Virtual Fonts, Virtuous Fonts

Alan Hoenig

Abstract

Virtual fonts allow us to use all digital fonts with TEX, even non-TEX ones, and do much more for us. What are virtual fonts? Several projects grant us necessary experience with them.
This document comprises somewhat less than half of the similarly named chapter which will appear in the book **TEX Unbound: LATEX and TEXStrategies for Fonts, Graphics, and More**, by Alan Hoenig, to be published in early 1997 by Oxford University Press. This excerpt is simplified so that it may be printed using the standard suite of TEX fonts; the original depends heavily on PostScript fonts and the author's style file for its typesetting. Consequently, some displays could not be included. For any questions or comments, please contact the author at `ajhjj@cunyvm.cuny.edu`.

When talking about computers, we use the adjective "virtual" to describe a thing that behaves like something else. Virtual disks are really memory blocks which simulate hard disks, while virtual memory uses a disk to mimic a computer's memory. A virtual font looks to TEX like any other font, but it really is pieced together from other fonts or collections of typographic elements. It may be

- a composite of several different fonts somehow mixed together (in a special way, according to precise rules);
- a single font whose characters are (for very good reasons) scrambled in some new order;
- a collection of constructed characters, each built from several components (like accented letters are), which behaves like a font;
- individual horizontal or vertical rules each of which is treated as a character in a font;
- a collection of text, graphics, or PostScript files, each of which is treated as a single character within the virtual font;
- a conglomerate of all (or some) of the above.

This and the next few chapters explore virtual fonts, consider occasions that need them, and provide procedures for constructing them. It's messy constructing virtual fonts by hand, but a few freely available resources make it easy.

The box (not included here—sorry) lists some virtual font projects. For most people, the only application of virtual fonts may be to perform the proper installation of outline fonts (that is, PostScript fonts) for use by TEX. (We will use the term *installation* to describe the entire process of making fonts usable by TEX.) Many tasks difficult or impossible to accomplish with macros become trivial when implemented via virtual fonts.

The next few sections explain the concept of virtual font, together with the related concepts of font tables and encoding tables, in detail. Thereafter, we will provide discussion and procedures for implementing most of the applications in the list above.

1 The virtual font concept

Let's begin by journeying to a different planet, one on which a system like TEX has been developed, but on which all languages contain only two distinct characters, which we can call 'e' and 'f', together with the double-f ligature 'ff'. A close examination of a font on this hypothetical planet makes it easier to understand the kinds of problems arising in real, terrestrial fonts, and how virtual fonts can solve them.

A table listing the characters of any TEX font would contain only three characters.

0	1	2
e	**f**	**ff**

These three characters have been numbered using the usual computer science convention which starts with 0. These numeric labels serve to identify the position in the font of each character.

These numeric positions also play an important role for TEX, for `dvi` files contain typesetting commands based *not* on the glyph name ('A', 'B', 'comma', or whatever) but on each numeric label. For any 'e' in the input file, the `dvi` file contains the instruction to typeset character 0 in the current font. The lowercase 'e' had better be in that position! This correspondence between character and character number is built into the TEX program, and that's true for both the distant planet and for ours.

Difficulties arise when we try to use a commercial font instead of Computer Modern. We will suppose that the commercial font we want contains three characters, but they are 'e', 'f', and '&'. To get the ligature, we need to purchase a separate font, which contains the 'ff' plus two other characters. A further difficulty surfaces when we examine the layout of the fonts.

0	1	2
f	**e**	**&**

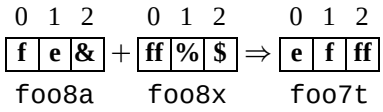
0	1	2
ff	**%**	**$**

The characters in the main font are in the wrong order, and this leads to disaster. To see why, let's select the commercial font, and now suppose we type `f`. TEX expects an 'f' to occupy position 1 of the font table, and so puts an instruction (in the `dvi` file) to typeset character 1. But character 1 in the non-TEX font is the glyph 'e', and that's what gets typeset—not the 'f' that we requested. Moreover, it does not appear that we can typeset the ff without an explicit call to the auxiliary font. Apparently, the input file will look different whether we typeset with the usual TEX fonts or with some other fonts, and this is unacceptable.

