ttf2tfm -- TrueType to TFM converter
ttf2pk -- TrueType to PK converter

These two auxiliary programs make TrueType fonts usable with TeX. ttf2tfm extracts the metric and kerning information of a TrueType font and converts it into metric files usable by TeX (quite similar to afm2tfm which is part of the dvips package). ttf2pk rasterizes the glyph outlines of a TrueType font into a bitmap font in PK format.

Since a TrueType font often contains more than 256 glyphs, some means are necessary to map a subset of the TrueType glyphs into a TeX font. To do this, two mapping tables are needed: the first maps from the TrueType font to a raw TeX font (this mapping table is used both by ttf2tfm and ttf2pk), and the second maps from the raw TeX font to another (virtual) TeX font providing all kerning and ligature information needed by TeX.

We sometimes refer to this first map as the `input' or `raw' map, and to the second as the `output' or `virtual' map.

This two stage mapping has the advantage that one raw font can be accessed with various TeX encodings (e.g. T1 and OT1) via the virtual font mechanism, and just one PK file is necessary.

For CJKV fonts, a different mechanism is provided (see section `Subfont definition files' below). Additionally, rotated glyphs for pseudo-vertical writing are supported -- if possible, vertical glyph presentation forms are used from the font's GSUB table.

ttf2tfm

Usage:

```
  ttf2tfm FILE[.ttf|.ttc] [OPTION]... [FILE[.tfm]]
```

Options (default values are given in brackets):

-c REAL	use REAL for height of small caps made with -V [0.8]
-e REAL	widen (extend) characters by a factor of REAL [1.0]
-E INT	select INT as the TTF encoding ID [1]
-f INT	select INT as the font index in a TTC [0]
-l	create 1st/2nd byte ligatures in subfonts
-L LIGFILE[.sfd]	create 1st/2nd byte ligatures in subfonts using LIGFILE
-n	use PS names of TrueType font
-N	use only PS names and no cmap
-O	use octal for all character codes in the vpl file
-p ENCFILE[.enc]	read ENCFILE for the TTF->raw TeX mapping
-P INT	select INT as the TTF platform ID [3]
-q	suppress informational output
-r OLDNAME NEWNAME	replace glyph name OLDNAME with NEWNAME
-R RPLFILE[.rpl]	read RPLFILE containing glyph replacement names
-s REAL	oblique (slant) characters by REAL, usually <<1 [0.0]

```
-t ENCFILE[.enc]    read ENCFILE for the encoding of the vpl file
-T ENCFILE[.enc]    equivalent to -p ENCFILE -t ENCFILE
-u                  output only characters from encodings, nothing extra
-v FILE[.vpl]       make a VPL file for conversion to VF
-V SCFILE[.vpl]     like -v, but synthesize smallcaps as lowercase
-w                  generate subfont enc. vectors containing glyph indices
-x                  rotate subfont glyphs by 90 degrees
-y REAL             move rotated glyphs down by a factor of REAL [0.25]
--help              print this message and exit
--version           print version number and exit
```

The usage is very similar to afm2tfm. Please consult the dvips info file for more details on the various parameters. Here we will concentrate on the differences between afm2tfm and ttf2tfm.

cmaps

Contrary to Type 1 PostScript fonts (but similar to the new CID-keyed PostScript fonts), most TrueType fonts have more than one native mapping table, also called `cmap', which maps the (internal) TTF glyph indices to the (external) TTF character codes. Common examples are a mapping table to Unicode encoded character positions and the standard Macintosh mapping. To specify this TrueType mapping table, use the options `-P' and `-E'. With `-P' you specify the platform ID; defined values are:

platform	platform ID (pid)
Apple Unicode	0
Macintosh	1
ISO	2
Microsoft	3

The encoding ID depends on the platform. For pid=0, we ignore the `-E' parameter (setting it to zero) since the mapping table is always Unicode version 2.0. For pid=1, the following table lists the defined values:

platform ID = 1

script	encoding ID (eid)
Roman	0
Japanese	1
Chinese	2
Korean	3
Arabic	4
Hebrew	5
Greek	6
Russian	7
Roman Symbol	8
Devanagari	9
Gurmukhi	10
Gujarati	11
Oriya	12

