The `fontinst` utility

Alan Jeffrey, Sebastian Rahtz, Ulrik Vieth

July 7, 1998

Contents

About the `fontinst` package

1 About this version of `fontinst`

This document describes version 1.8 of the `fontinst` utility, which was converted to `docstrip` format by Ulrik Vieth in June 1998. It is based on Sebastian Rahtz's "unofficial" `fontinst` releases v 1.6 and 1.7 as of February 1997 and May 1998, which, in turn, are based on Alan Jeffrey's versions v 1.500 and 1.504 of September 1995 and January 1996. In addition, this version also incorporates some code from Alan Jeffrey's last beta version, v 1.511, which dates back to June 1996, but was never released to the general public for some reason.

It should be noted that all these versions differ significantly from the last "official" version v 1.335 as of September 1994, which is still to be found in the `fontinst` directory on CTAN, but should be considered obsolete.

It is the intention of the `fontinst` hacker community, as represented on the `fontinst` mailing list, to adopt this version, v 1.8, as the new officially endorsed version, on which all further additions and enhancements can be based.

2 About previous `fontinst` releases

v 0.01 was begun by Alan Jeffrey in February 1993. This was the very first version recorded in the `CHANGES` file.

v 0.19 was completed in April 1993 and presented at the TUG '93 conference in July 1993. It is described in the proceedings in *TUGboat* 14#3 and represents one of the earliest release versions, in which the user interface was still subject to change.

v 1.0 was begun in August 1993 right after the conference and represents the beginning of a complete re-write from scratch.

v 1.333 was presented one year later at the TUG '94 conference and is described in the proceedings in *TUGboat* 15#3. It uses a different user interface, which is largely the same as in the present version, although the internals of the implementation have subsequently changed quite a bit.

v 1.335 was released in September 1994 and was the last "official" release version by Alan Jeffrey still to be found on CTAN. It is considered obsolete and should not be used any longer.

v 1.400 was prepared in November 1994 and was the first version which used `8r`-encoded raw fonts as a basis for virtual fonts.

v 1.500 appeared on CTAN in `psfonts/tools` in September 1995 together with Sebastian Rahtz's Unix shell scripts to automate the installation. This version included a completely revised implementation of `\latinfamily` and added the `\installrawfont` command to install ligfull raw fonts.

v 1.504 appeared on CTAN in `psfonts/tools` in January 1996. It added code to install (real or faked) small caps fonts and to fake narrow fonts, if they don't exists.

v 1.511 was one of the last beta versions by Alan Jeffrey which dates back to June 1996, but for some reason was never released. It uses the encodings `9o` and `9d` instead of `9t` and `9e` to install expertized oldstyle fonts, but this change never found its way into subsequent "unofficial" releases until v 1.8.

v 1.6 appeared on CTAN in `psfonts/tools` in February 1997 and was the first "unofficial" version of `fontinst` maintained by Sebastian Rahtz. It contains only minor changes compared to v 1.504, such as switching to lowercase file names, searching for `.pl` files before `.afm` files, and adding the `textcomp` installation to the distribution.

v 1.6 was re-issued on CTAN in September 1997, when the Unix shell scripts were replaced by Perl scripts. The `fontinst` package itself didn't change apart from minor changes of the debugging messages.

v 1.7 is the latest "unofficial version", which was released on CTAN in May 1998. It includes some changes to the mapping between `fontname` and LaTeX weight codes proposed by Ulrik Vieth, as well as some code to support the use of `8r`-encoded `.afm` files generated by `ttf2afm`.

3 About this fontinst release

- `fontinst.dtx` (the file you are reading) is used to generate the files `fontinst.sty`, `fontinst.ini` and `fontdoc.sty` from a single documented source.
- The functionality of `\textcompfamily` originally defined in `textcomp.tex` is now fully integrated into `\latinfamily`, which means that `\latinfamily` now installs the OT1, T1 and TS1 encodings by default.
- `\latinfamily{<FAM>j}{}` installs expertized fonts with oldstyle digits using the `9d` and `9o` encodings instead of `9e` and `9t`. The `\parse_family` macro now sets the switch `\if_oldstyle` when installing an expertized oldstyle family.
- `\latinfamily{<FAM>9}{}` (retained for backwards compatibility) is treated exactly as `\latinfamily{<FAM>j}{}`.
- A lot of debugging info was added to `\latinfamily` to make it easier to understand what is really going on.
- Instead of playing with `\setcommand\digit` trickery, oldstyle encodings are now installed by redefining `\encoding_shape`, so that different encoding files `OT1j.etx` or `T1j.etx` are used instead.
- Nearly all `.etx` and `.mtx` files were touched up, so that they can be typeset and printed properly. In particular the documentation of slots in `t1.etx`, `ts1.etx`, `8r.etx` and `8y.etx` was polsihed.
- `8y.etx` and `8y.mtx` were added and may be used as an optional alternative to `8r.etx` and `8r.mtx`, if `\raw_encoding` is redefined in `fontinst.rc`.

- `ts1.etx` and `\textcomp.mtx` were updated to reflect the latest TC fonts as of EC fonts 1.0. A number of slots were added and several glyphs are faked whenever this is easily possible.
- `\setleftrightkerning` is now defined in `fontinst.sty` instead of repeating the definition in several `.mtx` files.
- `fontdoc.sty` now support an automatic documentation feature for `.etx` files using default slot comments.

4 Notes on using `fontinst`

The primary purpose of `fontinst` is to simplify the installation of PostScript or TrueType text fonts. Leaving aside unusual variants which require special attention such as alternate or swash fonts, almost all standard font families can be installed automatically using the `\latinfamily` command, optionally making use of the corresponding expert fonts as well.

Depending on what kind of fonts you have and want to install, `\latinfamily` supports three different modes of operation:

`\latinfamily{<fam>}{}` installs a normal font family using `8a`-encoded standard fonts (reencoded to `8r`) and nothing else. It installs `.fd` files for the LaTeX families `8r<fam>`, `OT1<fam>`, `T1<fam>` and `TS1<fam>`, and generates virtual fonts for the `7t`, `8t` and `8c` encodings. This is the only option available for most typefaces which do not have an expert set.

`\latinfamily{<fam>x}{}` installs an expertized font family using `8a`-encoded standard fonts (reencoded to `8r`) and `8x`-encoded expert fonts. It installs `.fd` files for the LaTeX families, `OT1<fam>x`, `T1<fam>x` and `TS1<fam>x`, and generates virtual fonts for the `9t`, `9e` and `9c` encodings.

`\latinfamily{<fam>j}{}` installs an expertized font family with oldstyle digits using `8a`-encoded standard fonts (reencoded to `8r`) and `8x`-encoded expert fonts. It installs `.fd` files for the LaTeX families, `OT1<fam>j`, `T1<fam>j` and `TS1<fam>j`, and generates virtual fonts for the `9o`, `9d` and `9c` encodings. Since TS1 has oldstyle digits by default, the `9c`-encoded fonts should be the same as in the previous case. Finally, `\latinfamily{<fam>9}{}` is also supported as an alternative to `\latinfamily{<fam>j}{}` for backwards compatibility.

The whole installation process relies on certain assumptions about the symbol complement of Adobe's expert fonts. In particular, it is assumed that the expert fonts include the oldstyle digits and a complete set of small caps glyphs, which is an assumption that's not always statisfied for expert fonts by other suppliers. If these glyphs are not included in the expert fonts, the only way to get them is from real small caps fonts, but this requires some reshuffling of glyph names.

To support such unusual cases, this source file contains some optional code embedded between `<*oldstyle>` ... `</oldstyle>`, which extends the behaviour of `\latinfamily` for expertized encodings with oldstyle digits. Instead of relying only on the glyphs of the `8r`-reencoded raw font and the `8x`-encoded expert font, this version also looks for corresponding OsF or SC&OsF fonts and uses the default digits from those fonts as oldstyle digits.

5 Notes on small caps and olstyle fonts

The `\latinfamily` command is supposed to do a reasonably good job of installing a complete font family based on all the `.afm` files it can find. If it doesn't find a suitable font shape, it is sometimes possible to fake it by default substitutions. However, in the case of small caps fonts, there are several options which may require some clarification.

For the majority of typefaces, a font family typically consists only of of roman and italic fonts in several weights. Since real small caps fonts are not included, they have to be faked from the roman fonts, which is implemented by setting `\encoding_shape` to '`c`', so that different encoding files `OT1c.etx` or `T1c.etx` are used. Since these files call for glyph names such as '`Asmall`' which are not found in the roman font, the default substitutions in `latin.mtx` are eventaully used to approximate fake small caps glyphs by scaling and letterspacing. The outcome is just an approximation for a small caps font, but it is better than nothing.

For a small number of typefaces, the standard fonts are complemented by an expert collection, which ususally includes two sets of fonts. First, for each standard font there is a corresponding expert font containing additional glyphs such as extra ligatures and symbols, oldstyle digits and small capital letters. For Adobe expert fonts this set is sufficient to build a complete small caps font from the standard and expert glyphs. Furthermore, the expert collection usually also contains a number of real small caps font corresponding to the roman fonts and some OsF fonts corresponding to the italic fonts. If these fonts are available, there are several options how to install small caps fonts.

By default, `\latinfamily` first tries to find a real small caps font. If it is found, it is installed using the default encoding files and metric files, just like any roman or slanted font. However, once `\latinfamily` has taken this choice, it will fail to find a corresponding expert font, since it is actually looking for an expert font in small caps shape which doesn't exist. (In fact, it would be an error to substitute an expert font in normal shape.) The outcome will be a virtual font based only on glyphs from the real small caps raw font, which implies ending up with oldstyle digits as the default set of digits, but allows to inherit the kerning information of the real small caps font.

Another option for an expertized installation would be to make the real small caps fonts unavailable, so that `\latinfamily` will attempt to fake a small caps font using glyphs from the standard and expert fonts in normal shape. This means that `\encoding_shape` is again set to '`c`', so that `OT1c.etx` and `T1c.etx` are used, but this time a glyph named '`Asmall`' does exist in the expert font and will be used instead of faked one generated by scaling. The outcome will be a font based on normal and small caps glyphs from the standard and expert fonts. The oldstyle digits will only be used if they are called for, otherwise the default digits from the roman font are used. The only drawback of this approach is that the kerning around small capital letters will be based on the scaled kern amounts of the capital letters rather than on the kern pairs from the real small caps font.

Finally, the most promising approach of all these options would be to combine the glyphs from standard and expert fonts with kern pairs from the real small caps fonts. Although it is possible to achieve this with extensive reshuffling of glyph names and using a modified version of `latin.mtx`, it would be far too complicated to integrate into the `\latinfamily`, so it is not implemented in this version, although it would be a worthwile approach.

The `fontinst` package

6 Preliminaries

6.1 Initialization

Some default catcode assignments, in case we are using iniTeX.

```
⟨*pkg⟩
\catcode`\{=1
\catcode`\}=2
\catcode`\#=6
\catcode`\^=7
⟨/pkg⟩
```

`\fontinstversion` If we are running under iniTeX we cannot put the identification stuff any earlier than this. Note that `\fontinstversion` is not just used for indentification but also in `\needsfontinstversion`.

```
⟨*pkg | doc⟩
\def\fontinstversion{1.801}
\def\filedate{1998/07/07}
\def\docdate{1998/07/07}
⟨/pkg | doc⟩
```

If we are running under iniTeX or `plain`, we have to get around the LaTeX-specific `\ProvidesPackage` stuff.

```
⟨*pkg | doc⟩
\ifx\ProvidesPackage\undefined
   \def\NeedsTeXFormat#1{}
   \def\ProvidesPackage#1[#2]{}
\fi
⟨/pkg | doc⟩
```

Now we can identify ourselves as usual.

```
⟨doc⟩\ProvidesPackage{fontdoc}
⟨pkg⟩\ProvidesPackage{fontinst}
⟨pkg | doc⟩[\filedate\space v\fontinstversion\space
⟨doc⟩  fontinst documentation package]
⟨pkg⟩  fontinst installation package]
```

Save the current values of the catcodes: `@` and `_` are made letters, `~` is made a space, and space and newline are ignored.

```
⟨*pkg⟩
\edef\spacecatcode{\the\catcode`\ }
\edef\nlcatcode{\the\catcode`\^^M}
\edef\atcatcode{\the\catcode`\@}
\edef\underscorecatcode{\the\catcode`\_}
\edef\tildecatcode{\the\catcode`\~}

\catcode`\ =9
\catcode`\^^M=9
\catcode`\@=11
\catcode`\_=11
\catcode`\~=10
```

6.2 Plain TeX macros from `fontinst.ini`

If we're running in iniTeX we input some definitions taken from `plain`.

```
\ifx\@ne\undefined_command
   \input fontinst.ini\relax
\fi
⟨/pkg⟩
⟨*ini⟩
\chardef\active=13

\chardef\@ne=1
\chardef\tw@=2
\chardef\thr@@=3
\chardef\sixt@@n=16
\chardef\@cclv=255
\mathchardef\@cclvi=256
\mathchardef\@m=1000
\mathchardef\@M=10000
\mathchardef\@MM=20000

\count10=22 % allocates \count registers 23, 24, ...
\count11=9 % allocates \dimen registers 10, 11, ...
\count15=9 % allocates \toks registers 10, 11, ...
\count16=-1 % allocates input streams 0, 1, ...
\count17=-1 % allocates output streams 0, 1, ...
\count20=255 % allocates insertions 254, 253, ...
\countdef\insc@unt=20 % the insertion counter
\countdef\allocationnumber=21 % the most recent allocation
\countdef\m@ne=22 \m@ne=-1 % a handy constant
\def\wlog{\immediate\write\m@ne} % write on log file (only)

\countdef\count@=255
\dimendef\dimen@=0

\outer\def\newcount{\alloc@0\count\countdef\insc@unt}
\outer\def\newdimen{\alloc@1\dimen\dimendef\insc@unt}
\outer\def\newtoks{\alloc@5\toks\toksdef\@cclvi}
\outer\def\newread{\alloc@6\read\chardef\sixt@@n}
\outer\def\newwrite{\alloc@7\write\chardef\sixt@@n}

\def\alloc@#1#2#3#4#5{\global\advance\count1#1by\@ne
  \ch@ck#1#4#2% make sure there’s still room
  \allocationnumber=\count1#1%
  \global#3#5=\allocationnumber
  \wlog{\string#5=\string#2\the\allocationnumber}}
\outer\def\newinsert#1{\global\advance\insc@unt by\m@ne
  \ch@ck0\insc@unt\count
  \ch@ck1\insc@unt\dimen
  \ch@ck2\insc@unt\skip
  \ch@ck4\insc@unt\box
  \allocationnumber=\insc@unt
  \global\chardef#1=\allocationnumber
  \wlog{\string#1=\string\insert\the\allocationnumber}}
\def\ch@ck#1#2#3{\ifnum\count1#1<#2%
```

```
  \else\errmessage{No~room~for~a~new~#3}\fi}

\outer\def\newif#1{\count@\escapechar \escapechar\m@ne
  \expandafter\expandafter\expandafter
   \edef\@if#1{true}{\let\noexpand#1=\noexpand\iftrue}%
  \expandafter\expandafter\expandafter
   \edef\@if#1{false}{\let\noexpand#1=\noexpand\iffalse}%
  \@if#1{false}\escapechar\count@} % the condition starts out false
\def\@if#1#2{\csname\expandafter\if@\string#1#2\endcsname}
{\uccode`1=`i \uccode`2=`f \uppercase{\gdef\if@12{}}} % `if' is required

\newdimen\p@ \p@=1pt % this saves macro space and time
\newdimen\z@ \z@=0pt % can be used both for 0pt and 0

\def\space{~}
\let\bgroup={
\let\egroup=}

\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body \let\next\iterate \else\let\next\relax\fi \next}
\let\repeat=\fi % this makes \loop...\if...\repeat skippable

\def\supereject{\par\penalty-\@MM}
\outer\def\bye{\par\vfill\supereject\end}
\everyjob{%
\csname~if_file_exists\endcsname{fontinst.rc}\then
   \primitiveinput~fontinst.rc
\else
   \immediate\write16{No~file~fontinst.rc.}
\fi
}
⟨/ini⟩
```

6.3 Declaring variables and constants

Some temporary variables:

```
⟨*pkg⟩
\newcount\a_count
\newcount\b_count
\newcount\c_count
\newcount\d_count
\newcount\e_count
\newcount\f_count
\newcount\g_count
\newdimen\a_dimen
\newdimen\b_dimen
\newdimen\c_dimen
\newdimen\d_dimen
\newtoks\a_toks
\newtoks\b_toks
\newtoks\c_toks
```

\out_file

Some global variables:

```
\newwrite\out_file
```

Some constants:

```
\mathchardef\one_thousand=1000
\mathchardef\max_mathchardef="7FFF
```

6.4 Trigonometry macros

Include David Carlisle's trigonometry macros. First, do some hacks to get round all that `\NeedsTeXFormat{LaTeX2e}` stuff that David put in :-) In v1.8 these hacks been moved to an earlier place since we put in `\ProvidesPackage` ourselves.

```
\let\@tempdima\a_dimen
\let\@tempdimb\b_dimen
\input trig.sty
```

7 TEX hackery

7.1 Utiltiy macros

\x_cs
\x_relax
\g_let

```
\def\x_cs#1#2{\expandafter#1\csname#2\endcsname}
\let\x_relax=\relax
\def\g_let{\global\let}
```

\empty_command
\gobble_one
\gobble_two
\gobble_three
\identity_one
\first_of_two
\second_of_two

```
\def\empty_command{}
\def\gobble_one#1{}
\def\gobble_two#1#2{}
\def\gobble_three#1#2#3{}
\def\identity_one#1{#1}
\def\first_of_two#1#2{#1}
\def\second_of_two#1#2{#2}
```

\percent_char
\left_brace_char
\right_brace_char

```
\bgroup
   \catcode`\[=1
   \catcode`\]=2
   \catcode`\%=12
   \catcode`\{=12
   \catcode`\}=12
   \gdef\percent_char[%]
   \gdef\left_brace_char[{]
   \gdef\right_brace_char[}]
\egroup
```

\lose_measure

Get rid of a trailing `pt` when converting dimension.

```
{
   \catcode`\p=12
   \catcode`\t=12
   \gdef\lose_measure#1pt{#1}
}
```

\first_char Return the first character of a string.

```
\def\first_char#1#2={#1}
```

\add_to Append a element list to a token list.

```
\def\add_to#1#2{
   \ifx#1\relax
      \def#1{#2}
   \else
      \expandafter\def\expandafter#1\expandafter{#1#2}
   \fi
}
```

\never_do The command `\do`, protected from expansion.

```
\def\never_do{\noexpand\do}
```

7.2 Writing to `.fd` files

\open_out
\close_out
\out_line
\out_lline
\out_llline

```
\def\open_out#1{
   \immediate\openout\out_file=#1 \def\out_filename{#1}}
\def\close_out#1{
   \immediate\write16{#1~written~on~\out_filename.}
   \immediate\closeout\out_file}
\def\out_line#1{\immediate\write\out_file{#1}}
\def\out_lline#1{\out_line{\space\space\space#1}}
\def\out_llline#1{\out_lline{\space\space\space#1}}
```

7.3 Conditionals

\if_true
\if_false

In order to write macros that expand out to nested `\if`-statements, I say:

```
\ifblah...\then...\else...\fi
```

In order to match the `\fi`, `\then` has to be an `\if`.

```
\let\then=\iffalse
\def\if_false{\iffalse}
\def\if_true{\iftrue}
```

`\if_or...\or_else...\then` gives the disjunction of two booleans.

```
\def\if_or#1\or_else#2\then{
   #1\then
      \expandafter\if_true
   \else
      #2\then
         \expandafter\expandafter\expandafter\if_true
      \else
         \expandafter\expandafter\expandafter\if_false
      \fi
   \fi
}
```

\if_file_exists

\if_file_exists checks to see if a file exits, using \openin.

```
\def\if_file_exists#1\then{
   \immediate\openin1=#1\relax
   \ifeof1\relax
      \immediate\closein1
      \expandafter\if_false
   \else
      \immediate\closein1
      \expandafter\if_true
   \fi
}
```

\primitiveinput

If \@@input is defined, I'll assume it's the LaTeX version of the TeX \input primitive. I need this so that I can say \expandafter\primitiveinput foo, which doesn't work with the LaTeX version of \input.

```
\x_cs\ifx{@@input}\relax
   \let\primitiveinput=\input
\else
   \let\primitiveinput=\@@input
\fi
```

7.4 Setting variables

\setint
\setstr
\setdim
\setcommand

The macros:

```
\setint{INT}{INTEGER EXPRESSION}
\setstr{STR}{STRING}
\setdim{DIM}{DIMENSION}
\setcommand \COMMAND DEFINITION
```

define new macros \i-INT, \s-STR, \d-DIM or \COMMAND.