Virtual fonts have been created to deal with this (and other) exigencies. As far as an author is concerned, a virtual font is just another font. But it provides a mechanism whereby (behind the scenes), real fonts (*raw fonts*) can be combined so that the resulting *virtual font* conforms to the usual TEX conventions to eliminate any need for marking up the input file differently. In our example, a virtual font would

1. select the e and f from the main font, and re-order them in a TEX-acceptable way; and
2. include the ligature from the expert font in the last position in the table for the virtual font.

We call an auxiliary font containing ligatures and other special symbols an expert font. Furthermore, we'll follow the terrestrial convention of labelling raw fonts by appending '8a' or '8x' (expert) to them. So raw font `foo8a` and expert font `foo8x` come together in the virtual font `foo7t`. (Chapter 6 explains the conventions surrounding the notations 8a, 8x, and 7t.)



Font `foo7t` uses selected characters from the two raw fonts, and orders this selection in a way meaningful to TEX. Not all the characters from raw fonts need be part of the final virtual font.

Up until virtual fonts, words containing accents suppressed TEX's hyphenation algorithm. We can define an accented letter in a virtual font to be equivalent to any other letter, so hyphenation proceeds unimpeded—yet another advantage of virtual fonts.

To be sure, this unrealistic, alien font provides a contrived example. But with real commercial fonts, these same problems—writ large because real fonts have so many more characters—need the practical solutions that virtual fonts provide.

2 Digital fonts and font tables

Font tables for the Computer Modern TEX fonts and for PostScript outline fonts contain a maximum of 256 positions; 256 is one of the magic numbers of computer science. See figure 1 for examples of real font tables. (For a slightly different representation, refer to the font tables beginning on page 427 of *The TEXbook]/*. The PostScript font tables cannot be shown here; pardon.) A casual glance reveals significant differences between the layouts for the two fonts. Each slot of the Computer Modern font is filled (up till character 127), whereas there are many unfilled slots in the PostScript fonts. Some characters, like the uppercase Greek letters or the ff, ffi, and ffl ligatures, do not appear anywhere in the PostScript font while certain PostScript characters appear nowhere in the Computer Modern layout. Other characters are in disparate positions. All the Scandinavian ligatures (æ, and so on) appear in the fourth row of the Computer Modern font table, but cluster together near the very end of the PostScript font table.

Positions in any font table are numbered starting from zero up to 255. We know that terrestrial TEX selects characters *not* by the character name but according to its position in the font table, so a command to typeset an 'A' is relayed as an instruction to

> typeset the character from the currently selected font that occupies position 65 in that font,

since that's the numeric label of the 'A' slot. (All character positions are given here in decimal notation. Computer scientists may be more comfortable with a character's octal position, which is why that information also appears in the tables.)

When using PostScript outline fonts, it's useful to be able to typeset in a 'fake font'—our virtual font—which looks real to TEX but is in fact an amalgam of one or more raw, component fonts. We arrange this virtual font so the characters in the virtual font are in the same order as in any other Computer Modern font. That way, macros will seldom have to be redefined for different fonts, a particularly important issue for mathematics typesetting.

Associated with a font table is the *font encoding vector* or just the *encoding vector* or *code page*. The encoding vector is the list of the character names in the order in which they occur in the font table. For a TEX font (figure 1), the encoding vector is the list beginning

Gamma, Delta, Theta, Lambda, Epsilon, Pi, Sigma, Upsilon, Phi, Psi, Omega, ff, fi, fl, ffi, ffl, dotlessi, ...

and so on. If we let '`.notdef`' designate a font position for which no character is defined, then for a PostScript font, the encoding vector is a list that begins

.notdef, ..., .notdef (32 times in all),
space, exclam, quotedbl, numbersign, dollar, ...

3 What comprises a virtual font?

TEX does not deal with any characters beyond the metrics associated with a font. It expects to find this information in a `tfm` file, and so each virtual font must be accompanied by a font metric file in the usual way. This file should be placed in a suitable place.

