A PostScript Font Installation Package Written in TEX

Alan Jeffrey
School of Cognitive and Computing Sciences
University of Sussex
Falmer
Brighton
BN1 9QH
UK
alanje@cogs.susx.ac.uk

Abstract

This paper describes a font installation package written entirely in TEX. It can parse Adobe Font Metric and Font Encoding files, and convert them into Property List and Virtual Property List files, for processing with `pltotf` and `vptovf`. Since it is written in TEX, it is very customizable, and can deal with arbitrary font encodings, as well as mathematics fonts.

Introduction

This paper describes `fontinst` version 0.19, a prototype font installation package for PostScript fonts (or any other fonts with font metrics given in Adobe Font Metric format). This package:

- Is written in TEX, for maximum portability (at the cost of speed).
- Supports as much of the Cork encoding as possible.
- Allows fonts to be generated in an arbitrary encoding, 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 version 2 of the New Font Selection Scheme.
- Can be customized by the user to deal with arbitrary font encodings.

The most important difference between this package and other PostScript font installation packages (such as Rockiki's (1993) `afm2tfm` and Rahtz's (1993) `psnfss`) is that it is written in TEX rather than C, and so can be more easily customized by the user to deal with non-standard encodings and mathematical fonts. At the moment, only the T1 (Cork) encoding is supported, but mathematical fonts will be added once an 8-bit font standard can be agreed upon.

Usage

There are four ways to generate fonts using the `fontinst` package:

- The simplest method to install the 'vanilla' fonts (Times, Courier and Helvetica) with the T1 (Cork) encoding is to run TEX on `fontvani.tex`.
- If you want to install other T1 fonts, you can edit `fontvani.tex` to create a TEX file which installs your fonts.
- Alternatively, you can run TEX on the file `fontinst.tex` and get an interactive prompt, which asks you for details on the fonts you want to install.
- If you want to install some fonts in a non-Cork encoding, you can create new encoding files. These consist of: a macros file, a PostScript encoding vector, and a 'fudge' file containing all the information that TEX needs that isn't contained in the `afm` file.

In each case, the `fontinst` package creates a number of files:

- *filename*.pl contains the Property List of each PostScript font. You should convert it to a TEX font metric with pltotf, and then delete the pl file.
- *filename*.vpl contains the Virtual Property List of each TEX font. You should convert it to a TEX font metric and a Virtual Font with vptovf, and then delete the vpl file.
- *filename*.fd contains the LATEX Font Definitions for each family. If you are using version 2 of the New Font Selection Scheme, you can use these to access the font family by saying \fontfamily{*family name*}.
- *filename*.atx is a temporary file containing a translation of an afm file into a syntax that can be read by TEX, and can be deleted.
- *filename*.etx is a temporary file containing a translation of a PostScript encoding vector into a syntax that can be read by TEX, and can be deleted.

Vanilla Fonts

To install the vanilla fonts, you just copy the following afm files into a directory read by TEX, and run TEX on fontvani.tex.

```
Times-Roman   Times-Italic
Times-Bold    Times-BoldItalic
Courier      Courier-Oblique
Courier-Bold   Courier-BoldOblique
Helvetica     Helvetica-Oblique
Helvetica-Bold Helvetica-BoldOblique
```

This installs the Times, Courier and Helvetica families, in bold and normal weights, with roman, italic, and small caps variants. If you would like to install other PostScript fonts, the simplest thing to do is edit fontvani.tex. For example, to generate the Palatino fonts, you can say:

```
\makevanilla{ppt}
  {Palatino}{Palatino-Italic}
  {Palatino-Oblique}{Palatino-Bold}
  {Palatino-BoldItalic}
  {Palatino-BoldOblique}
```

Prompts

When you run TEX on fontinst.tex, you will be prompted for information about the fonts you are going to install. For each font family, you can specify a number of TEX fonts, which can in turn be built from a number of PostScript fonts. For example, the Times Roman (ptm) font family consists of the fonts:

- ptmrq roman, medium weight.
- ptmriq italic, medium weight.
- ptmrcq caps and small caps, medium weight.
- ptmbq roman, bold weight.
- ptmbiq italic, bold weight.
- ptmbcq caps and small caps, bold weight.

