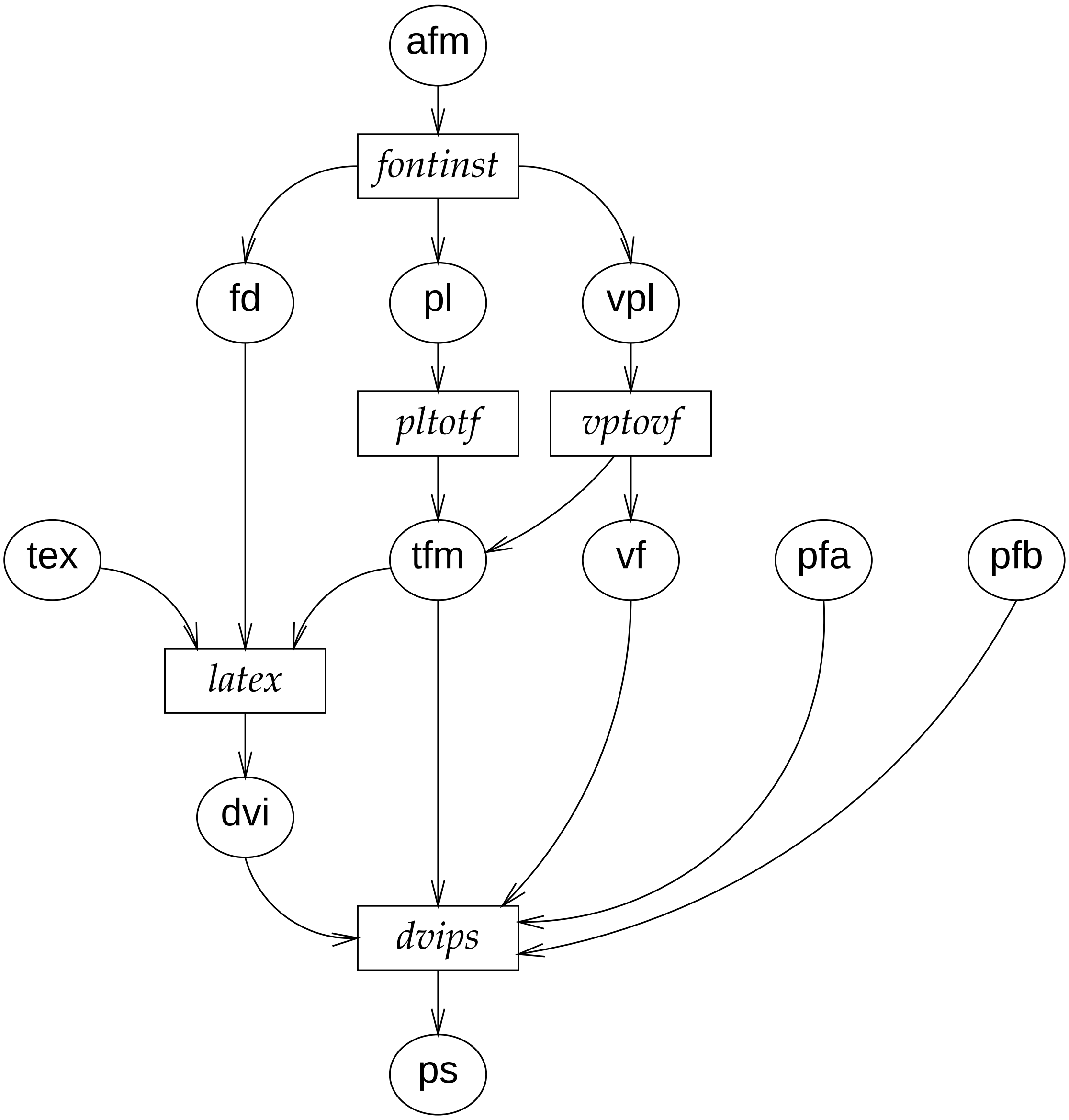
fontinst

Font installation software for TEX



Alan Jeffrey · fontinst v1.509 · June 1996

Contents

This manual describes the fontinst software for converting fonts from Adobe Font Metric format to forms readable by TEX. This manual should be distributed with the fontinst software, which is available by anonymous ftp from ftp://ftp.tex.ac.uk/tex-archive/fonts/utilities/fontinst, and on the various CD-ROMS containing material from the CTAN archives. Please do not contact the author directly for copies.

If you would like to report a bug with fontinst, please mail fontinst@cogs.susx.ac.uk. The mail will be sent to the fontinst mailing list. If you would like to be on the fontinst mailing list, please mail fontinst-request@cogs.susx.ac.uk.

