b}{sl}{}
\installfont{mjnbi9c}{mjnbi8r,mjnbi8x,textcomp}{ts1}{TS1}{mjnx}{b}{it}{}
\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,latin}{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 positions for figures. We have to keep in mind that the regular encoding vector for T1 encoding (`t1.etx`) references the figures as `zero`, `one`, `two` while

the hanging ('old style') figures in the expert font (which we want to be available by default) are labeled `zerooldstyle`, `oneoldstyle` and so on. In order to arrange the glyphs according to our wishes, we could read the regular font, clear the figures, read the expert font and rename the 'old style' figures. In this case, however, there is a simpler way: we use the special encoding vector `t1j.etx` which is essentially equivalent to `t1.etx` but automatically appends the suffix `oldstyle` to all figures.

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

We have regular optical small caps, so we use the metric file `latinsc.mtx` here. Instead of `t1c.etx` we use the encoding file `t1cj.etx` to make hanging figures the default. The remaining virtual fonts are built like the upright shape (34):

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

There are no optical small caps in the bold-weight expert fonts. Thus, when generating the bold small caps font, we use the metric file `latin.mtx` and the encoding file `t1cj.etx` to create mechanical small caps.

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

v.3 Inferior and superior figures

Expert fonts usually provide superior and inferior figures which can be combined with a special slash called a 'solidus' to typeset arbitrary text fractions like ½ or 17⁄53. Please note that they 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.[1] 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 positions 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's find out what `t1j.etx` does in detail and try to modify this approach according to our needs. This is `t1j.etx`:

```
\relax
\encoding
```

1. 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.

```
\setcommand\lc#1#2{#2}
\setcommand\uc#1#2{#1}
\setcommand\lctop#1#2{#2}
\setcommand\uctop#1#2{#1}
\setcommand\lclig#1#2{#2}
\setcommand\uclig#1#2{#1}
\setcommand\digit#1{#1oldstyle}
\inputetx{T1}
\endencoding
```

As you can see, `t1j.etx` is short. It does not define any encoding slots. All it does is predefine a few macros and use `\inputetx` to load `t1.etx` afterwards. The relevant part (and the only point at which it differs from what `t1.etx` does in this respect) is the line defining the `\digit` macro. To understand this mechanism, we need to take a 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 of figures are not given verbatim, they are used as an argument to the `\digit` macro. The default definition of this macro as given in `t1.etx` looks like this:

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

This means that the glyph labeled as `one` in the `afm` file will end up in the encoding position for the numeral 1 in the virtual font – and so on. `t1j.etx` predefines 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 position for the numeral 1 in the T1 encoded virtual font. When taking a look at 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 modify `t1j.etx` accordingly to create encoding vectors incorporating inferior and superior figures. Hence our encoding vector for T1 encoded fonts

featuring inferior figures (`t10.etx`, read: t-one-zero since 0 is the Fontname code for inferior figures) should look like this:

```
\relax
\encoding
\setcommand\lc#1#2{#2}
\setcommand\uc#1#2{#1}
\setcommand\lctop#1#2{#2}
\setcommand\uctop#1#2{#1}
\setcommand\lclig#1#2{#2}
\setcommand\uclig#1#2{#1}
\setcommand\digit#1{#1inferior}
\inputetx{t1}
\endencoding
```

All we need to do in `t10.etx` is use `\setcommand` to predefine the `\digit` macro as follows:

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

This will add the suffix `inferior` to all digits. For superior figures, the approach is similar. We create an encoding vector called `t11.etx` (read: t-one-one since 1 is the Fontname code for superior figures):

```
\relax
\encoding
\setcommand\lc#1#2{#2}
\setcommand\uc#1#2{#1}
\setcommand\lctop#1#2{#2}
\setcommand\uctop#1#2{#1}
\setcommand\lclig#1#2{#2}
\setcommand\uclig#1#2{#1}
\setcommand\digit#1{#1superior}
\inputetx{t1}
\endencoding
```

With `t10.etx` and `t11.etx` at hand, we may now create the font families `mjn0` and `mjn1` pretty much like we have generated `mjnj`. Let's put the new encoding vectors in our working directory and go back to the fontinst file:

```
\installfonts
\installfamily{T1}{mjn0}{}
\installfont{mjnr09e}{mjnr8r,mjnr8x,latin}{t10}{T1}{mjn0}{m}{n}{}
```

We add the code `0` to the name of the virtual font (`mjnr09e` here), use the encoding vector `t10.etx`, and adapt the NFSS font declaration (in this case `T1/mjn0/m/n`) accordingly. Other than that, the virtual fonts of the `mjn0` family are generated in the usual way:

```
\installfont{mjnro09e}{mjnro8r,mjnro8x,latin}{t10}{T1}{mjn0}{m}{sl}{}
\installfont{mjnri09e}{mjnri8r,mjnri8x,latin}{t10}{T1}{mjn0}{m}{it}{}
\installfont{mjnb09e}{mjnb8r,mjnb8x,latin}{t10}{T1}{mjn0}{b}{n}{}
\installfont{mjnbo09e}{mjnbo8r,mjnbo8x,latin}{t10}{T1}{mjn0}{b}{sl}{}
\installfont{mjnbi09e}{mjnbi8r,mjnbi8x,latin}{t10}{T1}{mjn0}{b}{it}{}
\endinstallfonts
```

Our fontinst file will omit the small caps shape to save some disk space. We have included a global shape substitution for the `sc` shape in the header, so `mjn0/sc` will be substituted by `mjn0/n` via a silent substitution in the font definition file. Since the figures of upright and small caps shapes do not differ at all and since we need the `mjn0` family for figures only, we can safely omit the small caps shape. For the `mjn1` family, we adapt the names of the virtual fonts (adding the Fontname code 1 to indicate superior figures), the encoding vector (`t11.etx`), and the NFSS declaration in a similar way:

```
\installfonts
\installfamily{T1}{mjn1}{}
\installfont{mjnr19e}{mjnr8r,mjnr8x,latin}{t11}{T1}{mjn1}{m}{n}{}
\installfont{mjnro19e}{mjnro8r,mjnro8x,latin}{t11}{T1}{mjn1}{m}{sl}{}
\installfont{mjnri19e}{mjnri8r,mjnri8x,latin}{t11}{T1}{mjn1}{m}{it}{}
\installfont{mjnb19e}{mjnb8r,mjnb8x,latin}{t11}{T1}{mjn1}{b}{n}{}
\installfont{mjnbo19e}{mjnbo8r,mjnbo8x,latin}{t11}{T1}{mjn1}{b}{sl}{}
\installfont{mjnbi19e}{mjnbi8r,mjnbi8x,latin}{t11}{T1}{mjn1}{b}{it}{}
\endinstallfonts
\bye
```