0 Γ 0	1 Δ 1	2 Θ 2	3 Λ 3	4 Ξ 4	5 Π 5	6 Σ 6	7 Υ 7	8 Φ 10	9 Ψ 11	10 Ω 12	11 ff 13	12 fi 14	13 fl 15	14 ffi 16	15 ffl 17
16 ı 20	17 ȷ 21	18 ` 22	19 ´ 23	20 ˘ 24	21 ¯ 25	22 ˚ 26	23 ¸ 27	24 ß 30	25 æ 31	26 œ 32	27 ø 33	28 Æ 34	29 Œ 35	30 Ø 36	31 Å 37
32 ˝ 40	33 ! 41	34 ” 42	35 # 43	36 $ 44	37 % 45	38 & 46	39 ’ 47	40 (50	41) 51	42 * 52	43 + 53	44 , 54	45 - 55	46 . 56	47 / 57
48 0 60	49 1 61	50 2 62	51 3 63	52 4 64	53 5 65	54 6 66	55 7 67	56 8 70	57 9 71	58 : 72	59 ; 73	60 ¡ 74	61 = 75	62 ¿ 76	63 ? 77
64 @ 100	65 A 101	66 B 102	67 C 103	68 D 104	69 E 105	70 F 106	71 G 107	72 H 110	73 I 111	74 J 112	75 K 113	76 L 114	77 M 115	78 N 116	79 O 117
80 P 120	81 Q 121	82 R 122	83 S 123	84 T 124	85 U 125	86 V 126	87 W 127	88 X 130	89 Y 131	90 Z 132	91 [133	92 “ 134	93] 135	94 ^ 136	95 ˙ 137
96 ‘ 140	97 a 141	98 b 142	99 c 143	100 d 144	101 e 145	102 f 146	103 g 147	104 h 150	105 i 151	106 j 152	107 k 153	108 l 154	109 m 155	110 n 156	111 o 157
112 p 160	113 q 161	114 r 162	115 s 163	116 t 164	117 u 165	118 v 166	119 w 167	120 x 170	121 y 171	122 z 172	123 – 173	124 — 174	125 ˝ 175	126 ~ 176	127 ¨ 177

Figure 1: A font table for Computer Modern Roman fonts (here, `cmb10`). The Roman numbers in the upper left of each box give the character number using the usual decimal representation. The italic numbers in the lower right are the octal equivalents.

The details behind the construction of the virtual characters appear in the actual virtual font file, a file with the extension `vf`. There needs to be a place place on a hard disk to store virtual fonts, in the same way that there are places for `tfm` files, format files, input files, and so on. The places have different names depending on whether your system is traditional or complies with the TDS standard (see chapter 6).

The actual virtual font `vf` file contains fragments of `dvi` language that specify the way that a virtual character should be created. That means that a character in a virtual font can be anything that occurs in a `dvi` file. In theory, one virtual character can typeset an entire page or document! Typically, virtual characters are not so complex. In the alien planet example, the virtual font simply remapped characters (placed them in a different and more suitable order) and merged characters together from raw fonts.

4 What we will need; preparation

We've seen in the previous chapter that *vfinst* takes care of the most common virtual font tasks—the installation of scalable fonts to make them usable by TEX. However, there are many more reasons to use virtual fonts, and so we begin a lengthy, conscientious look at virtual fonts. We need to understand, too, the sequence of steps that *vfinst* performs.

Firstly, virtual fonts are a feature of TEX3. In order to proceed, that version must be installed.

Many authors will be preparing documents for output on PostScript printing devices. Since TEX only knows how to write `dvi` files, we will always need a dvi-to-PostScript converter. Frequently these programs require an auxiliary map file to "map" the long font names to the short file names that are all that some operating systems, notably MS-DOS and its relatives, can handle. Because it is freely available, and available for all computer platforms, we will usually refer to *dvips* and its map file `psfonts.map`. Each of its entries pairs a short, DOS-acceptable name for a *raw* font with its long, given font name. These short aliases are the names that we should use in the process of virtual font creation. For each short alias in the map file, there must be a `tfm` file under that name.

The map file may serve other functions. It may aid in the process of downloading (see below), and it may be where we specify certain types of transformations on a font.

At print time, how does the printer get the information about the shapes of the characters in the document? For bitmap fonts, it's the responsibility of the printer driver to include the bitmap information in the instructions it transmits to the printer. For scalable fonts, the situation is different. The outline information on all fonts must be transmitted to the printer, for it is the printer that ultimately converts the outline to raster form for printing. In most PostScript-compatible printers, descriptions of 35 or so common fonts, including Times Roman and Helvetica, are resident—built-in—to the printer. If you use other, non-resident fonts, you will need to *download*—transmit—this font information to the printer, and this downloading can be accomplished in different ways. It is also possible to include the font information in the PostScript version of the document.