1 Introduction

The fontinst package is a set of TEX macros which allows users to install virtual fonts. It can convert fonts from Adobe Font Metric (afm) or TEX Property List (pl) format into Virtual Property List (vpl) format. These Virtual Fonts (vfs) can then be used by your favourite device driver, such as dvips.

The package:

- Is written in TEX, for maximum portability (at the cost of speed).
- Supports the OT1 (Computer Modern) and T1 (Cork) encodings.
- Allows fonts to be generated with arbitrary 'fake' characters; for example the 'ij' character can be faked if necessary by putting an 'i' next to a 'j'.
- Allows caps and small caps fonts with letter spacing and kerning.
- Allows kerning to be shared between characters, for example 'ij' can be kerned on the left as if it were an 'i' and on the right as if it were a 'j'. This is useful, since many PostScript fonts only include kerning information for characters without diacriticals.
- Allows the generation of math fonts with nextlarger, varchar, and arbitrary font dimensions.
- Allows more than one PostScript font to contribute to a TEX font, for example the 'ffi' ligatures for a font can be taken from the Expert encoding, if you have it.
- Automatically generates a fd file for use with LATEX 2ε.
- Can be customized by the user to deal with arbitrary font encodings.

Fontinst has been a stable piece of software since mid-1994. All further updates will be upwardly compatible with the interface described in this document.

2 Installation

To install fontinst, put the contents of the inputs, examples and pl directories into a directory read by TEX, for example /usr/local/texmf/tex/fontinst/*. You should also make sure the afm and pl files you plan to use can be read by TEX, for example in Unix csh using web2c TEX:

```
setenv TEXINPUTS $TEXMF/fonts/afm//:$TEXMF/fonts/source//::
```

That's it!

3 Usage

The simplest way to use the fontinst package is to start by running TEX on fontinst.sty, and then responding to the * prompt by saying:

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

This will install the Adobe Times fonts. It may take some time, and will produce:

- One pl file for each afm file (re-encoded into the 'TEX base encoding').
- One vpl file for each LATEX font.
- One fd file for each LATEX family.

These can be converted to a form readable by TEX using pltotf and vptovf, for example in Unix csh:

```
tutorial% foreach X(*.pl)
? pltotf $X $X:r.tfm
? end
tutorial% foreach X(*.vpl)
? vptovf $X $X:r.vf $X:r.tfm
? end
```

You should then:

- Move the tfm files to a TEX fonts directory (e.g. /usr/local/texmf/fonts/tfm/adobe/times).
- Move the vf files to a virtual fonts directory (e.g. /usr/local/texmf/fonts/vf/adobe/times).
- Move the fd files to a TEX inputs directory (e.g. /usr/local/texmf/tex/latex/fd).

Finally, you should tell your dvi-to-PostScript driver about the fonts. This will depend on your driver, for example with dvips you should add the following lines to your psfonts.map file (if it isn't there already):

```
ptmb8r Times-Bold "TeXBase1Encoding ReEncodeFont" <8r.enc
ptmbi8r Times-BoldItalic "TeXBase1Encoding ReEncodeFont" <8r.enc
ptmbo8r Times-Bold " .167 SlantFont TeXBase1Encoding ReEncodeFont " <8r.enc
ptmr8r Times-Roman "TeXBase1Encoding ReEncodeFont" <8r.enc
ptmri8r Times-Italic "TeXBase1Encoding ReEncodeFont" <8r.enc
ptmro8r Times-Roman " .167 SlantFont TeXBase1Encoding ReEncodeFont " <8r.enc
```

You can then use the fonts! For example, to make Adobe Times the default roman font in LATEX 2_ε, you say:

```
\renewcommand{\rmdefault}{ptm}
```

The details of the LATEX 2_ε font selection scheme are described in *LATEX 2_ε font selection* (distributed with LATEX 2_ε) and *The LATEX Companion* (Goossens, Mittelbach and Samarin, Addison-Wesley).

Once you have installed and used the Times fonts, you can install the other standard Adobe fonts (Times, Helvetica and Courier) in a similar way. You can also install Adobe Times and Symbol as TEX math fonts by running TEX on fontptcm.tex.

4 Installing your own Latin fonts

The fontinst package has a simple interface for installing most Latin fonts. Most users will never need to use any commands other than these. The commands are:

\latinfamily{*family*}{*commands*}

This installs a Latin family of fonts. For example, to install Adobe Times, you say:

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

The *commands* are issued by LATEX each time a font from that family is loaded. This is most often used with typewriter fonts, to switch off hyphenation. For example, Adobe Courier can be installed with:

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

Once the installation is over (which may take some time) the fonts can be used in LATEX by selecting an appropriate \fontfamily, for example Adobe Times can be selected with:

```
\fontfamily{ptm}\selectfont
```

If the fourth letter of the family name is 'x' then fontinst will use any Expert fonts in creating the fonts. If the fourth letter is 'j' (or for backward compatibility '9') then fontinst will use Expert fonts to create fonts with old style digits.