This is our complete fontinst file which will provide us with four font families: `mjnx`, `mjnj`, `mjn0`, and `mjn1`. Virtual fonts in T1 encoding are provided for all families, but TS1 encoded ones for `mjnx` only since they would be identical for all of our four font families anyway. Thus, we can simply use substitutions instead of creating duplicate virtual fonts. As mentioned in the third tutorial, however, fontinst does not provide family substitutions. We have to write font definition files manually to ensure that the lacking TS1 encoded fonts are substituted by their counterparts of the `mjnx` family so that the `textcomp` package will work with all of them. For the `mjnj` family, our font definition file for TS1 encoding (`ts1mjnj.fd`) looks like this:

```
\ProvidesFile{ts1mjnj.fd}
\DeclareFontFamily{TS1}{mjnj}{}
\DeclareFontShape{TS1}{mjnj}{m} {n} {<-> ssub * mjnx/m/n} {}
\DeclareFontShape{TS1}{mjnj}{m} {sc}{<-> ssub * mjnx/m/n} {}
\DeclareFontShape{TS1}{mjnj}{m} {sl}{<-> ssub * mjnx/m/sl}{}
\DeclareFontShape{TS1}{mjnj}{m} {it}{<-> ssub * mjnx/m/it}{}
\DeclareFontShape{TS1}{mjnj}{b} {n} {<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjnj}{b} {sc}{<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjnj}{b} {sl}{<-> ssub * mjnx/b/sl}{}
\DeclareFontShape{TS1}{mjnj}{b} {it}{<-> ssub * mjnx/b/it}{}
\DeclareFontShape{TS1}{mjnj}{bx}{n} {<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjnj}{bx}{sc}{<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjnj}{bx}{sl}{<-> ssub * mjnx/b/sl}{}
\DeclareFontShape{TS1}{mjnj}{bx}{it}{<-> ssub * mjnx/b/it}{}
\endinput
```

This is the equivalent for `mjn0`, the file `ts1mjn0.fd`:

```
\ProvidesFile{ts1mjn0.fd}
\DeclareFontFamily{TS1}{mjn0}{}
\DeclareFontShape{TS1}{mjn0}{m} {n} {<-> ssub * mjnx/m/n} {}
\DeclareFontShape{TS1}{mjn0}{m} {sc}{<-> ssub * mjnx/m/n} {}
\DeclareFontShape{TS1}{mjn0}{m} {sl}{<-> ssub * mjnx/m/sl}{}
```

```
\DeclareFontShape{TS1}{mjn0}{m} {it}{<-> ssub * mjnx/m/it}{}
\DeclareFontShape{TS1}{mjn0}{b} {n} {<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjn0}{b} {sc}{<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjn0}{b} {sl}{<-> ssub * mjnx/b/sl}{}
\DeclareFontShape{TS1}{mjn0}{b} {it}{<-> ssub * mjnx/b/it}{}
\DeclareFontShape{TS1}{mjn0}{bx}{n} {<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjn0}{bx}{sc}{<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjn0}{bx}{sl}{<-> ssub * mjnx/b/sl}{}
\DeclareFontShape{TS1}{mjn0}{bx}{it}{<-> ssub * mjnx/b/it}{}
\endinput
```

And finally, `ts1mjn1.fd` for the `mjn1` family:

```
\ProvidesFile{ts1mjn1.fd}
\DeclareFontFamily{TS1}{mjn1}{}
\DeclareFontShape{TS1}{mjn1}{m} {n} {<-> ssub * mjnx/m/n} {}
\DeclareFontShape{TS1}{mjn1}{m} {sc}{<-> ssub * mjnx/m/n} {}
\DeclareFontShape{TS1}{mjn1}{m} {sl}{<-> ssub * mjnx/m/sl}{}
\DeclareFontShape{TS1}{mjn1}{m} {it}{<-> ssub * mjnx/m/it}{}
\DeclareFontShape{TS1}{mjn1}{b} {n} {<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjn1}{b} {sc}{<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjn1}{b} {sl}{<-> ssub * mjnx/b/sl}{}
\DeclareFontShape{TS1}{mjn1}{b} {it}{<-> ssub * mjnx/b/it}{}
\DeclareFontShape{TS1}{mjn1}{bx}{n} {<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjn1}{bx}{sc}{<-> ssub * mjnx/b/n} {}
\DeclareFontShape{TS1}{mjn1}{bx}{sl}{<-> ssub * mjnx/b/sl}{}
\DeclareFontShape{TS1}{mjn1}{bx}{it}{<-> ssub * mjnx/b/it}{}
\endinput
```

As far as Latex is concerned, our setup is complete now. We still need a map file, though.

v.4 The map file

The syntax of map files has been discussed in detail before. The lines for the basic font set should therefore be obvious:

```
mjnr8r  JansonMT            "TeXBase1Encoding ReEncodeFont"           <8r.enc <mjnr8a.pfb
mjnri8r JansonMT-Italic     "TeXBase1Encoding ReEncodeFont"           <8r.enc <mjnri8a.pfb
mjnb8r  JansonMT-Bold       "TeXBase1Encoding ReEncodeFont"           <8r.enc <mjnb8a.pfb
mjnbi8r JansonMT-BoldItalic "TeXBase1Encoding ReEncodeFont"           <8r.enc <mjnbi8a.pfb
mjnro8r JansonMT      "0.167 SlantFont TeXBase1Encoding ReEncodeFont" <8r.enc <mjnr8a.pfb
mjnbo8r JansonMT-Bold "0.167 SlantFont TeXBase1Encoding ReEncodeFont" <8r.enc <mjnb8a.pfb
```

Mapping lines for expert fonts are simpler because there is no need for reencoding and no encoding vector will be included:

```
mjnr8x  JansonExpertMT                                          <mjnr8x.pfb
mjnri8x JansonExpertMT-Italic                                   <mjnri8x.pfb
mjnb8x  JansonExpertMT-Bold                                     <mjnb8x.pfb
mjnbi8x JansonExpertMT-BoldItalic                               <mjnbi8x.pfb
```

We do need slanted expert fonts as well, though:

```
mjnro8x JansonExpertMT      "0.167 SlantFont"                   <mjnr8x.pfb
mjnbo8x JansonExpertMT-Bold "0.167 SlantFont"                   <mjnb8x.pfb
```