\resetint
\resetstr
\resetdim
\resetcommand

The macros:

```
\resetint{INT}{INTEGER EXPRESSION}
\resetstr{STR}{STRING}
\resetdim{DIM}{DIMENSION}
\resetcommand \COMMAND DEFINITION
```

redefine the macros \i-INT, \s-STR, \d-DIM or \COMMAND.

\int
\str
\dim

The macros:

```
\int{INT}
\str{STR}
\dim{DIM}
\COMMAND
```

return the values of \i-INT, \s-STR, \d-DIM or \COMMAND.

\ifisint
\ifisstr
\ifisdim
\ifiscommand

The macros:

```
\ifisint{INT}\then
\ifisstr{STR}\then
\ifisdim{DIM}\then
\ifiscommand\COMMAND\then
```

return `\if_true` if `\i-INT`, `\s-STR` or `\d-DIM` have been defined, and `\if_false` otherwise.

`\unsetint` `\unsetstr` `\unsetdim` `\unsetcommand`

The macros:

```
\unsetint{INT}
\unsetstr{STR}
\unsetdim{DIM}
\unsetcommand \COMMAND
```

undefine `\i-INT`, `\s-STR`, `\d-DIM` or `\COMMAND`.

Integers are kept as `\mathchardef`s if possible. I compared the version where you try to keep ints as `\mathchardef`s with the version where you don't bother, and for a sample font without `\mathchardef`s I got:

114673 words of memory out of 150001
Time elapsed: 135.0 seconds

and with, I got:

114050 words of memory out of 150001
Time elapsed: 134.5 seconds

so I've saved a little memory and time. Not brilliant, but I may as well keep it in.

`\setint` `\setstr` `\setdim` `\setcommand`

```
\def\setint#1#2{
   \x_cs\ifx{i-#1}\relax
      \resetint{#1}{#2}
   \fi
}
\def\setstr#1#2{\x_cs\ifx{s-#1}\relax
   \x_cs\edef{s-#1}{#2}\fi}
\def\setdim#1#2{\x_cs\ifx{d-#1}\relax
   \a_dimen=#2\relax\x_cs\edef{d-#1}{\the\a_dimen}\fi}
\def\setcommand#1{\ifx#1\undefined_command
   \expandafter\def\expandafter#1\else
   \expandafter\def\expandafter\a_command\fi}
```

`\resetint` `\resetstr` `\resetdim` `\resetcommand`

```
\def\resetint#1#2{
   \eval_expr{#2}
   \ifnum\result<\max_mathchardef
      \ifnum\result<0
         \x_cs\edef{i-#1}{\the\result}
      \else
         \x_cs\mathchardef{i-#1}=\result
      \fi
   \else
      \x_cs\edef{i-#1}{\the\result}
   \fi
}
\def\resetstr#1#2{\x_cs\edef{s-#1}{#2}}
\def\resetdim#1#2{\a_dimen=#2\x_cs\edef{d-#1}{\the\a_dimen}}
\def\resetcommand#1{\def#1}
```

\int
\str
\dim

```
\def\int#1{\csname~i-#1\endcsname}
\def\str#1{\csname~s-#1\endcsname}
\def\dim#1{\csname~d-#1\endcsname}
```

\ifisint
\ifisstr
\ifisdim
\ifiscommand

```
\def\ifisint#1\then{\x_cs\ifx{i-#1}\relax\expandafter\if_false
   \else\expandafter\if_true\fi}
\def\ifisstr#1\then{\x_cs\ifx{s-#1}\relax\expandafter\if_false
   \else\expandafter\if_true\fi}
\def\ifisdim#1\then{\x_cs\ifx{d-#1}\relax\expandafter\if_false
   \else\expandafter\if_true\fi}
\def\ifiscommand#1\then{\ifx#1\undefined_command\expandafter\if_false
   \else\expandafter\if_true\fi}
```

\unsetint
\unsetstr
\unsetdim
\unsetcommand

```
\def\unsetint#1{\x_cs\let{i-#1}\relax}
\def\unsetstr#1{\x_cs\let{s-#1}\relax}
\def\unsetdim#1{\x_cs\let{d-#1}\relax}
\def\unsetcommand#1{\let#1=\undefined_command}
```

7.5 Integer expressions

\eval_expr
\eval_expr_to
\g_eval_expr_to

The macro:

```
\eval_expr{INTEGER EXPRESSION}
```

globally assigns `\result` to the value of `INTEGER EXPRESSION`, and changes the value of no other counters.

The macro:

```
\eval_expr_to{VARIABLE}{INTEGER EXPRESSION}
```

locally assigns the value of `INTEGER EXPRESSION` to `VARIABLE`. `\g_eval_expr_to` does the same globally.

```
\newcount\result
\def\eval_expr#1{\global\result=#1\relax}
\def\eval_expr_to#1#2{\eval_expr{#2}#1=\result}
\def\g_eval_expr_to#1#2{\eval_expr{#2}\global#1=\result}
```

\neg
\add
\sub
\mul
\div
\max
\min
\scale

These macros return an integer expression:

```
\neg{INTEGER EXPRESSION}
\add{INTEGER EXPRESSION}{INTEGER EXPRESSION}
\sub{INTEGER EXPRESSION}{INTEGER EXPRESSION}
\mul{INTEGER EXPRESSION}{INTEGER EXPRESSION}
\div{INTEGER EXPRESSION}{INTEGER EXPRESSION}
\max{INTEGER EXPRESSION}{INTEGER EXPRESSION}
\min{INTEGER EXPRESSION}{INTEGER EXPRESSION}
\scale{INTEGER EXPRESSION}{INTEGER EXPRESSION}
```

```
\def\neg#1{#1\global\result=-\result}
\def\add#1#2{#1{\a_count=\result\eval_expr{#2}
   \global\advance\result by \a_count}}
\def\sub#1#2{#1{\a_count=\result\eval_expr{#2}
   \advance\a_count by -\result
   \global\result=\a_count}}
\def\mul#1#2{#1{\a_count=\result\eval_expr{#2}
   \global\multiply\result by \a_count}}
\def\div#1#2{#1{\a_count=\result\eval_expr{#2}
   \divide\a_count by \result
   \global\result=\a_count}}
\def\max#1#2{#1{\a_count=\result\eval_expr{#2}
   \ifnum\a_count>\result \global\result=\a_count \fi}}
\def\min#1#2{#1{\a_count=\result\eval_expr{#2}
   \ifnum\a_count<\result \global\result=\a_count \fi}}
\def\scale#1#2{#1{\a_count=\result\eval_expr{#2}
   \global\multiply\result by \a_count
   \global\divide\result by \one_thousand}}
```

8 Reading and parsing input files

8.1 Converting an ENC file to an ETX file

\enctoext The macro:

```
\enctoetx{ENCFILE}{ETXFILE}
```

reads `ENCFILE.enc` and writes the same information to `ETXFILE.etx`, in a format TeX can read more easily.

Each `/GLYPH` command is recorded in a macro `\o-GLYPH`, which expands to a corresponding `\setslot{GLYPH} ... \endslot` statement.

```
\newif\ifmissingslots
\x_cs\def{o-.notdef}#1{\global\missingslotstrue}
```

\enctoext

```
{
   \catcode`\/=\active
   \catcode`\]=\active
   \gdef\enctoetx#1#2{{
      \catcode`\/=\active
      \catcode`\]=\active
      \def/##1[{
         \a_count=0
         \global\missingslotsfalse
         \def/####1~{
            \csname~o-####1\endcsname{
               \ifmissingslots
                  \out_line{\string\nextslot{\the\a_count}}
               \fi
               \global\missingslotsfalse
               \out_line{\string\setslot{####1}}
               \out_line{\string\endsetslot}
               \out_line{}
```

```
            }
            \advance\a_count by 1
         }
      }
      \def]~def{}
      \make_etx{#1}{#2}
   }}
}
```

`\make_etx` finishes the job of `\enctoext`.

```
\def\make_etx#1#2{
   \open_out{\temp_prefix#2.etx}
   \out_line{\percent_char~Filename:~#2.etx}
   \out_line{\percent_char~Created~by:~tex~\jobname}
   \out_line{\percent_char~Created~using:~\string\enctoetx{#1}{#2}}
   \out_line{}
   \out_line{\percent_char~This~file~contains~the
      information~of~#1.enc~in~a~form}
   \out_line{\percent_char~more~easily~read~by~TeX.
      It~is~used~by~the~fontinst~package.}
   \out_line{}
   \out_line{\percent_char~THIS~FILE~CAN~BE~DELETED.}
   \out_line{}
   \out_line{\string\relax}
   \out_line{}
   \out_line{\string\documentclass[twocolumn]{article}}
   \out_line{\string\usepackage{fontdoc}}
   \out_line{}
   \out_line{\string\begin{document}}
   \out_line{}
   \out_line{This~document~describes~the~#1~encoding.}
   \out_line{It~was~automatically~generated~by~the}
   \out_line{{\string\tt\space~fontinst}~package.}
   \out_line{}
   \out_line{\string\encoding}
   \out_line{}
   \out_line{\string\needsfontinstversion{\fontinstversion}}
   \out_line{}
   \primitiveinput #1.enc\relax
   \out_line{}
   \out_line{\string\endencoding}
   \out_line{}
   \out_line{\string\end{document}}
   \close_out{Encoding~vector}
}
```

8.2 Converting an AFM file to an MTX file

`\afmtomtx` The macro:

`\afmtomtx{AFMFILE}{MTXFILE}`

reads `AFMFILE.afm`, and writes the same information out to `MTXFILE.mtx`.

\afmtomtx

```
\def\afmtomtx#1#2{{
   \open_out{\temp_prefix#2.mtx}
   \def\raw_font_name{#2}
   \_italicfalse
   \setint{minimumkern}{0}
   \minimum_kern=\int{minimumkern}
   \out_line{\percent_char~Filename:~#2.mtx}
   \out_line{\percent_char~Created~by:~tex~\jobname}
   \out_line{\percent_char~Created~using:~\string\afmtomtx{#1}{#2}}
   \out_line{}
   \out_line{\percent_char~This~file~contains~the
      information~of~#1.afm~in~a~form}
   \out_line{\percent_char~more~easily~read~by~TeX.~
      It~is~used~by~the~fontinst~package.}
   \out_line{}
   \out_line{\percent_char~THIS~FILE~CAN~BE~DELETED.}
   \out_line{}
   \out_line{\string\relax}
   \out_line{\string\metrics}
   \out_line{}
   \out_line{\string\needsfontinstversion{\fontinstversion}}
   \out_line{}
   \catcode`\^^M=12
   \catcode`\ =10
   \expandafter\afm_line\primitiveinput #1.afm\relax
   \out_line{}
   \out_line{\string\endmetrics}
   \close_out{Metrics}
}}
```

Kerns below this value are ignored.

```
\newcount\minimum_kern
```

The variable `\if_italic` is set true if the font is italic.

```
\newif\if_italic
```

The command `\afm_line` reads to the end of the line, calls `\afm_command` on that line, then calls `\afm_line` again.

```
{\catcode`\^^M=12 \gdef\afm_line#1
{\afm_command#1~\end_of_line\afm_line}}
```

The command `\afm_command` reads the first word `FOO`, and calls `afm-FOO`. If this does not exist, then `\gobble_one_line` will eat up the rest of the line.

```
\def\afm_command#1~{\csname~afm-#1\endcsname\gobble_one_line}
\def\gobble_one_line#1\end_of_line{}
```

This all stops when we reach the command `EndFontMetrics`.

```
\x_cs\def{afm-EndFontMetrics}#1\afm_line{\endinput}
```

To define an AFM command, you say `\afm_def{COMMAND}(PATTERN){RESULT}`

```
\def\afm_def#1(#2)#3{\x_cs\def{afm-#1}
   \gobble_one_line#2\end_of_line{#3}}
```

For example, we can define the following AFM commands: (Fixed by Thierry Bouche <Thierry.Bouche@ujf-grenoble.fr> 1997/02/07, to get fontdimens comparable to EC fonts.)

```
\afm_def{CharWidth}(#1){\charxwidth=#1\relax}
\afm_def{ItalicAngle}(#1){\calculate_it_slant{#1}}
\afm_def{XHeight}(#1){
   \out_line{\string\setint{xheight}{#1}}}
\afm_def{CapHeight}(#1){
   \out_line{\string\setint{capheight}{#1}}}
\afm_def{Ascender}(#1){
   \out_line{\string\setint{ascender}{#1}}}
\afm_def{Descender}(#1){
   \out_line{\string\setint{descender_neg}{#1}}}
\afm_def{UnderlineThickness}(#1){
   \out_line{\string\setint{underlinethickness}{#1}}}
\afm_def{FontBBox}(#1~#2~#3~#4){
   \out_line{\string\setint{maxheight}{#4}}
   \out_line{\string\setint{maxdepth_neg}{#2}}}
\afm_def{IsFixedPitch}(#1){
   \if\first_char#1=f
   \else\out_line{\string\setint{monowidth}{1}}
   \fi
}
```

Processing kern pairs. If one of the glyph name starts with a dot as in `.notdef` or `.null` the kern pair is ignored.

```
\afm_def{KP}(#1~#2~#3~#4){
   \if\first_char#1=.\else
   \if\first_char#2=.\else
      \ifnum#3>\minimum_kern\relax
         \out_line{\string\setkern{#1}{#2}{#3}}
      \else\ifnum#3<-\minimum_kern\relax
         \out_line{\string\setkern{#1}{#2}{#3}}
      \fi\fi
   \fi\fi
}
\afm_def{KPX}(#1~#2~#3~#4){
   \if\first_char#1=.\else
   \if\first_char#2=.\else
      \ifnum#3>\minimum_kern\relax
         \out_line{\string\setkern{#1}{#2}{#3}}
      \else\ifnum#3<-\minimum_kern\relax
         \out_line{\string\setkern{#1}{#2}{#3}}
      \fi\fi
   \fi\fi
}
```

Processing char metrics.

```
\afm_def{C}(#1~;#2){\init_afm{#1}\do_list[#2]\afm_char}
\afm_def{CH}(#1~;#2){\init_afm{#1}\do_list[#2]\afm_char}
```

Processing composite chars.

```
\afm_def{CC}(#1~#2~;#3){\init_cc{#1}\do_list[#3]\cc_char}
```

When parsing a character, we set the values of the following variables:

```
\newcount\char_slot
\newcount\char_x_width
\newcount\x_width
\newcount\bbox_llx
\newcount\bbox_lly
\newcount\bbox_urx
\newcount\bbox_ury
\let\char_name=\empty_command
```

`\init_afm` initializes the variables the AFM character list writes to.

```
\def\init_afm#1{
   \char_slot=#1\relax
   \x_width=\char_x_width
   \bbox_llx=0
   \bbox_lly=0
   \bbox_urx=0
   \bbox_ury=0
   \let\char_name=\empty_command
}
```

`\afm_char` writes the values of these variables to the `.mtx` file, and saves them in a macro `\g-GLYPHNAME`, in the form:

```
{WIDTH}{HEIGHT}{DEPTH}{ITALIC}
```

These are needed, because the (grumble grumble) syntax of `CC` commands doesn't include the glyph measurements, so we have to remember them.

```
\def\afm_char{
   \a_count=-\bbox_lly
   \if_italic
      \b_count=-\x_width
      \advance\b_count by \bbox_urx
      \ifnum\b_count>0\else
         \b_count=0
      \fi
   \else
      \b_count=0
   \fi
   \out_line{
      \ifnum\char_slot>-1
         \string\setrawglyph
      \else
         \string\setnotglyph
      \fi
      {\char_name}
      {\raw_font_name}
      {10pt}
      {\the\char_slot}
      {\the\x_width}
      {\the\bbox_ury}
      {\the\a_count}
      {\the\b_count}
   }
}
```

\init_cc and \cc_char write out a composite character glyph.

```
\def\init_cc#1{%
   \out_line{\string\setglyph{#1}}
   \def\char_name{#1}
}
\def\cc_char{%
   \out_lline{\string\samesize{\char_name-not}}
   \out_line{\string\endsetglyph}
}
```

To set the italic angle, we need to calculate the tangent of the angle that the .afm file contains. This is done with David Carlisle's trig macros. We need to do a bit of trickery to strip off any spaces which may have crept into the end of the angle, since the trig macros don't like space at the end of their argument.

```
\def\calculate_it_slant#1{
   \edef\theangle{\strip_spaces#1~\end_strip_spaces}
   \CalculateTan{\theangle}
   \a_dimen=-\one_thousand sp
   \a_dimen=\UseTan{\theangle}\a_dimen
   \a_count=\a_dimen
   \out_line{\string\setint{italicslant}{\the\a_count}}
   \ifnum\a_count>0~\_italictrue \else \_italicfalse \fi
}
```

```
\def\strip_spaces#1~#2\end_strip_spaces{#1}
```

To process a list of commands separated by semi-colons, we call \do_list[LIST]. This works in a similar way to \afm_line.

```
\def\do_list[~#1~#2;~#3]{
   \csname~list-#1\endcsname\gobble_one_semi#2;
   \ifx\relax#3\relax\expandafter\gobble_one
   \else\expandafter\identity_one\fi
   {\do_list[~#3]}
}
\def\gobble_one_semi#1;{}
```

There is an analagous \list_def for defining commands to be used inside lists.

```
\def\list_def#1(#2)#3{\x_cs\def{list-#1}\gobble_one_semi#2~;{#3}}
```

For example, these are the commands that are used in giving character metrics:

```
\list_def{W}(#1~#2){\x_width=#1\relax}
\list_def{WX}(#1){\x_width=#1\relax}
\list_def{WY}(#1){}
\list_def{N}(#1){\def\char_name{#1}}
\list_def{B}(#1~#2~#3~#4){
   \bbox_llx=#1\relax
   \bbox_lly=#2\relax
   \bbox_urx=#3\relax
   \bbox_ury=#4\relax
}
\list_def{PCC}(#1~#2~#3){
   \out_lline{\string\glyphpcc{#1}{#2}{#3}}
}
```

8.3 Converting a PL file to an MTX file

\pltomtx The macro:

```
\pltomtx{PLFILE}{MTXFILE}
```

reads PLFILE.pl, and writes the same information out to MTXFILE.mtx. It can't cope with SKIP commands.