Bengali	13
Tamil	14
Telugu	15
Kannada	16
Malayalam	17
Sinhalese	18
Burmese	19
Khmer	20
Thai	21
Laotian	22
Georgian	23
Armenian	24
Maldivian	25
Tibetan	26
Mongolian	27
Geez	28
Slavic	29
Vietnamese	30
Sindhi	31
Uninterpreted	32

Here are the ISO encoding IDs:

platform ID = 2

encoding	encoding ID
ASCII	0
ISO 10646	1
ISO 8859-1	2

And finally, the Microsoft encoding IDs:

platform ID = 3

encoding	encoding ID
Symbol	0
Unicode 2.0	1
Shift JIS	2
GB 2312 (1980)	3
Big 5	4
KSC 5601 (Wansung)	5
KSC 5601 (Johab)	6
UCS-4	10

The program will abort if you specify an invalid platform/encoding ID pair. It will then show the possible pid/eid pairs. Please note that most fonts have at most two or three cmaps, usually corresponding to the pid/eid pairs (1,0), (3,0), or (3,1) in case of Latin based fonts. Valid Microsoft fonts should have a (3,1) mapping table, but some fonts exist (mostly Asian fonts) which have a (3,1) cmap not encoded in Unicode. The reason for this strange behavior is the fact that some old MS Windows versions will reject fonts having a non-Unicode cmap (since all non-Unicode Microsoft encoding IDs are for Asian specific MS Windows versions).

The `-P' and `-E' options to ttf2tfm must be equally specified for ttf2pk; the corresponding parameters in a map file are `Pid' and

`Eid', respectively.

The default pid/eid pair is (3,1).

If you use the `-N' switch, all cmaps are ignored, using only the PostScript names in the TrueType font. The corresponding option in a map file is `PS=Only'.

If you use the `-n' switch, the default glyph names built into ttf2tfm are replaced with the PS glyph names found in the font. In many cases this is not what you want because the glyph names in the font are often incorrect or non-standard. The corresponding option in a map file is `PS=Yes'.

input and output encodings

You must specify the encoding vectors from the TrueType font to the raw TeX font and from the raw TeX font to the virtual TeX font exactly as with afm2tfm, but you have more possibilities to address the character codes. [With `encoding vector' a mapping table with 256 entries in form of a PostScript vector is meant; see the file `T1-WGL4.enc' of this package for an example.] With afm2tfm, you must access each glyph with its Adobe glyph name, e.g. `/quotedsingle' or `/Acircumflex'. This has been extended with ttf2tfm; now you can (and sometimes must) access the code points and/or glyphs directly using the following syntax for specifying the character position in decimal, octal, or hexadecimal notation: `/.c<decimal-number>', `/.c0<octal-number>', or `/.c0x<hexadecimal-number>'. Examples: `/.c72', `/.c0646', `/.c0x48'. To access a glyph index directly, use the character `g' instead of `c' in the just introduced notation. Example: `/.g0x32'.

[Note: The `.cXXX' notation makes no sense if `-N' is used.]

Another possibility is to use the `-r old-glyphname new-glyphname' switch to rename a glyph. Example:

```
  ttf2tfm ... -r .g0xc7 dotlessi -r hungarumlaut dblacute ...
```

Nevertheless, it is not allowed to use the `.gXXX' or `.cXXX' glyph name construct for `new-glyphname'.

Alternatively, you can collect such replacement pairs in a file which should have `.rpl' as extension, using the `-R' option. The syntax is simple: Each line contains a pair `old-glyphname new-glyphname' separated by whitespace (without the quotation marks). The percent sign starts a line comment; you can continue a line with a backslash as the last character. An example for a replacement file is `VPS.rpl' (to be used in conjunction with `t5.enc' for Vietnamese) which is part of this package.

The `-r' and `-R' switches are ignored for subfonts or if no encoding tables are specified. For ttf2pk, the corresponding option to `-R' is `Replacement'. Single replacements are directly given as old_glyphname=newglyphname in a map file.

