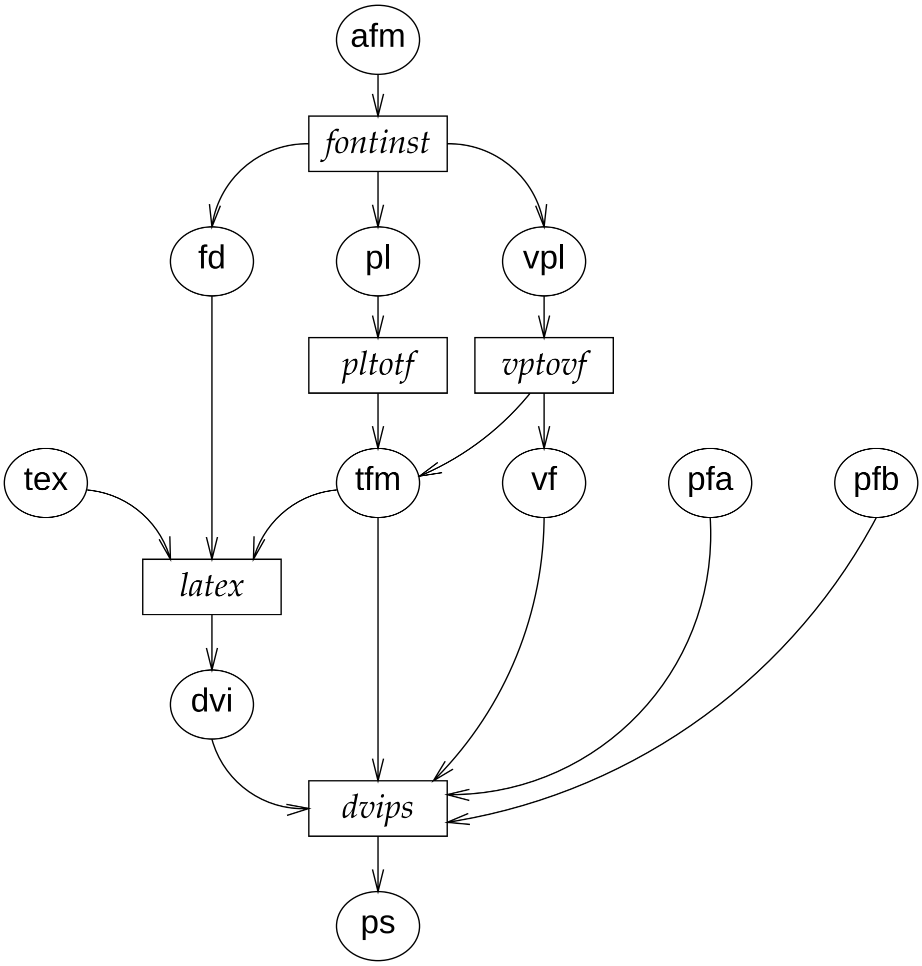
fontinst

Font installation software for TEX



Alan Jeffrey and Rowland McDonnell

fontinst v1.8 · 30 June 1998

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 written to create virtual fonts for use with TEX. Its main use is creating the files needed so you can use PostScript Type 1 fonts with LATEX.

FONTINST needs information about the fonts it works with. This information needs to be supplied in an Adobe Font Metric (`afm`) or TEX Property List (`pl`) file. `pl` files can be created from `tfm` files using TFTOPL, a program normally included with a TEX system.

The job that FONTINST does is complicated, but it can be used for many tasks by people who are not TEX font wizards. Having said that, you do need to understand at least the basics of LATEX 2ε's font selection mechanism, which is documented in `fntguide.tex`, part of the standard LATEX distribution. `ftp://ftp.tex.ac.uk/tex-archive/macros/latex/base/fntguide.tex` will fetch a copy if you don't have one to hand.

To get the most benefit out of FONTINST, it's important to understand and use Karl Berry's 'Fontname' naming scheme. The definitive version of this is available from your nearest CTAN server. The following URL will fetch all the files needed compressed into a single ZIP archive: `ftp://ftp.tex.ac.uk/tex-archive/info/fontname.zip`. I suggest that you print out the Fontname documentation and have it handy when you're learning about FONTINST.

The FONTINST 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.
- Can automatically generate 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.

1.1 What does fontinst do?

FONTINST is a tool written in TEX that can create the various extra files needed so you can use PostScript fonts with LATEX and TEX. It can read in `afm` files, and produces the necessary `vpl`, `pl`, and `fd` files to use the fonts (the human-readable `vpl` and `pl` files produced by FONTINST are turned into the machine-readable `vf` and `tfm` forms by VPTOVF and PLTOTF). It does not help you configure your DVI driver.

There also exists a `perl` front-end to FONTINST, intended specifically for use with a Unix TEX system, which takes care of routine tasks such as running VPTOVF and PLTOTF on the generated files after FONTINST has finished it's job. It also generates a font map file for use with DVIPS.

FONTINST's main job is creating `vf` files (virtual fonts). Not all TEX systems can use them. As far as I know, all current (1998) free and shareware TEX systems can; virtual fonts have been in widespread use with TEX since 1990. If you have a TeX system that can't use virtual fonts, FONTINST is most likely useless to you.

There are some nice things about having a tool written in TEX to do this: it's completely portable and you can modify its behaviour using TEX commands. The only real problem is that it's relatively slow: you can expect a typical FONTINST run to take something like 10–20 minutes on, say, a 40 MHz 80486SX PC or a 25 MHz 68LC040 Macintosh.

FONTINST can do its work on any font for which you have a corresponding `afm` or `tfm` metric file, so it's not limited to working with PostScript fonts; I have used it to produce the files I needed to use TrueType fonts with LATEX. Whether or not you can do this depends on whether or not you have suitable metric files and whether or not your TEX system can use TrueType fonts. In particular, the pdfTEX program supports TrueType fonts and includes a utility `ttf2afm` to generate `afm` files from `ttf` fonts.

Some people have used FONTINST to produce 'special effects' with normal TEX fonts. One example is the ECO set of fonts (available from CTAN: `ftp://ftp.tex.ac.uk/tex-archive/fonts/eco/`). These fonts are the same as the standard EC (European Modern) fonts, but with normal numerals replaced with old style numerals – 12345 rather than 12345 – everywhere except in maths mode.

1.2 Installation

To install FONTINST, put the contents of the `inputs/tex`, `inputs/etx`, `inputs/mtx` and `examples` directories into a directory read by TEX, for example `TEXMF/tex/generic/fontinst`.

When you use FONTINST, you need to make sure that the `afm` and `pl` files it will work on are in a directory searched by TEX.

If you are using `web2c` TEX on a Unix system with the TEX directory structure (TDS), you might put all the `afm` files in subdirectories of `TEXMF/fonts/afm/*`. And then say:

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

Note that `pl` files are not normally kept in TEX installations, so if you want to use MF fonts with FONTINST you have to generate the corresponding `pl` files from `tfm` files and put them in your working directory before running FONTINST.

You could adopt a similar strategy with other TEX systems: create directories for the required files and then change the relevant parameter (`input_folders` in the default configuration file with OzTEX, for example).

The approach I use is this: I write a file containing commands for FONTINST to process, and put the `afm` and `pl` files needed in the same directory as that file. When FONTINST has finished working, I delete the `afm` and `pl` files because they are not needed and waste space on my hard disc drive. Some application programs on some computers need `afm` files, so it's not always a good idea to remove them completely.