\pltomtx

```
\def\pltomtx#1#2{{
   \def\raw_font_name{#1}
   \open_out{\temp_prefix#2.mtx}
   \out_line{\percent_char~Filename:~#2.mtx}
   \out_line{\percent_char~Created~by:~tex~\jobname}
   \out_line{\percent_char~Created~using:~\string\pltomtx{#1}{#2}}
   \out_line{}
   \out_line{\percent_char~This~file~contains~the~
      information~of~#1.pl~in~a~form}
   \out_line{\percent_char~more~easily~read~by~TeX.~
      It~is~used~by~the~fontinst~package.}
   \out_line{}
   \out_line{\percent_char~THIS~FILE~CAN~BE~DELETED.}
   \out_line{}
   \out_line{\string\relax}
   \out_line{\string\metrics}
   \out_line{}
   \out_line{\string\needsfontinstversion{\fontinstversion}}
   \out_line{}
   \catcode`\(=0 \catcode`\)=9
   \let\/=\ignore_parens
   \let\do_pl_glyph=\relax
   \primitiveinput #1.pl\relax
   \do_pl_glyph
   \out_line{}
   \out_line{\string\endmetrics}
   \close_out{Metrics}
}}
```

To parse a .pl file, we first make (the escape character, make) ignored, then define the various PL commands.

We can ignore a parenthesis matched string by making (and) the group delimiters, then gobbling them up.

```
\def\ignore_parens{\bgroup\catcode`(=1 \catcode`)=2 \relax
   \expandafter\expandafter\expandafter\gobble_parens
      \iftrue\expandafter{\else}\fi}
\def\gobble_parens#1{\egroup}
```

Convert a PL real to an AFM unit, assuming it contains a decimal point.

```
\def\pl_real#1{\pl_realer(#1000)}
\def\pl_realer(#1.#2#3#4#5){#1#2#3#4}
```

Convert a PL int to a TeX int, assuming it's prefixed by C, D, O, or H.

```
\def\pl_int#1#2{
```

```
   \ifx#1C `#2
   \else\ifx#1D #2
   \else\ifx#1O '#2
   \else\ifx#1H "#2
   \else\errmessage{Unknown~PL~number~prefix~`#1'}
   \fi\fi\fi\fi
}
```

Many of the PL commands are ignored, and I'm assuming the `R`s are in the places `tftopl` puts them, which is a bit naughty of me.

```
\let\COMMENT=\ignore_parens
\let\FAMILY=\ignore_parens
\let\FACE=\ignore_parens
\let\CHECKSUM=\ignore_parens
\let\BOUNDARYCHAR=\ignore_parens
\let\LIG=\ignore_parens
\let\NEXTLARGER=\ignore_parens
\let\VARCHAR=\ignore_parens
```

When we reach a `CODINGSCHEME` instruction, we read the coding string, and read in the corresponding `ENCODING.etx` file.

The corresponding `ENCODING` is specified by `\declareencoding` statements (see below). Each `\declare_encdoing` defines a macro `\enc-CODINGSCHEME` which expands to `ENCODING`.

```
\def\CODINGSCHEME{\bgroup\catcode`\)=12\relax\CODINGSCHEME_cont}
\def\CODINGSCHEME_cont#1){
   \egroup
   \x_cs\ifx{enc-#1}\relax
      \errhelp{The~encoding~`#1'~has~not~been~declared.^^J
         You~should~declare~it~with~\string\declareencoding{#1}{ETXFILE}.^^J
         Press~<RETURN>~to~carry~on~with~fingers~crossed,^^J
         or~X~<RETURN>~to~exit.}
      \errmessage{Undeclared~encoding~`#1'}
   \else
      \def\do_slot{\x_cs\let{name-\the\slot_number}\slot_name}
      \catcode`\(=12 \catcode`\)=12
      \x_cs\inputetx{enc-#1}\relax
      \catcode`\(=0 \catcode`\)=9
   \fi
}
```

```
\def\DESIGNSIZE~#1~#2~{
   \a_dimen=#2pt
   \out_line{\string\setdim{designsize}{\the\a_dimen}}
}
\def\DESIGNUNITS~#1~#2~{
   \a_dimen=#2pt
   \out_line{\string\setdim{designunits}{\the\a_dimen}}
}
```

The following fontdimens are converted to ints and written out to the `.mtx` file.

```
\let\FONTDIMEN=\relax
\def\SLANT~R~#1~{\out_line{\string\setint{italicslant}{\pl_real{#1}}}}
```

```
\def\SPACE~R~#1~{\out_line{\string\setint{interword}{\pl_real{#1}}}}
\def\STRETCH~R~#1~{\out_line{\string\setint{stretchword}{\pl_real{#1}}}}
\def\SHRINK~R~#1~{\out_line{\string\setint{shrinkword}{\pl_real{#1}}}}
\def\XHEIGHT~R~#1~{\out_line{\string\setint{xheight}{\pl_real{#1}}}}
\def\QUAD~R~#1~{\out_line{\string\setint{quad}{\pl_real{#1}}}}
\def\EXTRASPACE~R~#1~{\out_line{\string\setint{extraspace}{\pl_real{#1}}}}
\def\DEFAULTRULETHICKNESS~R~#1~{
   \out_line{\string\setint{underlinethickness}{\pl_real{#1}}}
}
```

The following fontdimens are ignored.

```
\def\HEADER~#1~#2~#3~#4~{}
\def\SEVENBITSAFEFLAG~#1~{}
\def\PARAMETER~#1~#2~#3~#4~{}
\def\NUM#1~#2~#3~{}
\def\DENOM#1~#2~#3~{}
\def\SUP#1~#2~#3~{}
\def\SUB#1~#2~#3~{}
\def\SUBDROP~#1~#2~{}
\def\SUPDROP~#1~#2~{}
\def\DELIM#1~#2~#3~{}
\def\AXISHEIGHT~#1~#2~{}
\def\BIGOPSPACING#1~#2~#3~{}
```

The following ligtable commands are processed.

```
\def\LIGTABLE{\let\do=\never_do\let\temp_command\empty_command}
\def\LABEL~#1~#2~{\edef\temp_command{\temp_command\do{\pl_int{#1}{#2}}}}
\def\STOP{\let\temp_command\empty_command}
\def\SKIP~#1~#2~{\immediate\write16{Warning:~SKIP~instruction~ignored.}}
\def\KRN~#1~#2~R~#3~{
   \edef\do{\noexpand\write_pl_krn{\pl_int{#1}{#2}}{\pl_real{#3}}}
   \temp_command
   \let\do=\never_do
}

\def\write_pl_krn#1#2#3{
   \f_count=#1
   \g_count=#3
   \out_line{\string\setkern{\csname~name-\the\g_count\endcsname}
      {\csname~name-\the\f_count\endcsname}
      {#2}}
}
```

The following character metrics are processed.

```
\def\CHARWD~R~#1~{\b_count=\pl_real{#1}}
\def\CHARHT~R~#1~{\c_count=\pl_real{#1}}
\def\CHARDP~R~#1~{\d_count=\pl_real{#1}}
\def\CHARIC~R~#1~{\e_count=\pl_real{#1}}
\def\CHARACTER~#1~#2~{
   \do_pl_glyph
   \a_count=\pl_int{#1}{#2}
   \b_count=0
   \c_count=0
   \d_count=0
   \e_count=0
```

```
    \let\do_pl_glyph=\write_pl_glyph
}
```

```
\def\write_pl_glyph{
      \x_cs\ifx{name-\the\a_count}\relax\else
         \out_line{\string\setrawglyph
            {\csname~name-\the\a_count\endcsname}
            {\raw_font_name}
            {\the\a_dimen}
            {\the\a_count}
            {\the\b_count}
            {\the\c_count}
            {\the\d_count}
            {\the\e_count}}
      \fi
   }
```

9 Encoding files

\inputetx

The macro:

```
\inputetx{FILENAME}
```

inputs `FILENAME.etx`, ignoring anything between `\relax` and `\encoding`, and anything after `\endencoding`.

The file name is transformed to lowercase before opening.

\inputetx

```
\def\inputetx#1{
   \edef\lowercase_file{\lowercase{
     \edef\noexpand\lowercase_file{#1}}}
   \lowercase_file
   \slot_number=0
   \def\relax{\let\relax=\x_relax\iffalse}
   \let\encoding=\fi
   \let\endencoding=\endinput
   \primitiveinput \lowercase_file.etx\x_relax
   \let\relax=\x_relax
}
```

\setslot
\endsetslot
\slot_name

`\setslot{NAME} SLOT COMMANDS \endsetslot`

Sets `\slot_number` to the current slot, `\slot_name` to `NAME`, and calls `\do_slot` at the beginning of the slot and `\end_do_slot` at the end. By default, `\do_slot` and `\end_do_slot` do nothing, but this is over-ridden later.

```
\def\setslot#1{\edef\slot_name{#1}\do_slot}
\def\endsetslot{\end_do_slot\advance\slot_number by 1\relax}
```

```
\let\do_slot\empty_command
\let\do_new_slot\empty_command
\let\end_do_slot\empty_command
```

\nextslot
\skipslots
\slot_number

`\nextslot{INTEGER EXPRESSION}`
`\skipslots{INTEGER EXPRESSION}`

`\nextslot` sets the `\slot_number`. `\skipslots` addvances the `\slot_number`.

```
\newcount\slot_number
\def\nextslot#1{\slot_number=#1\relax}
\def\skipslots#1{\advance\slot_number by #1\relax}
```

\ligature
\nextlarger
\varchar
\comment

The `SLOT COMMANDS` are:

```
\ligature{LIG}{GLYPH}{GLYPH}
\nextlarger{GLYPH}
\varchar VARCHAR COMMANDS \endvarchar
\comment{TEXT}
\usedas{COMMAND}{TYPE}
```

By default, these do nothing, but are over-ridden later.

```
\let\ligature=\gobble_three
\let\nextlarger=\gobble_one
\let\varchar=\empty_command
\let\endvarchar=\empty_command
\let\comment=\gobble_one
\let\usedas=\gobble_two
```

\vartop
\varmid
\varbot
\varrep

The `VARCHAR COMMANDS` are:

```
\vartop{GLYPH}
\varmid{GLYPH}
\varbot{GLYPH}
\varrep{GLYPH}
```

```
\let\vartop=\gobble_one
\let\varmid=\gobble_one
\let\varbot=\gobble_one
\let\varrep=\gobble_one
```

\useexamplefont
\slotexample
\setslotcomment
\resetslotcomment
\unsetslotcomment

These macros were recently added in the `fontdoc` package. They do nothing in `fontinst`, just gobble their arguments.

```
\let\useexamplefont=\gobble_one
\let\slotexample=\empty_command
\let\setslotcomment=\gobble_one
\let\resetslotcomment=\gobble_one
\let\unsetslotcomment=\empty_command
```

10 Metric files

\inputmtx

The macro:

```
\inputmtx{FILENAME}
```

inputs `FILENAME.mtx`, ignoring anything between `\relax` and `\metrics`, and anything after `\endmetrics`.

\inputmtx

```
\def\inputmtx#1{
   \a_count=0
```

```
   \def\relax{\let\relax=\x_relax\iffalse}
   \let\metrics=\fi
   \let\endmetrics=\endinput
   \primitiveinput #1.mtx\x_relax
   \let\relax=\x_relax
}
```

10.1 Kerning information

The kerning information is kept in the macros `\l-NAME` and `\r-NAME`, containing information about how `NAME` kerns on the left and on the right, respectively. The `\l-NAME` macro should expand out to a series of `\k\r-NAME\AMOUNT` macros, and vice versa.

Where possible, we avoid re-scaling, which saves a bit of time and memory. With a sample font, the version where we didn't avoid re-scaling used:

> 114050 words of memory out of 150001
> Time elapsed: 134.5 seconds

whereas the version where we do avoid it used:

> 113786 words of memory out of 150001
> Time elapsed: 124.9 seconds

We keep the names of the glyphs to kern with as `\l-NAME` and `\r-NAME` to save on token space, and this got the resources used down to:

> 88574 words of memory out of 150001
> Time elapsed: 106.1 seconds

Keeping track of the kern amounts as `\AMOUNT` got the resources used down to:

> 75424 words of memory out of 150001
> Time elapsed: 97.2 seconds

Mind you, I then added all the `\transformfont` stuff, and it went back to:

> 77079 words of memory out of 150001
> Time elapsed: 97.7 seconds

`\setkern` `\setkern{GLYPH1}{GLYPH2}{INTEGER EXPERESSION}`

Sets a kern pair between `GLYPH1` and `GLYPH2` to the specified value, which is typically a value returned by `\kerning{GLYPH3}{GLYPH4}`.

```
\def\setkern#1#2#3{
   \x_cs\ifx{i-rawscale}\relax
      \expandafter\set_kern
         \csname~r-#1\expandafter\endcsname
         \csname~l-#2\endcsname{#3}
   \else
      \expandafter\set_kern
         \csname~r-#1\expandafter\endcsname
         \csname~l-#2\endcsname{\scale{#3}{\int{rawscale}}}
   \fi
}
```

```
\def\set_kern#1#2#3{
   \eval_expr{#3}
   \expandafter\set_kern_cs\csname\the\result\endcsname#1#2
}
\def\set_kern_cs#1#2#3{
   \add_to#2{\k#3#1}
   \add_to#3{\k#2#1}
}
```

\setleftkerning
\setrightkerning
\setleftrightkerning

`\setleftkerning{GLYPH1}{GLYPH2}{SCALED}`
`\setrightkerning{GLYPH1}{GLYPH2}{SCALED}`
`\setleftrightkerning{GLYPH1}{GLYPH2}{SCALED}`

Sets left or right kerning of `GLYPH1` to that of `GLYPH2` scaled by `SCALED`. `\setleftrightkerning` does both.

```
\def\setleftkerning#1#2#3{
   \eval_expr_to\b_count{#3}
   \expandafter\set_kerning
      \csname~l-#1\expandafter\endcsname
      \csname~l-#2\endcsname
}
\def\setrightkerning#1#2#3{
   \eval_expr_to\b_count{#3}
   \expandafter\set_kerning
      \csname~r-#1\expandafter\endcsname
      \csname~r-#2\endcsname
}
\def\setleftrightkerning#1#2#3{
   \eval_expr_to\b_count{#3}
   \expandafter\set_kerning
      \csname~l-#1\expandafter\endcsname
      \csname~l-#2\endcsname
   \expandafter\set_kerning
      \csname~r-#1\expandafter\endcsname
      \csname~r-#2\endcsname
}
\def\set_kerning#1#2{
   \if\b_count=\one_thousand
      \def\k##1##2{
         \set_kern_cs##2##1#1
      }
   \else
      \def\k##1##2{
         \set_kern##1#1{
            \scale\b_count{\expandafter\gobble_one\string##2}
         }
      }
   \fi
   #2
}
```

\kerning

`\kerning{GLYPH1}{GLYPH2}`

Returns the value of kern pair between `GLYPH1` and `GLYPH2` as an integer. Returns a value of zero if such a kern pair doesn't exist.

```
\def\kerning#1#2{0\relax
   \def\k##1{\csname~set-\string##1\endcsname\gobble_one}
   \bgroup
   \x_cs\def{set-\string\l-#2}##1##2{
      \global\result=\expandafter\gobble_one\string##2\egroup
   }
   \csname~r-#1\endcsname
   \csname~set-\string\l-#2\endcsname\gobble_one{00}
}
```

10.2 Glyph information

The glyph information is kept in the macros `\g-NAME`, which expands out to:

`{WIDTH}{HEIGHT}{DEPTH}{ITALIC}{MAPCOMMANDS}{MAPFONTS}`

where the `MAPCOMMANDS` will write out VPL `MAP` fragments to a `.vpl` file, and the `MAPFONTS` will produce any `MAPFONT` instructions that are needed.