This is our complete map file for Monotype Janson, `mjn.map`.

v.5 The style file

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

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{janson}
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\RequirePackage{nfssext}
\DeclareOption{oldstyle}{\renewcommand{\rmdefault}{mjnj}}
\DeclareOption{lining}{\renewcommand{\rmdefault}{mjnx}}
\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}
\newcommand{\exfs@try@family}[1]{%
  \expandafter\ifx\csname\f@encoding+#1\endcsname\relax
    \let\exfs@tempa\relax
    \begingroup
      \fontfamily{#1}\try@load@fontshape
      \expandafter\ifx\csname\f@encoding+#1\endcsname\relax
        \PackageWarning{nfssext}{%
          Font family \f@encoding/#1 unavailable,\MessageBreak
          ignoring font switch}%
      \else
        \gdef\exfs@tempa{\fontfamily{#1}\selectfont}%
      \fi
    \endgroup
    \exfs@tempa
  \else
    \fontfamily{#1}\selectfont
  \fi}
\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}
  \exfs@try@family{\expandafter\exfs@get@base\f@family\@nil x}}
```

The first part remains unchanged, but the `\lnstyle` macro has to cater for two font families which, depending on the font, may contain the lining figures: a basic font family with a three-character code (23) or an expert family with a four-character code ending with the letter `x` (24). Since we check if the requested font family is available, this will work for both Sabon and Janson.

```
\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, `\instyle` and `\sustyle`, which activate these figures locally.

v.6 Using the 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's 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 ½. When looking at an expert font in a font editor, you will see that expert fonts contain a few (fixed) text fractions. Some of them are included in TS1 encoding and supported by the `textcomp` package, but typing rather long commands such as `\textthreequarters` is not exactly convenient. Since there are only nine of them they are not terribly 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 ¹²⁄₃₄. When using the compact font switching macros in `nfssext.sty` instead of the standard NFSS commands, the macro would look like this:

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

What about using superior figures as footnote numbers? In this case, we need to redefine `\@makefnmark`. This is the 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}}
```

If you want to put a definition of `\@makefnmark` in a regular Latex input file (as opposed to a style file), do not forget to enclose it in `\makeatletter` and `\makeatother`.

TUTORIAL VI

EXPERT FONT SETS, EXTENDED SETUP

In this tutorial we will combine the tutorials III and V, that is, we will install a very comprehensive 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 package:

```
pmnr8a    Minion-Regular                A    143    morg____
pmnri8a   Minion-Italic                 A    143    moi_____
pmns8a    Minion-Semibold               A    143    mosb____
pmnsi8a   Minion-SemiboldItalic         A    143    mosbi___
pmnb8a    Minion-Bold                   A    143    mob_____
pmnbi8a   Minion-BoldItalic             A    143    mobi____
pmnc8a    Minion-Black                  A    143    mobl____
pmnrc8a   Minion-RegularSC              A    144    mosc____
pmnric8a  Minion-ItalicSC               A    144    moisc___
pmnriw7a  Minion-SwashItalic            A    144    moswi___
pmnsc8a   Minion-SemiboldSC             A    144    mosbs___
pmnsic8a  Minion-SemiboldItalicSC       A    144    mosic___
pmnsiw7a  Minion-SwashSemiboldItalic    A    144    mossb___
pmnbj8a   Minion-BoldOsF                A    144    mobos___
pmnbij8a  Minion-BoldItalicOsF          A    144    mobio___
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____
```

In addition to these 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 text set and we can omit them here.

VI.1 The fontinst file

With a very comprehensive set of fonts at our disposal, we will be fastidious. There will be no computed glyph shapes – no mechanical small caps and no slanted fonts – making this setup suitable for professional typesetting. Note that the bold and black weights do not feature optical small caps. Even though there are expert fonts for these weights, they do not contain any small caps glyphs. The bold weight is merely intended for applications requiring a very

strong contrast, for example to highlight the keywords in a dictionary, while the black weight of a typeface like Minion is only relevant for certain types of display work. Without further ado, we start off as usual:

```
\nonstopmode
\input fontinst.sty
\substitutesilent{bx}{sb}
\substitutenoisy{sc}{n}
\substitutenoisy{si}{it}
```

When looking at our font set it is obvious that semibold should be used as the main bold weight, hence we make it the default by substituting `sb` for `bx`. Since the bold and black weights do not feature optical small caps, we add appropriate substitutions for the `sc` and `si` (italic small caps) shapes.

```
\transformfont{pmnr8r}{\reencodefont{8r}{\fromafm{pmnr8a}}}
\transformfont{pmnrc8r}{\reencodefont{8r}{\fromafm{pmnrc8a}}}
\transformfont{pmnri8r}{\reencodefont{8r}{\fromafm{pmnri8a}}}
\transformfont{pmnric8r}{\reencodefont{8r}{\fromafm{pmnric8a}}}
\transformfont{pmns8r}{\reencodefont{8r}{\fromafm{pmns8a}}}
\transformfont{pmnsc8r}{\reencodefont{8r}{\fromafm{pmnsc8a}}}
\transformfont{pmnsi8r}{\reencodefont{8r}{\fromafm{pmnsi8a}}}
\transformfont{pmnsic8r}{\reencodefont{8r}{\fromafm{pmnsic8a}}}
\transformfont{pmnb8r}{\reencodefont{8r}{\fromafm{pmnb8a}}}
\transformfont{pmnbi8r}{\reencodefont{8r}{\fromafm{pmnbi8a}}}
\transformfont{pmnc8r}{\reencodefont{8r}{\fromafm{pmnc8a}}}
```

Reencoding: you know the drill. We reencode all base fonts using Adobe Standard as their native encoding. While the swash fonts are based on Adobe Standard as well, they contain a special set of glyphs and are handled like expert fonts.

```
\installfonts
\installfamily{T1}{pmnx}{}
\installfamily{TS1}{pmnx}{}
\installfont{pmnr9e}{pmnr8r,pmnr8x,latin}{t1}{T1}{pmnx}{m}{n}{}
\installfont{pmnri9e}{pmnri8r,pmnri8x,latin}{t1}{T1}{pmnx}{m}{it}{}
```

The setup of the upright and italic shapes does not differ from tutorial v.