1.3 Why do we need fontinst?

TEX refers to characters by number when it's typesetting. When you use a command like `\i`, TEX puts a number (16 if you're using OT1, 25 if you're using T1) into the `dvi` file. If you're using a font designed for use with TEX, this number will correspond to the character 'ı'. Assuming OT1 encoding for the moment, when you come to print out your `dvi` file, the DVI driver will see the number 16 in the `dvi` file, and select the character that sits in position 16 of the corresponding printer font file (a `pk` file in the case of normal TEX fonts). Unless something has gone wrong, that will result in the character 'ı' being placed on the page.

It's useful to think of these numbers and the actual characters corresponding to each number as sets called 'encodings'. A particular set of characters are assigned particular numbers. An example of an encoding is shown in table 1.

TEX began life using 7-bit fonts. This means the original TEX fonts used the numbers 0–127 to represent characters: 128 characters per font. TEX can now use 8-bit fonts: 256 numbers from 0–255, but even so, most typesetting with TEX still uses the original 7-bit encoding, now called 'OT1' (Old TEX 1 encoding). This has a correspondance between numbers and characters shown in table 1. The numbers used in that table are hexadecimal and octal because it makes for a neat table and anyway I stole the code to generate it from Donald Knuth and that's how he did it.

Returning to the example above, if you've a number 16 in your `dvi` file (expecting 'ı', a dotless i), but rather than printing with an OT1 encoded font, you print using a non re-encoded PostScript font in Adobe standard encoding, you'll get a blank, because the Adobe standard encoding has nothing in that character position.

There are several ways round this problem; I'll consider two cases here. If you are using LATEX you can tell it about a new encoding and re-define the commands that produce characters like 'ı' that live in different positions in

	´0	´1	´2	´3	´4	´5	´6	´7	
´00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	″0x
´01x	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl	
´02x	ı	ȷ	`	´	ˇ	˘	¯	˚	″1x
´03x	¸	ß	æ	œ	ø	Æ	Œ	Ø	
´04x	-	!	”	#	$	%	&	’	″2x
´05x	(	)	*	+	,	-	.	/	
´06x	0	1	2	3	4	5	6	7	″3x
´07x	8	9	:	;	¡	=	¿	?	
´10x	@	A	B	C	D	E	F	G	″4x
´11x	H	I	J	K	L	M	N	O	
´12x	P	Q	R	S	T	U	V	W	″5x
´13x	X	Y	Z	[	“	]	ˆ	˙	
´14x	‘	a	b	c	d	e	f	g	″6x
´15x	h	i	j	k	l	m	n	o	
´16x	p	q	r	s	t	u	v	w	″7x
´17x	x	y	z	–	—	˝	˜	¨	
	″8	″9	″A	″B	″C	″D	″E	″F	

Table 1: The OT1 font encoding

different encodings, or you can use a tool to re-encode the font so that it has the expected characters in the appropriate positions.

Re-encoding is the approach FONTINST uses: it can produce files to map the characters in the new font to one of TEX's existing encodings; this works with formats other than LATEX.

The first approach is used to define the standard encodings that LATEX uses. See, for example, the file `ot1enc.def` that comes with the current LATEX distribution, which defines the a few commands that refer to characters which aren't in the positions TEX would otherwise assume. This works only with modern versions of LATEX.

The second approach is used to allow you to use fonts in other encodings with any dialect of TEX. It has the some advantages over the first method: it works with any TEX format; and it improves portability, because you can typeset a document using a standard TEX encoding, sure that the same document will print correctly on a different kind of computer using a font with a different encoding. For example, you might say:

```
\usepackage{times}
```

in the preamble of your document. On my computer, that means my DVI driver will use a Macintosh encoded TrueType version of Times. On your computer, it might mean the dvi driver will use a Unicode encoded PostScript version of Times-Roman. The results will be identical in either case, without needing to modify the document.

1.4 How do you use fontinst?

FONTINST works on `afm` files named (more-or-less) according to Karl Berry's font naming scheme (see `ftp://ftp.tex.ac.uk/tex-archive/info/fontname` at CTAN). Let's say you want to use the Adobe Times fonts. You can get the metric files for this font from CTAN:

Location of file at CTAN	*Rename to*
`fonts/psfonts/adobeafm/base35/tib_____.afm`	`ptmb8a.afm`
`fonts/psfonts/adobeafm/base35/tibi____.afm`	`ptmbi8a.afm`
`fonts/psfonts/adobeafm/base35/tii_____.afm`	`ptmri8a.afm`
`fonts/psfonts/adobeafm/base35/tir_____.afm`	`ptmr8a.afm`

The new name is the name you should give the `tfm` files so that FONTINST understands what each file contains. The initial '`p`' means 'Adobe'; '`tm`' means 'Times'; '`b`' bold, '`r`' roman, '`i`' italic; and '`8a`' means 'Adobe standard encoding'.

The simplest use of FONTINST is to put the four `afm` files in the same directory as `fontinst.sty` and run TEX on `fontinst.sty`. At the `*` prompt type:

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

Some time later (about 17 minutes on my rather old computer), FONTINST will have finished, having created:

- Two `pl` files for each `afm` file
- One `vpl` file for each TEX font
- One `fd` file for each family

The `pl` files come in pairs: for example, `ptmb8a.pl` and `ptmb8r.pl`. The `8a` version has the same encoding as the original font; the `8r` version is re-encoded to `TeXBase1` (`8r`) encoding, and is the font that is the base on which the T1 and OT1 encoded versions are based on. The raw `8a` (Adobe standard) encoded font is not normally used.

These can be converted to TEX fonts using PLTOTF or VPTOVF. If you have OzTEX, launch OzMF, select PLTOTF (or VPTOVF) from the Tools menu, and say 'Do all files'.

If you use the `bash` shell on a Unix system, you can process all files using these one-liners at the `$` prompt:

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

(This assumes that PLTOTF and VPTOVF can deduce the file names of the corresponding `tfm` and `vf` files automatically.)

You should then:

- Move the `tfm` files to your TEX fonts directory (e.g. `TEXMFLOCAL/fonts/tfm/*`).
- Move the `vf` files to your virtual fonts directory (e.g. `TEXMFLOCAL/fonts/vf/*`).
- Move the `fd` files to your TEX inputs directory (e.g. `TEXMFLOCAL/tex/latex/psfonts/*`).

If your TEX installation is organized using the TEX directory structure (TDS), it is customary to subdivide the `tfm` and `vf` files into subdirectories by supplier and typeface name.

The `pl`, `vpl`, and `mtx` files are debris that can now be deleted. `mtx` files are font metric files FONTINST creates for its own use from `afm` and `pl` files. They're just more convenient for TEX to read than other forms – think of them as FONTINST readable `afm` and `pl` files.

By now, you have all the files in place to produce a `dvi` file using the new fonts. You can make Adobe Times the default roman font in your document by putting this in your preamble:

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

TEX will now happily produce a perfectly good `dvi` file including the new font, which your DVI driver will choke on because you've not yet told it about the new fonts. Exactly how you do this depends on the dvi driver, but they all need the same information: the name of a TEX font; the printer font name it corresponds to; some information to handle an re-encoding needed; and (in the case of a PostScript driver) perhaps an instruction to download the font to the printer. You don't need to download the Times font to a PostScript printer, because Times is built in to every PostScript printer.

