Surviving the TEX font encoding mess

Understanding the world of TEX fonts and mastering the basics of *fontinst*

Ulrik Vieth · Taco Hoekwater

EuroTEX '99 · Heidelberg

FAMOUS QUOTE:

English is useful because it is a mess. Since English is a mess,
it maps well onto the problem space, which is also a mess,
which we call reality. Similary, Perl was designed to be a mess,
though in the nicests of all possible ways.

— LARRY WALL

COROLLARY:

TEX fonts are mess, as they are a product of reality.
Similary, `fontinst` is a mess, not necessarily by design,
but because it has to cope with the mess we call reality.

Contents

I Overview of TEX font technology

- What is a font? What is a virtual font?
- Font file formats and conversion utilities
- Font attributes and classifications
- Font selection schemes
- Font naming schemes
- Font encodings
- What’s in a standard font? What’s in an expert font?
- Font installation considerations
- Why the need for reencoding?
- Which raw font encoding to use?
- What’s needed to set up fonts for use with TEX?

What is a font?

- in technical terms:
  - fonts have many different representations depending on the point of view
  - TEX typesetter: fonts metrics (`TFM`) and nothing else
  - DVI driver: virtual fonts (`VF`), bitmaps fonts(`PK`), outline fonts (`PFA`/`PFB` or `TTF`)
  - PostScript: Type 1 (outlines), Type 3 (anything), Type 42 fonts (embedded `TTF`)
- in general terms:
  - fonts are collections of glyphs (characters, symbols) of a particular design
  - fonts are organized into families, series and individual shapes
  - glyphs may be accessed either by character code or by symbolic names
  - encoding of glyphs may be fixed or controllable by encoding vectors
- font information consists of:
  - metric information (glyph metrics and global parameters)
  - some representation of glyph shapes (bitmaps or outlines)

What is a font … from the point of view of TEX?

- a font is described *only* by its metric information stored in `TFM` files
  - glyph metrics are accessed by font position, i.e. by character code
  - font encodings are fixed (font-specific), not changeable
  - mapping between glyphs and character codes happens at the macro level
  - macro packages need to know about font encodings and naming schemes
- font metric information consist of global and per-glyph information:
  - `FAMILY` and `CODINGSCHEME` parameters (not accessible from TEX)
  - global `\fontdimen` parameters (space, stretch, shrink, quad, etc.)
  - ligature and kerning table (interaction between glyphs)
  - glyph dimensions (width, height, depth, italic corrections)
- technical limitations of `TFM` format:
  - only 16 different heights or depths, 256 different widths
  - only 16 families of math fonts ($16\times256 = 4096$ math symbols)

What is a font … from the point of view of a DVI driver?

- a font is a file that contains a representation of glyph shapes
  - traditional approach: TEX-specific bitmap fonts stored in PK files
  - more modern approach: outline fonts (PostScript or TrueType)
- for bitmap fonts:
  - glyph shapes are represented as bitmaps of black and white pixels
  - glyph bitmaps are generated for specific resolutions and magnifications
  - glyph bitmaps are accessed by font position, i.e. by character code
  - font encodings are fixed (font-specific), not changeable
- for outline fonts:
  - printer-resident fonts or system fonts can be accessed directly
  - non-resident fonts have to be downloaded to the output file or device
  - processing and reencoding is left to the PostScript interpreter
  - rendering of outlines to pixels is left to the PostScript renderer

What is a font … from the point of view of PostScript?

- a font is a file that consists of programs to draw outlines of glyph shapes
  - glyph programs are stored in an encoded format in PFA/PFB files
  - glyph programs are accessed by symbolic names, such as `/germandbls`
  - mapping between glyphs and character codes by encoding vectors
  - outlines may be scaled or transformed (slanted, extended) as needed
- font encoding may be changed by encoding vectors: *reencoding*
  - glyphs may be hidden away from an encoding vector (unencoded glyphs)
  - glyphs may appear multiple times in an encoding vector
- font encodings used by default:
  - Standard encoding hides away 79 out of 228 standard characters
  - Expert encoding (subset) contains 165 (or 86) extra characters
  - reencoding is necessary to gain access to all glyphs in standard fonts

What is a virtual font?