5 The purpose of a simple installation

If we examine the font tables in this chapter, we see that the problem of constructing a virtual font from a PostScript font is not hopeless. Most characters are in the same positions, including all upper- and lowercase letters, digits, and much of the punctuation. We may divide the remaining characters in an outline font into two groups:

- special characters like fi, —, ¿, and the American quotation marks "" which are selected by TEX's ligature mechanism; and
- characters like æ, Œ, or ç which are invoked by control sequences or control words (here, `\ae`, `\OE`, and `\c{c}`).

(Actually, there's a third group—those characters present in `cmr10` but absent entirely from a standard Type1 font, such as the ligatures ff, ffi, and ffl. We'll see later how to deal with these.) We would like to make sure we have access to *these* members of a font **without** having to change the rules by which we create our source documents. Actually, just in case an author has been silly and used a nonstandard convention to typeset a symbol (such as getting ¿ by typing `\char62` or `\symbol{62}` rather than `?‘`), we would like the layout of the virtual font to adhere as closely as possible to the original TEX font layout.

The ligatures of the first group can be handled in a nonvirtual way by adjusting the font metric files so TEX plucks the ligature from the proper font position; no remapping is necessary. This requires a modification of the `tfm` only.

Characters accessed by TEX commands present more of a challenge. The definition for each such command relies upon being able to locate special characters by their position in the font table. TEX therefore expects œ to be character 27 in a font, since that's where it is in the Computer Modern family. When constructing ç, it expects the cedilla to be in position 24 for the same reason. Typically, though, these characters do not appear in those positions in the raw PostScript font (œ and cedilla occupy positions 250 and 203). Macros could be redefined, but it's a bad idea to have macro definitions depend on the current font. We require our virtual font utility to reorder—to remap—these characters in the font. For example, virtual character 27 consists of the raw character 250. That way, when the virtual font is the current font, `\oe` will correctly typeset the œ glyph.

The `afm2tfm` utility (part of *dvips*) is an excellent tool for creating this elementary kind of virtual font—a font consisting of the remapping of the characters in a single raw font. Because the source for this program has been made available, `afm2tfm` has been ported to every significant computer architecture, and executable binaries are freely available from friends or software archives (the same applies to *dvips* itself). But `afm2tfm` suffers from several disabilities: it can't create a virtual font out of more than one raw file, it can't create the `fd` font descriptors that LATEX now uses, and it doesn't mimic the original TEX font layout as closely as it might. Nevertheless, simple installations are so common that it is important to detail this process precisely.

The box (not included here—sorry) summarizes the procedure to follow to use `afm2tfm` to create virtual files from outline fonts. We use this procedure whenever this simple manipulation is sufficient for our needs. (More complicated finagling is best carried out with *fontinst*; see below.) This process involves using or creating several file types. If an outline font is `psfont`, that means the distribution diskette should include `psfont.afm` and `psfont.pfb`. It is necessary to rename the file name `psfont8a`, and from these we will be generating files `psfont7t.vpl`, `psfont7t.tfm`, and `psfont7t.vf`, the virtual file. We also generate a font metric file corresponding to the "raw" PostScript file `psfont8a.tfm`.

The program `afm2tfm` can also create pseudo-small caps fonts and other fonts which have undergone simple geometric transformations, like slanting or extension. Check the documentation to learn how.

Once we've created the virtual font and placed all the files where they belong, we access any virtual file just as if it were a normal TEX font (which it is). For example, we could declare

```
\font\foo=psfont7t at 10.5pt
```

in a `plain` TEX document and use it via the command `\foo` which has become a font changing command like `\it` or `\tt`. Although we never again refer to the raw font file explicitly, TEX do. Behind the scenes, whenever a TEX device driver resolves the meaning of a virtual font, it refers to the component raw fonts, the raw fonts must be present on our system.

6 Introduction to *fontinst*

The *fontinst* package, by Alan Jeffrey, does everything `afm2tfm` does and more. It can create a virtual font from several raw fonts, for example, and it automatically produces an auxiliary `fd` file used by LATEX's NFSS to select the font. The *fontinst* package is written entirely in TEX, and TEXegetes will enjoy perusing `fontinst.sty` to watch TEX do things it was never intended for. Writing it in the TEX language insures *fontinst* runs on every platform that TEX does. You can retrieve *fontinst* from any CTAN archive, under `fonts/utils`. The discussion in this chapter supplements `fontinst.tex`, the documentation of the package.