`\width` `\width{GLPYH}`
`\height` `\height{GLPYH}`
`\depth` `\depth{GLPYH}`
`\italic` `\italic{GLPYH}`

These macros return the width, height, depth and italic correction of `GLYPH` as an integer.

```
\def\width{\glyph_parameter\first_of_six}
\def\height{\glyph_parameter\second_of_six}
\def\depth{\glyph_parameter\third_of_six}
\def\italic{\glyph_parameter\fourth_of_six}
\def\mapcommands{\glyph_parameter\fifth_of_six}
\def\mapfonts{\glyph_parameter\sixth_of_six}
```

```
\def\glyph_parameter#1#2{
   \expandafter\expandafter\expandafter
         #1\csname~g-#2\endcsname
}
```

```
\def\first_of_six#1#2#3#4#5#6{#1}
\def\second_of_six#1#2#3#4#5#6{#2}
\def\third_of_six#1#2#3#4#5#6{#3}
\def\fourth_of_six#1#2#3#4#5#6{#4}
\def\fifth_of_six#1#2#3#4#5#6{#5}
\def\sixth_of_six#1#2#3#4#5#6{#6}
```

These are the commands allowed inside a glyph. They are initally set to `\relax`, so we can `\edef` with them safely.

```
\let\saved_scale\relax
\let\saved_mapfont\relax
\let\saved_raw\relax
\let\saved_rule\relax
\let\saved_special\relax
\let\saved_warning\relax
\let\saved_movert\relax
\let\saved_moveup\relax
```

```
\let\saved_push\relax
\let\saved_pop\relax
```

When the glyph is being constructed by

```
\setglyph{GLYPH} GLYPH COMMANDS \endsetglyph
```

the values of each of these parameters are kept in the following variables. Except for `\glyph_width`, these are kept globally, so they survive through `\push ... \pop` pairs. In addition, the current vertical offset is kept locally in `\glyph_voffset`.

```
\newcount\glyph_width
\newcount\glyph_height
\newcount\glyph_depth
\newcount\glyph_italic
\newtoks\glyph_map_commands
\newtoks\glyph_map_fonts
\newcount\glyph_voffset
```

`\setglyph` The `\setglyph{NAME}` command defines `\g-NAME`, unless `\g-NAME` has already been defined.

```
\def\setglyph#1{
   \ifisglyph{#1}\then
      \expandafter\gobble_glyph
   \else
      \x_cs\def{g-#1}{{\the\glyph_width}{\the
         \glyph_height}{\the\glyph_depth}{\the\glyph_italic}{\the
         \glyph_map_commands}{\the\glyph_map_fonts}}
      \resetglyph{#1}
   \fi
}
\long\def\gobble_glyph#1\endsetglyph{}
```

`\resetglyph` The `\resetglyph{NAME}` command redefines `\g-NAME`, regardless of whether `\g-NAME` has already been defined.

```
\def\resetglyph#1{
   \def\glyphname{#1}
   \glyph_width=0
   \global\glyph_height=0
   \global\glyph_depth=0
   \global\glyph_italic=0
   \glyph_voffset=0
   \global\glyph_map_commands={}
   \global\glyph_map_fonts={}
}
```

`\endsetglyph`
`\endresetglyph`

```
\def\endsetglyph{
   \x_cs\edef{g-\glyphname}
      {{\the\glyph_width}{\the\glyph_height}
       {\the\glyph_depth}{\the\glyph_italic}
       {\the\glyph_map_commands}{\the\glyph_map_fonts}}
}
\let\endresetglyph=\endsetglyph
```

\setrawglyph `\setrawglyph{GLYPH}{FONT}{SIZE}{SLOT}{WIDTH}{HEIGHT}{DEPTH}{ITALIC}`

These commands are generated automatically, when an `.mtx` file is written out by `\afmtomtx`, `\pltomtx`, or `\mtxtomtx`.

```
\def\setrawglyph#1#2#3#4#5#6#7#8{
   \x_cs\ifx{g-#1}\relax
      \x_cs\ifx{i-rawscale}\relax
         \x_cs\def{g-#1}{
            {#5}
            {#6}
            {#7}
            {#8}
            {\saved_raw{#2}{#4}{#1}}
            {\saved_mapfont{#2}{#3}}
         }
      \else
         \eval_expr_to\e_count{\int{rawscale}}
         \eval_expr_to\a_count{\scale{#5}\e_count}
         \eval_expr_to\b_count{\scale{#6}\e_count}
         \eval_expr_to\c_count{\scale{#7}\e_count}
         \eval_expr_to\d_count{\scale{#8}\e_count}
         \x_cs\edef{g-#1}{
            {\the\a_count}
            {\the\b_count}
            {\the\c_count}
            {\the\d_count}
            {\saved_scale{\the\e_count}
               {\saved_raw{#2}{#4}{#1}}}
            {\saved_scale{\the\e_count}
               {\saved_mapfont{#2}{#3}}}
         }
      \fi
   \fi
}
```

\setnotglyph

```
\def\setnotglyph#1#2#3#4#5#6#7#8{
   \x_cs\ifx{g-#1-not}\relax
      \x_cs\ifx{i-rawscale}\relax
         \x_cs\def{g-#1-not}{
            {#5}
            {#6}
            {#7}
            {#8}
            {}
            {}
         }
      \else
         \eval_expr_to\e_count{\int{rawscale}}
         \eval_expr_to\a_count{\scale{#5}\e_count}
         \eval_expr_to\b_count{\scale{#6}\e_count}
         \eval_expr_to\c_count{\scale{#7}\e_count}
         \eval_expr_to\d_count{\scale{#8}\e_count}
         \x_cs\edef{g-#1-not}{
            {\the\a_count}
```

```
            {\the\b_count}
            {\the\c_count}
            {\the\d_count}
            {}
            {}
         }
      \fi
   \fi
}
```

\unsetglyph The `\unsetglyph{NAME}` command makes `\g-NAME` undefined.

```
\def\unsetglyph#1{\x_cs\let{g-#1}\relax}
```

10.3 Glyph commands

The `GLYPH COMMANDS` are:

\glyph `\glyph{GLPYH}{SCALE}`

```
\def\glyph#1#2{
   \eval_expr_to\b_count{#2}
   \eval_expr_to\glyph_width{\add\glyph_width{\scale\b_count{\width{#1}}}}
   \g_eval_expr_to\glyph_height{\max\glyph_height
      {\add{\scale\b_count{\height{#1}}}\glyph_voffset}}
   \g_eval_expr_to\glyph_depth{\max\glyph_depth
      {\sub{\scale\b_count{\depth{#1}}}\glyph_voffset}}
   \g_eval_expr_to\glyph_italic{\scale\b_count{\italic{#1}}}
   \edef\temp_command{\the\glyph_map_commands
      \saved_scale{\the\b_count}{\mapcommands{#1}}}
   \global\glyph_map_commands\expandafter{\temp_command}
   \edef\temp_command{\the\glyph_map_fonts
      \saved_scale{\the\b_count}{\mapfonts{#1}}}
   \global\glyph_map_fonts\expandafter{\temp_command}
}
```

\glyphrule `\glyphrule{WIDTH}{HEIGHT}`

```
\def\glyphrule#1#2{
   \eval_expr_to\b_count{#1} \eval_expr_to\c_count{#2}
   \advance\glyph_width by \b_count
   \g_eval_expr_to\glyph_depth{\max\glyph_depth{-\glyph_voffset}}
   \g_eval_expr_to\glyph_height{\max\glyph_height{\add\glyph_voffset\c_count}}
   \global\glyph_italic=0
   \edef\temp_command{\the\glyph_map_commands
      \saved_rule{\the\b_count}{\the\c_count}}
   \global\glyph_map_commands\expandafter{\temp_command}
}
```

\glyphspecial `\glyphspecial{STRING}`
\glyphwarning `\glyphwarning{STRING}`

```
\def\glyphspecial#1{
   \edef\temp_command{\the\glyph_map_commands\saved_special{#1}}
   \global\glyph_map_commands\expandafter{\temp_command}}
\def\glyphwarning#1{
   \edef\temp_command{\the\glyph_map_commands\saved_warning{#1}}
   \global\glyph_map_commands\expandafter{\temp_command}}
```

\movert \moveup `\movert{XOFFSET}`
`\moveup{YOFFSET}`

```
\def\movert#1{
   \eval_expr{#1}
   \ifnum\result=0\else
      \advance\glyph_width by \result
      \edef\temp_command{\the\glyph_map_commands\saved_movert{\the\result}}
      \global\glyph_map_commands\expandafter{\temp_command}
   \fi}
\def\moveup#1{
   \eval_expr{#1}
   \ifnum\result=0\else
      \advance\glyph_voffset by \result
      \edef\temp_command{\the\glyph_map_commands\saved_moveup{\the\result}}
      \global\glyph_map_commands\expandafter{\temp_command}
   \fi}
```

\push \pop

```
\def\push{\bgroup
   \edef\temp_command{\the\glyph_map_commands\saved_push}
   \global\glyph_map_commands\expandafter{\temp_command}}
\def\pop{\egroup
   \edef\temp_command{\the\glyph_map_commands\saved_pop}
   \global\glyph_map_commands\expandafter{\temp_command}}
```

\resetwidth \resetheight \resetdepth \resetitalic `\resetwidth{WIDTH}`
`\resetheight{HEIGHT}`
`\resetdepth{DEPTH}`
`\resetitalic{ITALIC}`

```
\def\resetwidth#1{\movert{\sub{#1}\glyph_width}}
\def\resetheight{\g_eval_expr_to\glyph_height}
\def\resetdepth{\g_eval_expr_to\glyph_depth}
\def\resetitalic{\g_eval_expr_to\glyph_italic}
```

\glyphpcc Some syntactic sugar: `\glyphpcc` could do with optimization.
`\glyphpcc{GLYPH}{XOFFSET}{YOFFSET}`

```
\def\glyphpcc#1#2#3{
   \push
      \movert{#2}
      \moveup{#3}
      \glyph{#1}{\one_thousand}
   \pop
}
```

\samesize `\samesize{GLYPH}`

```
\def\samesize#1{
   \expandafter\expandafter\expandafter
      \same_size\csname g-#1\endcsname
}
\def\same_size#1#2#3#4#5#6{
   \movert{\sub{#1}\glyph_width}
   \global\glyph_height=#2
```

```
    \global\glyph_depth=#3
    \global\glyph_italic=#4
}
```

\ifisglyph The control flow command:

```
\def\ifisglyph#1\then{
    \x_cs\ifx{g-#1}\relax
       \expandafter\if_false
    \else
       \expandafter\if_true
    \fi
}
```

11 Writing output files

11.1 Converting an ETX file to a (V)PL file

\etxtovpl The macro:

```
\etxtovpl{ENCODING}{VPLFILE}
```

writes a virtual font (as a virtual property list) with the encoding ENCODING. (This macro is called by \installfont.)

\etxtopl The macro:

```
\etxtopl{ENCODING}{PLFILE}
```

writes a font (as a property list) with the encoding ENCODING. (This macro is called by \installrawfont.)

\etxtovpl
\etxtopl

```
\def\etxtovpl#1#2{{
    \def\vpl_extension{vpl}
    \def\vpl_title{COMMENT}
    \def\vpl_font{virtual~font}
    \def\vpl_Font{Virtual~font}
    \def\vpl_caller{\string\etxtovpl}
    \def\vpl_to_vf##1{vpltovf~##1.vpl~##1.vf~##1.tfm}
    \etx_to_font{#1}{#2}
}}
\def\etxtopl#1#2{{
    \def\vpl_extension{pl}
    \def\vpl_title{COMMENT}
    \def\vpl_font{font}
    \def\vpl_Font{Font}
    \def\vpl_caller{\string\etxtopl}
    \def\vpl_to_vf##1{pltotfm~##1.pl~##1.tfm}
    \let\make_mapfonts\gobble_one
    \let\do_character_map\relax
    \etx_to_font{#1}{#2}
}}
```

\etx_to_vpl

```
\def\etx_to_font#1#2{
   \def\do_slot{\resetint\slot_name\slot_number}
   \inputetx{#1}
   \open_out{#2.\vpl_extension}
   \out_line{(\vpl_title\space\vpl_font\space
      #2~created~by~fontinst~v\fontinstversion)}
   \out_line{}
   \out_line{(COMMENT~Filename:~#2.\vpl_extension)}
   \out_line{(COMMENT~Created~by:~tex~\jobname)}
   \out_line{(COMMENT~Created~using:~\vpl_caller{#1}{#2})}
   \out_line{}
   \out_line{(COMMENT~This~file~can~be~turned~into~a~\vpl_font~with)}
   \out_line{(COMMENT~\vpl_to_vf{#2})}
   \out_line{}
   \out_line{(COMMENT~THIS~FILE~CAN~THEN~BE~DELETED.)}
   \out_line{}
   \make_font{#1}
   \out_line{}
   \out_line{(COMMENT~END~OF~FILE~#2.\vpl_extension)}
   \close_out{\vpl_Font}
}
```

`\afmconvert\DIMEN=INTEGER EXPRESSION;` converts a count into a dimen, assuming the count is a number of AFM units. I'll assume that the largest dimension I'll have to deal with is 131pt, to try to minimize rounding errors.

```
\newdimen\scaled_design_size
```

```
\def\afm_convert#1=#2;{
   \eval_expr{#2}
   #1=\scaled_design_size
   \divide#1 by 8
   \multiply #1 by \result
   \divide #1 by \one_thousand
   \multiply#1 by 8
}
```

The commands `\vpl_real\dimen` or `\vpl_int\count` print a dimension or integer in (V)PL syntax.

```
\def\vpl_real#1{R~\expandafter\lose_measure\the#1}
\def\vpl_int#1{D~\the#1}
```

`\make_font{ENCODING}` makes a `.vpl` file.

```
\def\make_font#1{
   \make_header{#1}
   \make_mapfonts{#1}
   \make_fontdimens{#1}
   \make_ligtable{#1}
   \make_characters{#1}
   \make_tidy{#1}
}
```

```
\newcount\boundary_char
\newdimen\side_bearings
\newdimen\curr_bearings
```

```
\def\make_header#1{
   \global\font_count=0
   \setdim{designsize}{10pt}
   \scaled_design_size=\dim{designsize}
   \out_line{(DESIGNSIZE~\vpl_real\scaled_design_size)}
   \out_line{(DESIGNUNITS~\vpl_real\scaled_design_size)}
   \setstr{codingscheme}{UNKNOWN}
   \out_line{(CODINGSCHEME~\str{codingscheme})}
   \ifisint{boundarychar}\then
      \boundary_char=\int{boundarychar}
      \out_line{(BOUNDARYCHAR~\vpl_int\boundary_char)}
   \else
      \boundary_char=-1
   \fi
   \setint{letterspacing}{0}
   \afm_convert\side_bearings=\div{\int{letterspacing}}{2};
   \ifdim\side_bearings=0pt
      \let\do_character_sidebearings=\relax
   \fi
   \out_line{}
}
```

```
\def\make_mapfonts#1{
   \let\saved_scale\vpl_scale
   \let\saved_mapfont\vpl_mapfont
   \let\saved_raw\vpl_raw
   \let\saved_rule\vpl_rule
   \let\saved_special\vpl_special
   \let\saved_warning\vpl_warning
   \let\saved_movert\vpl_movert
   \let\saved_moveup\vpl_moveup
   \let\saved_push\vpl_push
   \let\saved_pop\vpl_pop
   \let\do_slot=\do_mapfont
   \inputetx{#1}
   \out_line{}
}
```

```
\def\make_fontdimens#1{
   \out_line{(FONTDIMEN}
      \ifisint{fontdimen(1)}\then
         \a_dimen=\int{fontdimen(1)}pt
         \divide\a_dimen by \one_thousand
         \out_lline{(PARAMETER~D~1~\vpl_real\a_dimen)}
      \fi
      \a_count=2
      \loop\ifnum\a_count<256
         {\ifisint{fontdimen(\the\a_count)}\then
            \afm_convert\a_dimen=\int{fontdimen(\the\a_count)};
            \out_lline{(PARAMETER~\vpl_int\a_count\space\vpl_real\a_dimen)}
         \fi}
      \advance\a_count by 1 \repeat
   \out_lline{)}
   \out_line{}
}
```

```
\def\make_ligtable#1{
   \bgroup
      \out_line{(LIGTABLE}
      \let\do_slot=\relax
      \let\end_do_slot=\vpl_kerning
      \let\ligature=\vpl_ligature
      \let\k=\vpl_kern
      \inputetx{#1}
      \out_lline{)}
   \egroup
   \out_line{}
}
```

```
\def\make_characters#1{
   \bgroup
      \let\do_slot=\do_character
      \let\end_do_slot=\end_do_character
      \let\nextlarger=\vpl_nextlarger
      \let\varchar=\vpl_varchar
      \let\endvarchar=\end_vpl_varchar
      \let\vartop=\vpl_vartop
      \let\varmid=\vpl_varmid
      \let\varbot=\vpl_varbot
      \let\varrep=\vpl_varrep
      \inputetx{#1}
   \egroup
}
```

```
\def\make_tidy#1{
   \a_count=0\loop\ifnum\a_count<\font_count
      \edef\temp_command{\csname~f-\the\a_count\endcsname}
      \global\x_cs\let\temp_command\relax
      \advance\a_count by 1
   \repeat
   \global\font_count=0
}
```

`\vpl_ligature{TYPE}{NAME}{NAME}` produces a ligtable entry for glyph `\slot_name`.

```
\def\vpl_ligature#1#2#3{
   \ifisint{#2}\then
      \ifisint{#3}\then
         \ifisglyph{#3}\then
            \a_count=\int{#2}
            \b_count=\int{#3}
            \vpl_liglabel
            \out_lline{(#1~\vpl_int\a_count\space \vpl_int\b_count)~
               (COMMENT~#2~#3)}
         \else
            \immediate\write16{Warning:~\string\ligature\space
               for~unknown~slot~‘#3’.}
         \fi
      \else
         \immediate\write16{Warning:~\string\ligature\space
            for~unknown~slot~‘#3’.}
      \fi
```

```
   \else
      \immediate\write16{Warning:~\string\ligature\space
         for~unknown~slot~`#2'.}
   \fi
}
```

`\vpl_kerning` writes out kerning instructions.

```
\def\vpl_kerning{\csname~r-\slot_name\endcsname\vpl_ligstop}
```

`\vpl_kern\l-NAME\AMOUNT` writes out a `KRN` instruction.

```
\def\vpl_kern#1#2{
   \edef\temp_command{\expandafter\gobble_three\string#1}
   \ifisint\temp_command\then
      \vpl_liglabel
      \a_count=\int\temp_command
      \afm_convert\a_dimen=\expandafter\gobble_one\string#2;
      \out_lline{
         (KRN~\vpl_int\a_count\space \vpl_real\a_dimen)~
         (COMMENT~\temp_command)
      }
   \fi
}
```

`\vpl_liglabel` writes out a `LIGLABEL` instruction if appropriate.

```
\def\out_liglabel{
   \out_lline{(LABEL~\vpl_int\slot_number)~(COMMENT~\slot_name)}
   \ifnum\slot_number=\boundary_char
      \out_lline{(LABEL~BOUNDARYCHAR)}
   \fi
   \let\vpl_liglabel=\relax
   \let\vpl_ligstop=\out_ligstop
}
\let\vpl_liglabel=\out_liglabel
```

`\vpl_ligstop` writes out a `LIGSTOP` instruction if appropriate.

```
\def\out_ligstop{\out_lline{(STOP)}
   \let\vpl_liglabel=\out_liglabel
   \let\vpl_ligstop=\relax}
\let\vpl_ligstop=\relax
```

`\do_mapfont` produces a `MAPFONT` entry for each font used by glyph `\slot_name`.

```
\def\do_mapfont{
   \ifisglyph\slot_name\then
      \mapfonts\slot_name
   \fi
}
```

The following commands can be used in a `MAPFONT`:

```
\def\vpl_scale#1#2{{
   \divide \scaled_design_size by 8
   \multiply \scaled_design_size by #1
   \divide \scaled_design_size by \one_thousand
   \multiply \scaled_design_size by 8
   #2
}}
```

```
\def\vpl_mapfont#1#2{
   \a_dimen=#2
   \x_cs\ifx{#1-\the\scaled_design_size}\relax
      \x_cs\xdef{#1-\the\scaled_design_size}{\the\font_count}
      \x_cs\xdef{f-\the\font_count}{#1-\the\scaled_design_size}
      \out_line{(MAPFONT~\vpl_int\font_count\space
         (FONTNAME~#1)~
         (FONTDSIZE~\vpl_real\a_dimen)~
         (FONTAT~\vpl_real\scaled_design_size))}
      \global\advance\font_count by 1
   \fi
}
```

```
\newcount\font_count
\newcount\next_mapfont
\newcount\prev_mapfont
```