If you use DVIPS, these lines added to your `psfonts.map` file will do the job:

```
ptmr8r   Times-Roman       "TeXBase1Encoding ReEncodeFont"  <8r.enc
ptmri8r  Times-Italic      "TeXBase1Encoding ReEncodeFont"  <8r.enc
ptmb8r   Times-Bold        "TeXBase1Encoding ReEncodeFont"  <8r.enc
ptmbi8r  Times-BoldItalic  "TeXBase1Encoding ReEncodeFont"  <8r.enc
ptmro8r  Times-Roman       "0.167 SlantFont TeXBase1Encoding ReEncodeFont"  <8r.enc
ptmbo8r  Times-Bold        "0.167 SlantFont TeXBase1Encoding ReEncodeFont"  <8r.enc
```

And now you can print a `dvi` file containing the new fonts. If you really do want to use the 'raw' `8a` encoded fonts for some reason, you need to add these lines to your `psfonts.map` file:

```
ptmr8a   Times-Roman
ptmri8a  Times-Italic
```

```
ptmb8a   Times-Bold
ptmbi8a  Times-BoldItalic
ptmro8a  Times-Roman       "0.167 SlantFont"
ptmbo8a  Times-Bold        "0.167 SlantFont"
```

Assuming that you're using the `fd` file that FONTINST has produced, and that you've asked for the Adobe Times family (`ptm`) in medium series (`m`) and upright shape (`n` for normal) using the NFSS font selection commands:

```
\renewcommand{\rmdefault}{ptm}
\rmfamily \mdseries \upshape
```

Assuming further that you are using the `fd` file `t1ptm.fd` produced by FONTINST, the TEX font (`tfm` file) `ptmr8t.tfm` will be selected by the above commands, as you can see from the relevant line in `t1ptm.fd`:

```
\DeclareFontShape{T1}{ptm}{m}{n}{<-> ptmr8t}{}
```

This is what happens:

- TEX typesets your document using the font metric file `ptmr8t.tfm`; this is the font that is put in the `dvi` file.
- DVIPS looks at the `dvi` file, and sees a reference to the font `ptmr8t`.
- DVIPS searches for a `vf` file corresponding to `ptmr8t`; when it finds `ptmr8t.vf`, it knows it has a virtual font on its hands.
- DVIPS follows the instructions in the `vf` file, which map characters in `ptmr8t.tfm` to characters in the font `ptmr8r.tfm`. That is, when it sees a number 25 in the `dvi` file (dotless i – 'ı' – in T1 encoding), it replaces it with a number 17, which is a dotless i in `8r` encoding.
- Then DVIPS looks up the name of each number according to the scheme given in the file `8r.enc`, and replaces each number with the name of the character, in this case, number 17 is listed as 'dotlessi'.
- And finally, DVIPS tells the printer to print the named character.

Not all DVI drivers can manage re-encoding as well as DVIPS can. For example, OzTEX's built-in non-PostScript `dvi` driver can only work with numbers, so if I'm using a PostScript font, I can't print characters (such as Eth) that don't have a number in Macintosh text encoding unless I use DVIPS and print on a PostScript printer. DVIPS works with character names, so it's not subject to this restriction. In the example above, OzTEX would replace the number 17 for 'dotlessi' in `8r` encoding with a number 245 for 'dotlessi' in Macintosh text encoding.

The details of the LATEX 2ε font selection scheme are described in *LATEX 2ε font selection* (distributed with LATEX 2ε as the file `fntguide.tex`) and *The LATEX Companion* (Goossens, Mittelbach and Samarin, Addison-Wesley).

The files you need to use Times, Helvetica, Courier, and the rest of the 'standard' PostScript fonts are distributed as part of the PSNFSS bundle available from CTAN, so there's no need to create new files to use these fonts.

A more involved example of FONTINST use can be seen in the file `fontptcm.tex` which creates the files you need to use a combination of Times, Symbol, Zapf Chancery and Computer Modern as TEX math fonts.

2 Installing your own font family

The FONTINST package has a command `\latinfamily` meant to do most of the work to install a 'normal' set (family) of roman text fonts from Adobe. Assuming you have a set of `afm` files to match the fonts you wish to use, the first step is to rename the `afm` files according to the Fontname naming scheme.

A 'normal' set of text fonts usually includes the basic upright roman version, bold, italic, and bold italic. Sometimes there will also be small caps versions, perhaps some 'expert' fonts, and maybe some other weights such as light, medium, semi bold, black or ultra bold.

The most important point to note is this: no matter what sort of computer you're using and no matter what font encoding it uses normally, `afm` files for text fonts are almost always in `8a` encoding (Adobe standard encoding), so the `afm` files when renamed normally end in `8a`.

A typical set of four `afm` files re-named for use with FONTINST is this:

```
ptmr8r.afm    Times-Roman
ptmri8r.afm   Times-Italic
ptmb8r.afm    Times-Bold
ptmbi8r.afm   Times-BoldItalic
```

Not all `afm` files use `8a` encoding. If you open an `afm` file using a text editor, you'll see a line looking like this somewhere near the top:

```
EncodingScheme AdobeStandardEncoding
```

and if you see exactly that, the `afm` file should end with `8a`. If you see something like this:

```
EncodingScheme FontSpecific
```

have a look at the name of the font in the `afm` file. If you see something like this:

```
FontName AGaramondExp-Regular
FullName Adobe Garamond Regular Expert
```

you have an 'expert' encoded font on your hands, and the `afm` file should end with `8x` to indicate this to FONTINST. An `8x` encoded font contains extra glyphs like old style numerals, small capital letters, more ligatures, and so on.

The `\latinfamily` command is used like this:

`\latinfamily{⟨family⟩}{⟨commands⟩}`

This installs a Latin family of fonts.

For example, to install Adobe Times, you say:

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

The *commands* 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 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.

For example, to install Adobe Garamond using expert fonts, you say:

```
\latinfamily{padx}{}
```

To install Adobe Garamond using expert fonts with oldstyle digits, you say:

```
\latinfamily{padj}{}
```

When you have expert fonts, and you've told FONTINST to use them, it will carry on as normal, but the resulting font family will have the name '`padx`' or '`padj`', and it will use expert glyphs whenever possible, so you'll have a real (rather than faked) small caps font, real (rather than faked) 'ffl' ligatures, and so on.

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 *supplier* (or foundry), such as '`p`' for Adobe.
- a *typeface*, such as '`ad`' for Adobe Garamond.
- up to two *variants*, such as '`j`' or '`x`' for 'old style digits' or 'expert'.

`b`	Bitstream
`f`	'free' (public domain)
`h`	Bigelow & Holmes
`i`	ITC
`l`	Linotype
`m`	Monotype
`p`	Adobe (`p` for PostScript)
`r`	'raw' (obsolete)
`u`	URW
`z`	bizarre

Table 2: A partial list of foundries

`a`	alternate
`d`	display, titling
`f`	fraktur, handtooled
`j`	oldstyle digits
`n`	informal, casual
`p`	ornaments
`s`	sans serif
`t`	typewriter
`w`	script, handwriting, swash
`x`	expert

Table 3: A partial list of variants

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

Table 4: A partial list of shapes

`ac`	Adobe Caslon
`ad`	Adobe Garamond
`ag`	Avantgarde
`bb`	Bembo
`bd`	Bodoni
`bk`	Bookman
`bv`	Baskerville
`ca`	Caslon
`ch`	Charter
`cr`	Courier
`fr`	Frutiger
`fu`	Futura
`gl`	Galliard
`gm`	Garamond
`gs`	Gill Sans
`hv`	Helvetica
`mn`	Minion
`lc`	Lucida
`lh`	Lucida Bright
`ls`	Lucida Sans
`nb`	New Baskerville
`nc`	New Century Schoolbook
`op`	Optima
`pl`	Palatino
`sy`	Symbol
`tm`	Times
`ut`	Utopia
`zc`	Zapf Chancery
`zd`	Zapf Dingbats

Table 5: A partial list of faces

So the family name '`padj`' indicates Adobe Garamond with old style digits. Note that the variants '`j`' or '`x`' are interpreted by FONTINST itself and do not appear in external font names, whereas other variants are passed through as part of the font names. (This is needed for families which have a sans serif or typewriter variant.)