For pid/eid pairs (1,0) and (3,1), both ttf2tfm and ttf2pk recognize built-in default Adobe glyph names; the former pair follows the names given in Appendix E of the book `Inside Macintosh', volume 6, the latter uses the names given in the TrueType Specification (WGL4, a Unicode subset). Note that Adobe glyph names are not unique and do sometimes differ: E.g., many PS fonts have the glyph `mu', whereas this glyph is called `mu1' in the WGL4 character set to distinguish it from the real Greek letter mu. You can find those mapping tables in the source code file `ttfenc.c'. Be also aware that OpenType (i.e. TrueType 2.0) fonts use an updated WGL4 table; we use the data from the latest published TrueType specification (1.66).

On the other hand, the switches `-n' and `-N' make ttf2tfm read in and use the PostScript names in the TrueType font itself (stored in the font's `post' table) instead of the default Adobe glyph names.

If you don't select an input encoding, the first 256 glyphs of the TrueType font with a valid entry in the selected cmap will be mapped to the TeX raw font (without the `-q' option ttf2tfm prints this mapping table to standard output), followed by all glyphs not yet addressed in the selected cmap. However, some code points for the (1,0) pid/eid pair are omitted since they do not represent glyphs useful for TeX: 0x00 (null), 0x08 (backspace), 0x09 (horizontal tabulation), 0x0d (carriage return), and 0x1d (group separator). The `invalid character' with glyph index 0 will be omitted too.

If you select the `-N' switch, the first 256 glyphs of the TrueType font with a valid PostScript name will be used in case no input encoding is specified. Again, some glyphs are omitted: `.notdef', `.null', and `nonmarkingreturn'.

If you don't select an output encoding, ttf2tfm uses the same mapping table as afm2tfm would use (you can find it in the source code file texenc.c); it corresponds to TeX typewriter text. Unused positions (either caused by empty code points in the mapping table or missing glyphs in the TrueType font) will be filled (rather arbitrarily) with characters present in the input encoding but not specified in the output encoding (without the `-q' option ttf2tfm prints the final output encoding to standard output). Use the `-u' option if you want only glyphs in the virtual font which are defined in the output encoding file, and nothing more.

One feature missing in afm2tfm has been added which is needed by the LaTeX T1 encoding: ttf2tfm will construct the glyph `Germandbls' (by simply concatenating to `S' glyphs) even for normal fonts if possible. It appears in the glyph list (written to stdout) as the last item, marked with an asterisk. Since this isn't a real glyph it will be available only in the virtual font.

For both input and output encoding, an empty code position is represented by the glyph name `.notdef'.

In encoding files, you can use `\' as the final character of a line to indicate that the input is continued on the next line. The backslash and the following newline character will be removed.

ttf2tfm returns 0 on success and 1 on error; warning and error messages are written to standard error.

other options

You can select the font in a TrueType font collection (which usually has the extension `.ttc') with `-f'; the default value, zero, specifies the first font. For fonts not being a collection this parameter is ignored.

The option `-l' makes ttf2tfm create ligatures in subfonts between first and second bytes of all the original character codes. Example: Character code 0xABCD maps to character position 123 in subfont 45. Then a ligature in subfont 45 between position 0xAB and 0xCD pointing to character 123 will be produced. The fonts of the Korean HLaTeX package use this feature. Note that this option generates correct ligatures only for TrueType fonts where the input cmap is identical to the output encoding. In case of HLaTeX, TTFs must have platform ID 3 and encoding ID 5.

Option `-L' is the same as `-l', but character codes for ligatures are specified in a ligature file. For example, `-L KS-HLaTeX' generates correct ligatures for the Korean HLaTeX package regardless of the platform and encoding ID of the used TrueType font (the file `KS-HLaTeX.sfd' is part of the ttf2pk package). Ligature files have the same format and extension as SFD files. Both `-L'and `-l' are ignored if not in subfont mode.