Each of these fonts may be built from more than one PostScript font, for example ptmrq might use Times-Roman for most characters, and the Expert set for the ffi and ffl ligatures.

When you run TEX on fontinst.tex you are prompted for information on the font family you would like to install. For each family, you are prompted for:

- \FamilyName, for example, Adobe Times Roman is ptm.
- \FamilyEncoding, for example T1.

Each family can include a number of fonts, and you will be prompted for information about each of them:

- \FontName, for example, Adobe Times Roman is ptmrq.
- \FontEncoding, for example T1ulc (for T1 upper and lower case) or T1csc (for T1 caps and small caps).
- \FontWeight, for example m (medium) or b (bold).
- \FontShape, for example n (normal), sl (oblique), it (italic) or sc (caps and small caps).

Each TEX font can be built from a number of PostScript fonts. For each PostScript font you will be asked for:

- \AFMName, for example Adobe Times is Times-Roman.
- \AFMShortName, for example Adobe Times Roman is ptmr0.
- \AFMEncoding, for example adobe (for Adobe Standard Encoding) or expert (for Adobe Expert Encoding).

Using fontinst in Other Macro Packages

If you run TEX on fontinst.tex, you will be prompted interactively about the fonts you want to install. Sometimes this is not what you want, for example fontvani.tex uses the macros defined in fontinst.tex without running the prompt. This is achieved by having fontinst.tex check to see if a macro \noprompt is defined. So if you want to use fontinst.tex yourself, you should say:

```
\def\noprompt{!}
\input fontinst
```

The most useful commands given by `fontinst.tex` are:

- `\makefamily{`*commands*`}` This generates a font family named `\FamilyName` with encoding `\FamilyEncoding` using the `\maketexfont` commands.
- `\maketexfont{`*commands*`}` This generates a TEX font named `\FontName` with encoding `\FontEncoding`, weight `\FontWeight` and shape `\FontShape` using the `\makerawfont` commands.
- `\makerawfont` This generates a PostScript font named `\AFMName` with short name `\AFMShortName` and encoding `\AFMEncoding`.

For example, to generate a family consisting of just Adobe Times Roman you could say:

```
\def\FamilyName{ptm}
\def\FamilyEncoding{T1}
\makefamily{
  \def\FontName{ptmr}
  \def\FontEncoding{T1ulc}
  \def\FontWeight{m}
  \def\FontShape{n}
  \maketexfont{
    \def\AFMName{Times-Roman}
    \def\AFMShortName{rptmr}
    \def\AFMEncoding{adobe}
  }
}
```

Installing a New Encoding

The main advantage of using a font installation package written in TEX is that it is very customizable. To install a font in a new encoding, you just have to generate a new `enc` file, a new `mac` file and a new `fud` file. The `enc` file is just a PostScript encoding vector, as described in the *PostScript Language Reference Manual*. The `mac` file just defines any macros you may wish to use in the `fud` file. The most important file is the `fud` file, that contains all the font information for a TEX font that is not present in the `afm` file. This includes:

- The coding scheme name.
- The boundary character.
- The font dimensions.
- The ligatures.
- The `varchar` and `nextlarger` entries.
- How to kern glyphs such as 'ffi' which aren't given kerning information in the `afm` file.
- How to fake glyphs such as 'ffi' which aren't defined in the PostScript font.