`\do_character` produces a character entry for glyph `\slot_name` in slot `\slot_number`. First of all, the version with letterspacing:

```
\def\do_character{
   \x_cs\ifx{g-\slot_name}\relax
      \expandafter\gobble_setslot
   \else
      \ifx\slot_name\notdef_name\else
         \out_line{(CHARACTER~\vpl_int\slot_number\space
            (COMMENT~\slot_name)}
         \afm_convert\a_dimen=\width\slot_name;
         \do_character_sidebearings
         \out_lline{(CHARWD~\vpl_real\a_dimen)}
         \afm_convert\a_dimen=\height\slot_name;
         \out_lline{(CHARHT~\vpl_real\a_dimen)}
         \afm_convert\a_dimen=\depth\slot_name;
         \out_lline{(CHARDP~\vpl_real\a_dimen)}
         \afm_convert\a_dimen=\italic\slot_name;
         \ifnum\a_dimen>0
            \out_lline{(CHARIC~\vpl_real\a_dimen)}
         \fi
         \do_character_map
      \fi
   \fi
}
```

```
\def\do_character_sidebearings{
   \ifisint{\slot_name-spacing}\then
      \afm_convert\curr_bearings=\int{\slot_name-spacing};
      \divide\curr_bearings by 2
   \else
      \curr_bearings=\side_bearings
   \fi
   \afm_convert\a_dimen=\width\slot_name;
   \advance\a_dimen by 2\curr_bearings
}
```

```
\def\do_character_map{
   \global\prev_mapfont=0 \out_lline{(MAP}
      \ifdim \curr_bearings=0pt
```

```
         \mapcommands\slot_name
      \else
         \out_llline{(MOVERIGHT~\vpl_real\curr_bearings)}
         \mapcommands\slot_name
         \out_llline{(MOVERIGHT~\vpl_real\curr_bearings)}
      \fi
   \out_llline{)}
}
```

Now the version without:

```
\def\do_character_no_letterspacing{
   \x_cs\ifx{g-\slot_name}\relax
      \expandafter\gobble_setslot
   \else
      \ifx\slot_name\notdef_name\else
         \out_line{(CHARACTER~\vpl_int\slot_number\space
            (COMMENT~\slot_name)}
         \afm_convert\a_dimen=\width\slot_name;
         \out_lline{(CHARWD~\vpl_real\a_dimen)}
         \afm_convert\a_dimen=\height\slot_name;
         \out_lline{(CHARHT~\vpl_real\a_dimen)}
         \afm_convert\a_dimen=\depth\slot_name;
         \out_lline{(CHARDP~\vpl_real\a_dimen)}
         \afm_convert\a_dimen=\italic\slot_name;
         \ifnum\a_dimen>0 \out_lline{(CHARIC~\vpl_real\a_dimen)} \fi
         \global\prev_mapfont=0 \out_lline{(MAP}
            \mapcommands\slot_name
         \out_llline{)}
      \fi
   \fi
}
\def\gobble_setslot#1\endsetslot{\endsetslot}
```

```
\def\end_do_character{
   \ifisglyph\slot_name\then
      \out_lline{)}
   \fi
}
```

```
\def\notdef_name{.notdef}
```

`\vpl_nextlarger{NAME}` produces a `NEXTLARGER` entry.

```
\def\vpl_nextlarger#1{
   \ifisint{#1}\then
      \out_lline{(NEXTLARGER~D~\the\int{#1})~(COMMENT~#1)}
   \else
      \immediate\write16{Warning:~\string\nextlarger\space
         for~unknown~slot~`#1'}
   \fi
}
```

`\vpl_varchar VARCHAR COMMANDS \end_vpl_varchar` produces a `VARCHAR` entry.

```
\def\vpl_varchar{\out_lline{(VARCHAR}}
```

```
\def\vpl_vartop#1{
   \ifisint{#1}\then
      \out_lline{(TOP~D~\the\int{#1})~(COMMENT~#1)}
   \else
      \immediate\write16{Warning:~\string\vartop\space
         for~unknown~slot~‘#1’}
   \fi
}
\def\vpl_varmid#1{
   \ifisint{#1}\then
      \out_lline{(MID~D~\the\int{#1})~(COMMENT~#1)}
   \else
      \immediate\write16{Warning:~\string\varmid\space
         for~unknown~slot~‘#1’}
   \fi
}
\def\vpl_varbot#1{
   \ifisint{#1}\then
      \out_lline{(BOT~D~\the\int{#1})~(COMMENT~#1)}
   \else
      \immediate\write16{Warning:~\string\varbot\space
         for~unknown~slot~‘#1’}
   \fi
}
\def\vpl_varrep#1{
   \ifisint{#1}\then
      \out_lline{(REP~D~\the\int{#1})~(COMMENT~#1)}
   \else
      \immediate\write16{Warning:~\string\varrep\space
         for~unknown~slot~‘#1’}
   \fi
}
```

```
\def\end_vpl_varchar{\out_llline{)}}
```

The following commands can be used in a glyph:

```
\def\vpl_raw#1#2#3{
   \global\next_mapfont=\csname~#1-\the\scaled_design_size\endcsname\relax
   \ifnum\next_mapfont=\prev_mapfont\else
      \out_lline{(SELECTFONT~\vpl_int\next_mapfont)~
         (COMMENT~#1~at~\the\scaled_design_size)}
   \fi
   \out_lline{(SETCHAR~D~#2)~(COMMENT~#3)}
   \global\prev_mapfont=\next_mapfont
}
```

```
\def\vpl_rule#1#2{
   \afm_convert\a_dimen=#1;
   \afm_convert\b_dimen=#2;
   \out_lline{(SETRULE~\vpl_real\b_dimen\space~\vpl_real\a_dimen)}
}
```

```
\def\vpl_special#1{
   \out_lline{(SPECIAL~#1)}}
\def\vpl_warning#1{
   \out_lline{(SPECIAL~Warning:~#1)}
```

```
    \immediate\write16{Warning:~#1.}
}
```

```
\def\vpl_movert#1{
    \afm_convert\a_dimen=#1;
    \out_llline{(MOVERIGHT~\vpl_real\a_dimen)}
}
\def\vpl_moveup#1{
    \afm_convert\a_dimen=#1;
    \out_llline{(MOVEUP~\vpl_real\a_dimen)}
}
```

```
\def\vpl_push{\out_llline{(PUSH)}}
\def\vpl_pop{\out_llline{(POP)}}
```

11.2 Converting an MTX file to a PL file

\mtxtopl The macro:

```
\mtxtopl{MTXFILE}{PLFILE}
```

writes a font from the `\setrawglyph` instructions in `MTXFILE` to `PLFILE`. It ignores any font dimensions and kerning, so the resulting font is only useful for generating virtual fonts from. (This macro is called by `\transformfont`.)

\mtxtopl

```
\def\mtxtopl#1#2{{
    \open_out{#2.pl}
    \out_line{(COMMENT~raw~font~#2~created~by~fontinst~v\fontinstversion)}
    \out_line{}
    \out_line{(COMMENT~Filename:~#2.pl)}
    \out_line{(COMMENT~Created~by:~tex~\jobname)}
    \out_line{(COMMENT~Created~using:~\string\mtxtopl{#1}{#2})}
    \out_line{}
    \out_line{(COMMENT~This~file~can~be~turned~into~a~TeX~font~with)}
    \out_line{(COMMENT~pltotfm~#2.pl~#2.tfm)}
    \out_line{}
    \out_line{(COMMENT~THIS~FILE~CAN~THEN~BE~DELETED.)}
    \out_line{}
    \out_line{(DESIGNSIZE~R~10.0)}
    \out_line{(DESIGNUNITS~R~10.0)}
    \out_line{}
    \scaled_design_size=10pt
    \let\setglyph=\iffalse
    \let\endsetglyph=\fi
    \let\setkern=\gobble_three
    \let\setrawglyph=\pl_raw_glyph
    \inputmtx{#1}
    \out_line{}
    \out_line{(COMMENT~END~OF~FILE~#2.pl)}
    \close_out{Raw~font}
}}
```

```
\def\pl_raw_glyph#1#2#3#4#5#6#7#8{
    \a_count=#4
```

```
    \afm_convert\a_dimen=#5;
    \afm_convert\b_dimen=#6;
    \afm_convert\c_dimen=#7;
    \afm_convert\d_dimen=#8;
    \out_line{(CHARACTER~\vpl_int\a_count\space(COMMENT~#1)}
    \out_lline{(CHARWD~\vpl_real\a_dimen)}
    \out_lline{(CHARHT~\vpl_real\b_dimen)}
    \out_lline{(CHARDP~\vpl_real\c_dimen)}
    \out_lline{(CHARIC~\vpl_real\d_dimen)}
    \out_lline{()}
}
```

12 Font installation commands and `.fd` files

`\installfonts`
`\endinstallfonts`

The macro:

```
\installfonts INSTALL COMMANDS \endinstallfonts
```

is the top-level interface for installing a number of fonts and creating `.fd` files for them.

`\installfamily`
`\installfont`
`\installrawfont`

The `INSTALL COMMANDS` are:

```
\installfamily{ENCODING}{FAMILY}{FD-COMMANDS}
\installfont{FONT-NAME}{FILE-LIST}{ETX}
  {ENCODING}{FAMILY}{SERIES}{SHAPE}{SIZE}
\installrawfont{FONT-NAME}{FILE-LIST}{ETX}
  {ENCODING}{FAMILY}{SERIES}{SHAPE}{SIZE}
```

Each `\installfamily` command causes the generation of an `.fd` file for `ENCODING` and `FAMILY`, which is writen out by the time `\endinstallfonts` is processed.

Each `\installfont` generates a `.vpl` font by calling `\etxtovpl` and adds an `.fd` entry. Each `\installrawfont` generates a ligfull `.pl` font by calling `\etxtopl` and adds an `.fd` entry. (Raw `.pl` fonts, containing only the glyph metrics without any ligaturing or kerning information, are also generated by `\mtxtopl` called from `\transformfont` statements.)

`\installfonts`

Initializes the `\family_toks` token register, which is used to store the information which is written out to `.fd` files at the end of the job.

```
\newtoks\family_toks
\def\installfonts{
    \bgroup
    \global\family_toks={}
    \gdef\prev_file_list{}
}
```

`\installfamily`

`\installfamily{ENCODING}{FAMILY}{FD-COMMANDS}`

Adds the command `\fd_family{ENCODING}{FAMILY}{FD-COMANNDS}` to the token list `\family_toks` and defines a macro `\ENCODING-FAMILY`.

```
\def\installfamily#1#2#3{
    \global\family_toks=\expandafter{\the\family_toks\fd_family{#1}{#2}{#3}}
    \global\x_cs\let{#1-#2}\empty_command
}
```

\installfont
\installrawfont

```
\installfont{FONT-NAME}{FILE-LIST}{ETX}
  {ENCODING}{FAMILY}{SERIES}{SHAPE}{SIZE}
\installrawfont{FONT-NAME}{FILE-LIST}{ETX}
  {ENCODING}{FAMILY}{SERIES}{SHAPE}{SIZE}
```

```
\def\installfont#1#2#3#4#5#6#7#8{
   \install_font{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{\etxtovpl}
}
\def\installrawfont#1#2#3#4#5#6#7#8{
   \install_font{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{\etxtopl}
}
```

\install_font

```
\install_font{FONT-NAME}{FILE-LIST}{ETX}
  {ENCODING}{FAMILY}{SERIES}{SHAPE}{SIZE}{CONVERTER}
```

Installs a font shape by writing out a (V)`.pl` font using the given converter function and adding an entry to the `.fd` file.

```
\def\install_font#1#2#3#4#5#6#7#8#9{
   \xdef\curr_file_list{#2}
   \ifx\prev_file_list\curr_file_list\else
      \egroup\bgroup
      \expandafter\input_mtx_files\curr_file_list,
         \relax,\end_input_mtx_files
      \global\let\prev_file_list=\curr_file_list
   \fi
   #9{#3}{#1}
   \x_cs\ifx{#4-#5-#6-#7}\relax
      \let\do_shape=\relax
      \x_cs\xdef{#4-#5}{
         \x_cs\ifx{#4-#5}\relax\else
            \csname#4-#5\endcsname
         \fi
         \do_shape{#4}{#5}{#6}{#7}
      }
   \fi
   \let\do_size=\relax
   \x_cs\xdef{#4-#5-#6-#7}{
      \x_cs\ifx{#4-#5-#6-#7}\relax\else
         \csname#4-#5-#6-#7\endcsname
      \fi
      \do_size{#8}{#1}
   }
}
```

\endinstallfonts

Finish the installation by processing the `\family_toks` token register, which contains the accumulated information to be written out to `.fd` files.

```
\def\endinstallfonts{
   \let\do_shape=\fd_shape
   \let\do_size=\fd_size
   \the\family_toks
   \global\family_toks{}
   \egroup
}
```

\input_mtx_files

```
\input_mtx_files{FILE-LIST}
```

Process a comma-separated list of .mtx files and call \imput_mtx_file for each file.

```
\def\input_mtx_files#1,#2\end_input_mtx_files{
   \if\relax#1
      \expandafter\gobble_one
   \else
      \expandafter\identity_one
   \fi{
      \input_mtx_file{#1}
      \input_mtx_files#2\end_input_mtx_files
   }
}
```

\input_mtx_file \input_mtx_file{FILE}

Process an .mtx file specified in the FILE-LIST argument of \installfont or \installrawfont. If an .mtx file doesn't exists, it is generated on the fly from a corresponding .pl or .afm file with \pltomtx or \afmtomtx. (.mtx files generated from .afm files are also written out as raw .pl files with \mtxtopl.)

```
\def\input_mtx_file#1{
   \if\relax#1\relax\else
      \get_file_name #1~scaled~1000~;
      \if_file_exists{\file_name.mtx}\then
         \inputmtx\file_name
      \else
         % changed the order (SPQR, 15/01/97) because of cmr*.afm
         % files found in TEXMF/fonts/afm hierarchy
         \if_file_exists{\file_name.pl}\then
               \pltomtx\file_name\file_name
         \else
            \if_file_exists{\file_name.afm}\then
               \afmtomtx\file_name\file_name
               \mtxtopl\file_name\file_name
            \fi
         \fi
         \inputmtx\file_name
      \fi
   \fi
}
```

\get_file_name FILENAME scaled SCALE JUNK ;

```
\def\get_file_name #1~scaled~#2~#3;{
   \edef\file_name{#1}
   \ifnum#2=\one_thousand
      \unsetint{rawscale}
   \else
      \setint{rawscale}{#2}
   \fi
}
```

\fd_family \fd_family{ENCODING}{FAMILY}{FD-COMMANDS}

Writes out an .fd file for the specified ENCODING and FAMILY processing the accumulated information and default substitutions.

```
\def\fd_family#1#2#3{
   \a_toks{#3}
   \edef\lowercase_file{\lowercase{
     \edef\noexpand\lowercase_file{#1#2.fd}}}
   \lowercase_file
   \open_out{\lowercase_file}
   \out_line{\percent_char~Filename:~\lowercase_file}
   \out_line{\percent_char~Created~by:~tex~\jobname}
   \out_line{\percent_char~Created~using~fontinst~v\fontinstversion}
   \out_line{}
   \out_line{\percent_char~THIS~FILE~SHOULD~BE~PUT~IN~A~TEX~INPUTS~DIRECTORY}
   \out_line{}
   \out_line{\string\ProvidesFile{\lowercase_file}}
   \out_lline{[
      \the\year/
      \ifnum10>\month0\fi\the\month/
      \ifnum10>\day0\fi\the\day\space
      Fontinst~v\fontinstversion\space
      font~definitions~for~#1/#2.
   ]}
   \out_line{}
   \out_line{\string\DeclareFontFamily{#1}{#2}{\the\a_toks}}
   {
      \csname #1-#2\endcsname
      \out_line{}
      \let\do_shape=\substitute_shape
      \csname #1-#2\endcsname
      \let\do_shape=\remove_shape
      \csname #1-#2\endcsname
   }
   \x_cs\g_let{#1-#2}\relax
   \out_line{}
   \out_line{\string\endinput}
   \close_out{Font~definitions}
}
```

\fd_shape \fd_shape{ENCODING}{FAMILY}{SERIES}{SHAPE}

```
\def\fd_shape#1#2#3#4{
   \out_line{}
   \out_line{\string\DeclareFontShape{#1}{#2}{#3}{#4}\left_brace_char}
   \csname #1-#2-#3-#4\endcsname
   \x_cs\g_let{#1-#2-#3-#4}\empty_command
   \out_line{\right_brace_char{}}
}
```

\fd_size \fd_size{SIZE}{FONT-NAME}

```
\def\fd_size#1#2{
   \x_cs\ifx{siz-#1}\relax
      \out_lline{#1~#2}
   \else
      \out_lline{\csname siz-#1\endcsname\space #2}
   \fi
}
```

\remove_shape \remove_shape{ENCODING}{FAMILY}{SERIES}{SHAPE}

```
\def\remove_shape#1#2#3#4{
   \x_cs\g_let{#1-#2-#3-#4}\relax
}
```

\substitute_shape \substitute_shape{ENCODING}{FAMILY}{SERIES}{SHAPE}

```
\def\substitute_shape#1#2#3#4{
   \edef\orig_shape{#4}
   \substitute_series{#1}{#2}{#3}{\orig_shape}
   \x_cs\ifx{sub-\orig_shape}\relax\else
      \edef\subst_shape{\csname sub-\orig_shape\endcsname}
      \x_cs\ifx{#1-#2-#3-\subst_shape}\relax
         \out_line{
            \string\DeclareFontShape{#1}{#2}{#3}{\subst_shape}{
               <->\csname typ-\orig_shape\endcsname\space *~#2/#3/\orig_shape
            }{}
         }
         \x_cs\let{#1-#2-#3-\subst_shape}\empty_command
         \substitute_shape{#1}{#2}{#3}{\subst_shape}
      \fi
   \fi
}
```

\substitute_series \substitute_series{ENCODING}{FAMILY}{SERIES}{SHAPE}

```
\def\substitute_series#1#2#3#4{
   \edef\orig_series{#3}
   \x_cs\ifx{sub-\orig_series}\relax\else
      \edef\subst_series{\csname sub-\orig_series\endcsname}
      \x_cs\ifx{#1-#2-\subst_series-#4}\relax
         \out_line{
            \string\DeclareFontShape{#1}{#2}{\subst_series}{#4}{
               <->\csname typ-\orig_series\endcsname\space *~#2/\orig_series/#4
            }{}
         }
         \x_cs\let{#1-#2-\subst_series-#4}\empty_command
         \substitute_series{#1}{#2}{\subst_series}{#4}
      \fi
   \fi
}
```