```
\installfont{pmnrc9e}%
  {kernoff,pmnr8r,pmnr8x,kernon,glyphoff,pmnrc8r,glyphon,resetsc,latinsc}%
  {t1c}{T1}{pmnx}{m}{sc}{}
\installfont{pmnric9e}%
  {kernoff,pmnri8r,pmnri8x,kernon,glyphoff,pmnric8r,glyphon,resetsc,latinsc}%
  {t1c}{T1}{pmnx}{m}{si}{}
```

There is one problem with taking optical small caps from an expert font as demonstrated in tutorial v: there are no kerning pairs between the uppercase alphabet and the small caps replacing the lowercase letters. Without dedicated small caps fonts there is nothing we can do about that. Now that we have both expert and small caps fonts, however, we could take an approach similar to the one outlined in tutorial iii, adding the expert fonts 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. 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 so that PDF and Postscript files with embedded fonts will be slightly smaller. The input file list should be more or less self-explanatory: we use `kernoff.mtx` to ignore the kerning data while reading the respective base and expert fonts. Then we add `kernon.mtx` to re-activate the kerning commands and a special metric file called `glyphoff.mtx` to ignore the glyph data. After that, we read the corresponding small caps font and re-activate the glpyh commands. Finally, we add `resetsc.mtx` as well as `latinsc.mtx`. Our encoding file is `t1c.etx`.

```
\installfont{pmns9e}{pmns8r,pmns8x,latin}{t1}{T1}{pmnx}{sb}{n}{}
\installfont{pmnsi9e}{pmnsi8r,pmnsi8x,latin}{t1}{T1}{pmnx}{sb}{it}{}
\installfont{pmnsc9e}%
  {kernoff,pmns8r,pmns8x,kernon,glyphoff,pmnsc8r,glyphon,resetsc,latinsc}%
  {t1c}{T1}{pmnx}{m}{sc}{}
\installfont{pmnsic9e}%
  {kernoff,pmnsi8r,pmnsi8x,kernon,glyphoff,pmnsic8r,glyphon,resetsc,latinsc}%
  {t1c}{T1}{pmnx}{m}{si}{}
```

We repeat these steps for the semibold weight.

```
\installfont{pmnb9e}{pmnb8r,pmnb8x,latin}{t1}{T1}{pmnx}{b}{n}{}
\installfont{pmnbi9e}{pmnbi8r,pmnbi8x,latin}{t1}{T1}{pmnx}{b}{it}{}
\installfont{pmnc9e}{pmnc8r,pmnc8x,latin}{t1}{T1}{pmnx}{eb}{n}{}
```

The bold and black weights are handled differently because there are no optical small caps. We will simply omit the respective shapes. The black weight will be mapped to the `eb` series of the NFSS. After finishing T1 encoding we continue with TS1. Our setup for TS1 encoding does not differ from tutorial v either:

```
\installfont{pmnr9c}{pmnr8r,pmnr8x,textcomp}{ts1}{TS1}{pmnx}{m}{n}{}
\installfont{pmnri9c}{pmnri8r,pmnri8x,textcomp}{ts1}{TS1}{pmnx}{m}{it}{}
\installfont{pmns9c}{pmns8r,pmns8x,textcomp}{ts1}{TS1}{pmnx}{sb}{n}{}
\installfont{pmnsi9c}{pmnsi8r,pmnsi8x,textcomp}{ts1}{TS1}{pmnx}{sb}{it}{}
\installfont{pmnb9c}{pmnb8r,pmnb8x,textcomp}{ts1}{TS1}{pmnx}{b}{n}{}
\installfont{pmnbi9c}{pmnbi8r,pmnbi8x,textcomp}{ts1}{TS1}{pmnx}{b}{it}{}
\installfont{pmnc9c}{pmnc8r,pmnc8x,textcomp}{ts1}{TS1}{pmnx}{eb}{n}{}
\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,latin}{t1j}{T1}{pmnj}{m}{n}{}
\installfont{pmnri9d}{pmnri8r,pmnri8x,latin}{t1j}{T1}{pmnj}{m}{it}{}
```

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}
  {kernoff,pmnr8r,pmnr8x,kernon,glyphoff,pmnrc8r,glyphon,resetosf,resetsc,latinsc}%
  {t1cj}{T1}{pmnj}{m}{sc}{}
```

```
\installfont{pmnric9d}
  {kernoff,pmnri8r,pmnri8x,kernon,glyphoff,pmnric8r,glyphon,resetosf,resetsc,latinsc}%
  {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 we use the encoding file `t1cj.etx` and add the metric file `resetosf.mtx`.

```
\installfont{pmns9d}{pmns8r,pmns8x,latin}{t1j}{T1}{pmnj}{sb}{n}{}
\installfont{pmnsi9d}{pmnsi8r,pmnsi8x,latin}{t1j}{T1}{pmnj}{sb}{it}{}
\installfont{pmnsc9d}
  {kernoff,pmns8r,pmns8x,kernon,glyphoff,pmnsc8r,glyphon,resetosf,resetsc,latinsc}%
  {t1cj}{T1}{pmnj}{sb}{sc}{}
\installfont{pmnsic9d}
  {kernoff,pmnsi8r,pmnsi8x,kernon,glyphoff,pmnsic8r,glyphon,resetosf,resetsc,latinsc}%
  {t1cj}{T1}{pmnj}{sb}{si}{}
```

Again, we repeat these steps for the semibold weight.

```
\installfont{pmnb9d}{pmnb8r,pmnb8x,latin}{t1j}{T1}{pmnj}{b}{n}{}
\installfont{pmnbi9d}{pmnbi8r,pmnbi8x,latin}{t1j}{T1}{pmnj}{b}{it}{}
\installfont{pmnc9d}{pmnc8r,pmnc8x,latin}{t1j}{T1}{pmnj}{eb}{n}{}
\endinstallfonts
```

The bold and black weights are essentially handled like those of the `pmnx` family, only differing in the choice of the encoding file.

```
\installfonts
\installfamily{T1}{pmn0}{}
\installfont{pmnr09e}{pmnr8r,pmnr8x,latin}{t10}{T1}{pmn0}{m}{n}{}
\installfont{pmnri09e}{pmnri8r,pmnri8x,latin}{t10}{T1}{pmn0}{m}{it}{}
\installfont{pmns09e}{pmns8r,pmns8x,latin}{t10}{T1}{pmn0}{sb}{n}{}
\installfont{pmnsi09e}{pmnsi8r,pmnsi8x,latin}{t10}{T1}{pmn0}{sb}{it}{}
\installfont{pmnb09e}{pmnb8r,pmnb8x,latin}{t10}{T1}{pmn0}{b}{n}{}
\installfont{pmnbi09e}{pmnbi8r,pmnbi8x,latin}{t10}{T1}{pmn0}{b}{it}{}
\installfont{pmnc09e}{pmnc8r,pmnc8x,latin}{t10}{T1}{pmn0}{eb}{n}{}
\endinstallfonts
```

In addition to `pmnx` and `pmnj`, we also add dedicated font families incorporating inferior and superior figures. Since inferior figures are found in the expert fonts, our approach here does not differ from the one introduced in section v.3.