When an `afm` file is read, the following parameters are set:

- `\afmunits` is the length of one `afm` unit. There are usually 1000 `afm` units to the em-quad.
- `\itslant` is the italic slant, measured in points. This is normally assigned to font dimension 1.
- `\xheight` is the x-height of the font, measured in `afm` units. This is usually assigned to font dimension 5.
- `\capheight` is the capital height of the font, measured in `afm` units.
- `\ascender` is the ascender height of the font, measured in `afm` units.
- `\underlinethickness` is the rule width of the font, measured in `afm` units.
- `\iffixedpitch` is true iff the font is monoweight.
- `\getchar{`*glyph*`}` globally sets the following parameters:
  - `\chardp`, `\charht`, `\charic` and `\charwd` are the dimensions of the character and its italic correction. These are given in points.
  - `\map` is a token list consisting of the `MAP` instructions used to generate the glyph. For example, to set character 123 from font 0, followed by character 45 from font 2, `\map` would be set to:

    ```
    (SETCHAR D 123)
    (SELECTFONT D 2)
    (SETCHAR D 45)
    ```

  - `\startfont` is the font number the character expects to start in, and `\stopfont` is the font number the character expects to stop in. For example, in the above case, `\startfont` would be 0 and `\stopfont` would be 2.

The commands that can be used to change the TEX font generated by `fontinst.tex` are:

- `\codingscheme{`*scheme name*`}` sets the coding scheme of the font, for example:

  ```
  \codingscheme{EXTENDED TEX FONT
    ENCODING - LATIN}
  ```

- `\boundarychar{`*glyph*`}` sets the boundary character of the font, for example:

  ```
  \boundarychar{percent}
  ```

- `\fontdimens{`*font dimension commands*`}` sets the font dimensions of the font. Within the font dimension commands, you can say `\parameter{`*number*`}{`*dimen*`}` to set each parameter. For example:

```
\fontdimens{
  \getchar{space}
  \parameter{1}{\itslant}
  \parameter{2}{\charwd}
  \parameter{3}{.5\charwd}
  \parameter{4}{.33333\charwd}
  \parameter{5}{\xheight\afmunits}
  \parameter{6}{1000\afmunits}
  \parameter{7}{.33333\charwd}
}
```

- `\ligature{`*glyph*`}{`*lig commands*`}` sets the ligatures for a glyph. Within the lig commands, you can say `\lig{`*glyph*`}{`*glyph*`}{`*type*`}`. The ligature type is given in `pl` syntax, that is one of:

```
LIG  /LIG /LIG> LIG/
LIG/> /LIG/ /LIG/> /LIG/>>
```

  For example, the ligatures for 'f' could be given:

```
\ligature{f}{
  \iffixedpitch\else
    \lig{i}{fi}{LIG}
    \lig{f}{ff}{LIG}
    \lig{l}{fl}{LIG}
  \fi
}
```

- `\lkern{`*glyph*`}{`*lkern commands*`}` sets how characters should kern on the left. Within the lkern commands, you can use `\scale{`*number*`}{`*commands*`}` to set the scale, and `\do{`*glyph*`}` to set a kern. For example, to say that 'i' and 'ij' should kern on the left like 'i' you can say:

```
\lkern{i}{
  \scale{1}{\do{i}\do{ij}}
}
```

  The `\scale` command is provided for fonts such as caps and small caps, where you may wish to scale the kerning of a character. For example, to say that 'T' should kern 85% as much as 'T' you could say:

```
\lkern{T}{
  \scale{1}{\do{T}}
  \scale{0.85}{\do{Tsmall}}
}
```

  This command is useful for glyphs like 'Á', which most PostScript fonts do not include kerning information for.

- `\rkern` is just like `\lkern` but for kerns on the right. For example, to say that 'ij' kerns on the right like 'j' you can say:

```
\rkern{j}{
  \scale{1}{\do{j}\do{ij}}
}
```

- `\lrkern` combines an `\lkern` and a `\rkern`. For example, to say that '%' should kern like a word boundary, you can say:

```
\lrkern{space}{
  \scale{1}{\do{percent}}
}
```

- `\nextlarger{`*glyph*`}{`*glyph*`}` specifies the next element in a `NEXTLARGER` list. For example to say that $\sum$ is followed by $\sum$ you can say:

```
\nextlarger{textsum}{displaysum}
```