Before using these commands, you will need to make sure that you have the Adobe Font Metric (afm) files for the fonts, and that they have appropriate names. The fontinst package uses the LATEX convention for naming fonts, and uses a *font family* name which consists of:

- a *foundry*, such as 'p' for Adobe.
- a *face*, such as 'tm' for Times.
- up to two *variants*, such as 'j' for 'old style digits'.

So the family name 'ptmj' indicates Adobe Times with old style digits.

The foundry must be one letter, and the face must be two (this is an attempt to fit all filenames into MS–DOS format). Each variant is one letter. The full list of foundries, faces and variants is given in Karl Berry's 'Filenames for fonts' (available by anonymous ftp from ftp://ftp.tex.ac.uk/tex-archive/info/fontname), but the more common ones are given in Tables 1–3.

The fontinst package uses Karl Berry's naming scheme for afm files. The full naming scheme is rather more flexible than the subset used by fontinst, which uses filenames consisting of:

- a *foundry*, such as 'p' for Adobe.
- a *face*, such as 'tm' for Times.

a	Autologic	b	Bitstream	h	Bigelow & Holmes
l	Linotype	m	Monotype	p	Adobe
z	bizarre				

Table 1: A partial list of foundries

ac	ACalson	ad	AGaramond	as	Aldus
bb	Bembo	bd	Bodoni	bk	Bookman
bv	Baskerville	ca	Caslon	ch	Charter
cr	Courier	cs	Century Schoolbook	fr	Frutiger
fu	Futura	gj	Granjon	gl	Galliard
gm	Garamond	gs	Gill Sans	hv	Helvetica
lc	Lucida	nc	New Century Schoolbook	pl	Palatino
pn	Plantin	pp	Perpetua	rw	Rockwell
st	Stone	sy	Symbol	tm	Times
un	Univers	ut	Utopia	zc	Zapf Chancery

Table 2: A partial list of faces

j	oldstyle digits	n	informal, casual
w	script, handwritten, swash	x	expert

Table 3: A partial list of variants

- a *weight*, such as 'r' for regular.
- up to two *shapes*, such as 'i' for italic.
- an *encoding*, such as '7t' for Knuth's 7-bit TEX encoding.
- an optional *width*, such as 'c' for Compressed.
- a *file extension*, such as '.tfm' for TEX Font Metric.

So the filename name 'ptmri7tc.tfm' indicates Adobe Times regular italic compressed, with old style digits, in the 7-bit TEX encoding.

The full list of shapes, encodings and weights is given in Karl Berry's 'Filenames for fonts', but the more common ones are given in Tables 4–6.

For example, to install Adobe AGaramond including the Expert fonts, you would need to rename the afm files:

c	small caps	i	italic	o	oblique (i.e., slanted)

Table 4: A partial list of shapes

7t	TEX 7-bit text OT1	9t	TEX 7-bit text with expert glyphs
8r	TEX 8-bit 'raw'	9e	TEX 8-bit text with expert glyphs
8t	TEX 8-bit text T1	9d	TEX 7-bit text with expert glyphs and old-style digits
8a	Adobe Standard	9o	TEX 8-bit text with expert glyphs and old-style digits
8x	Adobe Expert		

Table 5: A partial list of encodings

b	bold	i	extra light	l	light
s	semibold	x	extra bold		

Table 6: A partial list of weights

Adobe's name	*fontinst name*
AGaramond-Bold.afm	padb8a.afm
AGaramond-BoldItalic.afm	padbi8a.afm
AGaramond-Italic.afm	padri8a.afm
AGaramond-Regular.afm	padr8a.afm
AGaramond-Semibold.afm	pads8a.afm
AGaramond-SemiboldItalic.afm	padsi8a.afm
AGaramondExp-Bold.afm	padb8x.afm
AGaramondExp-BoldItalic.afm	padbi8x.afm
AGaramondExp-Italic.afm	padri8x.afm
AGaramondExp-Regular.afm	padr8x.afm
AGaramondExp-Semibold.afm	pads8x.afm
AGaramondExp-SemiboldItalic.afm	padsi8x.afm

You can then run TEX on the following document to install the Garamond family:

```
\input fontinst.sty
\latinfamily{padx}{}
\latinfamily{padj}{}
\bye
```

Not all font families can be installed using the Latin family commands, so the rest of this document describes the full fontinst syntax, and is intended for 'power users'.