The *supplier* must be one letter, and the *typeface* 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, typefaces, shapes 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 2–5.

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 *supplier*, such as '`p`' for Adobe.
- a *typeface*, such as '`hv`' for Helvetica.
- a *weight*, such as '`r`' for regular.

`b`	bold
`c`	black
`d`	demibold
`h`	heavy
`k`	book
`l`	light
`m`	medium
`r`	regular
`s`	semibold
`u`	ultra bold
`x`	extra bold

Table 6: A partial list of weights

`c`	condensed
`n`	narrow
`w`	wide
`x`	extended

Table 7: A partial list of widths

`8a`	Adobe Standard
`8x`	Adobe Expert
`8r`	TEX 8-bit 'raw' (`TeXBase1`)
`8y`	TEX 8-bit 'raw' (`TeXnANSI`)
`7t`	TEX 7-bit text (`OT1`)
`7m`	TEX 7-bit math italic (`OML`)
`7y`	TEX 7-bit math symbol (`OMS`)
`7v`	TEX 7-bit math extension (`OMX`)
`8t`	TEX 8-bit text (`T1`)
`8c`	TEX 8-bit text symbols (`TS1`)
`9t`	TEX 7-bit text with expert glyphs
`9o`	TEX 7-bit text with expert glyphs and old-style digits
`9e`	TEX 8-bit text with expert glyphs
`9d`	TEX 8-bit text with expert glyphs and old-style digits
`9c`	TEX 8-bit symbols with expert glyphs and old-style digits

Table 8: A partial list of encodings

- up to two *shapes* or *variants*, such as '`o`' for oblique.
- an *encoding*, such as '`7t`' for Knuth's 7-bit TEX encoding.
- an optional *width*, such as '`n`' for narrow.
- a *file extension*, such as '`.tfm`' for TEX Font Metric.

So the filename name '`phvro7tn.tfm`' indicates Adobe Helvetica regular oblique narrow, 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 Garamond including the expert fonts, you would need to rename the `afm` files:

Adobe name	*ATM name*	*Fontinst name*
`AGaramond-Bold.afm`	`gdb_____.afm`	`padb8a.afm`
`AGaramond-BoldItalic.afm`	`gdbi____.afm`	`padbi8a.afm`
`AGaramond-Italic.afm`	`gdi_____.afm`	`padri8a.afm`
`AGaramond-Regular.afm`	`gdrg____.afm`	`padr8a.afm`
`AGaramond-Semibold.afm`	`gdsb____.afm`	`pads8a.afm`
`AGaramond-SemiboldItalic.afm`	`gdsbi___.afm`	`padsi8a.afm`
`AGaramondExp-Bold.afm`	`geb_____.afm`	`padb8x.afm`
`AGaramondExp-BoldItalic.afm`	`gebi____.afm`	`padbi8x.afm`
`AGaramondExp-Italic.afm`	`gei_____.afm`	`padri8x.afm`
`AGaramondExp-Regular.afm`	`gerg____.afm`	`padr8x.afm`
`AGaramondExp-Semibold.afm`	`gesb____.afm`	`pads8x.afm`
`AGaramondExp-SemiboldItalic.afm`	`gesbi___.afm`	`padsi8x.afm`
`AGaramond-RegularSC.afm`	`gdsc____.afm`	`padrc8a.afm`
`AGaramond-SemiboldSC.afm`	`gdsbs___.afm`	`padsc8a.afm`

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

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

Not all font families can be installed using the `\latinfamily` command, so the rest of this document describes the full FONTINST syntax, and is intended for 'power users'.

3 Defining terms

This is rather a large and perhaps tedious section. You might be tempted to skip it so you can get to some more direct information on how to use FONTINST. That's fine if you understand everything about how TEX handles fonts. If not, I suggest you at least skim though this section.

3.1 What's a font?

Once upon a time, this question was easily answered: a font is a set of type in one size, style, etc. There used to be no ambiguity, because a font was a collection of chunks of type metal kept in a drawer, one drawer for each font.

These days, with digital typesetting, things are more complicated. What a font 'is' isn't easy to pin down. A typical use of a PostScript font with TEX might use these elements:

- Type 1 printer font file
- Bitmap screen font file
- Adobe font metric file (`afm` file)
- TEX font metric file (`tfm` file)
- Virtual font file (`vf` file)
- font definition file (`fd` file)

Looked at from a particular point of view, each of these files 'is' the font. So what's going on?

3.1.1 Type 1 printer font files

These files contain the information needed by your printer to draw the shapes of all the characters in a font. They're typically files with a `pfa` or `pfb` extension; on Macs they're usually in files of type 'LWFN' which usually have icons that look like a laser printer. The information in all these files is basically the same: the only difference is in its representation. `pfa` stands for 'printer font ASCII', while `pfb` stands for 'printer font binary'. That is, `pfa` files contain plain text information, while `pfb` files contain the same information encoded as machine-readable binary data.

If you have Adobe Type Manager (ATM) installed on your computer, ATM will use these files to draw an accurate representation of the letters on the screen of your computer when you are previewing a TEX document.

Printer font files are not used directly by TEX at all – TEX just prepares a `dvi` file that refers to the fonts by name and the characters by number: TEX knows nothing about the shapes involved. The DVI driver uses the printer font files when you ask it to print the `dvi` file. This means that you can produce a `dvi` file which uses, say, Palatino, even if you do not have the Type 1 printer font file for this font on your computer. You will need to find a computer that does have Palatino before you can print it or preview it, though.

(This isn't exactly true for the recently developped pdfTEX program, which integrates some of the functionality of a DVI driver.)

3.1.2 Bitmap screen font files

These files contain a low-resolution bitmap for drawing a representation of the font on the screen of your computer if ATM is not installed. In the TEX world, these files are only used for screen previews by the DVI driver. They are kept in font suitcase files on Macintoshes.

3.1.3 Adobe font metric files (`afm` files)

These files are text files which contain information about the size of each character in a font, kerning and ligature information, and so on. They can't be used by TEX directly, but the information they contain is essential if you want to use a font with TEX. FONTINST can create from an `afm` file the necessary `tfm` and `vf` files so you can use a font with TEX. Once you have created all the files you need to use a font with TEX, you can remove the corresponding `afm` files from your computer unless you have other software that needs them.

The job of turning an `afm` file into a set of `tfm` and `vf` files is one of the main uses for FONTINST. Most of this document is concerned with this process, so don't worry if it seems a bit vague at the moment.

3.1.4 TEX font metric files (`tfm` files)

These are binary data files in a format designed for use by TEX which contain (more-or-less) the same information as `afm` files: the size of each character in a font (font metric data), kerning, and ligature information.

When you select a font in TEX, you are telling TEX to typeset using a particular `tfm` file; from TEX's point of view, a `tfm` file (and nothing else) *is* a font. TEX itself doesn't see printer font files, screen bitmaps, `pk` files, `vf` files, or anything else to do with fonts: only `tfm` files.