PostScript encoding vectors containing glyph indices of subfonts, primarily used to embed TrueType fonts in pdfLaTeX, can be created with option `-w'. ttf2tfm takes the TFM names and replaces the suffix with `.enc'; that is, for files `foo01.tfm', `foo02.tfm', ... it creates `foo01.enc', `foo02.enc', ... at the same place.

To produce glyphs rotated by 90 degrees counter-clockwise, use `-x'. If the font contains a GSUB table (with feature `vert') to specify vertical glyph presentation forms, both ttf2pk and ttf2tfm will use it. This will work only in subfont mode. The y-offset of rotated glyphs can be specified with the `-y' option; its parameter gives the fractional amount of shifting downwards (the unit is one EM). If not specified, a value of 0.25 (em) is used.

ttf2pk

Usage:

```
  ttf2pk [-q] [-n] FONT DPI
  ttf2pk -t [-q] FONT
```

Options:

```
-q               suppresses informational output
```

```
-n              only use `.pk' as extension
-t              test for FONT (returns 0 on success)
--help          print this message and exit
--version       print version number and exit
```

The FONT parameter must correspond to an entry in a map file recorded in the configuration file ttf2pk.cfg (see below for details), otherwise error code 2 is returned -- this can be used for scripts like mktexpk to test whether the given font name is a (registered) TrueType font.

Another possibility is to use the `-t' switch which will print the line of a map file corresponding to FONT and return 0 on success (`-q' suppresses any output).

DPI specifies the intended resolution (we always assume a design size of 10pt).

ttf2pk.cfg

ttf2pk uses a small configuration file called ttf2pk.cfg; in each line it contains a keyword with its value, separated by whitespace. Comment lines can start with any of the following characters: `*', `#', `;', and `%'. Leading whitespace is ignored.

Currently, only one keyword, `map', is recognized in this file; it takes a map file name as a parameter. If no extension is given to the map file name, `.map' is appended. No whitespace is allowed in the map file name. The `map' keyword can be given more than once to specify multiple map files; if the map file name is prepended by a plus sign, it is added to the list of map files to be used. Example:

```
  map  foo
  map +bar
```

This makes ttf2pk to first read `foo.map', then `bar.map'.

If the configuration file is not found, ttf2pk tries to use `ttfonts.map' instead.

map files

Parameters specified to ttf2tfm are preserved for ttf2pk in map files -- ttf2tfm writes out to standard output, as the last line, a proper entry for a map file.

As an example, a call to

```
  ttf2tfm arial -s 0.25 -P 1 -E 0 -r .g0xc7 caron \
                -p 8r.enc -t T1-WGL4.enc -v arialsx arials
```

will produce the following line:

```
  arials   arial Slant=0.25 Encoding=8r.enc Pid=1 Eid=0 .g0xc7=caron
```

The output encoding given with `-t' for the virtual font `arialsx' is immaterial to ttf2pk (nevertheless, input encoding files must have the same format as with ttf2tfm, and all said above about encoding files holds).

Here a table listing the various ttf2tfm parameters and its corresponding entries in a map file:

-s	Slant
-e	Extend
-p	Encoding
-f	Fontindex
-P	Pid
-E	Eid
-n	PS=Yes
-N	PS=Only
-R	Replacement
-x	Rotate=Yes
-y	Y-Offset

Single replacement glyph names given to ttf2tfm with the `-r' switch are directly specified with old-glyphname=new-glyphname. For subfonts or if no encoding file is given, replacement glyphs are ignored.

One additional parameter in a map file is unique to ttf2pk: `Hinting', which can take the values `On' or `Off'. Some fonts (e.g. the CJKV part of cyberbit.ttf) are rendered incorrectly if hinting is activated. Default is `On' (you can also use `Yes', `No', `1', and `0').

The format of map files is simple. Each line defines a font; first comes the TeX font name, then its TrueType font file name, followed by the parameters in any order. Case is significant (even for parameter names); the parameters are separated from its values by an equal sign, with possible whitespace surrounding it. ttf2pk reads in a map file line by line, continuing until the TeX font specified on the command line is found, otherwise the programs exits with error code 2. Thus you can use any character invalid in a TeX font name to start a comment line.