5 Customization

The fontinst package reads a file fontinst.rc if it exists. This can contain your own customizations.

You can create a fontinst format by running iniTEX on fontinst.sty then saying \dump.

6 Syntax

There are three types of files used by the fontinst package:

- *fontinst files* contain commands to install fonts. For example, fonttime.tex is a *fontinst file*.
- *encoding files* contain information about an encoding, including the code table, ligatures, and font dimensions. For example, T1.etx is an *encoding file*.
- *metric files* contain information about glyphs, including glyph dimensions, composite characters, and kerning. For example, latin.mtx is a *metric file*.

Any of these files can include the *general commands*, and can use the *integer expressions*, defined in Section 6.2.

6.1 General commands

The following commands can be used anywhere:

\needsfontinstversion{*version*}
: This issues a warning if the current version of the fontinst package is less than *version*.

\setdim{*dim*}{*dimension*}
: If the dimension variable *dim* is currently undefined, it is defined to be the current value of *dimension*.

\setint{*int*}{*integer expression*}
: If the integer variable *int* is currently undefined, it is defined to be the current value of *integer expression*.

\setstr{*str*}{*string*}
: If the string variable *str* is currently undefined, it is defined to be the current value of *string*.

\setcommand{*command*}{*definition*}
: If the command *command* is currently undefined, it is defined to be the *definition*. This uses the same syntax for parameters as the TEX \def command.

\resetdim{*dim*}{*dimension*}
: The dimension variable *dim* is defined to be the current value of *dimension*.

\resetint{*int*}{*integer expression*}
: The integer variable *int* is defined to be the current value of *integer expression*.

\resetstr{*str*}{*string*}
: The string variable *str* is defined to be the current value of *string*.

\resetcommand{*command*}{*definition*}
: This is a synonym for \def.

\ifisint{*int*}\then
: Expands out to \iftrue if the integer variable *int* is defined, and \iffalse otherwise.

\ifisdim{*dim*}\then
: Expands out to \iftrue if the dimension variable *dim* is defined, and \iffalse otherwise.

\ifisstr{*str*}\then
: Expands out to \iftrue if the string variable *str* is defined, and \iffalse otherwise.

\ifisglyph{*glyph*}\then
: Expands out to \iftrue if the glyph variable *glyph* is defined, and \iffalse otherwise.

\ifiscommand{*command*}\then
: Expands out to \iftrue if the command *command* is defined, and \iffalse otherwise.

\unsetdim{*dim*}
: Makes *dim* an undefined dimension.

\unsetint{*int*}
: Makes *int* an undefined integer.

\unsetstr{*str*}
: Makes *str* an undefined string.

\unsetcommand{*command*}
: Makes *command* an undefined command.

6.2 Integer expressions

The *integer expressions* provide a user-friendly syntax for TEX arithmetic. They are used to manipulate any integers, including glyph dimensions, which are given in afm units, that is 1000 to the design size. TEX pl fonts have their dimensions converted to afm units automatically.

The *integer expressions* are:

⟨number⟩
: A TEX ⟨number⟩.

\int{*int*}
: The value of the integer variable *int*.

\width{*glyph*}
: The width of the glyph variable *glyph*.

\height{*glyph*}
: The height of the glyph variable *glyph*.

\depth{*glyph*}
: The depth of the glyph variable *glyph*.

\italic{*glyph*}
: The italic correction of the glyph variable *glyph*.

\kerning{*left*}{*right*}
: The kerning between the *left* and *right* glyph variables.

\neg{*integer expression*}
: The negation of the *integer expression*.

\add{*integer expression*}{*integer expression*}
The sum of the two *integer expression*s.

\sub{*integer expression*}{*integer expression*}
The first *integer expression* minus the second.

\mul{*integer expression*}{*integer expression*}
The product of the two *integer expression*s.

\div{*integer expression*}{*integer expression*}
The first *integer expression* divided by the second.

\scale{*integer expression*}{*integer expression*}
The first *integer expression* times the second, divided by 1000.

7 Fontinst files

A *fontinst file* is any TEX document which inputs the fontinst macros. The commands available are:

\installfonts
install commands
\endinstallfonts
This makes a font family, using the *install commands*. There can be any number of \installfonts commands in a fontinst file.