TEX uses these `tfm` files to decide where to put characters when typesetting. From TEX's point of view, `tfm` files are fonts, even though they contain no information about the shape of letters, and are not used by anything except

TEX – once you have produced a `dvi` file, you don't need the `tfm` files to print it out. (This is a slight lie: DVIPS can read `tfm` files corresponding to PostScript and TrueType fonts so it can modify the metrics slightly to improve the letterspacing at your chosen output resolution. This is an optional minor tweak and not an essential part of the output process.)

3.1.5 Property list files (`pl` files)

`pl` files are human-readable text files which contain all the font metric, kerning, ligature, and other information needed to create a `tfm` file. You can convert between the two file formats using TFTOPL and PLTOTF.

3.1.6 Virtual font files (`vf` files)

These are binary data files in a format designed for use by TEX `dvi` drivers. They're main purpose in life is to let you use fonts in different encodings to the standard TEX encodings. These files are used by dvi driver software only.

They are used only by `dvi` drivers to work out what it should *really* print when you ask for a particular character. They are arcane creatures, but FONTINST deals with the details for you, and creating and using them is what this document is about, so don't worry if this doesn't make sense yet. (I don't understand the details about the innards of `vf` files, but you'll find out that that doesn't matter if you just want to use FONTINST).

Each `vf` file has a `tfm` file with the same name. To use a virtual font, you select the `tfm` file as the font to use in your document. When the `dvi` driver comes across this `tfm` file in the `dvi` file, it looks up the `vf` file and uses that to decide what to do.

3.1.7 Virtual property list files (`vpl` files)

`vpl` files are human-readable text files which contain all the font metric, kerning, mapping, and other information needed to create a `vf` and `tfm` pair.

VPTOVF will create a `vf`/`tfm` pair from a `vpl` file. VFTOVP will create a `vpl` from a `vf`/`tfm` pair. VFTOVP also needs to be able to read all the `tfm` files that are referred to by a `vf` to recreate the `vpl`– it looks at the checksums to verify that everything's okay.

3.1.8 Font definition files (`fd` files)

These are files containing commands to tell LATEX which `tfm` files to associate with a request for a font using LATEX's font selection commands.

For example, here is a small and edited part of the `fd` file supplied with PSNFSS to allow you to use the Adobe Times font in T1 encoding:

```
\ProvidesFile{t1ptm.fd}
    [1997/02/11 Fontinst v1.6 font definitions for T1/ptm.]

\DeclareFontFamily{T1}{ptm}{}

\DeclareFontShape{T1}{ptm}{m}{n} {<-> ptmr8t}{}
\DeclareFontShape{T1}{ptm}{m}{it}{<-> ptmri8t}{}
...
\DeclareFontShape{T1}{ptm}{b}{n} {<-> ptmb8t}{}
\DeclareFontShape{T1}{ptm}{b}{it}{<-> ptmbi8t}{}
...
```

What this means is: when you use LATEX to select the font family `ptm` in T1 encoding in the medium series (`m`) and normal shape (`n`), TEX uses the font `ptmr8t.tfm`. Similarly, if you select bold italic, TEX uses `ptmbi8t.tfm`.

LATEX works out which `fd` file to load based on the current encoding and font family selected. If you've selected T1 encoded `ptm` like this:

```
\fontencoding{T1}\fontfamily{ptm}\selectfont
```

LATEX loads the file `t1ptm.fd` (if it doesn't exist, you're in trouble). As you can see above, this file contains information so that LATEX knows which `tfm` file to use. So if you ask for, say, `T1/ptm/b/it` (T1 encoded Times-Roman, bold series, italic shape), you get the font `ptmbi8t`.

You can find more about `fd` files and LATEX's font selection commands at CTAN: `ftp://ftp.tex.ac.uk/tex-archive/macros/latex/base/fntguide.tex` and `ftp://ftp.tex.ac.uk/tex-archive/info/simple-nfss.tex` are both useful.

3.2 What does fontinst do?

FONTINST creates `vpl` and `pl` files from `afm` or `pl` files to map any glyph or combination of glyphs in the original font files to any slot in the output font file. There, isn't that better? Off you go now...

If you're still confused, I'll explain a few things.

Glyph This is a jargon word referring to what most people think of as a character. But is an acute accent really a character? And what about an acute accent over an 'e'? You can refer to the letter 'e' as a glyph; the acute accent '´' as a glyph; and the accented letter 'é' as yet another glyph.

Encoding A particular arrangement of glyphs in a font used by a computer. You're probably familiar with ASCII encoding, which has the letter 'A' in slot 65, 'B' in slot 66, and so on. That's it, really. TEX uses several different encodings. The most common ones are OT1 (the original TEX 7 bit encoding) and T1 (the newer TEX 8 bit encoding).

The thing is that the average PostScript font comes in Adobe standard encoding, which, for example, has the glyph dotless i 'ı' in slot 245. But TEX T1 encoding expects the glyph o dieresis 'ö' in that slot, and wants dotless i in slot 25. So if you tried to use a raw PostScript font with TEX, any time you tried to get a 'ö', you'd get a 'ı'; and every time you tried to get a 'ı', you'd get a blank, because Adobe standard encoding says that slot 25 is empty. The process of dealing with this problem is called 're-encoding', and is what FONTINST helps with.

Slot As mentioned above, this is a numbered position in an encoding. For example, slot 33 in ASCII encoding contains the glyph '!'.

This might not make much sense yet; the best thing to do is relax. There's a lot of things that need to be dealt with when you're setting up LATEX to use a new font, so you can expect to be a bit confused until you've done it a few times.

3.3 What do you do with fontinst?

If you're using FONTINST, the usual steps you need to take to use an ordinary PostScript text font with LATEX are these:

1. Give the `afm` files an approriate name.
2. Use FONTINST to produce an `8r` encoded `pl` files from these `afm` files.
3. Use FONTINST to create T1 and OT1 encoded `pl` and `vpl` files from the `8r` encoded `pl` filea (this procedure will also create suitable `fd` files).
4. Use PLTOTF to turn each `pl` file into a `tfm` file.
5. Use VPTOVF to turn each `vpl` file into a pair of `vf` and `tfm` files.
6. Move the `tfm`, `vf`, and `fd` files into the appropriate directories so LATEX can see them.
7. Tell your DVI driver about the new font (edit DVIPS's `psfonts.map` file, or OzTEX's `Default` configuration file.
8. Perhaps write a package file to make selecting the new font a little easier.
9. Test it.

4 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`.

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

- *fontinst files* contain commands to process fonts metrics so you can use a font with TEX. For example, `fontptcm.tex` is a *fontinst file*.
- *encoding files* contain information about an encoding, including the code table, ligatures, and font dimensions. For example, `8r.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 4.2.

4.1 General commands

The following *general 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*⟩`}`
`\setint{`⟨*int*⟩`}{`⟨*integer expression*⟩`}`
`\setstr{`⟨*str*⟩`}{`⟨*string*⟩`}`

If the dimension variable ⟨*dim*⟩ is currently undefined, it is defined to be the current value of ⟨*dimension*⟩.

If the integer variable ⟨*int*⟩ is currently undefined, it is defined to be the current value of ⟨*integer expression*⟩.

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*⟩`}`
`\resetint{`⟨*int*⟩`}{`⟨*integer expression*⟩`}`
`\resetstr{`⟨*str*⟩`}{`⟨*string*⟩`}`

The dimension variable ⟨*dim*⟩ is defined to be the current value of ⟨*dimension*⟩.