We use *fontinst* by preparing a simple `plain` TEX file. Typically, this file will be short, and will consist of a command to `\input` the `fontinst.sty`, followed by a variety of commands which tell *fontinst* how to create the virtual font. Normally, `vf` and `tfm` files are binary files, file types which TEX cannot write. Therefore, *fontinst* reads and writes property list files and special metric and encoding files instead. These are all in Ascii, and the property files in particular are ASCII equivalents to `vf` and `tfm` files with extensions `vpl` and `pl`. Part of your TEX installation should include the utilities `vptovf` and `pltotf` (together with their inverses `vftovp` and `tftopl`), and we would then use utilities these to create the font files we need.

After each successful run of *fontinst* there will be three new kinds of files in your working directory.

- `pl` files—one for each raw font—which feeds into `pltotf` to create a `tfm` file;
- `vpl` files—one for each virtual font—which feeds into `vptovf` to create one `vf` and one `tfm` file; and
- a `fd` *font descriptor file*—one for each font family—which NFSS will use to relate the font attributes to individual fonts.

(There are also some new `mtx` files and the usual `log` file that you can delete.) It is necessary to run all `vpl` files through `vptovf` and all `pl` files through `pltotf` to generate the binary metric files that TEX needs. A map file, such as `psfonts.map` for *dvips*, must be updated; see chapter 6.

All `tfm` files belong with your other `tfm` files. The `vf` files belong in a special place as well, where *dvips* expects to find virtual files. The `fd` files belong in a TEX inputs directory.

6.1 Installing fontinst

The *fontinst* package consists of a the core file `fontinst.sty` together with some documentation, some samples and many examples. You may well receive the package as a zipped collection of files already organized in its own directory structure. I found it convenient to create a `.../fontinst` directory in which I unpacked *fontinst*. One or two levels down is a new directory called `inputs`. In addition to `fontinst.sty` itself, there are a collection of files with extensions `mtx` and `etx`. Move these files to one of your TEX input directories to complete the installation.

Goals

The *fontinst* package provides a new language for the creation of virtual fonts of all types. Our goal in this and subsequent chapters shall be to develop familiarity with these procedures so we can install any font with (relatively) little work.

Although *fontinst* works *much* slower than `afm2tfm`, it is much faster than creating `vpl` files by hand.

7 Simple font installation with fontinst

7.1 New commands

Figure 2 displays one way to use *fontinst* to install the Times Roman fonts that are resident in every PostScript printer. Most *fontinst* installation files resemble this display.

Much of this file is standard boilerplate. The first line

```
\input fontinst.sty
```

makes *fontinst* known to TEX.

The pair of commands `\installfonts` and `\endinstallfonts` (with no arguments) surrounds the sequence of commands that do the bulk of the work. One or more `\installfamily` commands now follow. The first argument specifies the encoding, the second the family designation, and the third a set of commands that will be executed each time the family is loaded. See the *fontinst* documentation for further details on this third argument; it will be empty in nearly all our work.

```
\installfamily{encoding}{family}
               {fd-commands}
```

The workhorse command in any installation file is the `\installfont` command, which takes eight parameters. The last parameter allows us to specify size information for the font. For scalable fonts, it is nearly always empty because scalable fonts are, well, scalable to any size. (Bitmap fonts, created specifically for different sizes, require non-empty entries.) Parameters 4 through 7 provide space for the encoding, family, series, and shape values that *fontinst* uses to create the NFSS `fd` file. Consult the previous chapter and examples in this and subsequent chapters to see how these parameters fill out. The very first parameter stores the file name of the virtual font you want to create.

That leaves the second and third parameters. In order to understand their significance, we need a small digression to consider the process of font creation.

7.2 Creating fonts

There are two aspects to font creation:

1. **Metric:** We need procedures for constructing each glyph or character in the virtual font.
2. **Encoding:** We need to decide on the order of the glyphs in the font, and specify any additional rules that the characters need to live by. For example, rules might concern ligatures (any time an i follows a single f, replace it by fi; any time an *A* appears at the beginning of a word, replace it by a swash variant), or math symbols (any time interior material gets too tall, replace a delimiter by the next larger size).

For *fontinst*, these instructions should be in *metric files*, with an `mtx` extension, and *encoding files*, with extension `etx`. In the second position of the `\installfont` command, we place a list of metric files to be inserted. *fontinst* reads them to find out how to construct the characters. The third position records the name of an encoding file, which *fontinst* reads to learn which characters to include, how to order them, and what ligature and other special rules to follow.