\substitutesilent{*from*}{*to*}
This declares a LATEX font subsubstitution, that the series or shape *from* should be substituted if necessary by the series or shape *to*. When the font substitution is made, no warning will be made. For example, to say that the series *bx* can be replaced by the series *b*, you say:

```
\substitutesilent{bx}{b}
```

The following declarations are standard:

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

\substitutenoisy{*from*}{*to*}
This is the same as \substitutesilent, but gives a warning when the substitution is made by LATEX. For example:

```
\substitutenoisy{ui}{it}
```

The following declarations are standard:

```
\substitutenoisy{ui}{it}
```

\transformfont{*font-name*}{*transformed font*}

This makes a raw transformed font, for example expanded, slanted, or re-encoded. *It is the responsibility of the device driver to implement this transform.* A *transformed font* is given by:

transformed font	::=	\fromafm{*afm*}
	|	\frommtx{*mtx*}
	|	\frompl{*pl*}
	|	\scalefont{*integer expression*}{*transformed font*}
	|	\xscalefont{*integer expression*}{*transformed font*}
	|	\yscalefont{*integer expression*}{*transformed font*}
	|	\slantfont{*integer expression*}{*transformed font*}
	|	\reencodefont{*etx*}{*transformed font*}

For example, to create an oblique, 8r-encoded version of Adobe Times called ptmro8r you say:

```
\transformfont{ptmro8r}{
  \slantfont{167}{
    \reencodefont{8r}{
      \fromafm{ptmr8a}
    }
  }
}
```

This will create ptmr8a.pl, ptmro8r.pl, ptmr8a.mtx and ptmro8r.mtx, which can then be used as raw fonts in \installfonts. You will have to inform your device driver about the transformed font, using the syntax appropriate for that driver. For example, in dvips you add a line to psfonts.map:

```
ptmro8r Times-Roman ".167 SlantFont TeXBase1Encoding ReEncodeFont" <8r.enc
```

\declaresize{*size*}{*fd-size-range*}

This declares a new size, and gives the fd commands for it. For example, fontinst.sty declares the following sizes:

```
\declaresize{}{<->}
\declaresize{5}{<5>}
\declaresize{6}{<6>}
\declaresize{7}{<7>}
\declaresize{8}{<8>}
\declaresize{9}{<9>}
\declaresize{10}{<10>}
\declaresize{11}{<10.95>}
\declaresize{12}{<12>}
```

```
\declaresize{14}{<14.4>}
\declaresize{17}{<17.28>}
\declaresize{20}{<20.74>}
\declaresize{25}{<24.88>}
```

\declareencoding{*string*}{*etx*}

This declares which etx file corresponds to which encoding string. For example, fontinst.sty declares the following encoding strings:

```
\declareencoding{EXTENDED TEX FONT ENCODING - LATIN}{T1}
\declareencoding{TEX TEXT}{OT1}
\declareencoding{TEX TEXT WITHOUT F-LIGATURES}{OT1}
\declareencoding{TEX TYPEWRITER TEXT}{OT1tt}
\declareencoding{TEX MATH ITALIC}{OML}
\declareencoding{TEX MATH SYMBOLS}{OMS}
\declareencoding{TEX MATH EXTENSION}{OMX}
\declareencoding{TEXBASE1ENCODING}{8r}
```

7.1 Install commands

The *install commands* describe the fonts, glyphs and encodings used to build fonts. The *install commands* are:

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

This produces a LATEX family with the given encoding and family, for example to create the Cork-encoded Times family, you say:

```
\installfamily{T1}{ptm}{}
```

The *fd-commands* are executed every time a font in that family is loaded, for example to stop the Courier font from being hyphenated you say:

```
\installfamily{T1}{pcr}{\hyphenchar\font=-1}
```

\installfont{*font-name*}{*file-list*}{*etx*} {*encoding*}{*family*}{*series*}{*shape*}{*size*}

This produces a TEX virtual font called *font-name* from a comma-separated *file-list* which should be .mtx, .afm or .pl filenames, with an optional 'scaled *scale*' suffix. Any .afm files are also converted into .pl files, for use as 'raw' fonts. The resulting TEX font is encoded using *etx*.etx, and can be accessed in LATEX 2ε with the given *encoding*, *family*, *series* and *shape*. The *size* is either declared by \declaresize, or is an fd size specification. For example, to install the Cork-encoded Times Roman font, you say:

```
\installfont{ptmrq}{ptmr0,latin}{T1}
  {T1}{ptm}{m}{n}{}
```

To install a TEX text-encoded Times Roman font with a scaled version of Symbol for the Greek letters, you say:

```
\installfont{ptmr}{ptmr0,psyr scaled 1100,latin}{OT1}
  {OT1}{ptm}{m}{n}{}
```

To install a version of cmr10 with old style digits, you say:

```
\installfont{cmro10}{cmr10,cmmi10}{OT19}
  {OT1}{cmr9}{m}{n}{<10> <10.95>}
```

This instruction sets the rawscale variable used by \setrawglyph, \setnotglyph and \setkern.