The integer variable ⟨*int*⟩ is defined to be the current value of ⟨*integer expression*⟩.

The string variable ⟨*str*⟩ is defined to be the current value of ⟨*string*⟩.

`\resetcommand{`⟨*command*⟩`}{`⟨*definition*⟩`}`

The command ⟨*command*⟩ is defined to be the ⟨*definition*⟩, regardless of whether it was already defined or not. This is a synonym for the TEX `\def` command.

`\ifisint{`⟨*int*⟩`}\then`
`\ifisdim{`⟨*dim*⟩`}\then`
`\ifisstr{`⟨*str*⟩`}\then`
`\ifisglyph{`⟨*glyph*⟩`}\then`
`\ifiscommand{`⟨*command*⟩`}\then`

Expands out to `\iftrue` if the integer variable ⟨*int*⟩ is defined, and `\iffalse` otherwise.

Expands out to `\iftrue` if the dimension variable ⟨*dim*⟩ is defined, and `\iffalse` otherwise.

Expands out to `\iftrue` if the string variable ⟨*str*⟩ is defined, and `\iffalse` otherwise.

Expands out to `\iftrue` if the glyph variable ⟨*glyph*⟩ is defined, and `\iffalse` otherwise.

Expands out to `\iftrue` if the command ⟨*command*⟩ is defined, and `\iffalse` otherwise.

`\unsetdim{`⟨*dim*⟩`}`
`\unsetint{`⟨*int*⟩`}`
`\unsetstr{`⟨*str*⟩`}`
`\unsetcommand{`⟨*command*⟩`}`

Makes ⟨*dim*⟩, ⟨*int*⟩, ⟨*str*⟩, or ⟨*command*⟩ an undefined dimension, integer, string or command.

4.2 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*⟩

Returns the value of a TEX ⟨*number*⟩ (as explained ib *The TEXbook*).

`\int{`⟨*int*⟩`}`

Returns the value of the integer variable ⟨*int*⟩.

`\width{`⟨*glyph*⟩`}`
`\height{`⟨*glyph*⟩`}`
`\depth{`⟨*glyph*⟩`}`
`\italic{`⟨*glyph*⟩`}`

Returns the width, height, depth, or italic correction of the glyph variable ⟨*glyph*⟩.

`\kerning{`⟨*left*⟩`}{`⟨*right*⟩`}`

Returns the kerning between the ⟨*left*⟩ and ⟨*right*⟩ glyph variables.

`\neg{`⟨*integer expression*⟩`}`
`\add{`⟨*integer expression*⟩`}{`⟨*integer expression*⟩`}`
`\sub{`⟨*integer expression*⟩`}{`⟨*integer expression*⟩`}`
`\mul{`⟨*integer expression*⟩`}{`⟨*integer expression*⟩`}`
`\div{`⟨*integer expression*⟩`}{`⟨*integer expression*⟩`}`
`\scale{`⟨*integer expression*⟩`}{`⟨*integer expression*⟩`}`

`\neg` returns the negation of the ⟨*integer expression*⟩.

`\ad` returns the sum of the two ⟨*integer expression*⟩s.

`\sub` returns the first ⟨*integer expression*⟩ minus the second.

`\mul` returns the product of the two ⟨*integer expression*⟩s.

`\div` returns the first ⟨*integer expression*⟩ divided by the second.

`\scale` returns the first ⟨*integer expression*⟩ times the second, divided by 1000.

5 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{`⟨*to*⟩`}{`⟨*from*⟩`}` `\substitutenoisy{`⟨*to*⟩`}{`⟨*from*⟩`}`

This declares a LATEX font substitution, that the series or shape ⟨*to*⟩ should be substituted if necessary by the series or shape ⟨*from*⟩. `\substitutesilent` means that when the font substitution is made, no warning will given.

`\substitutenoisy` is the same as `\substitutesilent`, but gives a warning when the substitution is made by LATEX.

For example, to say that the series `bx` can be replaced by the series `b`, you say:

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

To say that the shaoe `ui` can be replaced by the shape `i`, you say:

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

The following weight substitutions are standard:

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

The following shape substitutions are standard:

```
\substitutenoisy{ui}{it}
\substitutesilent{it}{sl}
\substitutesilent{sl}{it}
```

`\transformfont{`⟨*font-name*⟩`}{`⟨*transformed font*⟩`}`

This makes a raw transformed font, for example expanded, slanted, condensed or re-encoded. *It is the responsibility of the device driver to implement this transform.* Each `\transformfont` command writes out an `mtx` file and a raw `pl` file for ⟨*font-name*⟩.

A ⟨*transformed font*⟩ is given by the following commands:

`\fromafm{`⟨*afm*⟩`}`
`\frompl{`⟨*pl*⟩`}`
`\frommtx{`⟨*mtx*⟩`}`

This reads the metrics of a font which is about to be transformed from an external file. Both `\fromafm` and `\frompl` write out an `mtx` file corresponding to the `afm` or `pl` file. In addition, `\formafm` also writes out a raw `pl` file, containing just the glyph metrics but no kerning information.

`\scalefont{`⟨*integer expression*⟩`}{`⟨*transformed font*⟩`}`
`\xscalefont{`⟨*integer expression*⟩`}{`⟨*transformed font*⟩`}`
`\yscalefont{`⟨*integer expression*⟩`}{`⟨*transformed font*⟩`}`
`\slantfont{`⟨*integer expression*⟩`}{`⟨*transformed font*⟩`}`

This applies a geometric transformation to the font metrics of ⟨*transformed font*⟩. The scale factor or slant factor are given in 1000 units to the design size. Typical examples are 167 for slanted fonts or 850 for condensed fonts.

`\reencodefont{`⟨*etx*⟩`}{`⟨*transformed font*⟩`}`

This rearranges the encoding vector of ⟨*transformed font*⟩.

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

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

This will create `ptmr8a.mtx`, `ptmr8a.pl`, `ptmro8r.mtx` and `ptmro8r.pl`, which can then be used as raw fonts in `\installfont` commands. The same transformation can also be achieved in two steps:

```
\transformfont{ptmr8r}{\reencodefont{8r}{\fromafm{ptmr8a}}}
\transformfont{ptmro8r}{\slantfont{167}{\frommtx{ptmr8r}}}
```

This will create `ptmr8a.mtx`, `ptmr8a.pl`, `ptmr8r.mtx`, `ptmr8r.pl`, `ptmro8r.mtx` and `ptmro8r.pl`.