- `\varchar{`*main*`}{`*top*`}{`*mid*`}{`*rep*`}{`*bot*`}` gives a `VARCHAR` entry for a glyph. If an entry is empty, it is omitted. For example, to say how large left brackets are built, you can say:

```
\varchar{lbracktop}{lbracktop}
  {lbrackmid}{}{lbrackbot}
```

- `\defchar{`*glyph*`}{`*commands*`}` gives the default definition of a glyph. If the glyph is not defined in the PostScript font, then this definition is used instead. The commands should define the parameters given above for `\getchar`. For example, the 'compound word mark' character is defined:

```
\defchar{compwordmark}{
  \global\charht=0pt
  \global\charwd=0pt
  \global\chardp=0pt
  \global\charic=0pt
  \global\map{}
}
```

  In giving the default character definitions, it is useful to define macros in the `mac` file. For example, `T1.mac` defines a command `\doublechar` which joins characters together. For example, `T1ulc.fud` contains:

```
\defchar{fi}{\doublechar{f}{i}{0}}
\defchar{ffi}{\doublechar{f}{fi}{0}}
```

  This says that 'fi' can be faked by putting an 'f' next to an 'i', and that an 'ffi' can be faked by putting an 'f' next to an 'fi'. Since fakes can be nested, this means that some fonts will generate 'ffi' out of an 'f', an 'f' and an 'i'.

- `\missingchar` is the character used if there is no sensible fake, for example for 'ȷ'. The default is a half-em black box '■'.

An Overview of `fontinst.tex`

The most important file in the `fontinst` package is `fontinst.tex`, which provides TEX macros for parsing `afm` and `enc` files, for faking characters, and for writing `pl` and `vpl` files. The most important commands are:

- `\makeatx{`*filename*`}` reads *filename*`.afm` and writes the same information to *filename*`.atx`, in a form which can be parsed more easily by TEX. For example, a file which begins:

  ```
  StartFontMetrics 2.0
  FontName Times-Roman
  ItalicAngle 0.0
  IsFixedPitch false
  ```

  will be converted to a file which begins:

  ```
  \fontname{Times-Roman}
  \itslant=0pt
  \fixedpitchfalse
  ```

  This macro is an interesting example of writing a parser in TEX, and contains a lot of hacking with `\catcodes`. One annonying feature is that `afm` files give italic angles in degrees, where `pl` files use gradiants. To convert from one to the other, we use Phil Taylor's (1989) excellent trignometry macros
- `\makeetx{`*filename*`}` reads *filename*`.enc` and writes the same information to *filename*`.etx`, in a form which can be parsed more easily by TEX. For example, an encoding file which begins:

  ```
  /T1Encoding [ /grave /acute ...
  ```

  will be converted to a file which begins:

  ```
  \characternumber{grave}{0}
  \characternumber{acute}{1}
  ```

  This is quite a simple parser.
- `\readafm{`*afm*`}{`*enc*`}{`*pl*`}` reads *afm*`.atx` and *enc*`.etx` (making them if necessary), and stores the results in macros, which are used by `\makepl` and `\makevpl`.
- `\makepl{`*encoding*`}{`*commands*`}` reads in the `afm` files given by the *commands* and writes a `pl` file. For example, the 'raw' Times-Roman font can be generated with:

  ```
  \makepl{adobe}{
    \readafm{Times-Roman}{adobe}{ptmr0}
  }
  ```
- `\makevpl{`*encoding*`}{`*commands*`}` reads in the `afm` files given by the *commands* and writes a `vpl` file. It also reads the file *encoding*`.fud` to find the font fudges. For example, the Times-Roman font can be generated with:

  ```
  \makepl{T1ulc}{
    \readafm{Times-Roman}{adobe}{ptmr0}
    \readafm{Times-Expert}{expert}{ptmrx}
  }
  ```

The code for these macros is fairly gory, especially the parsers, since TEX was never really intended for these tasks!