7.2 The Latin commands

The Latin font commands are just syntactic sugar for the *fontinst file* commands. For example, \latinfamily{ptm}{} is syntactic sugar for:

```
\transformfont{ptmr8r}{\reencodefont{8r}{\fromafm{ptmr8a}}}
\transformfont{ptmri8r}{\reencodefont{8r}{\fromafm{ptmri8a}}}
\transformfont{ptmb8r}{\reencodefont{8r}{\fromafm{ptmb8a}}}
\transformfont{ptmbi8r}{\reencodefont{8r}{\fromafm{ptmbi8a}}}
\transformfont{ptmro8r}{\slantfont{167}{\frommtx{ptmr8r}}}
\transformfont{ptmbo8r}{\slantfont{167}{\frommtx{ptmb8r}}}
\installfonts
  \installfamily{T1}{ptm}{}
  \installfamily{OT1}{ptm}{}
  \installfont{ptmr8t}{ptmr8r,latin}{T1}{T1}{ptm}{m}{n}{}
  \installfont{ptmr7t}{ptmr8r,latin}{OT1}{OT1}{ptm}{m}{n}{}
  \installfont{ptmrc8t}{ptmr8r,latin}{T1c}{T1}{ptm}{m}{sc}{}
  \installfont{ptmrc7t}{ptmr8r,latin}{OT1c}{OT1}{ptm}{m}{sc}{}
  \installfont{ptmri8t}{ptmri8r,latin}{T1}{T1}{ptm}{m}{it}{}
  \installfont{ptmri7t}{ptmri8r,latin}{OT1i}{OT1}{ptm}{m}{it}{}
  \installfont{ptmro8t}{ptmro8r,latin}{T1}{T1}{ptm}{m}{sl}{}
  \installfont{ptmro7t}{ptmro8r,latin}{OT1}{OT1}{ptm}{m}{sl}{}
  \installfont{ptmb8t}{ptmb8r,latin}{T1}{T1}{ptm}{b}{n}{}
  \installfont{ptmb7t}{ptmb8r,latin}{OT1}{OT1}{ptm}{b}{n}{}
  \installfont{ptmbc8t}{ptmb8r,latin}{T1c}{T1}{ptm}{b}{sc}{}
  \installfont{ptmbc7t}{ptmb8r,latin}{OT1c}{OT1}{ptm}{b}{sc}{}
  \installfont{ptmbi8t}{ptmbi8r,latin}{T1}{T1}{ptm}{b}{it}{}
  \installfont{ptmbi7t}{ptmbi8r,latin}{OT1i}{OT1}{ptm}{b}{it}{}
  \installfont{ptmbo8t}{ptmbo8r,latin}{T1}{T1}{ptm}{b}{sl}{}
  \installfont{ptmbo7t}{ptmbo8r,latin}{OT1}{OT1}{ptm}{b}{sl}{}
\endinstallfonts
```

8 Encoding files

An *encoding file* (or .etx file) is a TEX document consisting of:

\relax
ignored material
\encoding
encoding commands
\endencoding
ignored material
This describes the encoding of a font, using the *encoding commands*.

Since the encoding file ignores any material between \relax and \encoding, an *encoding file* can also be a LATEXdocument.

8.1 Encoding commands

The *encoding commands* are:

\nextslot{*number*}
Sets the number of the next slot. If there is no \nextslot command, the number is the successor of the previous slot.

\setslot{*glyph*}
slot commands
\endsetslot
Sets the slot of the *glyph*. The *slot commands* describe the glyph, and give its usage in TEX.

\inputetx{*file*}
Inputs the *encoding commands* of *file*.etx.

8.2 Encoding variables

The *encoding files* may set the following integer variables:

boundarychar
The slot of the boundary character.

fontdimen(*n*)
The value of the *n*th font dimension.

letterspacing
The extra space to be added between every glyph.

In addition, they will set an integer variable for each slot in the encoding. The slot number for *glyph* is \int{*glyph*}.

The *encoding files* may set the following string variables:

codingscheme
The font coding scheme.

8.3 Slot commands

The *slot commands* are:

\comment{*text*}
A comment, which is ignored by fontinst.