```
\installfonts
\installfamily{T1}{pmn1}{}
\installfont{pmnr19e}{pmnr8r,pmnr8x,latin}{t11}{T1}{pmn1}{m}{n}{}
\installfont{pmnri19e}{pmnri8r,pmnri8x,latin}{t11}{T1}{pmn1}{m}{it}{}
\installfont{pmns19e}{pmns8r,pmns8x,latin}{t11}{T1}{pmn1}{sb}{n}{}
\installfont{pmnsi19e}{pmnsi8r,pmnsi8x,latin}{t11}{T1}{pmn1}{sb}{it}{}
\installfont{pmnb19e}{pmnb8r,pmnb8x,latin}{t11}{T1}{pmn1}{b}{n}{}
\installfont{pmnbi19e}{pmnbi8r,pmnbi8x,latin}{t11}{T1}{pmn1}{b}{it}{}
\installfont{pmnc19e}{pmnc8r,pmnc8x,latin}{t11}{T1}{pmn1}{eb}{n}{}
\endinstallfonts
```

The same holds true for superior figures.

```
\installfonts
\installfamily{T1}{pmnw}{}
\installfont{pmnriw9d}{pmnri8r,unsetcaps,pmnriw7a,pmnri8x,latin}{t1j}{T1}{pmnw}{m}{it}{}
\installfont{pmnsiw9d}{pmnsi8r,unsetcaps,pmnsiw7a,pmnsi8x,latin}{t1j}{T1}{pmnw}{sb}{it}{}
```

```
\endinstallfonts
\bye
```

In order to incorporate the italic swashes we create an additional font family called `pmnw`. We read the respective base font and clear the slots of the capital letters using `unsetcaps.mtx`. Note that `unsetcaps.mtx` was apparently written with OT1 encoding in mind as it merely clears the slots of the English alphabet. The slots of capital letters with an accent are not cleared. In this case, however, this does not matter as Minion merely provides swash capitals for the English alphabet anyway, so the stock `unsetcaps.mtx` file is sufficient for the job. After that we add the respective swash font and finally the expert font as usual. We employ `t1j.etx` to use hanging figures with the swash family. In this particular case using OT1 encoding might actually be useful as this encoding constructs accented letters from the English alphabet. Here is the respective part of the file for OT1 encoding:

```
\installfonts
\installfamily{OT1}{pmnw}{}
\installfont{pmnriw9o}{pmnri8r,unsetcaps,pmnriw7a,pmnri8x,latin}{ot1j}{OT1}{pmnw}{m}{it}{}
\installfont{pmnsiw9o}{pmnsi8r,unsetcaps,pmnsiw7a,pmnsi8x,latin}{ot1j}{OT1}{pmnw}{sb}{it}{}
\endinstallfonts
```

Note that using a setup including OT1 encoding for one font family only will require switching the encoding explicitly when selecting the swash fonts:

```
\fontencoding{OT1}\fontfamily{pmnw}\selectfont
```

The `pmnw` family as generated by fontinst will only cover two shapes in either case. Since fontinst does not support family substitutions we cannot take the missing shapes from `pmnj` in the fontinst file. We have to edit the respective font definition file, `t1pmnw.fd`, after running fontinst. For T1 encoding it should look as follows:

```
\ProvidesFile{t1pmnw.fd}
\DeclareFontFamily{T1}{pmnw}{}
\DeclareFontShape{T1}{pmnw}{m} {n} {<-> ssub * pmnj/m/n}  {}
\DeclareFontShape{T1}{pmnw}{m} {sc}{<-> ssub * pmnj/m/sc} {}
\DeclareFontShape{T1}{pmnw}{m} {sl}{<-> ssub * pmnj/m/it} {}
\DeclareFontShape{T1}{pmnw}{m} {it}{<-> pmnriw9d}         {}
\DeclareFontShape{T1}{pmnw}{m} {si}{<-> ssub * pmnj/m/si} {}
\DeclareFontShape{T1}{pmnw}{sb}{n} {<-> ssub * pmnj/sb/n} {}
\DeclareFontShape{T1}{pmnw}{sb}{sc}{<-> ssub * pmnj/sb/sc}{}
\DeclareFontShape{T1}{pmnw}{sb}{sl}{<-> ssub * pmnj/sb/it}{}
\DeclareFontShape{T1}{pmnw}{sb}{it}{<-> pmnsiw9d}         {}
\DeclareFontShape{T1}{pmnw}{sb}{si}{<-> ssub * pmnj/sb/si}{}
\DeclareFontShape{T1}{pmnw}{b} {n} {<-> ssub * pmnj/b/n}  {}
\DeclareFontShape{T1}{pmnw}{b} {sc}{<-> ssub * pmnj/b/sc} {}
\DeclareFontShape{T1}{pmnw}{b} {sl}{<-> ssub * pmnj/b/it} {}
\DeclareFontShape{T1}{pmnw}{b} {it}{<-> ssub * pmnj/b/it} {}
\DeclareFontShape{T1}{pmnw}{b} {si}{<-> ssub * pmnj/b/si} {}
\DeclareFontShape{T1}{pmnw}{eb}{n} {<-> ssub * pmnj/eb/n} {}
\DeclareFontShape{T1}{pmnw}{eb}{sc}{<-> ssub * pmnj/eb/sc}{}
\DeclareFontShape{T1}{pmnw}{eb}{sl}{<-> ssub * pmnj/eb/it}{}
\DeclareFontShape{T1}{pmnw}{eb}{it}{<-> ssub * pmnj/eb/it}{}
```

```
\DeclareFontShape{T1}{pmnw}{eb}{si}{<-> ssub * pmnj/eb/si}{}
\DeclareFontShape{T1}{pmnw}{bx}{n} {<-> ssub * pmnw/sb/n} {}
\DeclareFontShape{T1}{pmnw}{bx}{sc}{<-> ssub * pmnw/sb/sc}{}
\DeclareFontShape{T1}{pmnw}{bx}{sl}{<-> ssub * pmnw/sb/it}{}
\DeclareFontShape{T1}{pmnw}{bx}{it}{<-> ssub * pmnw/sb/it}{}
\DeclareFontShape{T1}{pmnw}{bx}{si}{<-> ssub * pmnw/sb/si}{}
\endinput
```

Only the pmnx family offers TS1 encoded fonts as the glyphs found in this encoding are identical across all font families. To make sure that all font families work as expected, however, we need font definition files containing family substitutions which cannot be defined in a fontinst file. For the pmnj family:

```
\ProvidesFile{ts1pmnj.fd}
\DeclareFontFamily{TS1}{pmnj}{}
\DeclareFontShape{TS1}{pmnj}{m} {n} {<-> ssub * pmnx/m/n}  {}
\DeclareFontShape{TS1}{pmnj}{m} {sc}{<-> ssub * pmnx/m/n}  {}
\DeclareFontShape{TS1}{pmnj}{m} {sl}{<-> ssub * pmnx/m/sl} {}
\DeclareFontShape{TS1}{pmnj}{m} {it}{<-> ssub * pmnx/m/it} {}
\DeclareFontShape{TS1}{pmnj}{sb}{n} {<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmnj}{sb}{sc}{<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmnj}{sb}{sl}{<-> ssub * pmnx/sb/sl}{}
\DeclareFontShape{TS1}{pmnj}{sb}{it}{<-> ssub * pmnx/sb/it}{}
\DeclareFontShape{TS1}{pmnj}{b} {n} {<-> ssub * pmnx/b/n}  {}
\DeclareFontShape{TS1}{pmnj}{b} {sc}{<-> ssub * pmnx/b/n}  {}
\DeclareFontShape{TS1}{pmnj}{b} {sl}{<-> ssub * pmnx/b/sl} {}
\DeclareFontShape{TS1}{pmnj}{b} {it}{<-> ssub * pmnx/b/it} {}
\DeclareFontShape{TS1}{pmnj}{eb}{n} {<-> ssub * pmnx/eb/n} {}
\DeclareFontShape{TS1}{pmnj}{eb}{sc}{<-> ssub * pmnx/eb/n} {}
\DeclareFontShape{TS1}{pmnj}{eb}{sl}{<-> ssub * pmnx/eb/sl}{}
\DeclareFontShape{TS1}{pmnj}{eb}{it}{<-> ssub * pmnx/eb/it}{}
\DeclareFontShape{TS1}{pmnj}{bx}{n} {<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmnj}{bx}{sc}{<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmnj}{bx}{sl}{<-> ssub * pmnx/sb/sl}{}
\DeclareFontShape{TS1}{pmnj}{bx}{it}{<-> ssub * pmnx/sb/it}{}
\endinput
```

For the pmnw family:

```
\ProvidesFile{ts1pmnw.fd}
\DeclareFontFamily{TS1}{pmnw}{}
\DeclareFontShape{TS1}{pmnw}{m} {n} {<-> ssub * pmnx/m/n}  {}
\DeclareFontShape{TS1}{pmnw}{m} {sc}{<-> ssub * pmnx/m/n}  {}
\DeclareFontShape{TS1}{pmnw}{m} {sl}{<-> ssub * pmnx/m/sl} {}
\DeclareFontShape{TS1}{pmnw}{m} {it}{<-> ssub * pmnx/m/it} {}
\DeclareFontShape{TS1}{pmnw}{sb}{n} {<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmnw}{sb}{sc}{<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmnw}{sb}{sl}{<-> ssub * pmnx/sb/sl}{}
\DeclareFontShape{TS1}{pmnw}{sb}{it}{<-> ssub * pmnx/sb/it}{}
\DeclareFontShape{TS1}{pmnw}{b} {n} {<-> ssub * pmnx/b/n}  {}
\DeclareFontShape{TS1}{pmnw}{b} {sc}{<-> ssub * pmnx/b/n}  {}
\DeclareFontShape{TS1}{pmnw}{b} {sl}{<-> ssub * pmnx/b/sl} {}
\DeclareFontShape{TS1}{pmnw}{b} {it}{<-> ssub * pmnx/b/it} {}
\DeclareFontShape{TS1}{pmnw}{eb}{n} {<-> ssub * pmnx/eb/n} {}
\DeclareFontShape{TS1}{pmnw}{eb}{sc}{<-> ssub * pmnx/eb/n} {}
\DeclareFontShape{TS1}{pmnw}{eb}{sl}{<-> ssub * pmnx/eb/sl}{}
\DeclareFontShape{TS1}{pmnw}{eb}{it}{<-> ssub * pmnx/eb/it}{}
\DeclareFontShape{TS1}{pmnw}{bx}{n} {<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmnw}{bx}{sc}{<-> ssub * pmnx/sb/n} {}
```

```
\DeclareFontShape{TS1}{pmnw}{bx}{sl}{<-> ssub * pmnx/sb/sl}{}
\DeclareFontShape{TS1}{pmnw}{bx}{it}{<-> ssub * pmnx/sb/it}{}
\endinput
```

For the pmn0 family:

```
\ProvidesFile{ts1pmn0.fd}
\DeclareFontFamily{TS1}{pmn0}{}
\DeclareFontShape{TS1}{pmn0}{m} {n} {<-> ssub * pmnx/m/n}  {}
\DeclareFontShape{TS1}{pmn0}{m} {sc}{<-> ssub * pmnx/m/n}  {}
\DeclareFontShape{TS1}{pmn0}{m} {sl}{<-> ssub * pmnx/m/sl} {}
\DeclareFontShape{TS1}{pmn0}{m} {it}{<-> ssub * pmnx/m/it} {}
\DeclareFontShape{TS1}{pmn0}{sb}{n} {<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmn0}{sb}{sc}{<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmn0}{sb}{sl}{<-> ssub * pmnx/sb/sl}{}
\DeclareFontShape{TS1}{pmn0}{sb}{it}{<-> ssub * pmnx/sb/it}{}
\DeclareFontShape{TS1}{pmn0}{b} {n} {<-> ssub * pmnx/b/n}  {}
\DeclareFontShape{TS1}{pmn0}{b} {sc}{<-> ssub * pmnx/b/n}  {}
\DeclareFontShape{TS1}{pmn0}{b} {sl}{<-> ssub * pmnx/b/sl} {}
\DeclareFontShape{TS1}{pmn0}{b} {it}{<-> ssub * pmnx/b/it} {}
\DeclareFontShape{TS1}{pmn0}{eb}{n} {<-> ssub * pmnx/eb/n} {}
\DeclareFontShape{TS1}{pmn0}{eb}{sc}{<-> ssub * pmnx/eb/n} {}
\DeclareFontShape{TS1}{pmn0}{eb}{sl}{<-> ssub * pmnx/eb/sl}{}
\DeclareFontShape{TS1}{pmn0}{eb}{it}{<-> ssub * pmnx/eb/it}{}
\DeclareFontShape{TS1}{pmn0}{bx}{n} {<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmn0}{bx}{sc}{<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmn0}{bx}{sl}{<-> ssub * pmnx/sb/sl}{}
\DeclareFontShape{TS1}{pmn0}{bx}{it}{<-> ssub * pmnx/sb/it}{}
\endinput
```

For the pmn1 family:

```
\ProvidesFile{ts1pmn1.fd}
\DeclareFontFamily{TS1}{pmn1}{}
\DeclareFontShape{TS1}{pmn1}{m} {n} {<-> ssub * pmnx/m/n}  {}
\DeclareFontShape{TS1}{pmn1}{m} {sc}{<-> ssub * pmnx/m/n}  {}
\DeclareFontShape{TS1}{pmn1}{m} {sl}{<-> ssub * pmnx/m/sl} {}
\DeclareFontShape{TS1}{pmn1}{m} {it}{<-> ssub * pmnx/m/it} {}
\DeclareFontShape{TS1}{pmn1}{sb}{n} {<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmn1}{sb}{sc}{<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmn1}{sb}{sl}{<-> ssub * pmnx/sb/sl}{}
\DeclareFontShape{TS1}{pmn1}{sb}{it}{<-> ssub * pmnx/sb/it}{}
\DeclareFontShape{TS1}{pmn1}{b} {n} {<-> ssub * pmnx/b/n}  {}
\DeclareFontShape{TS1}{pmn1}{b} {sc}{<-> ssub * pmnx/b/n}  {}
\DeclareFontShape{TS1}{pmn1}{b} {sl}{<-> ssub * pmnx/b/sl} {}
\DeclareFontShape{TS1}{pmn1}{b} {it}{<-> ssub * pmnx/b/it} {}
\DeclareFontShape{TS1}{pmn1}{eb}{n} {<-> ssub * pmnx/eb/n} {}
\DeclareFontShape{TS1}{pmn1}{eb}{sc}{<-> ssub * pmnx/eb/n} {}
\DeclareFontShape{TS1}{pmn1}{eb}{sl}{<-> ssub * pmnx/eb/sl}{}
\DeclareFontShape{TS1}{pmn1}{eb}{it}{<-> ssub * pmnx/eb/it}{}
\DeclareFontShape{TS1}{pmn1}{bx}{n} {<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmn1}{bx}{sc}{<-> ssub * pmnx/sb/n} {}
\DeclareFontShape{TS1}{pmn1}{bx}{sl}{<-> ssub * pmnx/sb/sl}{}
\DeclareFontShape{TS1}{pmn1}{bx}{it}{<-> ssub * pmnx/sb/it}{}
\endinput
```

VI.2 Text ornaments

The Minion expert package includes a dedicated ornament font, `pmnrp.pfb`. As discussed before in section IV.3, we do not really need fontinst when installing symbol fonts. Since no reencoding is required and there are no virtual fonts, `afm2tfm` is sufficient for the job:

```
afm2tfm pmnrp.afm pmnrp.tfm
```

Using the fonts with Latex requires a font definition file, though. Symbol fonts are not based on any particular encoding, so we use the encoding code `U` (uncoded, unknown) in this case. This is `upmnp.fd`:

```
\ProvidesFile{upmnp.fd}
\DeclareFontFamily{U}{pmnp}{}
\DeclareFontShape{U}{pmnp}{m}{n}{<-> pmnrp}{}
\endinput
```

VI.3 The map file

While the map file for Minion is much longer than the one in the last tutorial, it is conceptually similar:

```
pmnr8r   Minion-Regular            "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnr8a.pfb
pmnrc8r  Minion-RegularSC          "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnrc8a.pfb
pmnri8r  Minion-Italic             "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnri8a.pfb
pmnric8r Minion-ItalicSC           "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnric8a.pfb
pmns8r   Minion-Semibold           "TeXBaseEncoding ReEncodeFont" <8r.enc <pmns8a.pfb
pmnsc8r  Minion-SemiboldSC         "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnsc8a.pfb
pmnsi8r  Minion-SemiboldItalic     "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnsi8a.pfb
pmnsic8r Minion-SemiboldItalicSC   "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnsic8a.pfb
pmnb8r   Minion-Bold               "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnb8a.pfb
pmnbj8r  Minion-BoldOsF            "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnbj8a.pfb
pmnbi8r  Minion-BoldItalic         "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnbi8a.pfb
pmnbij8r Minion-BoldItalicOsF      "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnbij8a.pfb
pmnc8r   Minion-Black              "TeXBaseEncoding ReEncodeFont" <8r.enc <pmnc8a.pfb
pmncj8r  Minion-BlackOsF           "TeXBaseEncoding ReEncodeFont" <8r.enc <pmncj8a.pfb
pmnr8x   MinionExp-Regular                                                <pmnr8x.pfb
pmnri8x  MinionExp-Italic                                                 <pmnri8x.pfb
pmns8x   MinionExp-Semibold                                               <pmns8x.pfb
pmnsi8x  MinionExp-SemiboldItalic                                         <pmnsi8x.pfb
pmnb8x   MinionExp-Bold                                                   <pmnb8x.pfb
pmnbi8x  MinionExp-BoldItalic                                             <pmnbi8x.pfb
pmnc8x   MinionExp-Black                                                  <pmnc8x.pfb
```

The only difference are the swash and ornament fonts. Both types of fonts are not reencoded, hence their mapping is similar to that of expert fonts:

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

VI.4 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 (34–36):

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{nfssext}
\newcommand{\exfs@try@family}[1]{%
  \expandafter\ifx\csname\f@encoding+#1\endcsname\relax
    \let\exfs@tempa\relax
    \begingroup
      \fontfamily{#1}\try@load@fontshape
      \expandafter\ifx\csname\f@encoding+#1\endcsname\relax
        \PackageWarning{nfssext}{%
          Font family \f@encoding/#1 unavailable,\MessageBreak
          ignoring font switch}%
      \else
        \gdef\exfs@tempa{\fontfamily{#1}\selectfont}%
      \fi
    \endgroup
    \exfs@tempa
  \else
    \fontfamily{#1}\selectfont
  \fi}
\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}
  \exfs@try@family{\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's 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. Let's 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, let's redefine `\itshape`. Since we are about to change Latex internals, we duly note that in the log file:

```
\PackageInfo{nfssext}{Redefining '\string\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 this case it will switch to `si` instead, provided that `si` is available. `\scshape` does it the other way around:

```
\PackageInfo{nfssext}{Redefining '\string\scshape'}
\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`:

```
\PackageInfo{nfssext}{Redefining '\string\upshape'}
\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}
```

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

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

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.

VI.5 A high-level interface for ornaments

Technically, ornament fonts are comparable to the euro fonts discussed in section IV.3. To typeset the first ornament of Minion, for example, we could use the following construct:

```
{\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 explicity 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) to the ornament which is 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) 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 neccessarily 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, this ornament is 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:

```
\DeclareRobustCommand{\ornament}[1]{%
  \expandafter\ifx\csname\exfs@base@family @orn\@roman#1\endcsname\relax
    \PackageWarning{nfssext}{%
      Ornament #1 undefined for font family '\exfs@base@family'\MessageBreak
      Setting debug mark}%
    \rule{1ex}{1ex}%
  \else
    \begingroup
      \edef\exfs@tempb{\csname\exfs@base@family @orn\@roman#1\endcsname}%
      \expandafter\expandafter\expandafter\exfs@split@orndef
        \expandafter\string\exfs@tempb\@nil
      \selectfont\char\exfs@tempa
    \endgroup
  \fi}
\endinput
```

First of all, we check if the desired ornament has been declared (86) and issue a warning if not (87–89). We also typeset a mark (90) to facilitate debugging in this case. If it has been declared we expand the declaration (93), parse it (94–95), switch fonts, and finally typeset the ornament (96). We use grouping to keep the font change local.

VI.6 The style file

The style file for Minion is similar to the ones suggested in section III.3 and V.5. The only difference is the declaration of the text ornaments. This is the first part of `minion.sty`:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{minion}
\RequirePackage[T1]{fontenc}
\RequirePackage{textcomp}
\RequirePackage{nfssext}
\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}
```

CODE TABLES

The tables below 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. 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 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 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.

TABLE 1: Font weights

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 2: Font widths

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 3: Font variants

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 4: Font encodings

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

CONTRIBUTING

Contact me at `lehman@gmx.net` if you are interested in helping to improve and extend this guide. Please drop me a line before you begin so that I can coordinate contributions in order to avoid overlapping efforts. If you would like to contribute, please note that writing good documentation is quite a time consuming thing to do, especially if you are writing in a language that is not your mother tongue. Therefore I would appreciate it if at least native English speakers could provide contributions in a final form so that I can simply add them to the text without major changes. I could use contributions for the following topics in particular:

TEX DISTRIBUTIONS FOR WINDOWS/MAC – The installation instructions in tutorial I are written with Tetex on a Unix box in mind. I'm not familiar with Tex distributions for Windows or the Mac, but if anyone would like to contribute a few paragraphs I will gladly include them in the first tutorial. Things that immediately come to mind include:

- Directory structure and Tex trees: how to maintain a local tree, how to configure kpathsea or any equivalent thereof (the section on file installation in the first tutorial).
- Standard maintenance tools: how to refresh the file databases etc. Are there any additional tools that need to be configured when installing new fonts? If so, how do you do that?
- Viewers and their configuration: what is the default DVI viewer? Does it support Postscript fonts? If so, how do you configure it? Does it read dvips's map files?

ADDITIONAL TUTORIALS – I have been planning to add one more tutorial to this guide. The contents are pretty much settled by now. Tutorial VII will deal with non-standard ligatures and alternate lettershapes. Basically, it will be an introduction to encoding vector hacking and to some of the advanced capabilities of virtual fonts. In addition to that, I'm open to all kinds of contributions suiting the concept and the scope of this guide. Possible topics which come to mind include:

- Scaling fonts. This tutorial could discuss scaling at both install-time[1] and 'run-time', that is, when the font definition file is read.[2]

1. Possible applications of install-time scaling include lining figures. If hanging figures are not available for a given typeface, lining figures need to be employed even within mixed case text where they appear too large, attracting too much attention. This problem is usually alleviated by setting them smaller than the surrounding text. A dedicated virtual font created by fontinst might put smaller lining figures in the default encoding slots for figures.
2. Run-time scaling is particularly useful for sans serif and monospaced typefaces since they

- Tracking fonts. This tutorial would discuss tracking with fontinst. A typical application of tracking is spaced out small caps fonts.
- Installing math fonts. I'm not sure whether this can be discussed in an adequate way in only 15–25 pages. It would certainly take an individual with a profound understanding of math fonts to write such a tutorial in a concise and lucid way.

often need to be scaled to blend in with the main text font. Since the appropriate scale factor depends on the combination of typefaces used in a specific text, adequate scaling is not feasible at install-time in this case. `helvet.sty`, along with the font definition files for Helvetica, is a good example for a possible implementation of run-time scaling.

LICENSE

GNU Free Documentation License
Version 1.2, November 2002

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.

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 5 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.

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 3.

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

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.

4. Modifications

You may copy and distribute a modified version of the document under the conditions of sections 2 and 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, in the form shown in the addendum below.
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.

5. Combining documents

You may combine the document with other documents released under this license, under the terms defined in section 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.'

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.

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 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.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the document under the terms of section 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 4) to preserve its title (section 1) will typically require changing the actual title.

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.

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/

REVISION HISTORY

1.10	2003-03-27	Added license
1.00	2003-03-25	Final release for fontinst 1.8
		Added tutorial VI
		Updated notes on contributions
0.80	2003-03-23	Added corrections and suggestions by Timothy Eyre
		Revised section I.5, splitting off section I.6
		Added section III.4
		Updated notes on contributions
0.68	2003-02-09	Revised section IV.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 II.2
		Revised section II.3
		Revised introduction
0.60	2003-01-11	Revised tutorial III
		Added discussion of kerning issues to section III.1
0.54	2003-01-04	Revised discussion of OT1 encoding in tutorial I
		Added minor changes to code tables
0.52	2003-01-02	Added table of widths to code tables
		Revised notes on code tables
0.50	2002-12-30	Added tutorial V
		Added code tables
		Added revision history
0.43	2002-10-25	First public pre-release featuring tutorials I–IV
		Added installation instructions to section IV.3
		Added section IV.4
0.40	2002-08-11	Added tutorial IV
0.30	2002-05-12	Added tutorial III
0.20	2002-04-17	Unreleased draft including tutorials I and II