Examples

Table 1 shows the Times Roman font in T1 encoding, as produced by the `fontinst` package. Note that there are a number of missing characters:

0 ȷ Ŋ đ ŋ Đ Þ ð þ

Four of these characters (Đ, Þ, ð and þ) are available in the Times font, but are not in the default Adobe encoding. These characters can be used if you have a `dvi` to PostScript converter such as `dvips` which can re-encode fonts. Unfortunately, re-encoding the font to use the ISO Latin-1 encoding results in the loss of the characters:

fi fl ‹ › “ ” „ Ł ł Œ œ – —

This means that any encoding which we re-encode the raw PostScript fonts with is going to have to be non-standard. Sigh...

Figure 1 shows what can be achieved with TEX and PostScript.

Bugs

The `fontinst` package is currently available for β-testing, and has a number of 'features' which should be dealt with at some point...

- The documentation is very scrappy, and the code is badly commented.
- It takes seven minutes to generate a font on a Macintosh Classic.
- The interactive prompt is very unfriendly.
- The error handling is non-existent (and some of the errors are rather odd, for example a missing `enc` file will result in the complaint 'File blah.*afm* not found.'
- The accent positioning in italic fonts is pretty poor.
- Some characters, such as 'Lcaron' (Ľ) are pretty poor in monoweight fonts.

Test of ptmrq at 10pt on June 10, 1993 at 1533

	′0	′1	′2	′3	′4	′5	′6	′7	
′00x	`	´	ˆ	˜	¨	˝	°	ˇ	″0x
′01x	˘	¯	˙	¸	˛	,	‹	›	
′02x	“	”	„	«	»	–	—		″1x
′03x	■	ı	■	ff	fi	fl	ffi	ffl	
′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	[	\	]	^	_	
′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	{	|	}	~	-	
′20x	Ă	Ą	Ć	Č	Ď	Ě	Ę	Ğ	″8x
′21x	Ĺ	Ľ	Ł	Ń	Ň	■	Ő	Ŕ	
′22x	Ř	Ś	Š	Ş	Ť	Ţ	Ű	Ů	″9x
′23x	Ÿ	Ź	Ž	Ż	IJ	İ	■	§	
′24x	ă	ą	ć	č	ď	ě	ę	ğ	″Ax
′25x	Í	ľ	ł	ń	ň	■	ő	ŕ	
′26x	ř	ś	š	ş	ť	ţ	ű	ů	″Bx
′27x	ÿ	ź	ž	ż	ij	¡	¿	£	
′30x	À	Á	Â	Ã	Ä	Å	Æ	Ç	″Cx
′31x	È	É	Ê	Ë	Ì	Í	Î	Ï	
′32x	■	Ñ	Ò	Ó	Ô	Õ	Ö	Œ	″Dx
′33x	Ø	Ù	Ú	Û	Ü	Ý	■	SS	
′34x	à	á	â	ã	ä	å	æ	ç	″Ex
′35x	è	é	ê	ë	ì	í	î	ï	
′36x	■	ñ	ò	ó	ô	õ	ö	œ	″Fx
′37x	ø	ù	ú	û	ü	ý	■	ß	
	″8	″9	″A	″B	″C	″D	″E	″F	

Table 1: The Times Roman font generated by `fontinst`

In the first broadsheet section

MacKenzie of the Sun
'I will never do anything that would simply raise the circulation'

- **Home** Peter Imbert: the middle class and dual morality
  Why MacGregor's rail bill is going to hit the buffers
- **Sport** India go all-square
- **Analysis** Party labouring on
- **Commentary** English tests

In the Guardian 2 tabloid

Burchill of the Mail
'All I ever wanted from life was love and money'

- **Arts** De Niro talks tough
- **Obituary** Audrey Hepburn
- **Europe** Russians are losing the battle of Stalingrad
- **Environment** The Oxleas Nine fight for more than a wood
- **Notes & Queries,** radio and TV