\substitutesilent \substitutesilent{TO}{FROM}
\substitutenoisy \substitutenoisy{TO}{FROM}

Specifies a default substitution for family or shape `TO`, which is substituted by family or shape `FROM`.

```
\def\substitutesilent#1#2{
   \x_cs\def{sub-#2}{#1}
   \x_cs\def{typ-#2}{ssub}
}
\def\substitutenoisy#1#2{
   \x_cs\def{sub-#2}{#1}
   \x_cs\def{typ-#2}{sub}
}
```

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

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

I don't think we want these since in OT1 encoding it will cause the old `<sterling>` vs `<dollar>` problem. (ASAJ) — Oh yes we do. (SPQR)

```
\substitutesilent{sl}{it}
\substitutesilent{it}{sl}
```

12.1 Default encodings and font sizes

\declareencoding `\declareencoding{CODINGSCHEME}{ENCODING}`

Declare a macro `\enc-CODINGSCHEME` which expands to `ENCODING`. This is used to determine the encoding in `\pltomtx`.

```
\def\declareencoding#1#2{\x_cs\edef{enc-#1}{#2}}
```

Old TEX text font encodings.

```
\declareencoding{TEX~TEXT}{OT1}
\declareencoding{TEX~TEXT~WITHOUT~F-LIGATURES}{OT1}
\declareencoding{TEX~TYPEWRITER~TEXT}{OT1TT}
```

Old TEX math font encodings.

```
\declareencoding{TEX~MATH~ITALIC}{OML}
\declareencoding{TEX~MATH~SYMBOLS}{OMS}
\declareencoding{TEX~MATH~EXTENSION}{OMX}
\declareencoding{LATEX~SYMBOLS}{lasy}
```

Euler math font encodings.

```
\declareencoding{TEX~TEXT~SUBSET}{eufrak}
\declareencoding{TEX~MATH~ITALIC~SUBSET}{eurm}
\declareencoding{TEX~MATH~SYMBOLS~SUBSET}{euscr}
\declareencoding{EULER~SUBSTITUTIONS~ONLY}{euex}
```

New TEX text font encodings.

```
\declareencoding{EXTENDED~TEX~FONT~ENCODING~-~LATIN}{T1}
\declareencoding{TEX~TEXT~COMPANION~SYMBOLS~1---TS1}{TS1}
```

Rencoded PostScript font encdings.

```
\declareencoding{TEXBASE1ENCODING}{8r}
\declareencoding{TEX~TYPEWRITER~AND~WINDOWS~ANSI}{8y}
```

\declaresize `\declaresize{FONTSIZE}{LATEXSIZE}`

```
\def\declaresize#1#2{\x_cs\edef{siz-#1}{#2}}
```

Default sizes. An empty size argument is equivalent to `<->`, for use with scalable fonts.

```
\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>}
```

13 Font transformations

\transformfont

The macro:

```
\transformfont{FONT-NAME}{TRANSFORMED FONT}
```

transforms a raw font.

\fromafm \frommtx \frompl \scalefont \xscalefont \yscalefont \slantfont \reencodefont

The `TRANSFORMED FONT` commands are:

```
\fromafm{AFMFILE}
\frompl{PLFILE}
\frommtx{MTXFILE}
\scalefont{INTEGER EXPRESSION}{TRANSFORMED FONT}
\xscalefont{INTEGER EXPRESSION}{TRANSFORMED FONT}
\yscalefont{INTEGER EXPRESSION}{TRANSFORMED FONT}
\slantfont{INTEGER EXPRESSION}{TRANSFORMED FONT}
\reencodefont{ETX}{TRANSFORMED FONT}
```

Each `\transformfont` command generates an `.mtx` file for `FONT-NAME` and a corresponding raw `.pl` file, which is written out by `\mtxtopl`.

Each `\fromafm` or `\frommtx` command also generates an `.mtx` file, which is written out by `\afmtomtx` or `\pltomtx`. In addition, `\fromafm` also generates a corresponding raw `.pl` file by `\mtxtopl`.

\transformfont

```
\def\transformfont#1#2{{
   \unsetstr{afm-name}
   \unsetstr{etx-name}
   \resetint{x-scale}{\one_thousand}
   \resetint{y-scale}{\one_thousand}
   \resetint{slant-scale}{0}
   #2
   \mtxtomtx{\str{afm-name}}{#1}
   \mtxtopl{#1}{#1}
}}
```

\fromafm \frompl \frommtx

```
\def\fromafm#1{
   \setstr{afm-name}{#1}
   \afmtomtx{#1}{#1}
   \mtxtopl{#1}{#1}
}
```

```
\def\frompl#1{
   \setstr{afm-name}{#1}
   \pltomtx{#1}{#1}
}
\def\frommtx#1{
   \setstr{afm-name}{#1}
}
```

\scalefont
\xscalefont
\yscalefont
\slantfont
\reencodefont

```
\def\scalefont#1#2{
   \resetint{x-scale}{#1}
   \resetint{y-scale}{\int{x-scale}}
   #2
}
\def\xscalefont#1#2{
   \resetint{x-scale}{#1}
   #2
}
\def\yscalefont#1#2{
   \resetint{y-scale}{#1}
   #2
}
\def\slantfont#1#2{
   \resetint{slant-scale}{#1}
   #2
}
\def\reencodefont#1#2{
   \resetstr{etx-name}{#1}
   #2
}
```

\mtxtomtx The macro:

```
\mtxtomtx{MTX}{MTX}
```

converts the first `.mtx` file to the second, using the current values of `\int{x-scale}`, `\int{y-scale}`, `\int{slant-scale}` and `\str{etx-name}`.

NOTE: this doesn't convert arbitrary `.mtx` files, just ones created with `\afmtomtx` or `\pltomtx`.

\mtxtomtx

```
\def\mtxtomtx#1#2{{
   \ifisstr{etx-name}\then
      \def\do_slot{\resetint\slot_name\slot_number}
      \inputetx{\str{etx-name}}
   \fi
   \open_out{\temp_prefix#2.mtx}
   \def\raw_font_name{#2}
   \out_line{\percent_char~Filename:~#2.mtx}
   \out_line{\percent_char~Created~by:~tex~\jobname}
   \out_line{\percent_char~Created~using:~\string\mtxtomtx{#1}{#2}}
   \out_line{}
   \out_line{\percent_char~This~file~is~used~by~the~fontinst~package.}
   \out_line{}
```

```
   \out_line{\percent_char~THIS~FILE~CAN~BE~DELETED.}
   \out_line{}
   \out_line{\string\relax}
   \out_line{\string\metrics}
   \out_line{}
   \out_line{\string\needsfontinstversion{\fontinstversion}}
   \out_line{}
   \let\setint=\mtxtomtx_setint
   \let\setrawglyph=\mtxtomtx_setrawglyph
   \let\setnotglyph=\mtxtomtx_setrawglyph
   \let\setkern=\mtxtomtx_setkern
   \let\setglyph=\mtxtomtx_setglyph
   \let\glyphpcc=\mtxtomtx_glyphpcc
   \let\samesize=\mtxtomtx_samesize
   \let\endsetglyph=\mtxtomtx_endsetglyph
   \inputmtx{#1}
   \out_line{}
   \out_line{\string\endmetrics}
   \close_out{Transformed~metrics}
}}

\def\mtxtomtx_setint#1#2{
   \def\temp_command{#1}
   \ifx\temp_command\italicslant_name
      \eval_expr_to\a_count{
         \add{\int{slant-scale}}
         {\div{\mul{\int{x-scale}}{#2}}{\int{y-scale}}}
      }
   \else
      \eval_expr_to\a_count{\scale{#2}{\int{y-scale}}}
   \fi
   \out_line{\string\setint{#1}{\the\a_count}}
}
\def\italicslant_name{italicslant}

\def\mtxtomtx_setrawglyph#1#2#3#4#5#6#7#8{
   \eval_expr_to\a_count{\scale{#5}{\int{x-scale}}}
   \eval_expr_to\b_count{\scale{#6}{\int{y-scale}}}
   \eval_expr_to\c_count{\scale{#7}{\int{y-scale}}}
   \eval_expr_to\d_count{
      \add{\scale{#8}{\int{x-scale}}}
      {\scale\b_count{\int{slant-scale}}}
   }
   \ifisstr{etx-name}\then
      \ifisint{#1}\then
         \e_count=\int{#1}
      \else
         \e_count=-1
      \fi
   \else
      \e_count=#4
   \fi
   \out_line{
      \ifnum\e_count>-1
         \string\setrawglyph
```

```
      \else
         \string\setnotglyph
      \fi
         {#1}{\raw_font_name}{#3}{\the\e_count}
         {\the\a_count}{\the\b_count}{\the\c_count}{\the\d_count}
   }
}
```

```
\def\mtxtomtx_setkern#1#2#3{
   \eval_expr{\scale{\int{x-scale}}{#3}}
   \out_line{\string\setkern{#1}{#2}{\the\result}}
}
```

```
\def\mtxtomtx_setglyph#1{\out_line{\string\setglyph{#1}}}
\def\mtxtomtx_samesize#1{\out_lline{\string\samesize{#1}}}
\def\mtxtomtx_endsetglyph{\out_line{\string\endsetglyph}}
```

```
\def\mtxtomtx_glyphpcc#1#2#3{
   \eval_expr_to\b_count{\scale{\int{y-scale}}{#3}}
   \eval_expr_to\a_count{
      \add{\scale{\int{x-scale}}{#2}}
      {\scale\b_count{\int{slant-scale}}}
   }
   \out_lline{\string\glyphpcc{#1}{\the\a_count}{\the\b_count}}
}
```

14 Installing Latin families

`\latinfamily`

The macro:

```
\latinfamily{FAMILY}{COMMANDS}
```

installs a Latin font family.

```
\def\latinfamily#1#2{{
   \edef\temp_command{#1}
   \expandafter\parse_family\temp_command
      \empty_command\empty_command\end_parse_family
   \installfonts
      \if_file_exists{\raw_encoding.mtx}\then
         \installfamily{\raw_encoding}{\latex_family}{#2}
      \fi
      \installfamily{OT1}{\latex_family}{#2}
      \installfamily{T1}{\latex_family}{#2}
⟨*textcomp⟩
      \installfamily{TS1}{\latex_family}{#2}
⟨/textcomp⟩
      \latin_weights
   \endinstallfonts
}}
```

`\parse_family` `\font_family` `\font_variant` `\raw_variant` `\latex_variant`

`\parse_family FAMILY\end_parse_family`

Intializes `\latex_family` to `FAMILY`, `\font_family` to first three characters of `FAMILY`, `\font_variant` and `\raw_variant` to fourth and fifth character (if present).

SPQR 02/95: If the fourth parameter is `x` or `9`, then its an expert indication, not a variant. So initialy set `\raw_variant` to `#4`, but unset it if the expert test succeeds. When `#4` is used for a real variant, it passes through.

SPQR 05/95: If it *is* an expert font, then the variantness is expressed by the encoding, not a variant letter, so unset `\font_variant`.

UV 06/98: Added `\if_oldstyle_` code from Alan's v 1.511 beta. If the fourth parameter is `9`, treat it as if `j` was given.

Note that instead of using `\setcommand\digit`, we now use separate encoding files for `T1j.etx`, etc.

```
\def\parse_family #1#2#3#4#5\end_parse_family{
   \gdef\font_family{#1#2#3}
   \gdef\font_variant{#4#5}
   \gdef\raw_variant{#4#5}
   \gdef\latex_family{#1#2#3#4#5}
   \_expert_false
   \_oldstyle_false
   \ifx#4x
      \_expert_true
      \gdef\raw_variant{#5}
      \gdef\font_variant{#5}
      %\setcommand\digit##1{##1}
   \else\ifx#4j
      \_expert_true
      \_oldstyle_true
      \gdef\raw_variant{#5}
      \gdef\font_variant{#5}
      %\setcommand\digit##1{##1oldstyle}
   \else\ifx#49
      \gdef\latex_family{#1#2#3j#5}
      \_expert_true
      \_oldstyle_true
      \gdef\raw_variant{#5}
      \gdef\font_variant{#5}
      %\setcommand\digit##1{##1oldstyle}
   \else
      \_expert_false
      \_oldstyle_false
      %\setcommand\digit##1{##1}
   \fi\fi\fi
⟨*debug⟩
   \if_oldstyle_
   \immediate\write16{INFO>~parse~family~<#1#2#3><#5>~(oldstyle)}
   \else\if_expert_
   \immediate\write16{INFO>~parse~family~<#1#2#3><#5>~(expert)}
   \else
   \immediate\write16{INFO>~parse~family~<#1#2#3><#4#5>}
   \fi\fi
⟨/debug⟩
}
```

`\if_expert_`
`\if_oldstyle_`

```
\newif\if_expert_
\newif\if_oldstyle_
```

14.1 Processing a list of weights, widths and shapes

\latin_weight \font_weight \latex_weight

`\latin_weight{FONTWEIGHT}{LATEXWEIGHT}`

```
\def\latin_weight#1#2{
   \gdef\font_weight{#1}%
   \gdef\latex_weight{#2}%
   \latin_widths
}
```

\latin_width \font_width \latex_width

`\latin_width{FONTWIDTH}{LATEXWIDTH}`

```
\def\latin_width#1#2{
   \gdef\font_width{#1}
   \gdef\latex_width{#2}
   \latin_shapes
}
```

\latin_shape \font_shape \raw_shape \encoding_shape \latex_shape \encoding_switches

`\latin_shape{FONTSHAPE}{RAWSHAPE}{ENCODINGSHAPE}{LATEXSHAPE}{SWITCHES}`

Calls `\fake_width_WIDTH` and `\fake_shape_SHAPE` to generate an `8r`-reencoded `.mtx` file. If successful, calls `\latin_encodings`.

UV 06/98: Added a fifth argument `SWITCHES`, which may be used to set `\if_textcomp_` before it is evaluated in `\latin_encodings`.

```
\def\latin_shape#1#2#3#4#5{
   \gdef\font_shape{#1}
   \gdef\raw_shape{#2}
   \gdef\encoding_shape{#3}
   \if_oldstyle_
      \gdef\encoding_shape{#3\oldstyle_shape}
   \fi
   \gdef\latex_shape{#4}
   \gdef\encoding_switches{#5}
⟨*debug⟩
   \immediate\write16{
      INFO>~to~make~LaTeX~font~shape~<\latex_family,
      \latex_weight,\latex_shape,\latex_width>~seek~
      \font_family\font_weight\raw_shape
         \raw_variant\raw_encoding\font_width.mtx}
⟨/debug⟩
   \csname fake_width_\font_width\endcsname
   \csname fake_shape_\raw_shape\endcsname
   \if_file_exists{
      \font_family\font_weight\raw_shape
         \raw_variant\raw_encoding\font_width.mtx
   }\then
      \latin_encodings
   \fi
}
```

14.2 Faking font shapes

(CK) changed font faking code; based on SPQR's code (still in this file). I am not sure that know what I am doing here; let's hope I won't be caught. Also changed: Order of `\font_width` and `\font_encoding` in MANY places. Also added tests which prevent multiple calls to the same font faking routine. (Seems to be necessary in some cases and can't hurt.) — (ASAJ) I think CK knows more about what he's doing than he's letting on.

`\fake_shape_` Check if an `8a`-encoded `.afm` file exists for current combination of font attributes, and, if so, call `\fake_shape_raw_encoding`. For expert fonts, also check if an `8x`-encoded `.afm` file exists, and call `\fake_shape_expert_encoding`, if approporiate.

UV, 06/98: Cut down lengthy `\fake_shape_` routine into several subroutines `\fake_shape_raw_encoding` and optionally `\fake_shape_expert_encoding` and `\fake_shape_oldstyle_encoding`.

UV, 06/98: Added experimental code to install OsF or SC&OsF fonts.

```
\def\fake_shape_{
   \if_file_exists{
      \font_family\font_weight\raw_shape\raw_variant
         \adobe_encoding\font_width.afm
   }\then
      \fake_shape_raw_encoding
   \fi
   \if_expert_
      \if_file_exists{
         \font_family\font_weight\raw_shape\raw_variant
            \expert_encoding\font_width.afm
      }\then
         \fake_shape_expert_encoding
      \fi
   \fi
⟨*oldstyle⟩
   \if_oldstyle_
      \gdef\oldstyle_variant{\oldstyle_shape}
      \if_file_exists{
         \font_family\font_weight\raw_shape\oldstyle_variant
            \adobe_encoding\font_width.afm
      }\then
         \fake_shape_oldstyle_encoding
      \else
         \gdef\oldstyle_variant{\caps_shape}
         \if_file_exists{
            \font_family\font_weight\raw_shape\oldstyle_variant
               \adobe_encoding\font_width.afm
         }\then
            \fake_shape_oldstyle_encoding
         \fi
      \fi
   \fi
⟨/oldstyle⟩
}
```

`\fake_shape_raw_encoding` Called from `\fake_shape_` if an `8a`-encoded `.afm` file was found. Invokes

```
\transformfont{<FONT>8r}
  {\reencodefont{8r}{\fromafm{<FONT>8a}}}
```

to generate an `8r`-reencoded `.mtx` file (and a raw `.pl` file), followed by

```
\installrawfont{<FONT>8r}{<FONT>8r,8r}{8r}...
```

to install a ligfull `.pl` file.

```
\def\fake_shape_raw_encoding{
   \if_file_exists{
      \font_family\font_weight\raw_shape\raw_variant
         \raw_encoding\font_width.mtx
   }\then % no action required
   \else
⟨*debug⟩
      \immediate\write16{
         INFO>~run~\string\transformfont\space\space
         <\font_family\font_weight\raw_shape\raw_variant
            \raw_encoding\font_width>~from~
         <\font_family\font_weight\raw_shape\raw_variant
            \adobe_encoding\font_width>
      }
⟨/debug⟩
      \transformfont{
         \font_family\font_weight\raw_shape\raw_variant
            \raw_encoding\font_width
      }{
         \reencodefont{\raw_encoding}{
            \fromafm{
               \font_family\font_weight\raw_shape\raw_variant
                  \adobe_encoding\font_width
            }
         }
      }
      \if_file_exists{ \raw_encoding.mtx }\then
⟨*debug⟩
         \immediate\write16{
            INFO>~run~\string\installrawfont\space
            <\font_family\font_weight\font_shape\raw_variant
               \raw_encoding\font_width>
            <\font_family\font_weight\raw_shape\raw_variant
               \raw_encoding\font_width,
             \raw_encoding>
            <\raw_encoding>
            <\raw_encoding>
            <\latex_family>
            <\latex_weight\latex_width>
            <\latex_shape>}
⟨/debug⟩
         \installrawfont{
            \font_family\font_weight\raw_shape\raw_variant
               \raw_encoding\font_width
         }{ \font_family\font_weight\raw_shape\raw_variant
               \raw_encoding\font_width,
            \raw_encoding
         }{ \raw_encoding
         }{ \raw_encoding
         }{ \latex_family
         }{ \latex_weight\latex_width
         }{ \latex_shape
         }{}
```

```
      \fi
   \fi
}
```