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{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{EXTENDED TEX FONT ENCODING - LATIN}{T1}
\declareencoding{TEX TEXT COMPANION SYMBOLS 1---TS1}{TS1}
\declareencoding{TEXBASE1ENCODING}{8r}
\declareencoding{TEX TYPEWRITER AND WINDOWS ANSI}{8y}
```

5.1 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 2_ε 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 T1-encoded Times Roman font (using `T1.etx` and `latin.mtx`), you say:

```
\installfont{ptmr8t}{ptmr8r,latin}{T1}
  {T1}{ptm}{m}{n}{}
```

To install a OT1-encoded Times Roman font with a scaled version of Symbol for the Greek letters, you say:

```
\installfont{zptmrsy}{ptmr8r,psyr scaled 1100,latin}{OT1}
  {OT1}{ptm}{m}{n}{}
```

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

`\installrawfont{`⟨*font-name*⟩`}{`⟨*file-list*⟩`}{`⟨*etx*⟩`}`
`{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}{`⟨*size*⟩`}`

This is similar to `\installfont` except that it produces a TEX raw font as `pl` file rather than a virtual font.

For example, to install an 8r-encoded Times Roman raw font (using `8r.etx` and `8r.mtx`), you say:

```
\installfont{ptmr8t}{ptmr8r,8r}{8r}
  {8r}{ptm}{m}{n}{}
```

5.2 The `\latinfamily` command

The `\latinfamily` command is essentially a short-cut to save you preparing a huge file with many different FONTINST commands in it.

It takes `afm` or `mtx` files as the source of font metric data to work with. Usually, you have a set of `afm` files. They must be named according to a subset of the Fontname naming scheme. To illustrate the process, here is an edited part of the console log from a use of `\latinfamily`:

```
\latinfamily{pad}{}
```

This log does not show FONTINST 'in action'; it's just to illustrate which fonts are looked for when you use the `\latinfamily` command.

```
INFO> to make LaTeX font shape <pad,m,n,> seek padr8r.mtx
INFO> to make LaTeX font shape <pad,m,sc,> seek padrc8r.mtx
INFO> to make LaTeX font shape <pad,m,sl,> seek padro8r.mtx
INFO> to make LaTeX font shape <pad,m,it,> seek padri8r.mtx
INFO> to make LaTeX font shape <pad,m,n,c> seek padr8rn.mtx
INFO> to make LaTeX font shape <pad,m,sc,c> seek padrc8rn.mtx
INFO> to make LaTeX font shape <pad,m,sl,c> seek padro8rn.mtx
INFO> to make LaTeX font shape <pad,m,it,c> seek padri8rn.mtx
```

The important point to notice is that FONTINST needs an `8r` encoded `mtx` file for each font when you are using the `latinfamily` command. If it can't find an `8r` encoded `mtx` file, it'll look for for an `8a` encoded `afm` file. It will automatically turn the file it finds into an `8r` encoded `mtx` file. So when FONTINST says '`seek padr8r.mtx`', it is in fact looking for `padr8r.mtx` and `padr8a.afm`. Whatever it finds, it will end up with `padr8r.mtx` to work on.

The first line of the log shows that FONTINST is trying to create a `vpl` file for `pad/m/n`. That is, font family `pad` (Adobe Garamond), font series `m` (normal 'book' or 'regular' weight), and font shape `n` (normal upright).

If it finds what it's looking for, it will create the files:

```
padr7t.vpl
padr8t.vpl
padr8c.vpl
```

And add these lines to the given `fd` files:

```
OT1pad.fd:    \DeclareFontShape{OT1}{pad}{m}{n}{<-> padr7t}{}
 T1pad.fd:    \DeclareFontShape{T1} {pad}{m}{n}{<-> padr8t}{}
TS1pad.fd:    \DeclareFontShape{TS1}{pad}{m}{n}{<-> padr8c}{}
```

This means you will have three new fonts to use in LATEX: the OT1, T1 and TS1 encoded versions of `pad/m/n`. You'll be able to select (say) `T1/pad/m/n` by saying:

```
\fontencoding{T1}\fontfamily{pad}\fontseries{m}\fontshape{n}\selectfont
```

This is the clumsiest way of selecting that particular font, but I've done it to illustrate exactly what's happening.

The next line:

```
INFO> to make LaTeX font shape <pad,m,sc,> seek padrc8r.mtx
```

shows that FONTINST is trying to install a small caps font. If you have a real small caps metric file named `padrc8r.mtx` (don't forget it'll look for an `8a` encoded `afm` file), FONTINST will go ahead and create the `vpl` file and `fd` file entry as expected.

But you don't normally have a real small caps font, so FONTINST will quite happily produce a fake small caps font. To do this, it looks for a suitable metric file by dropping the '`c`':

> 'Hmm... I can't find `padrc8r`, so I'll look for `padr8r`.'

And you will eventually have:

```
padrc7t.vpl
padrc8t.vpl
```

And add these lines to the given `fd` files:

```
OT1pad.fd:    \DeclareFontShape{OT1}{pad}{m}{sc}{<-> padrc7t}{}
 T1pad.fd:    \DeclareFontShape{T1} {pad}{m}{sc}{<-> padrc8t}{}
```

(Note that it won't install a TS1-encoded small caps font because TS1 is a text symbol font, which would look the same in the upright and small caps shape.)

The next log line shows FONTINST trying to create a `vpl` for the oblique version of Adobe Garamond:

```
INFO> to make LaTeX font shape <pad,m,sl,> seek padro8r.mtx
```

It's quite usual for an oblique version to be unavailable, but FONTINST has a way round this: it can fake an oblique font from the corresponding 'straight' version:

> 'Oh dear: I can't find `padro8r`, so I'll look for `padr8r` and use clever maths to fake a slanted version.'

This is not as straightforward as the small caps case. FONTINST only works out what the metrics ought to be if the entire font is slanted to the right. It's up to the DVI driver to actually print a slanted font. DVIPS can do this.

You will eventually have:

```
padro7t.vpl
padro8t.vpl
padro8c.vpl
```

And these lines added to the given `fd` files:

```
OT1pad.fd:    \DeclareFontShape{OT1}{pad}{m}{sl}{<-> padro7t}{}
 T1pad.fd:    \DeclareFontShape{T1} {pad}{m}{sl}{<-> padro8t}{}
TS1pad.fd:    \DeclareFontShape{TS1}{pad}{m}{sl}{<-> padro8c}{}
```

The next line is straightforward:

```
INFO> to make LaTeX font shape <pad,m,it,> seek padri8r.mtx
```

If FONTINST can't find a suitable metrics file (`padri8r.mtx` or `padri8a.afm`), it carries on without doing anything. If it does find a suitable metrics file, it churns away until you will eventually have:

```
padri7t.vpl
padri8t.vpl
padri8c.vpl
```

And these lines added to the given `fd` files:

```
OT1pad.fd:    \DeclareFontShape{OT1}{pad}{m}{it}{<-> padri7t}{}
 T1pad.fd:    \DeclareFontShape{T1} {pad}{m}{it}{<-> padri8t}{}
TS1pad.fd:    \DeclareFontShape{TS1}{pad}{m}{it}{<-> padri8c}{}
```

The next line is a bit different. FONTINST is now trying to create a `vpl` file for a *condensed* font:

```
INFO> to make LaTeX font shape <pad,m,n,c> seek padr8rn.mtx
```

If it finds a suitable metric file (Adobe Garamond, medium weight, normal upright shape, condensed), it will eventually produce:

```
padr7tn.vpl
padr8tn.vpl
padr8cn.vpl
```

And these lines added to the given `fd` files:

```
OT1pad.fd:    \DeclareFontShape{OT1}{pad}{mc}{n}{<-> padr7tn}{}
 T1pad.fd:    \DeclareFontShape{T1} {pad}{mc}{n}{<-> padr8tn}{}
TS1pad.fd:    \DeclareFontShape{TS1}{pad}{mc}{n}{<-> padr8cn}{}
```

There is no standard LaTeX command like `\bfseries` to select the medium condensed (`mc`) series created here. If you want to use this font, you must do something like:

```
\fontfamily{pad}\fontseries{mc}\selectfont
```

If it doesn't find a suitable metric file for a narrow series, FONTINST will just skip over and continue, unless you specifically tell it to fake a narrow series.

And so the process continues: FONTINST attempts to create `vpl` files for condensed versions of all the font shapes met so far, and then goes on to:

```
INFO> to make LaTeX font shape <pad,b,n,> seek padb8r.mtx
```

And again, if it finds a suitable metric file (`padb8r.mtx` or `padb8a.afm`), it'll potter off and create the files:

```
padb7t.vpl
padb8t.vpl
padb8c.vpl
```

And these lines added to the given `fd` files:

```
OT1pad.fd:    \DeclareFontShape{OT1}{pad}{b}{n}{<-> padb7t}{}
 T1pad.fd:    \DeclareFontShape{T1} {pad}{b}{n}{<-> padb8t}{}
TS1pad.fd:    \DeclareFontShape{TS1}{pad}{b}{n}{<-> padb8c}{}
```

With this step done, fontinst will try to create `vpl` files for the small caps, slanted, and italic versions of `pad/b`; and then it'll try to create condensed versions of all those:

```
INFO> to make LaTeX font shape <pad,b,n,> seek padb8r.mtx
INFO> to make LaTeX font shape <pad,b,sc,> seek padbc8r.mtx
INFO> to make LaTeX font shape <pad,b,sl,> seek padbo8r.mtx
INFO> to make LaTeX font shape <pad,b,it,> seek padbi8r.mtx
INFO> to make LaTeX font shape <pad,b,n,c> seek padb8rn.mtx
INFO> to make LaTeX font shape <pad,b,sc,c> seek padbc8rn.mtx
INFO> to make LaTeX font shape <pad,b,sl,c> seek padbo8rn.mtx
INFO> to make LaTeX font shape <pad,b,it,c> seek padbi8rn.mtx
```

If it manages to find the files it needs to create the `vpl` files to use all those fonts with LaTeX, you'll end up with the following lines in the T1 `fd` file (I've ignored the OT1 `fd` file to save some space):

```
\DeclareFontShape{T1} {pad}{b} {n} {<-> padb8t}{}
\DeclareFontShape{T1} {pad}{b} {sc}{<-> padbc8t}{}
\DeclareFontShape{T1} {pad}{b} {sl}{<-> padbo8t}{}
\DeclareFontShape{T1} {pad}{b} {it}{<-> padbi8t}{}
\DeclareFontShape{T1} {pad}{bc}{n} {<-> padb8tn}{}
\DeclareFontShape{T1} {pad}{bc}{sc}{<-> padbc8tn}{}
\DeclareFontShape{T1} {pad}{bc}{sl}{<-> padbo8tn}{}
\DeclareFontShape{T1} {pad}{bc}{it}{<-> pckbi8tn}{}
```

To translate into English: Adobe Garamond bold in 'normal', small caps, slanted, and italic versions, as well as condensed versions of all four.

Again, because there's no convenient way of selecting the condensed versions with existing LaTeX commands, you need to say something like:

```
\fontfamily{pad}\fontseries{bc}\selectfont
```

to use the bold condensed (`bc`) versions of this font; you can of course use `\itshape`, `\scshape`, `\slshape`, and `upshape` to switch between the italic, small caps, slanted, and 'normal' versions of Adobe Garamond bold condensed once you've got `pad/bc` selected.

So far, you've seen `\latinfamily` look at two different weights and two different widths. For each weight, `\latinfamily` will try and install eight different fonts as you can see above. It will try and install the same eight different fonts for each of the following different weights:

LaTeX	*Fontname*	*description*
`ul`	`a`	ultra light
`el`	`i`	extra light
`l`	`l`	light
`m`	`k`, `r`	book, regular
`mb`	`m`	medium
`db`	`d`	demi bold
`sb`	`s`	semi bold
`b`	`b`	bold
`eb`	`c`, `h`, `x`	black, heavy, extra bold
`ub`	`u`	ultra bold

The LaTeX column contains the label that will be used in the `\DeclareFontShape` command to specify the font series. The Fontname column contains the width specifier used to name the font metric file that FONTINST will look for in that case.

In other words, at some stage FONTINST will look for:

```
INFO> to make LaTeX font shape <pad,sb,n,> seek pads8r.mtx
```

and if it finds a suitable metric file (`pads8r.mtx` or `pads8a.afm`), it will create:

```
pads7t.vpl
pads8t.vpl
pads8c.vpl
```

and `fd` file entries like this:

```
OT1pad.fd:    \DeclareFontShape{OT1}{pad}{sb}{n}{<-> pads7t}{}
 T1pad.fd:    \DeclareFontShape{T1} {pad}{sb}{n}{<-> pads8t}{}
TS1pad.fd:    \DeclareFontShape{TS1}{pad}{sb}{n}{<-> pads8c}{}
```

and since `sb` is not a normal LaTeX font series, you'll need to use something like:

```
\fontfamily{pad}\fontseries{sb}\selectfont
```

to use this font.

6 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 LATEX document.

6.1 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.

`\skipslots{⟨number⟩}`

Advances the number of the next slot. New feature v1.8

```
\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`.

6.2 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.

`\int{`⟨*glyph*⟩`}`

The slot number for ⟨*glyph*⟩.

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

`codingscheme`

The font coding scheme.

6.3 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 ⟨*ligtype*⟩s 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: New feature Obsolete?!

```
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*⟩. The ⟨*varchar commands*⟩ are:

```
\vartop{⟨glyph⟩}
\varmid{⟨glyph⟩}
\varbot{⟨glyph⟩}
\varrep{⟨glyph⟩}
```

Sets the top, middle, bottom, or repeated ⟨*glyph*⟩ of the VARCHAR.

7 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*⟩.

7.1 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}{850}
\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⟩}
\setrightkerning{⟨glyph⟩}{⟨glyph⟩}{⟨integer expression⟩}
```

Sets the amount by which the first glyph should mimic how the second glyph kerns on the left or right, for example:

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

Sets the amount by which the first glyph should mimic how the second glyph kerns on the right, for example:

`\setleftrightkerning{⟨glyph⟩}{⟨glyph⟩}{⟨integer expression⟩}`

Sets the amount by which the first glyph should mimic how the second glyph kerns on both sides, for example: New feature v1.8

```
\setleftrightkerning{Asmall}{A}{850}
```

`\inputmtx{⟨file⟩}`

Inputs the ⟨*metric commands*⟩ of ⟨*file*⟩`.mtx`.

7.2 Metric variables

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

```
ascender
capheight
descender
xheight
```

The height of the tallest lower-case letter.

The height of the tallest capital letter.

The depth of the lowest lower-case letter.

The x-height of lower-case letters without ascenders.

`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.

⟨*glyph*⟩`-spacing`

The letter spacing for the glyph ⟨*glyph*⟩.

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

`designsize`

The design size of the font.

7.3 Glyph commands

The ⟨*glyph commands*⟩ are:

`\glyph{`⟨*glyph*⟩`}{`⟨*integer expression*⟩`}`

Sets the named glyph ⟨*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 ‘missingglyph’}
\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}
\endsetglyph
```

`\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⟩}
\resetheight{⟨integer expression⟩}
\resetdepth{⟨integer expression⟩}
\resetitalic{⟨integer expression⟩}
```

Sets the width, height, depth, or 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 sign can be defined to be a text summation $\sum$ scaled 120% with 0.5 pt extra height and depth using:

```
\setglyph{summationdisplay}
  \glyph{summationtext}{1200}
  \resetheight{\add{\height{summationdisplay}}{50}}
  \resetdepth{\add{\depth{summationdisplay}}{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
```

8 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!