Schematically, a `\installfont` instruction looks like this.

```
\installfont{font-name}{metric-files}
   {encoding-file}{encoding}
   {family-name}{series}{shape}{size}
```

7.3 Metric files

The task of preparing metric files is lightened because *fontinst* reads three types of metric files:

1. `mtx` files, using a format specific to *fontinst*;
2. `afm` files, the ASCII metric files that come with each scalable outline font; and
3. `pl` files, the ASCII equivalents to a TEX `tfm` file.

fontinst reads the first two types automatically, but you will need to use the program `tftopl` (which should accompany your version of TEX) to create this file. For example, type

```
\input fontinst.sty

\installfonts
   \installfamily{OT1}{ptm}{}
   \installfont{ptmr7t}{ptmr8a,latin}{OT1}{OT1}{ptm}{m}{n}{}
   \installfont{ptmrc7t}{ptmr8a,latin}{OT1c}{OT1}{ptm}{m}{sc}{}
   \installfont{ptmri7t}{ptmri8a,latin}{OT1}{OT1}{ptm}{m}{it}{}
   \installfont{ptmb7t}{ptmb8a,latin}{OT1}{OT1}{ptm}{bx}{n}{}
   \installfont{ptmbc7t}{ptmb8a,latin}{OT1c}{OT1}{ptm}{bx}{sc}{}
   \installfont{ptmbi7t}{ptmbi8a,latin}{OT1}{OT1}{ptm}{bx}{it}{}
\endinstallfonts
\bye
```

Figure 2: One way to install Times Roman. This examples uses the original TEX encoding but does not include any expert fonts. Two series are installed—regular and bold. Within each series, three shapes are installed—upright, small caps (which use encoding file `OT1c.etx`), and italic.

```
tftopl cmr10.tfm cmr10.pl
```

to do the obvious thing.

In *fontinst* prior definitions take precedence over subsequent definitions. That is, if any construct appears more than once in a series of files that *fontinst* reads, only the first one counts; later definitions are silently ignored. Therefore, *the order in which fontinst reads files is critical!* This philosophy is central to the way *fontinst* works, as we'll see.

The file `latin.mtx` is the "metric file of last resort." It provides instructions for creating 401 glyphs found in Latin alphabets. Of those 401, some are unfakable—there's no way to print characters like 'A' unless the A is in the font, but many other glyphs can be faked. Accented letters can be built from letters and accents, and small caps can be taken from an uppercase font set at 80% of the current design size. Of course, there isn't room for all 401 of these characters in a single font anyway. (The limit is 256.) But because many of these characters have been previously defined in metric files, *fontinst* will ignore many of the definitions in `latin.mtx`—remember, glyph constructs have no effect if defined previously. But if you have neglected to define a glyph that you later call for, the definitions in `latin.mtx` serve as safety net. That is why all the `\installfont` commands in figure 2 and in virtually every *fontinst* example contain lists of metric files that terminate with a call to `latin.mtx`.

7.4 Encoding files

Once the metric files have done their job (of constructing the glyphs), a single encoding file chooses the group of characters that belong in the font and the proper order (encoding). This file also specifies certain ligature and other rules for the font to abide by.

Encoding files tend have names to reflect their encoding. Thus, the encoding file for the OT1 encoding is simply `OT1.etx`. Similar files, `OT1c.etx` and `OT19.etx`, would set up a small caps and an old-style figures font using OT1 encoding. There are several more variants in the *fontinst* distribution.

8 Progressive examples

It's time to consider examples using *fontinst* to create virtual fonts.

8.1 Simple font installation

The simplest way to use *fontinst* is to run TEX on the file `fontinst.sty` and to then type

```
\latinfamily{ptm}{}
\bye
```

in response to TEX's star prompt `*`. This works presuming that all the fonts in the `ptm` family (Times Roman) have been named in accordance with Karl Berry's font naming scheme and that all font metric files are in places that TEX can read from.

This method is best for authors who plan never to need any more exotic fonts than these. Subsequent examples are designed to show of the power of *fontinst* and to teach its intricacies in a tutorial manner.

8.2 Easy DC fonts

The Cork encoding, denoted by T1, refers to the standard agreed upon at a TEX meeting held in Cork, Ireland in September 1990. At that time, agreement was reached for sets of 256-character fonts for use by TEX. (The TEX standard had at that time only been extended to 256 character fonts for a short time.) The `dcr` fonts look like the usual computer Modern fonts, but these fonts have been extended according to the Cork standard. Virtual fonts provide an easy way to generate `dcr` fonts from raw, Computer Modern fonts.