\fake_shape_expert_encoding

Called from \fake_shape_ if an 8x-encoded .afm file was found. Invokes

```
\fromafm{<FONT>8x}
```

to generate an 8x-encoded .mtx file (and a raw .pl file).

```
\def\fake_shape_expert_encoding{
   \if_file_exists{
      \font_family\font_weight\raw_shape\raw_variant
         \expert_encoding\font_width.mtx
   }\then % no action required
   \else
      \fromafm{
         \font_family\font_weight\raw_shape\raw_variant
            \expert_encoding\font_width
      }
   \fi
}
```

\fake_shape_oldstyle_encoding

Called from \fake_shape_ if an OsF or SC&OsF variant of an 8a-encoded .afm file was found. Invokes

```
\transformfont{<FONT>j8r}
  {\reencodefont{8r}{\fromafm{<FONT>j8a}}}
```

to generate an 8r-reencoded .mtx file (and a raw .pl file).

```
⟨*oldstyle⟩
\def\fake_shape_oldstyle_encoding{
   \if_file_exists{
      \font_family\font_weight\raw_shape\oldstyle_variant
         \adobe_encoding\font_width.mtx
   }\then % no action required
   \else
⟨*debug⟩
      \immediate\write16{
         INFO>~run~\string\transformfont\space\space
         <\font_family\font_weight\raw_shape\oldstyle_variant
            \raw_encoding\font_width>~from~
         <\font_family\font_weight\raw_shape\oldstyle_variant
            \adobe_encoding\font_width>
      }
⟨/debug⟩
      \transformfont{
         \font_family\font_weight\raw_shape\oldstyle_variant
            \raw_encoding\font_width
      }{
         \reencodefont{\raw_encoding}{
            \fromafm{
               \font_family\font_weight\raw_shape\oldstyle_variant
                  \adobe_encoding\font_width
            }
```

```
          }
       }
    \fi
}
⟨/oldstyle⟩
```

`\fake_shape_c` Check if an `8a`-encoded `.afm` file exists for the small caps shape. If so, call `\fake_shape_` to generate an `8r`-reencoded MTX file and to install a ligfull `.pl` file. If not, reset `\raw_shape` to the default shape and `\encoding_shape` to small caps before calling `\fake_shape_` to install a faked small caps font.

```
\def\fake_shape_c{
   \if_file_exists{
      \font_family\font_weight\raw_shape\raw_variant
         \adobe_encoding\font_width.afm
   }\then
      \fake_shape_
   \else
      % If real smallcaps font doesn't exist, fake it from the roman.
      \gdef\raw_shape{}
      \if_oldstyle_
         \gdef\encoding_shape{cj}
      \else
         \gdef\encoding_shape{c}
      \fi
      \fake_shape_
   \fi
}
```

`\fake_shape_o` Check if an `8a`-encoded `.afm` file exists for the oblique shape. If so, call `\fake_shape_` to generate an `8r`-reencoded MTX file and to install a ligfull `.pl` file. If not, call macros to fake a oblique fonts.

```
\def\fake_shape_o{
   \if_file_exists{
      \font_family\font_weight\raw_shape\raw_variant
         \adobe_encoding\font_width.afm
   }\then
      \fake_shape_
   \else
      \fake_shape_o_raw_encoding
     \if_expert_
      \fake_shape_o_expert_encoding
     \fi
⟨*oldstyle⟩
     \if_oldstyle_
      \fake_shape_o_oldstyle_encoding
     \fi
⟨/oldstyle⟩
   \fi
}
```

`\fake_shape_o_raw_encoding` Called from `\fake_shape_o` if the oblique shape needs to be faked. Invokes

```
\transformfont{<FONT>o8r}
  {\slantfont{SLANT}{\frommtx{<FONT>8r}}}
```

to generate an 8r-reencoded .mtx file (and a raw .pl file), followed by

```
\installrawfont{<FONT>o8r}{<FONT>o8r,8r}{8r}...
```

to install a ligfull .pl file.

```
\def\fake_shape_o_raw_encoding{
   \if_file_exists{
      \font_family\font_weight\raw_variant
         \raw_encoding\font_width.mtx
   }\then
      \if_file_exists{
         \font_family\font_weight\font_shape\raw_variant
            \raw_encoding\font_width.mtx
      }\then % no action required
      \else
⟨*debug⟩
         \immediate\write16{
            INFO>~run~\string\transformfont\space\space
            <\font_family\font_weight\font_shape\raw_variant
               \raw_encoding\font_width>~from~
            <\font_family\font_weight\raw_variant
               \raw_encoding\font_width>~(faking~oblique)
         }
⟨/debug⟩
         %%% WARNING: famtool.pl relies on this message format!!!
         \immediate\write16{
            Faking~oblique~font~
            \font_family\font_weight\font_shape\raw_variant
               \raw_encoding\font_width
            \space from~
            \font_family\font_weight\raw_variant
               \raw_encoding\font_width
         }
         \transformfont{
            \font_family\font_weight\font_shape\raw_variant
               \raw_encoding\font_width
         }{
            \slantfont{\SlantAmount}{
               \frommtx{
                  \font_family\font_weight\raw_variant
                     \raw_encoding\font_width
               }
            }
         }
         \if_file_exists{ \raw_encoding.mtx }\then
⟨*debug⟩
            \immediate\write16{
               INFO>~run~\string\installrawfont\space
               <\font_family\font_weight\font_shape\raw_variant
                  \raw_encoding\font_width>
               <\font_family\font_weight\font_shape\raw_variant
                  \raw_encoding\font_width,
                \raw_encoding>
               <\raw_encoding>
```

```
               <\raw_encoding>
               <\latex_family>
               <\latex_weight\latex_width>
               <\latex_shape>}
⟨/debug⟩
            \installrawfont{
               \font_family\font_weight\font_shape\raw_variant
                  \raw_encoding\font_width
            }{ \font_family\font_weight\font_shape\raw_variant
                  \raw_encoding\font_width,
               \raw_encoding
            }{ \raw_encoding
            }{ \raw_encoding
            }{ \latex_family
            }{ \latex_weight\latex_width
            }{ \latex_shape
            }{}
         \fi
      \fi
   \fi
}
```

\fake_shape_o_expert_encoding Called from `\fake_shape_o` if the oblique shape needs to be faked. Invokes

```
\transformfont{<FONT>o8x}
  {\slantfont{SLANT}{\frommtx{<FONT>8x}}}
```

to generate an 8x-reencoded `.mtx` file (and a raw `.pl` file).

```
\def\fake_shape_o_expert_encoding{
   \if_file_exists{
      \font_family\font_weight\raw_variant
         \expert_encoding\font_width.mtx
   }\then
      \if_file_exists{
         \font_family\font_weight\font_shape\raw_variant
            \expert_encoding\font_width.mtx
      }\then % no action required
      \else
⟨*debug⟩
         \immediate\write16{
            INFO>~run~\string\transformfont\space\space
            <\font_family\font_weight\font_shape\raw_variant
               \expert_encoding\font_width>~from~
            <\font_family\font_weight\raw_variant
               \expert_encoding\font_width>~(faking~oblique)
         }
⟨/debug⟩
         %%% WARNING: famtool.pl relies on this message format!!!
         \immediate\write16{
            Faking~oblique~font~
            \font_family\font_weight\font_shape\raw_variant
               \expert_encoding\font_width
            \space from~
            \font_family\font_weight\raw_variant
```

```
               \expert_encoding\font_width
         }
         \transformfont{
            \font_family\font_weight\font_shape\raw_variant
               \expert_encoding\font_width
         }{
            \slantfont{\SlantAmount}{
               \frommtx{
                  \font_family\font_weight\raw_variant
                     \expert_encoding\font_width
               }
            }
         }
      \fi
   \fi
}
```

ake_shape_o_oldstyle_encoding Called from `\fake_shape_o` if the oblique shape needs to be faked. Invokes

```
\transformfont{<FONT>oj8r}
  {\slantfont{SLANT}{\frommtx{<FONT>j8r}}}
```

to generate an `8r`-reencoded `.mtx` file (and a raw `.pl` file).

```
⟨*oldstyle⟩
\def\fake_shape_o_oldstyle_encoding{
   \if_file_exists{
      \font_family\font_weight\oldstyle_variant
         \raw_encoding\font_width.mtx
   }\then
      \if_file_exists{
         \font_family\font_weight\font_shape\oldstyle_variant
            \raw_encoding\font_width.mtx
      }\then % no action required
      \else
⟨*debug⟩
         \immediate\write16{
            INFO>~run~\string\transformfont\space\space
            <\font_family\font_weight\font_shape\oldstyle_variant
               \raw_encoding\font_width>~from~
            <\font_family\font_weight\oldstyle_variant
               \raw_encoding\font_width>~(faking~oblique)
         }
⟨/debug⟩
         %%% WARNING: famtool.pl relies on this message format!!!
         \immediate\write16{
            Faking~oblique~font~
            \font_family\font_weight\font_shape\oldstyle_variant
               \raw_encoding\font_width
            \space from~
            \font_family\font_weight\oldstyle_variant
               \raw_encoding\font_width
         }
         \transformfont{
            \font_family\font_weight\font_shape\oldstyle_variant
```

```
            \raw_encoding\font_width
      }{
         \slantfont{\SlantAmount}{
            \frommtx{
               \font_family\font_weight\oldstyle_variant
                  \raw_encoding\font_width
            }
         }
      }
   \fi
\fi
}
</oldstyle>
```

`\fake_shape_i` Call `\fake_shape_` to generate an `8r`-reencoded `.mtx` file and a ligfull `.pl` file, if a corresponding `8a`-encoded `.afm` file exists. Otherwise, do nothing, since an italic shape can't be faked.

```
\let\fake_shape_i\fake_shape_  % We must do this again!
```

14.3 Faking font widths

`\if_fake_narrow_`

```
\newif\if_fake_narrow_
\_fake_narrow_false
```

`\fakenarrow` `\fakenarrow{WIDTH}`

Sets the expansion factor used to generate faked narrow fonts. If it isn't set, do not attempt to install faked narrow fonts.

```
\def\fakenarrow#1{
   \_fake_narrow_true
   \gdef\fake_narrow_width{#1}
}
```

`\fake_width_` Do nothing for the default width or the condensed width.

```
\def\fake_width_{}
\def\fake_width_c{}
```

`\fake_width_n` If we are faking narrow fonts, check if an `8a`-encoded `.afm` file exists for the current shape in narrow width, and if not call `\fake_shape_n_raw_encoding` and `\fake_shape_n_expert_encoding`.

```
\def\fake_width_n{
   \if_fake_narrow_
      \if_file_exists{
         \font_family\font_weight\raw_shape\raw_variant
            \adobe_encoding\font_width.afm
      }\then % no action required
      \else
         \fake_width_n_raw_encoding
        \if_expert_
         \fake_width_n_expert_encoding
        \fi
      \fi
```

```
   \fi
}
```

`\fake_width_n_raw_encoding` Called from `\fake_width_n` if the narrow width needs to be faked. Invokes

```
\transformfont{<FONT>8rn}
  {\xscalefont{WIDTH}{\frommtx{<FONT>8r}}}
```

to generate an `8r`-encoded `.mtx` file for a faked narrow font, followed by

```
\installrawfont{<FONT>8rn}{<FONT>8rn,8r}{8r}...
```

to install a ligfull `.pl` file.

```
\def\fake_width_n_raw_encoding{
   \if_file_exists{
      \font_family\font_weight\font_shape\raw_variant
         \raw_encoding.mtx
   }\then
      \if_file_exists{
         \font_family\font_weight\font_shape\raw_variant
            \raw_encoding\font_width.mtx
      }\then % no action required
      \else
⟨*debug⟩
         \immediate\write16{
            INFO>~run~\string\transformfont\space\space
            <\font_family\font_weight\font_shape\raw_variant
               \raw_encoding\font_width>~from~
            <\font_family\font_weight\font_shape\raw_variant
               \raw_encoding>~(faking~narrow)
         }
⟨/debug⟩
         %%% WARNING: famtool.pl relies on this message format!!!
         \immediate\write16{
            Faking~narrow~font~
            \font_family\font_weight\font_shape\raw_variant
               \raw_encoding\font_width
            \space from~
            \font_family\font_weight\font_shape\raw_variant
               \raw_encoding
         }
         \transformfont{
            \font_family\font_weight\font_shape\raw_variant
               \raw_encoding\font_width
         }{
            \xscalefont{\fake_narrow_width}{
               \frommtx{
                  \font_family\font_weight\font_shape\raw_variant
                     \raw_encoding
               }
            }
         }
         \if_file_exists{ \raw_encoding.mtx }\then
⟨*debug⟩
            \immediate\write16{
```

```
                INFO>~run~\string\installrawfont\space
                <\font_family\font_weight\font_shape\raw_variant
                   \raw_encoding\font_width>
                <\font_family\font_weight\font_shape\raw_variant
                   \raw_encoding\font_width,
                 \raw_encoding>
                <\raw_encoding>
                <\raw_encoding>
                <\latex_family>
                <\latex_weight\latex_width>
                <\latex_shape>}
⟨/debug⟩
             \installrawfont{
                \font_family\font_weight\font_shape\raw_variant
                   \raw_encoding\font_width
             }{ \font_family\font_weight\font_shape\raw_variant
                   \raw_encoding\font_width,
                \raw_encoding
             }{ \raw_encoding
             }{ \raw_encoding
             }{ \latex_family
             }{ \latex_weight\latex_width
             }{ \latex_shape
             }{}
          \fi
       \fi
    \fi
}
```

\fake_width_n_expert_encoding Called from `\fake_width_n` if the narrow width needs to be faked. Invokes

```
\transformfont{<FONT>8xn}
  {\xscalefont{WIDTH}{\frommtx{<FONT>8x}}}
```

to generate an `8x`-encoded `.mtx` file for a faked narrow font.

```
\def\fake_width_n_expert_encoding{
   \if_file_exists{
      \font_family\font_weight\font_shape\raw_variant
         \expert_encoding.mtx
   }\then
      \if_file_exists{
         \font_family\font_weight\font_shape\raw_variant
            \expert_encoding\font_width.mtx
      }\then % no action required
      \else
⟨*debug⟩
         \immediate\write16{
            INFO>~run~\string\transformfont\space\space
            <\font_family\font_weight\font_shape\raw_variant
               \expert_encoding\font_width>~from~
            <\font_family\font_weight\font_shape\raw_variant
               \expert_encoding>~(faking~narrow)
         }
⟨/debug⟩
```

```
         %%% WARNING: famtool.pl relies on this message format!!!
         \immediate\write16{
            Faking~narrow~font~
            \font_family\font_weight\font_shape\raw_variant
               \expert_encoding\font_width
            \space from~
            \font_family\font_weight\font_shape\raw_variant
               \expert_encoding
         }
         \transformfont{
            \font_family\font_weight\font_shape\raw_variant
               \expert_encoding\font_width
         }{
            \xscalefont{\fake_narrow_width}{
               \frommtx{
                  \font_family\font_weight\font_shape\raw_variant
                     \expert_encoding
               }
            }
         }
      \fi
   \fi
}
```

14.4 Installing reencoded fonts

\latin_encoding
\font_encoding
\latex_encoding
\latex_mtx

`\latin_encoding{FONTENC}{EXPERTISED-ENC}{OLDSTYLE-ENC} {LATEXENC}{LATEXMTX}`

If this is an expertised family `EXPERTISED-ENC` is used instead of `FONTENC`. If this is an expertised family with oldstyle digits `OLDSTYLE-ENC` is used instead of `EXPERTISED-ENC`.

UV 06/98: Added code for oldstyle encodings from Alan's v 1.511. The `\font_encoding` is changed when `\if_oldstyle_` is true.

UV 06/98: Added anonther parameter for the default metrics file (which is usually either `latin.mtx` or `textcomp.mtx`).

UV 06/98: Added experimental code for OsF or SC&OsF fonts.

```
\def\latin_encoding#1#2#3#4#5{
   \gdef\latex_encoding{#4}
   \gdef\latex_mtx{#5}
   \if_oldstyle_
      \gdef\font_encoding{#3}
   \else\if_expert_
      \gdef\font_encoding{#2}
   \else
      \gdef\font_encoding{#1}
   \fi\fi
   \gdef\expert_font{}
   \gdef\oldstyle_font{}
   \if_expert_
      \if_file_exists{
         \font_family\font_weight\raw_shape\raw_variant
            \expert_encoding\font_width.mtx
      }\then
         \gdef\expert_font{
```

```
            \font_family\font_weight\raw_shape\raw_variant
               \expert_encoding\font_width,
         }
      \fi
   \fi
⟨*oldstyle⟩
   \if_oldstyle_
      \if_file_exists{
         \font_family\font_weight\raw_shape\oldstyle_variant
            \raw_variant\raw_encoding\font_width.mtx
      }\then
         \gdef\oldstyle_font{
            unsetnum,
            \font_family\font_weight\raw_shape\oldstyle_variant
               \raw_variant\raw_encoding\font_width,
            resetosf,
         }
      \fi
   \fi
⟨/oldstyle⟩
⟨*debug⟩
   \immediate\write16{
      INFO>~run~\string\installfont\space
      <\font_family\font_weight\font_shape\font_variant
         \font_encoding\font_width>
      <\font_family\font_weight\raw_shape\raw_variant
         \raw_encoding\font_width,
      \if_oldstyle_
       \ifx\raw_shape\caps_shape
         resetosf,
       \fi
      \fi
      \expert_font
      \oldstyle_font
      \latex_mtx>
      <\latex_encoding\encoding_shape>
      <\latex_encoding>
      <\latex_family>
      <\latex_weight\latex_width>
      <\latex_shape>}
⟨/debug⟩
   \installfont{
      \font_family\font_weight\font_shape\font_variant
         \font_encoding\font_width
   }{
      \font_family\font_weight\raw_shape\raw_variant
         \raw_encoding\font_width,
      \if_oldstyle_
       \ifx\raw_shape\caps_shape
         resetosf,
       \fi
      \fi
      \expert_font
      \oldstyle_font
```

```
    \latex_mtx
  }{
    \latex_encoding\encoding_shape
  }{
    \latex_encoding
  }{
    \latex_family
  }{
    \latex_weight\latex_width
  }{
    \latex_shape
  }{}
}
```

14.5 Default weights, widths and shapes

Fontname: `weight.map`		NFSS: *LaTeX Companion*, p. 190	
`a`	Thin Hairline	`ul`	Ultra Light
`j`	ExtraLight	`el`	Extra Light
`l`	Light	`l`	Light
`r`	Regular Roman	`m`	Medium
`k`	Book	`m`	Medium
`m`	Medium	`mb`	(was: `m`)
`d`	Demi	`db`	(was: `sb`)
`s`	Semibold	`sb`	Semibold
`b`	Bold	`b`	Bold
`h`	Heavy Heavyface	`eb`	(was missing)
`c`	Black	`eb`	(was missing)
`x`	ExtraBold ExtraBlack	`eb`	Extra Bold
`u`	Ultra UltraBlack	`ub`	Ultra Bold
`p`	Poster		(still missing)

`\latin_weights` Each call to `\latin_weight` maps a Fontname weight code (`\font_weight`) to a LaTeX weight code (`\latex_weight`). Non-existing weights are ignored (or substituted when the `.fd` files are written out by `\endinstallfont`).