In both map files and encoding files, use `\' as the final character of a line to indicate that the input is continued on the next line. The backslash and the following newline character will be removed.

ttf2pk will abort if it can't find and read the TeX font metrics file of the given TeX font name.

Subfont definition files

CJKV (Chinese/Japanese/Korean/old Vietnamese) fonts usually contain several thousand glyphs; to use them with TeX it is necessary to split such large fonts into subfonts. Subfont definition files (usually having the extension `.sfd') are a simple means to do this smoothly. A subfont file name usually consists of a prefix, a subfont infix, and a postfix (which is empty in most cases), e.g.

```
    ntukai23 -> prefix: ntukai, infix: 23, postfix: (empty)
```

Here the syntax of a line in an SFD file, describing one subfont:

```
  <whitespace> <infix> <whitespace> <ranges> <whitespace> `\n'

  <infix> := anything except whitespace. It's best to use only
             alphanumerical characters.
  <whitespace> := space, formfeed, carriage return, horizontal and
                  vertical tabs -- no newline characters.
  <ranges> := <ranges> <whitespace> <codepoint> |
              <ranges> <whitespace> <range> |
              <ranges> <whitespace> <offset> <whitespace> <range>

  <codepoint> := <number>
  <range> := <number> `_' <number>
  <offset> := <number> `:'

  <number> := hexadecimal (prefix `0x'), decimal, or octal
              (prefix `0')
```

A line can be continued on the next line with a backslash ending the line. The ranges must not overlap; offsets have to be in the range 0-255.

Example:

The line

```
    03   10: 0x2349 0x2345_0x2347
```

assigns to the code positions 10, 11, 12, and 13 of the subfont having the infix `03' the character codes 0x2349, 0x2345, 0x2346, and 0x2347, respectively.

The SFD files in the distribution are customized for the CJK package for LaTeX.

You have to embed the SFD file into the TFM font name (at the place where the infix will appear) surrounded by two `@' signs, on the command line resp. a map file; both ttf2tfm and ttf2pk switch then to subfont mode.

It is possible to use more than a single SFD file by separating them with commata and no whitespace; for a given subfont, the first file is scanned for an entry, then the next file, and so on. Later entries override entries found earlier (possibly only partially). For example, the first SFD file sets up range 0x10-0xA0, and the next one modifies entries 0x12 and 0x25. As can be easily seen, this algorithm allows for adding and replacing, but not for removing entries.

Subfont mode disables the options `-n', `-N', `-p', `-r', `-R', `-t', `-T', `-u', `-v', `-w', and `-V' for ttf2tfm; similarly, no `Encoding' and `Replacement' parameter resp. single replacement glyph names are allowed in a map file.

ttf2tfm will create ALL subfont TFM files specified in the SFD files (provided the subfont contains glyphs) in one run.

Example:

The call

```
ttf2tfm ntukai.ttf ntukai@Big5,Big5-supp@
```

will use `Big5.sfd' and `Big5-supp.sfd', producing the subfont files ntukai01.tfm, ntukai02.tfm etc.

ttf2pk should be then called on the subfonts directly:

```
ttf2pk ntukai01 600
ttf2pk ntukai02 600
...
```

Some notes on file searching

Both ttf2pk and ttf2tfm use either the kpathsea, emtexdir, or MiKTeX library for searching files (emtexdir will work only on operating systems which have an MS-DOSish background, i.e. MS-DOS, OS/2, Windows; MiKTeX is specific to MS Windows).

During compilation, you have to define HAVE_KPATHSEA, HAVE_EMTEXDIR, or MIKTEX to activate the specific file search code.

As a last resort, both programs can be compiled without a search library; the searched files must be then in the current directory or specified with a path. Default extensions will be appended also (with the exception that only `.ttf' is appended and not `.ttc').

kpathsea

Please note that older versions of kpathsea (<3.2) have no special means to search for TrueType fonts and related files, thus we use the paths for PostScript related stuff. The actual version of kpathsea is displayed on screen if you call either ttf2pk or ttf2tfm with the `--version' command line switch.