- virtual fonts consist of metrics (`TFM`) and typesetting instructions (`VF`)
  - virtual fonts appear like normal fonts from the point of view of TEX
  - virtual fonts are interpreted by DVI drivers (or the pdfTEX back-end)
- typical applications of virtual fonts:
  - reordering glyphs from a single font: *remapping* (not *reencoding*!)
  - combining glyphs from multiple raw fonts in a single font
  - faking unavailable glyphs by combining multiple glyphs
  - faking unavailable font shapes using transformed versions
- specific applications of virtual fonts:
  - adding ff-ligatures from expert fonts to standard fonts
  - adding small caps or old style figures to standard fonts
  - putting accent glyphs on top of unaccented letters
  - faking small caps by scaling and letterspacing

Font file formats

- traditional METAFONT bitmap fonts:
  - TFM, PL: TEX font metrics (binary format), property lists (textual format)
  - VF, VPL: virtual fonts (binary format), virtual property lists (textual format)
  - GF, PK: generic fonts, packed fonts (bitmap formats)
- PostScript Type 1 outline fonts:
  - AFM: Adobe font metrics (textual format)
  - PFM: printer font metrics (binary format)
  - PFA: printer font ASCII (encoded glyph programs in textual format)
  - PFB: printer font binary (encoded glyph programs in binary format)
- TrueType outline fonts:
  - TTF: TrueType font (includes both metrics and glyph programs)
  - T42: Type 42 font, TrueType font embedded in PostScript wrapper

Font conversion utilities

- TEXware / METAFONTware utilities:
  - `tftopl`, `pltotf`: convert TFM to PL and back
  - `vftovp`, `vptovf`: convert VF/TFM to VPL and back
  - `gftopk`, `pktogf`: convert GF to PK and back
- PostScript utilities:
  - `afm2tfm` (included with `dvips`): convert (and reencode) AFM to TFM
  - `gsf2pk` (included with `xdvi`): render PFA or PFB fonts to PK
  - `t1binary`, `t1ascii` (from `t1utils`): convert PFA to PFB and back
  - `t1disasm`, `t1asm` (from `t1utils`): decode or encode PFA or PFB
- TrueType utilities:
  - `ttf2afm` (included with `pdftex`): generate AFM for TTF fonts
  - FreeType project: `ttf2tfm`, `ttf2pk`, `ttf2pfb`, etc.
- *fontinst* [to be discussed later]

Font attributes and classifications

- fonts may be described by the following *font attributes*:

– *family*	typeface name
– *series*	combination of *weight* and *width*
– *weight*	regular, bold, semibold, light, demi, book, medium, black, etc.
– *width*	normal, condensed, compressed, extended, expanded, etc.
– *shape*	normal (upright), slanted (oblique), italic
– *variant*	small caps, old style figures
– *glyph set*	standard, expert, alternate, swash, etc.
– *encoding*	[to be discussed later]

- font attributes are reflected in font names:
  - PostScript font names, e.g. `Minion-SemiboldItalicSC`
  - Berry font naming scheme, e.g. `pmnsic8a`, `pmnsic8r`, `pmnsic8t`
  - LaTeX font selection scheme, e.g. `\usefont{T1}{pmn}{sb}{scit}`

Font selection schemes

- traditional plain TEX font selection scheme:
  - specific font commands (`\tenrm`) are used to access specific fonts (`cmr10`)
  - generic font commands (`\rm`) are mapped to specific commands (`\tenrm`)
  - font commands select combinations of family, series, shape and size
- LATEX 2_ε font selection scheme:
  - mapping of font commands to files through `*.fd` files (*font definitions*)
  - font attributes (encoding, family, series, shape) are decoupled
  - `\usefont` command selects specific combinations of font attributes:
    e.g. `\fontsize{10}{12}\usefont{OT1}{cmr}{m}{n}`
  - generic font commands select or switch font attributes independently:
    e.g. `\fontfamily{cmr}`, `\fontseries{m}`, `\fontshape{n}`
    e.g. `\rmfamily`, `\mdseries`, `\upshape`
  - font attributes may be substituted by default values:
    e.g. `\rmdefault`, `\seriesdefault`, `\shapedefault`
  - font changes take effect only after `\selectfont` command

Font naming schemes

- PostScript font names are given in verbose format (32 chars) like this:

  `FamilySupplier-SeriesShapeVariant`

- PostScript fonts are named according to vendor-specific naming schemes
- TeX fonts are named according to Karl Berry's font name scheme like this:

  `S FF W [V] EE [W] [DD]`

- How the Berry font name scheme is composed:

  `S`: supplier, `FF`: family, `W`: weight, `V`: variants (as needed),
  `EE`: encoding, `W` width (if any), `DD`: design size (if any)

- Problems of the Berry font name scheme:
  - designed to be compatible with 8+3 file systems (more or less)
  - limited to no more than 26 suppliers and 26×36 families
  - all meaningful supplier and family codes are already taken
  - no one-to-one mapping of weights onto LaTeX font selection codes
  - distinction between shapes, design and encoding variants is messy

Decoding the Berry font naming scheme (I)

some *supplier* codes:

`a`	Autologic
`b`	Bitstream
`c`	Compugraphic
`d`	DTC
`e`	Apple
`f`	‘free’ / public
`g`	GNU
`h`	Bigelow & Holmes
`i`	ITC
`j`	Microsoft
`k`	Softkey
`l`	Linotype
`m`	Monotype
`n`	IBM
`o`	Corel
`p`	Adobe
`r`	‘raw’ (obsolete)
`s`	Sun
`t`	Paragraph
`u`	URW
`w`	Wolfram
`z`	bizarre

some *family* codes:

`a1`	Arial
`ac`	Adobe Caslon
`ad`	Adobe Garamond
`ag`	AvantGarde
`bb`	Bembo
`bd`	Bodoni
`bk`	Bookman
`bv`	Baskerville
`ca`	Caslon
`ch`	Charter
`cr`	Courier
`dt`	Dante
`fr`	Frutiger
`fu`	Futura
`gl`	Galliard
`gm`	Garamond
`gs`	Gill Sans
`gv`	Giovanni
`gy`	Goudy
`hv`	Helvetica

more *family* codes:

`lc`	Lucida
`lh`	Lucida Bright
`ls`	Lucida Sans
`lx`	Lucida Fax
`mn`	Minion
`my`	Myriad
`nb`	New Baskerville
`nc`	New Century Schoolbook
`ns`	Times New Roman PS
`nt`	Times New Roman
`op`	Optima
`pi`	‘Pi’ fonts (symbols)
`pl`	Palatino
`sb`	Sabon
`sy`	Symbol
`un`	Univers
`ut`	Utopia
`tm`	Times
`zc`	Zapf Chancery
`zd`	Zapf Dingbats

Decoding the Berry font naming scheme (II)

some *weight* codes:

`l`	light
`r`	regular
`k`	book
`m`	medium
`d`	demi
`s`	semibold
`b`	bold
`c`	black

some *width* codes:

`c`	condensed
`p`	compressed
`n`	narrow
`-`	normal
`e`	expanded
`x`	extended

some *variant* codes:

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

some *shape* codes:

`c`	small caps
`i`	italic
`o`	oblique, slanted
`u`	unslanted italic

some *encodings*:

`8a`	Adobe standard encoding
`8x`	Adobe expert encoding
`8r`	`TeXBase1Encoding`
`8y`	`TeXnAnsiEncoding` (`LY1`)
`7t`	7-bit TeX text (`OT1`)
`7m`	7-bit math letters (`OML`)
`7y`	7-bit math symbols (`OMS`)
`7v`	7-bit math extension (`OMX`)
`8t`	8-bit TeX text (`T1`)
`8c`	8-bit TeX text symbols (`TS1`)
`7a`	Alternate or Swash Caps
`7c`	DFr (= Deutsche Fraktur)

Mapping the Berry font naming schemes to LaTeX font attributes

Berry font names and LaTeX *weight* codes:

`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	`eb`	(was missing)
`c`	Black	`eb`	(was missing)
`x`	Extra, ExtraBlack	`eb`	Extra Bold
`u`	Ultra, UltraBlack	`ub`	Ultra Bold
`p`	Poster	–	(still missing)

Berry font names and LaTeX *width* codes:

`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, Regular	`m`	Medium
–	–	`sx`	Semi Expanded
`e`	Expanded	`x`	Expanded
`x`	Extended	`x`	..
`v`	Extra Expanded	`ex`	Extra Expanded
–	–	`ux`	Ultra Expanded
`w`	Wide	–	–

Font encodings (I)

- 7-bit TEX-specific encodings:
  - `7t` (`OT1`): 7-bit text fonts, e.g. `cmr`, `cmsl`
  - `7t` (`OT1i`): 7-bit text with variant glyphs (£ vs. $), e.g. `cmmi`
  - `7t` (`OT1c`): 7-bit text with small caps glyphs, e.g. `cmcsc`
  - `7t` (`OT1t`): 7-bit text without f-ligatures, e.g. `cmtt`
- 8-bit TEX-specific encodings:
  - `8t` (`T1`): 8-bit text fonts, e.g. `ecrm`, `ecsl`, `ecit`
  - `8c` (`TS1`): 8-bit text symbol fonts, e.g. `tcrm`, `tcsl`, `tcit`
- 8-bit default encodings for PostScript / TrueType fonts:
  - `8a`: Adobe standard or SC+OsF encoding
  - `8x`: Adobe expert or expert subset encoding
- 8-bit raw encodings for PostScript / TrueType fonts:
  - `8r`: `TeXBase1Encoding`
  - `8y`: `TeXnANSIEncoding` (`LY1`)

Font encodings (II)

- target encodings for font installation:
  - `7t` (OT1): 7-bit text (standard)
  - `9t` (OT1): 7-bit text (standard + expert)
  - `9o` (OT1): 7-bit text (standard + expert + oldstyle)
  - `8t` (T1): 8-bit text (standard)
  - `9e` (T1): 8-bit text (standard + expert)
  - `9d` (T1): 8-bit text (standard + expert + oldstyle)
  - `8c` (TS1): 8-bit text symbols (standard)
  - `9c` (TS1): 8-bit text symbols (standard + expert)
  - `8i` (TS0): 8-bit text symbols subset (standard)
  - `9i` (TS0X): 8-bit text symbols subset (standard + expert)
- non-Latin 1 encodings (Greek, Cyrillic, etc.)

What’s in a standard font?

- What’s in a standard font?
  - regular numerals
  - ASCII and Latin 1 capital letters (slots 65–90, 160–191)
  - ASCII and Latin 1 lowercase letters (slots 97–122, 224–255)
  - ASCII and Latin 1 symbols (slots 32–127, 160–191)
  - miscellaneous symbols, including fi- and fl-ligatures
- What’s in an SC+OsF font?
  - oldstyle figures (instead of regular numerals)
  - ASCII and Latin 1 capital letters (slots 65–90, 160–191)
  - ASCII and Latin 1 small caps letters (slots 97–122, 224–255)
- What’s in an OsF font?
  - oldstyle figures (instead of regular numerals)
  - ASCII and Latin 1 capital letters (slots 65–90, 160–191)
  - ASCII and Latin 1 lowercase letters (slots 97–122, 224–255)

What’s in an expert font?

- What’s in an expert font?
  - oldstyle figures, superior and inferior figures
  - ASCII and Latin 1 small caps letters (slots 97–122, 224–255)
  - miscellaneous symbols, including ff-, ffi- and ffl-ligatures
- What’s in an expert subset font?
  - oldstyle figures, superior and inferior figures
  - empty slots instead of small caps letters
  - miscellaneous symbols, including ff-, ffi- and ffl-ligatures
- How are glyph names organized?
  - small caps letters: `/Asmall` instead of `/a`
  - oldstyle figures: `/zerooldstyle` instead of `/zero`

Font installation considerations

- Given a set of standard fonts only:
  - regular fonts: can be implemented except for ff-ligatures
  - small caps fonts: can be faked using scaled fonts and letterspacing
  - oldstyle figures: cannot be implemented or faked at all
- Given a set of standard and expert fonts:
  - regular fonts: implemented using ff-ligatures from expert fonts
  - small caps fonts: implemented using small caps from expert fonts
  - oldstyle figures: implemented using oldstyle figs from expert fonts
- Given a set of standard and expert subset fonts:
  - regular fonts: implemented using ff-ligatures from expert fonts
  - small caps fonts: can be faked using scaled fonts and letterspacing
  - oldstyle figures: implemented using oldstyle figs from expert fonts

Why the need for reencoding?