For each virtual `dcr` font, a corresponding `cmr` font acts as the single raw font. We will need the property list `pl` file as well.

Here are the steps to create a virtual `dcr10` from a raw `cmr10` font. The installation file `makedcr.tex` should resemble

```
% This is makedcr.tex, for use with fontinst.
\input fontinst.sty
  \installfonts
    \installfamily{T1}{dcr}{}
    \installfont{dcr10}{cmr10,latin}{T1}{T1}%
                {dcr}{m}{n}{}
```

```
  \endinstallfonts
\bye
```

although you'll need additional `\installfont` statements for members of this family which are italic, boldface, and so on.

Following the successful execution of the *fontinst* run, enter these statements at the prompt:

```
tftopl cmr10.tfm cmr10.pl
tex makedcr
vptovf dcr10.vpl dcr10.vf dcr10.tfm
rm *.log *.pl *.vpl *.mtx
```

after which you'll need to move the `tfm` and `vf` files to their proper places. In words, we need first the ASCII property list file, after which we can invoke TEX and *fontinst*. Thereafter, we create binary font files using the virtual property `vpl` produced by *fontinst*. Finally, we clean up. (Unix syntax is shown.) This example does not require an addendum to `psfonts.map` unless you are using scalable versions of the Computer Modern fonts.

Drawbacks of easy dcr10

During the creation of `dcr10.vpl`, *fontinst* reports that 34 glyphs are missing—that is, of the full complement of characters that do belong in a T1-encoded font, *fontinst* complained 34 times that it couldn't make the glyph. All of these are various diacritics (ring, ASCII tilde, and so on) and accented letters that use these missing diacritics, but a few are more problematic, including the sterling symbol and French quotations. If you access these characters, the mock `dcr10` font will not be suitable.

Moreover, there is no premium on disc space from using these fonts. The `vf` and `tfm` files require roughly 4k and 5.7k apiece, comparable with an actual `pk` file at a laser printer resolution.

8.3 Installing outline fonts

The *vfinst* utility takes care of scalable font installation, but we are now in a position to understand a simple installation ourselves. We may begin by renaming the font files to conform to a TEX font naming standard. Suppose we have Adobe Garamond Roman fonts to install. We rename the regular font files to `padr8a.pfb` and `padr8a.afm`, for example.

As an example, we can create the OT1-encoded font `padr7t` from these. This new font will belong to font family `pad` and have NFSS designations of `m` and `n` (medium series, normal shape). With this information, we prepare an installation file that looks like

```
% This is file makepad.tex
\input fontinst.sty

\installfonts
  \installfamily{OT1}{pad}{}
  \installfont{padr7t}{padr8a,latin}{OT1}%
  {OT1}{pad}{m}{n}{}
\endinstallfonts
```

although a real installation will likely contain several `\installfont` commands. The `\installfont` command is quite straightforward. It:

- constructs a font for family `pad`;
- uses glyph information from `padr8a.afm`, and supplements it (if necessary) with instructions from `latin.mtx`;
- applies the OT1 encoding to it; and
- uses the four parameters `OT1`, `pad`, `m`, and `n` for the NFSS `fd` file.

Incorporating expert fonts

For the vast majority of outline fonts, the only way to get the ff, ffi, and ffl ligatures is from an expert font, because these characters are rarely present in a base font. However, `latin.mtx` does create mock characters with these names because slots are provided for these ligatures in the font by the encoding files. Therefore, the way to get the honest double-f ligatures is simply to include the expert font name in the list of metric files in an `\installfont` command. That is, the skeletal installation file listed above would look something like

```
% This is file makepad.tex
\input fontinst.sty

\installfonts
  \installfamily{OT1}{pad}{}
  \installfont{padr7t}{padr8a,padr8x,latin}{OT1}%
    {OT1}{pad}{m}{n}{}
\endinstallfonts
```

Note the presence of `padr8x`, the expert font for Garamond regular.

The box (not included here; apologies) summarizes the bookkeeping involved in completing the installation. this discussion is presented for pedagogical completeness only, for in this case it's better to use `\latinfamily` (see above, section 8.1) or PSNFSS or *vfinst* (refer to chapter 6).