\ligature{*ligtype*}{*glyph*}{*glyph*}
Specifies a ligature of type *ligtype* from the current glyph followed by the first glyph to the second glyph. The *ligtypes* are as in vpl files (see the vptovf Web source for more details). For example:

```
\setslot{ff}
  \ligature{LIG}{i}{ffi}
  \ligature{LIG}{l}{ffl}
  \comment{The ‘ff’ ligature.}
\endsetslot
```

\usedas{*type*}{*control sequence*}
Sets the TEX control sequence for this slot, with the *type* taken from:

```
char       accent     mathord
mathbin    mathrel    mathopen
mathclose  mathpunct  mathvariable
mathaccent mathdelim
```

\nextlarger{*glyph*}
Sets a NEXTLARGER entry from the current slot to the *glyph*.

\varchar
varchar commands
\endvarchar
Sets a VARCHAR entry for the current slot, using the *varchar commands*.

8.4 Varchar commands

The *varchar commands* are:

\vartop{*glyph*}
Sets the top *glyph* of the VARCHAR.

\varmid{*glyph*}
Sets the middle *glyph* of the VARCHAR.

\varbot{*glyph*}
Sets the bottom *glyph* of the VARCHAR.

\varrep{*glyph*}
Sets the repeated *glyph* of the VARCHAR.

9 Metric files

A *metric file* (or .mtx file) is a TEX document consisting of:

\relax
ignored material
\metrics
metric commands
\endmetrics
ignored material

This describes the glyphs in a font, using the *metric commands*.

9.1 Metric commands

The *metric commands* are:

\setglyph{*name*}
glyph commands
\endsetglyph

If the glyph called *name* is undefined, it is built using the *glyph commands* given below, for example:

```
\setglyph{IJ}
  \glyph{I}{1000}
  \glyph{J}{1000}
\endsetglyph
\setglyph{Asmall}
  \glyph{A}{700}
\endsetglyph
```

\resetglyph{*name*}
glyph commands
\endsetglyph

Gives the definition of the glyph called *name* using the *glyph commands*.

\unsetglyph{*name*}

Makes the glyph called *name* undefined.

\setrawglyph{*name*}{*font*}{*dimen*}{*integer*}
{*integer*}{*integer*}{*integer*}{*integer*}

This sets a glyph called *name* from the *font*, which has the given design size, slot, width, height, depth and italic correction. If the integer variable rawscale is set, the glyph will be scaled by that amount. This command will usually be generated automatically from an afm or pl file.

\setnotglyph{*name*}{*font*}{*dimen*}
{*integer*}{*integer*}{*integer*}{*integer*}

This sets a glyph called *name*-not, which is present in the *font*, but is not in the default encoding. It takes the same arguments as \setrawglyph, although the slot will normally be -1. This command will usually be generated automatically from an afm file.

\setkern{*glyph*}{*glyph*}{*integer expression*}
Sets a kern between the two glyphs, scaled by the current value of rawscale, if it has been set.

Note that there is a bug with the current implementation: if more than one kern value is given for the same pair of glyphs, then both are written to the (V)PL file. Really it should only be the first pair (to fit with all the other \set... commands). This bug is not likely to be fixed soon.

\setleftkerning{*glyph*}{*glyph*}{*integer expression*}
Sets the amount by which the first glyph should mimic how the second glyph kerns on the left, for example:

```
\setleftkerning{Asmall}{A}{850}
\setleftkerning{IJ}{I}{1000}
```

\setrightkerning{*glyph*}{*glyph*}{*integer expression*}
Sets the amount by which the first glyph should mimic how the second glyph kerns on the right, for example:

```
\setrightkerning{Asmall}{A}{850}
\setrightkerning{IJ}{J}{1000}
```

\inputmtx{*file*}
Inputs the *metric commands* of *file*.mtx.

9.2 Metric variables

The *metrics files* may set the following integer variables:

ascender
The height of the tallest lower-case letter.

capheight
The height of the tallest capital letter.

descender
The depth of the lowest lower-case letter.

italicslant
The ratio of the italic slant, given in units of rightward movement for each 1000 units of upward movement.

minimumkern
Any kern smaller than this amount is ignored.

monowidth
This variable is set if the font is monowidth.

underlinethickness
The width of the underline rule.

xheight
The x-height.

***glyph*-spacing**
The letter spacing for the *glyph*.

The *metrics files* may set the following dimension variables:

designsize
The design size of the font.

9.3 Glyph commands

The *glyph commands* are:

\glyph{*glyph*}{*integer expression*}
Sets the named glyph at the given scale, with 1000 as the natural size. This:

- Advances the current glyph width.
- Sets the current glyph height to be at least the height of the named glyph, adjusted for the current vertical offset.
- Sets the current glyph depth to be at least the depth of the named glyph, adjusted for the current vertical offset.
- Sets the current glyph italic correction to be the same as the set glyph.