- 7-bit TEX-specific encodings (`OT1`) are inadequate for accented languages
- optimal encoding should be based on Latin 1 as much as possible
- optimal encoding should make best use of all available glyphs
- Adobe standard encoding (`8a`) hides away too many glyphs
- 8-bit TEX-specific encodings (`T1` and `TS1`) go beyond standard glyph set
- best use of available glyph set can be made through reencoding to raw fonts
- raw font encodings can be used directly for typesetting, if desired
- virtual fonts are needed to combine standard glyphs with expert glyphs
- `7t` (`OT1`) can be implemented either by reencoding or remapping
- `8t` (`T1`) requires faking of non-Latin 1 glyphs through virtual fonts
- `8c` (`TS1`) includes glyphs that cannot be implemented through faking
- `8i` (`TS0`) is the subset which can be implemented without faking

Which raw font encoding to use?

- `8r` and `8y` both provide full access to all glyphs available in standard fonts
- `8r` and `8y` are based on ASCII (32–127) and Latin 1 (160-255)
- `8r` and `8y` differ in placement of extra glyphs and symbols
- `8r` and `8y` include slots for ff-ligatures (usually absent from standard fonts)
- `8r` is widely used for raw fonts in CTAN metrics since 1995
- `8y` is proposed as an alternative approach by Y&Y ("`8y` is `8r` done right")
- `8r` is only used as a raw font encoding, not directly for typesetting
- `8y` is also used as a LaTeX output encoding for typesetting (`LY1`)
- `8y` mostly follows `OT1` layout in lower half (including some glyphs twice)
- `8y` avoids complications of `T1`, `TS1` regarding non-standard glyphs
- `8y` is functionally equivalent or even superior to `8r` for standard glyphs
- `8y` still requires virtual fonts to make use of expert glyphs
- `LY1` may be the best choice for Latin 1, but `T1` also supports Latin 2

What’s needed to set up fonts for use with TEX?

- PostScript fonts: Have `AFM` files ready or convert `PFM` files to `AFM` files
- TrueType fonts: Extract font metrics to `AFM` files using `ttf2afm`
- Install font metrics (`AFM`) and font programs (PFA/PFB or TTF)
- Reencode fonts to raw encoding (`8r` or `8y`) to make all glyphs accessible
- Transform raw fonts as needed to fill missing shapes (`SlantFont`)
- Generate TEX font metrics (`TFM`) for each reencoded or transformed font
- Generate virtual fonts (`VF`, `TFM`) to implement usual TEX encodings
- Install generated font metrics (`TFM`) and virtual fonts (`VF`, `TFM`)
- Generate and install font definition files (`*.fd`) for LATEX
- Generate or update font map files for `dvips` and `pdftex`

II Installing TEX fonts with *fontinst*

- Overview of *fontinst* — What *fontinst* can do or can't do
- History and development of *fontinst*
- Installing and setting up *fontinst* — Running *fontinst*
- Low-level *fontinst* commands: `\transformfont`, `\install(raw)font`
- High-level *fontinst* commands: `\installfamily`, `\latinfamily`
- How fonts are installed in `\latinfamily`
- Understanding the details of font installation
- Perl front-ends for *fontinst*
- Some little-known *fontinst* tricks
- Font installation examples step by step

Overview of *fontinst*

- What is *fontinst*?
  - a general-purpose utility for (PostScript) font installation
  - developed by Alan Jeffrey, now maintained by volunteer group
  - development coordinated through *fontinst* mailing list
- Features of *fontinst*:
  - written entirely in TEX for portability at the cost of speed
  - operates on font metric information in textual format
  - reads `AFM` or `PL` files and writes out `PL` or `VPL` files
  - uses `ETX` files to specify source and target encodings
  - uses `MTX` files to record metric and kerning information
  - allows reencoding, transforming and scaling fonts as needed
  - supports installation of reencoded and transformed raw fonts
  - supports installation of virtual fonts based on raw fonts
  - allows manipulating glyph metrics and kerns through `MTX` files