40p
Friday
January 22
1993
Published in London
and Manchester

The Guardian

Businesses press for base rate cut □ Pound plunges with fall in production

Jobless total on brink of 3m

Larry Elliott, Mark Milner and Patrick Wintour

WHEN THE people? I a prog? Why the program. Who has other people will tell you to move to owner of projects and programmers will be taxed onv eniety.
Used to do the same indepeople pay forbid programmers to make a liviness that mo any support. But thosee if you ever the world in genet, and Int.
I have with Unix would answer has companies will tax he program which I am wroney. I view this that the gNU remain freed to be able to make this sort of demand tooks, or rejectually ree without servicult.
People ind the ways that programmers I talk that people will programerson must prohibited, though commonly they depair a suppor example.
And cent for trib a computer hun, this of more min per spend money materially and spir suppoorly inessmenefitting mutual, the desire that with the percent of the work for the fixed by a Unix percent of the with others.
I rule requences are delp.
Schoolse. The essential contation. We hope compared with the kind you mechanisms onto the ne bettelly only on a whole would be funded with and did not they now.

Major under fire on pit closures

Keith Harper
Labour Editor

HAPPEN INTO. I a prog? Why the program. Who has other people will tell you to move to owner of projects and programmers will be taxed onv eniety.
Used to do the same indepeople pay forbid programmers to make a liviness that mo any support. But thosee if you ever the world in genet, and Int.
I have with Unix would answer has companies will tax he program which I am wroney. I view this that the gNU remain freed to be able to make this sort of demand tooks, or rejectually ree without servicult.
People ind the ways that programmers I talk that people will programerson must prohibited, though commonly they depair a suppor example.
And cent for trib a computer hun, this of more min per spend money materially and spir suppoorly inessmenefitting mutual, the desire that with the percent of the work for the fixed by a Unix percent of the with others.
I rule requences are delp.
Schoolse. The essential contation. We hope compared with the kind you mechanisms onto the ne bettelly only on a whole would be funded with and did not they now.

This from. I a prog? Why the program. Who has other people will tell you to move to owner of projects and programmers will be taxed onv eniety.
Used to do the same indepeople pay forbid programmers to make a liviness that mo any support. But thosee if you ever the world in genet, and Int.
I have with Unix would answer has companies will tax he program which I am wroney. I view this that the gNU remain freed to be able to make this sort of demand tooks, or rejectually ree without servicult.
People ind the ways that programmers I talk that people will programerson must prohibited, though commonly they depair a suppor example.
And cent for trib a computer hun, this of more min per spend money materially and spir suppoorly inessmenefitting mutual, the desire that with the percent of the work for the fixed by a Unix percent of the with others.
I rule requences are delp.
Schoolse. The essential contation. We hope compared with the kind you mechanisms onto the ne bettelly only on a whole would be funded with and did not they now.

Figure 1: An example TEX document

- Producing oblique fonts by optical effects is not supported. (But I'm not sure this isn't a good thing!)
- Composite character instructions in the `afm` file are ignored.
- The support for math and Expert fonts is untested, and is awaiting an agreement on suitable encodings for 8-bit math and Expert fonts.
- I've made some assumptions about the format of `afm` files, for example that italic angles lie between 0 and 90°.
- The `vpl` files generated can have arbitrarily long lines in them, caused by long `\map` instructions. This may cause a problem on some systems.

This software is available by anonymous ftp from `ftp.cogs.susx.ac.uk` in `pub/tex/fontinst`. All comments are welcome!

References

Adobe Systems. *PostScript Language Reference Manual*, 2nd edition, Addison Wesley, 1990.

Rahtz, Sebastian. *Notes on setup of the New Font Selection Scheme 2 to use PostScript fonts*, distributed with the `psnfss2` package, 1993. Will be made available when version 2 of NFSS is released.

Rockiki, Tomas. *Dvips: A TEX Driver*, distributed with `dvips`, 1993. Available for anonymous ftp from `ftp.tex.ac.uk`.

Taylor, Phil. "Trigonometry.TEX" in *TEXhax*, September 1989. Included in the `fontinst` package.