The named glyph must have already been defined, otherwise an error will occur. For example:

```
\setglyph{fi}
  \glyph{f}{1000}
  \glyph{i}{1000}
\endsetglyph
```

\glyphrule{*integer expression*}{*integer expression*}
Sets a rule of the given width and height, for example:

```
\setglyph{underline}
  \glyphrule{333}{40}
\endsetglyph
```

\glyphspecial{*text*}
Sets a driver-dependent \special, for example:

```
\setglyph{crest}
  \glyphspecial{Filename: crest.eps}
\endsetglyph
```

\glyphwarning{*text*}
Sets a warning \special, and produces a warning message each time the glyph is used, for example:

```
\setglyph{missingglyph}
  \glyphrule{500}{500}
  \glyphwarning{missing glyph}
\endsetglyph
```

\movert{*integer expression*}

Moves right by the given amount, and advances the current glyph width, for example:

```
\setglyph{Asmall}
  \movert{50}
  \glyph{A}{700}
  \movert{50}
\setendglyph
```

\moveup{*integer expression*}

Moves up by the given amount, and advances the current vertical offset. Each glyph should always end at vertical offset zero, for example:

```
\setglyph{onehalf}
  \moveup{500}
  \glyph{one}{700}
  \moveup{-500}
  \glyph{slash}{1000}
  \moveup{-200}
  \glyph{two}{700}
  \moveup{200}
\endsetglyph
```

\push
glyph commands
\pop

Performs the *glyph commands* without adjusting the current position or glyph width, for example:

```
\setglyph{aacute}
  \push
    \movert{\div{\sub{\width{a}}{\width{acute}}}{2}}
    \glyph{acute}{1000}
  \pop
  \glyph{a}{1000}
\endsetglyph
```

\glyphpcc{*glyph*}{*integer expression*}{*integer expression*}

This is generated from PCC instructions in an afm file, and is syntactic sugar for:

```
\push
\movert{first integer expression}
```

\moveup{*second integer expression*}
\glyph{*glyph*}{1000}
\pop

\resetwidth{*integer expression*}
Sets the width of the current glyph.

\resetheight{*integer expression*}
Sets the height of the current glyph.

\resetdepth{*integer expression*}
Sets the depth of the current glyph.

\resetitalic{*integer expression*}
Sets the italic correction of the current glyph.

\samesize{*glyph*}
Sets the dimensions of the current glyph to be the same as *glyph*.

Inside the definition of *glyph*, you can use expressions such as \width{*glyph*}, which will refer to the glyph defined so far. For example, a display summation can be defined to be a text summation $\sum$ scaled 120% with 0.5pt extra height and depth using:

```
\setglyph{sumdisplay}
  \glyph{textsum}{1200}
  \resetheight{\add{\height{sumdisplay}}{50}}
  \resetdepth{\add{\depth{sumdisplay}}{50}}
\endsetglyph
```

Within a \resetglyph, these expressions will refer to the previous definition of the glyph. For example, you can add sidebearings to the letter 'A' with:

```
\resetglyph{A}
  \movert{25}
  \glyph{A}{1000}
  \movert{25}
\endresetglyph
```

10 Future work

The fontinst software is now fairly stable, but there are some things I would like to do sometime...

- See if it's possible to automatically generate 'first cut' math fonts.
- Add a way to use Alternate sets and SC fonts.
- Find out if there's a way to generate smaller VPLs for T1 small caps fonts.
- Allow reals rather than just integers in AFM files.
- Fix the bug with multiple \setkern entries.

A more detailed list is distributed as the file TODO.

Acknowledgements

I'd like to thank all of the fontinst α-testers, especially Karl Berry, Damian Cugley, Steve Grahthwohl, Yannis Haralambous, Alan Hoenig, Rob Hutchings, Constantin Kahn, Peter Busk Laursen, Ciarán Ó Duibhín, Hilmar Schlegel, Paul Thompson, Norman Walsh and John Wells, who made excellent bug-catchers!

Thanks to Barry Smith, Frank Mittelbach, and especially Sebastian Rahtz for many useful email discussions on how virtual fonts should interact with LaTeX 2_ε.

Thanks to Karl Berry and Damain Cugley for detailed comments on this documentation.

Thanks to David Carlisle for the use of his trig macros for calculating trignometry.

Warranty and distribution

There is no warranty for the fontinst package, to the extent permitted by applicable law. Except when otherwise stated in writing, the author provides the program 'as is' without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the package prove defective, you assume the cost of all necessary servicing, repair or correction.

In no event unless required by applicable law or agreed to in writing will the author be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

Redistribution of unchanged files is allowed provided that all files listed in the MANIFEST file are distributed.

If you receive only some of these files from someone, or if you receieve altered files, then complain!