The standard values are given in the table. They may be changed, but you'd better know what you're doing.

UV, 04/98: Changed the processing order: Do the most common shapes first. Added new mappings for `c` and `h`, changed mapping for `m` to newly invented LaTeX weight `mb`.

```
\def\latin_weights{
  \latin_weight{r}{m}
  \latin_weight{k}{m}
  \latin_weight{b}{b}
  \latin_weight{s}{sb}
  \latin_weight{d}{db} % was {d}{sb}, SPQR changed
  \latin_weight{m}{mb} % was {m}{m},  UV changed
  \latin_weight{c}{eb} % UV added
  \latin_weight{h}{eb} % UV added
  \latin_weight{x}{eb}
  \latin_weight{u}{ub}
  \latin_weight{l}{l}
```

```
    %\latin_weight{j}{el}
    %\latin_weight{a}{ul} % UV added
}
```

Fontname: `width.map`		NFSS: *LaTeX Companion*, p. 190	
`t`	Thin	–	–
`o`	Ultra Condensed	`uc`	Ultra Condensed
`u`	Ultra Compressed	`uc`	. .
`q`	Extra Compressed	`ec`	Extra Condensed
`c`	Condensed	`c`	Condensed
`p`	Compressed	`c`	. .
`n`	Narrow	`c`	. .
–	–	`sc`	Semi Condensed
`r`	Normal, Medium, Regular	`m`	Medium
–	–	`sx`	Semi Expanded
`e`	Expanded	`x`	Expanded
`x`	Extended	`x`	. .
`v`	Extra Expanded	`ex`	Extra Expanded
–	–	`ux`	Ultra Expanded
`w`	Wide	–	–

`\latin_widths` Each call to `\latin_width` maps a Fontname width code (`\font_width`) to a LaTeX width code (`\latex_width`). Non-existing narrow fonts are faked only if `\fakenarrow` is specifically called for.

```
\def\latin_widths{
    \latin_width{}{}
    \latin_width{n}{c}
    %latin_width{c}{c}
    %latin_width{x}{x}
}
```

`\latin_shapes` The fifth argument of `\latin_shape` is stored in the variable `\encoding_switches`.

```
\def\latin_shapes{
    \latin_shape{} {} {} {n} {\_textcomp_true }
    \latin_shape{c}{c}{} {sc}{\_textcomp_false}
    \latin_shape{o}{o}{} {sl}{\_textcomp_true }
    \latin_shape{i}{i}{i}{it}{\_textcomp_true }
}
```

`\latin_encodings` `\encoding_switches` is evaluated immediatley before the `\if_textcomp_` test to set it as appropriate for the curent shape.

```
\def\latin_encodings{
    \latin_encoding{7t}{9t}{9o}{OT1}{latin}
    \latin_encoding{8t}{9e}{9d}{T1} {latin}
⟨*textcomp⟩
    \encoding_switches
    \if_textcomp_
       \latin_encoding{8c}{9c}{9c}{TS1}{textcomp}
    \fi
⟨/textcomp⟩
}
```

\if_textcomp_ Switch to control whether or not to install a text companion encoding. It is set or unset for each shape by `\latin_shapes` and evaluated in `\latin_encodings`.

```
\newif\if_textcomp_
```

\raw_encoding
\adobe_encoding
\expert_encoding

```
\def\raw_encoding{8r}
\def\adobe_encoding{8a}
\def\expert_encoding{8x}
```

\caps_shape
\oldstyle_shape

```
\def\caps_shape{c}
\def\oldstyle_shape{j}
```

\SlantAmount

```
\def\SlantAmount{167}
```

15 Version numbers

\needsfontinstversion The macro:

```
\needsfontinstversion{NUMBER}
```

checks the version number.

```
\def\needsfontinstversion#1{{
   \a_dimen=#1pt
   \b_dimen=\fontinstversion~pt\relax
   \ifnum\a_dimen>\b_dimen
      \immediate\write16{}
      \immediate\write16{Warning:~This~file~needs~fontinst~version~#1.}
      \immediate\write16{Warning:~You~are~using~version~\fontinstversion.}
      \immediate\write16{Warning:~This~may~cause~errors.}
      \immediate\write16{}
   \fi
}}
```

The directory for temporary files.

```
\def\tempfileprefix#1{\def\temp_prefix{#1}}
\tempfileprefix{}
```

16 Debugging

\NOFILES `\NOFILES` switches off file generation, and causes fontinst only to generate empty files.

```
\def\NOFILES{
   \def\transformfont##1##2{
      \touch_file{##1.mtx}
      \touch_file{##1.pl}
   }
   \def\installfonts{}
   \def\endinstallfonts{}
```

```
   \def\installfont##1##2##3##4##5##6##7##8{%
      \touch_file{##1.vpl}
   }
   \def\installrawfont##1##2##3##4##5##6##7##8{%
      \touch_file{##1.pl}
   }
   \def\installfamily##1##2##3{
      \touch_file{##1##2.fd}
   }
}
```

\touch_file

```
\def\touch_file#1{
   \edef\lowercase_file{\lowercase{
     \edef\noexpand\lowercase_file{#1}}}
   \lowercase_file
   \open_out{\lowercase_file}
   \out_line{\percent_char~TEST~FILE.}
   \out_line{\percent_char~Created~whilst~debugging~fontinst.}
   \close_out{Test~file}
}
```

By default, show as much error info as you can. (I assume `fontinst` users are TEXperts.)

```
\errorcontextlines=999
```

Set `\newlinechar` for `\errhelp` messages.

```
\newlinechar=`\^^J
```

17 Final cleanup

If `\bye` hasn't been defined, define it (for LATEX).

```
\ifx\bye\undefined_command
   \def\supereject{\par\penalty-\@MM}
   \outer\x_cs\def{bye}{\par\vfill\supereject\@@end}
\fi
```

Restore the catcodes we changed.

```
\catcode`\@=\atcatcode
\catcode`\^^M=\nlcatcode
\catcode`\ =\spacecatcode
\catcode`\~=\tildecatcode
```

We input the `fontinst.rc` file, if it exists. (Search order changed 1997/02/07 <Thierry.Bouche@ujf-grenoble.fr>.)

UV, 06/1998: What is this `fontinst.rc` file good for? It turns out that you can use it to do `\def\raw_encoding{8y}` if you prefer to install your fonts the other way.

```
\if_file_exists{fontinst.rc}\then
   \primitiveinput fontinst.rc
   \catcode`\_=\underscorecatcode
\else
```

```
    \catcode`\_=\underscorecatcode
    \immediate\write16{No file fontinst.rc.}
\fi
⟨/pkg⟩
```

The `fontdoc` package

18 Overview

The purpose of the `fontdoc` package is to support typesetting of `.etx` and `.mtx` files intended for use with `fontinst`. The typical format of these files looks something like this:

```
\documentclass[twocolumn]{article}
\usepackage{fontdoc}

\begin{document}
TeX commands
\encoding or \metrics
fontinst commands
\endencoding or \endmetrics
TeX commands
\end{document}
```

To make it work, `fontdoc` has to define all the user-level `fontinst` commands in terms of typesetting instructions.

19 Implementation

Some global variables:

```
⟨*doc⟩
\newdimen\a@dimen
\newcount\a@count
```

Some useful macros and constrol structures:

```
\def\x@cs#1#2{\expandafter#1\csname#2\endcsname}
```

`\setcommand` `\resetcommand` Macros that set define new commands.

```
\def\setcommand#1{\ifx#1\undefined@command
   \expandafter\def\expandafter#1\else
   \expandafter\def\expandafter\a@command\fi}
\def\resetcommand#1{\def#1}
```

`\ifisglyph` `\ifisint` `\ifisdim` `\ifisstr` `\then` All conditionals are treated as true.

```
\def\ifisglyph#1\then{\iftrue}
\def\ifisint#1\then{\iftrue}
\def\ifisdim#1\then{\iftrue}
\def\ifisstr#1\then{\iftrue}
\let\then=\iffalse
```

`\setdim` `\setint` `\setstr` Macros that set integers, dimens or strings.

```
\def\setdim#1#2{\a@dimen=#2\relax\Bheading{Default} #1 = \the\a@dimen}
\def\setint#1#2{\Bheading{Default} #1 = $\expression0{#2}$}
\def\setstr#1#2{\Bheading{Default} #1 = #2}
```

\resetdim \resetint \resetstr

Macros that reset integers, dimens or strings.

```
\def\resetdim#1#2{\a@dimen=#2\relax\Bheading{Value} #1 = \the\a@dimen}
\def\resetint#1#2{\Bheading{Value} #1 = $\expression0{#2}$}
\def\resetstr#1#2{\Bheading{Value} #1 = #2}
```

Utilities for typesetting nested arithmetic expressions.

```
\def\expression#1#2{\a@count=#1\relax#2}
```

```
\def\priority#1#2{
   \ifnum\a@count<#1
      \let\next=\identity
   \else
      \let\next=\bracket
   \fi
   \next{#2}}
\def\identity#1{#1}
\def\bracket#1{(#1)}
```

Plain TeX's `\int` and `\div` are redfined as `\plainint` and `\plaindiv`, since we are going to use them for other purposes.

```
\let\plainint=\int
\let\plaindiv=\div
```

\int \width \height \depth \italic \kerning

Macros that return integer expressions are printed in symbolic form.

```
\def\int#1{{\fam0#1}}
\def\width#1{w({\fam0#1})}
\def\height#1{h({\fam0#1})}
\def\depth#1{d({\fam0#1})}
\def\italic#1{i({\fam0#1})}
\def\kerning#1#2{k({\fam0#1})({\fam0#1})}
```

\neg \add \sub \mul \div \max \min \scale

Nested arithmetic expressions are typeset using `\expression` and `\priority`.

```
\def\neg#1{\priority8{-\expression6{#1}}}
\def\add#1#2{\priority2{\expression2{#1}+\expression2{#2}}}
\def\sub#1#2{\priority2{\expression3{#1}-\expression3{#2}}}
\def\mul#1#2{\priority4{\expression4{#1}\times\expression4{#2}}}
\def\div#1#2{\priority4{\expression5{#1}/\expression5{#2}}}
\def\max#1#2{\priority6{\expression6{#1}\sqcup\expression6{#2}}}
\def\min#1#2{\priority7{\expression7{#1}\sqcap\expression7{#2}}}
\def\scale#1#2{\div{\mul{#1}{#2}}{1000}}
```

`\Aheading` and `\Bheading` are used to typeset various headings.

```
\def\Aheading#1{\par\medskip\noindent{\bf#1}\par}
\def\Bheading#1{\par\noindent{\bf#1}}
```

\needsfontinstversion

`\needfontinstversion` is printed out as a comment.

```
\def\needsfontinstversion#1{\Bheading{Needs fontinst v#1}}
```

\encoding \endencoding \slot@number

The `\encoding` and `\endencoding` macros define a group and initialize the `\slot@number` counter.

```
\def\encoding{\bgroup\slot@number=0}
\def\endencoding{\egroup}
```

\nextslot
\skipslots

`\nextslot` sets the `\slot@number`. `\skipslots` advances the `\slot@number`.

```
\newcount\slot@number
\def\nextslot#1{\slot@number=#1\relax}
\def\skipslots#1{\advance\slot@number by #1\relax}
```

\setslotcomment
\resetslotcomment
\unsetslotcomment
\useexamplefont
\slotexample
\slot@comment
\slot@font

In this version of `fontinst`, we have added an interface for automatic documentation of encoding files, which has been developed by Matthias Clasen as part of his work on math fonts. The implementation was slightly modified and integrated into this version by Ulrik Vieth.

`\setslotcomment` defines a default slot comment, stored in the variable `\slot@comment`, which is subsequently used to annotate all `\setslot` commands. The slot comment can be changed by `\resetslotcomment` or turned off by `\unsetslotcomment`.

`\useexamplefont` defines a default font, `\slot@font`, which may be referenced by calling `\slotexample` in slot comments to display the character or symbol allocated to the current slot.

Taking advantage of this mechanism, it is possible to write:

```
\useexamplefont{FONT}

\setslotcomment{The symbol `\slotexample'.}
\setslot{FOO}\endsetslot
\setslot{BAR}\endsetslot
\resetslotcomment{The character `\slotexample'.}
\setslot{BAZ}\endsetslot
```

instead of having to write:

```
\usepackage{PACKAGE-to-use-symbols-from-FONT}

\setslot{FOO}\comment{The symbol `\foo'.}\endsetslot
\setslot{BAR}\comment{The symbol `\bar'.}\endsetslot
\setslot{BAZ}\comment{The character `\baz'.}\endsetslot
```

```
\newif\ifslot@comment@
\slot@comment@false
\def\slot@comment{}
\def\setslotcomment#1{%
  \slot@comment@true
  \gdef\slot@comment{#1}}
\def\resetslotcomment#1{%
  \gdef\slot@comment{#1}}
\def\unsetslotcomment{%
  \slot@comment@false
  \gdef\slot@comment{}}
%  \end{macrocode}
%
%    \begin{macrocode}
\let\slot@font=\nullfont
\def\useexamplefont#1{\font\slot@font=#1 }
\def\slotexample{{\slot@font\char\the\slot@number}}
```

\setslot
\endsetslot
\slot@name

`\setslot ... \endslot` commands print out the slot number, optionally followed by a default slot comment.

```
\def\setslot#1{\def\slot@name{#1}%
   \Aheading{Slot \the\slot@number\space `#1'}%
   \ifslot@comment@ \comment{\slot@comment}\fi}
\def\endsetslot{\advance\slot@number by 1\relax}
```

`\comment` Slot comments are typeset as a text paragraph.

```
\def\comment{\par\noindent}
```

`\ligature` Symbolic typesetting of `\ligature` programs requires special processing.

```
\def\ligature#1{\Bheading{Ligature} \csname doc-#1\endcsname}
```

```
\x@cs\def{doc-LIG}#1#2{%
  $\fam0 \slot@name * #1 \rightarrow #2$}
\x@cs\def{doc-/LIG}#1#2{%
  $\fam0 \slot@name * #1 \rightarrow \slot@name * #2$}
\x@cs\def{doc-LIG/}#1#2{%
  $\fam0 \slot@name * #1 \rightarrow #2 * #1$}
\x@cs\def{doc-/LIG/}#1#2{%
  $\fam0 \slot@name * #1 \rightarrow \slot@name * #2 * #1$}
\x@cs\def{doc-/LIG>}#1#2{%
  $\fam0 \slot@name * #1 \rightarrow \slot@name + #2$}
\x@cs\def{doc-LIG/>}#1#2{%
  $\fam0 \slot@name * #1 \rightarrow #2 + #1$}
\x@cs\def{doc-/LIG/>}#1#2{%
  $\fam0 \slot@name * #1 \rightarrow \slot@name + #2 * #1$}
\x@cs\def{doc-/LIG/>>}#1#2{%
  $\fam0 \slot@name * #1 \rightarrow \slot@name + #2 + #1$}
```

`\nextlarger` `\varchar` `\vartop` `\varmid` `\varbot` `\varrep` — `\nextlarger` and `\varchar ... \endvarchar` commands.

```
\def\nextlarger#1{\Bheading{Next larger} #1}
\def\varchar{\Bheading{Extensible glyph:}}
\def\vartop#1{\Bheading{Top} #1}
\def\varmid#1{\Bheading{Middle} #1}
\def\varbot#1{\Bheading{Bottom} #1}
\def\varrep#1{\Bheading{Repeated} #1}
\let\endvarchar\relax
```

`\metrics` `\endmetrics` — The `\metrics` and `\endmetrics` macros just define a group.

```
\def\metrics{\bgroup}
\def\endmetrics{\egroup}
```

`\setkern` `\setleftkerning` `\setrightkerning` `\setleftrightkerning` — Setting kern pair manipulations.

```
\def\setkern#1#2#3{%
   \Bheading{Kern} ${\fam0 #1 + #2} \rightarrow \expression0{#3}$}
\def\setleftkerning#1#2#3{%
   \Bheading{Kern} #1 on the left like #2 scaled $\expression0{#3}$}
\def\setrightkerning#1#2#3{%
   \Bheading{Kern} #1 on the right like #2 scaled $\expression0{#3}$}
\def\setleftrightkerning#1#2#3{%
   \Bheading{Kern} #1 on the left and right like #2 scaled $\expression0{#3}$}
```

`\setrawglyph` Setting raw glyphs.

```
\def\setrawglyph#1#2#3#4#5#6#7#8{\Aheading{Glyph `#1'}
   \Bheading{Taken from} slot #4 in font #2}
```

\setglyph \resetglyph \endsetglyph \endresetglyph

Setting and resetting glpyhs.

```
\def\setglyph#1{\Aheading{Glyph `#1'}}
\def\resetglyph#1{\Aheading{Reset glyph `#1'}}
\def\endsetglyph{}
\def\endresetglyph{}
```

\glyph \glyphrule \glyphspecial \glyphwarning

Glyph commands used in `\setglyph ... \endsetglyph`.

```
\def\glyph#1#2{\Bheading{Glyph} `#1' scaled $\expression0{#2}$}
\def\glyphrule#1#2{%
  \Bheading{Rule} $\expression0{#1}$ by $\expression0{#2}$}
\def\glyphspecial#1{\Bheading{Special} `#1'}
\def\glyphwarning#1{\Bheading{Warning} `#1'}
```

\movert \moveup

```
\def\movert#1{\Bheading{Rt}~$\expression0{#1}$}
\def\moveup#1{\Bheading{Up}~$\expression0{#1}$}
```

\push \pop

```
\def\push{\Bheading{Push}}
\def\pop{\Bheading{Pop}}
```

\resetwidth \resetheight \resetdepth \resetitalic

```
\def\resetwidth#1{\Bheading{Reset width to $\expression0{#1}$}}
\def\resetheight#1{\Bheading{Reset height to $\expression0{#1}$}}
\def\resetdepth#1{\Bheading{Reset depth to $\expression0{#1}$}}
\def\resetitalic#1{\Bheading{Reset italic to $\expression0{#1}$}}
```

Some last-minute commands.

```
\raggedright
\raggedbottom
\def\plain{{\tt plain}}
⟨/doc⟩
```

Index

Numbers written in italic refer to the page where the corresponding entry is described, the ones underlined to the code line of the definition, the rest to the code lines where the entry is used.