Here is a table of the file type and the corresponding kpathsea variables. TTF2PKINPUTS and TTF2TFMINPUTS are program specific environment variables introduced in kpathsea version 3.2:

.ttf and .ttc	TTFONTS
ttf2pk.cfg	TTF2PKINPUTS
.map	TTF2PKINPUTS
.enc	TTF2PKINPUTS, TTF2TFMINPUTS
.rpl	TTF2PKINPUTS, TTF2TFMINPUTS

.tfm	TFMFONTS
.sfd	TTF2PKINPUTS, TTF2TFMINPUTS

And here the same for pre-3.2-versions of kpathsea:

.ttf and .ttc	T1FONTS
ttf2pk.cfg	TEXCONFIG
.map	TEXCONFIG
.enc	TEXPSHEADERS
.rpl	TEXPSHEADERS
.tfm	TFMFONTS
.sfd	TEXPSHEADERS

Finally, the same for pre-3.0-versions:

.ttf and .ttc	DVIPSHEADERS
ttf2pk.cfg	TEXCONFIG
.map	TEXCONFIG
.enc	DVIPSHEADERS
.rpl	DVIPSHEADERS
.tfm	TFMFONTS
.sfd	DVIPSHEADERS

Please consult the info files for kpathsea for details on these variables. The decision whether to use the old or the new scheme will be done during compilation.

You should set the TEXMFCNF variable to the directory where your texmf.cnf configuration file resides.

The default TDS location for the files in the data subdirectory is

```
  $TEXMF/ttf2tfm
```

(or $TEXMF/ttf2pk; you should either make a symbolic link

```
  % ln -s $TEXMF/ttf2tfm $TEXMF/ttf2pk
```

or set the variable TTF2PKINPUTS to $TEXMF/ttf2tfm for newer kpathsea versions)

Here is the proper command to find out to which value a kpathsea variable is set (we use `TTFONTS' as an example). This is especially useful if a variable isn't set in texmf.cnf or in the environment, thus pointing to the default value which is hard-coded into the kpathsea library.

```
  % kpsewhich --progname=ttf2tfm --expand-var='$TTFONTS'
```

We select the program name also since it is possible to specify variables which are searched only for a certain program -- in our example it would be `TTFONTS.ttf2tfm'.

A similar but not identical method is to say

```
  % kpsewhich --progname=ttf2tfm --show-path='truetype fonts'
```

[A full list of format types can be obtained by saying `kpsewhich --help' on the command line prompt.] This is exactly how ttf2tfm (and ttf2pk) searches for files; the disadvantage is that all variables are expanded which can cause a very long string.

emtexdir

Here the list of suffixes and its related environment variables to be set in autoexec.bat (resp. in config.sys for OS/2):

.ttf and .ttc	TTFONTS
ttf2pk.cfg	TTFCFG
.map	TTFCFG
.enc	TTFCFG
.rpl	TTFCFG
.tfm	TEXTFM
.sfd	TTFCFG

With other words, all files in the `data' subdirectory should be moved to a place in your emtex tree with TTFCFG pointing to this directory.

If one of the variables isn't set, a warning message is emitted. The current directory will always be searched. As usual, one exclamation mark appended to a directory path causes subdirectories one level deep to be searched, two exclamation marks causes all subdirectories to be searched. Example:

```
  TTFONTS=c:\fonts\truetype!!;d:\myfonts\truetype!
```

Constructions like `c:\fonts!!\truetype' aren't possible.

MiKTeX

Both ttf2tfm and ttf2pk have been fully integrated into MiKTeX. Please refer to the documentation of MiKTeX for more details on file searching.

A full example

Here an example how to handle the font `verdana.ttf' and its variants.

1. Construct the font name

[This is the most complicated part -- in case you are too lazy to construct font names compliant to TeX's `fontname' scheme, just use your own names.]

Using the `ftdump' utility (which is part of FreeType 1) you can find out the PostScript name of the specific TTF which is probably the best choice to adapt TrueType fonts to the PostScript-oriented `fontname' scheme.

In our example, the PostScript name is `Verdana'.

`fontname' uses the scheme

```
  S TT W [V...] [N] [E] [DD]
```

as documented in `fontname.texi' resp. `fontname.dvi'. Now you have to check the various mapping files:

```
   S: supplier.map: `j'  for `Microsoft'
  TT: typeface.map: `vn' for `Verdana'
   W: weight.map:   `r'  for `Regular Roman',
                    `b'  for `bold'
   V,
   N: variant.map:  `8r' for the raw base font
                    `8t' for the virtual font
                         (i.e., LaTeX's T1 encoding)
                    [additionally an inserted `c' for small caps,
                     `o' for slanted (`oblique'), or `i' for italic
                     fonts]
```

Here the standard combinations:

```
  `jvnr8r' for the default base font.
    `jvnr8t'  for the virtual default font.
    `jvnrc8t' for  the virtual  font with small  caps.  [As  you can
              see, no additional raw font is needed.]
  `jvnro8r' for the slanted base font.
    `jvnro8t' for the virtual slanted font.
```

The corresponding variants are:

```
  bold:            verdanab.ttf -> jvnb{8r,8t}
    small caps:                    jvnbc8t
    slanted:                       jvnbo{8r,8t}
  italic:          verdanai.ttf -> jvni{8r,8t}
  bold and italic: verdanaz.ttf -> jvnbi{8r,8t}
```

```
* NOTE: Be careful to use different names for the virtual font and the
*       raw font!  In the above example, `*8r' and `*8t' is used.  You
*       could also  use the postfix  `*-raw', to name an  example, for
*       the raw font if you don't follow the `fontname' naming scheme.
```

2. Font definition files

The FD file should be called `t1jvn.fd' (as you can see, this is T1 encoding). It is very similar to `t1ptm.fd', part of the PSNFSS package (which can be found in almost all TeX distributions). A `verdana.sty' file can also be modeled after `times.sty'.

```
3. Calling ttf2tfm
------------------

  To make the  example simpler, we use `T1-WGL4.enc'  for both the raw
  and  the  virtual encoding.   This  should  be  sufficient for  most
  TrueType  fonts mapped to  T1 encoding.   Other packages  may define
  other encodings  (e.g. the `t2' package available  from CTAN defines
  mapping files for Cyrillic encodings) -- it may also be necessary to
  use the `-n' or `-N' switch together with replacement glyph names to
  access all glyph names in the TrueType font.

  To create `jvnr8r' and `jvnr8t', just call

    ttf2tfm verdana -T T1-WGL4 -v jvnr8t jvnr8r
    vptovf jvnr8t

  For `jvnrc8t', do

    ttf2tfm verdana -T T1-WGL4 -V jvnrc8t jvnr8r
    vptovf jvnrc8t

  Note  that almost  always some  warnings will  appear  about missing
  glyphs.

  The last line written to stdout by ttf2tfm is a suitable entry for a
  map file -- let's call  it `verdana.map'.  Since ttf2pk doesn't care
  about virtual fonts, both calls below produce the same.

  Now  just  repeat this  procedure.   For  slanted  fonts you  should
  additionally use the switch `-s 0.176' (of course you can change the
  slanting amount to make it fit your needs).


4. Modifying ttf2pk.cfg
-----------------------

  The  final step is  to add  `verdana.map' to  ttf2pk's configuration
  file.  Simply insert the following line at the end of `ttf2pk.cfg':

    map +verdana


Problems
========

Most vptovf  implementations allow only  100 bytes for the  TFM header
(the limit is  1024 in the TFM file itself): 8  bytes for checksum and
design size, 40 bytes for the  family name, 20 bytes for the encoding,
and 4  bytes for  a face byte.   There remain  only 28 bytes  for some
additional information which is  used by ttf2tfm for an identification
string (which  is essentially  a copy of  the command line),  and this
limit is always exceeded.

The optimal solution is to increase the value of `max_header_bytes' in
the file vptovf.w  (and probably pltotf.w) to, say,  400 and recompile
vptovf  (and pltotf).  Otherwise  you will  get some  (harmless) error
```

messages like

```
  This HEADER index is too big for my present table size
```

which can be safely ignored.

--- end of ttf2pk.doc ---