What *fontinst* can do or can't do

- What *fontinst* can do:
  - convert PostScript font metrics (`AFM`) to internal *fontinst* format (`MTX`)
  - reencode standard-encoded fonts (`8a`) to raw encoding (`8r` or `8y`)
  - transform raw fonts as needed to create slanted or narrow fonts
  - generate TEX font metrics (`TFM`) for each reencoded or transformed font
  - generate virtual fonts (`VF`, `TFM`) to implement usual TEX encodings
  - generate LATEX font definition files (`*.fd`)
- What *fontinst* can't do:
  - generate font map file for `dvips` and `pdftex`
  - add checksums to `PL` and `VPL` files for consistency checks
  - convert `PL` and `VPL` files to binary `TFM` and `VF` files
  - install font metrics, virtual fonts and font definition files

History and development of *fontinst* (I)

- Version 0.xx (ASAJ) started in Feb. 1993, presented at TUG ’93
- Version 1.00 (ASAJ) started after TUG ’93, complete rewrite
- Version 1.3xx (ASAJ) presented at TUG ’94
  - `\latinfamily` implemented using `8a`-encoded base fonts
- Version 1.400 (ASAJ) started in Nov. 1994
  - re-implementation of `\latinfamily` using `8r`-encoded raw fonts
- Version 1.500 (SPQR) released in Sept. 1995
  - first CTAN release of PostScript metrics using `8r`-encoded raw fonts
- Version 1.5xx (ASAJ) unreleased Jun. 1996
  - added support for expertized oldstyle fonts
- Version 1.6 (SPQR) released in Feb. 1997
  - added `\textcompfamily` for TS1 encoding (`8c`)

History and development of *fontinst* (II)

- Version 1.8xx (UV, Jun. 1998)
  - converted macro sources to `DOCSTRIP` format
  - merged development lines of 1.5xx and 1.6 versions
  - integrated `\textcompfamily` into `\latinfamily`
  - integrated support for expertized oldstyle fonts
  - updated user documentation (Rowland McDonnell)
- Version 1.9xx (LH, to be released in 1999)
  - modularized `DOCSTRIP` sources
  - fixed some long-standing known bugs
  - added some experimental features related to kerns
  - updated source documentation (Lars Hellström)

Installing and setting up *fontinst*

- *fontinst* is included in many TEX distributions (TEX Live, `teTeX`, `fpTeX`)
- *fontinst* distribution available from `CTAN:fonts/utilites/fontinst`
- Contents of the *fontinst* distribution:
  - `fontinst.sty`: primary *fontinst* macro package for use with plain TEX
  - `fontinst.ini`: extra *fontinst* module for use with `INITEX`
  - `fontinst.rc`: local configuration or modification file (optional)
  - `*.etx`: encoding definitions for most common encodings
  - `*.mtx`: metric files used to install common encodings
- Installing the *fontinst* distribution:
  - typical installation path: `$TEXMF/tex/fontinst/base/`
  - `TEXINPUTS` search path used to find distributed `ETX` and `MTX` files
  - `TEXINPUTS` search path used to find `AFM` font metrics as well (!)
  - favorite approach: regard *fontinst* as a special TEX format: *fontinst*-TEX uses `TEXINPUTS.fontinst` search path

Running *fontinst*

- How *fontinst* works:
  - *fontinst* macro package is loaded from a (temporary) TEX file
  - reads encodings and glyph commands from auxiliary files
  - reads font metric files and stores them in auxiliary files
  - writes font metric files for generated fonts
- Example *fontinst* control file:

```
\input fontinst.sty           % loads fontinst.sty and fontinst.rc

\transformfont  commands      % creates MTX files from AFM or PL
\installfonts
  \installfamily  commands    % records FD  files to be created
  \installrawfont commands    % creates PL  files from MTX and ETX
  \installfont    commands    % creates VPL files from MTX and ETX
\endinstallfonts              % creates FD  files
\bye
```

Low-level *fontinst* commands: `\transformfont`

- Overview of `\transformfont`:
  - converts font metrics to internal *fontinst* format (MTX files)
  - reads font metrics from existing MTX files, AFM files or PL files
  - supports reencoding and geometric transformations of font metrics
- Syntax of `\transformfont`:

```
\transformfont{<font>}{<commands>}      % writes transformed MTX
 \frommtx{<font>}                       % reads from existing MTX
 \fromafm{<font>}                       % reads from AFM, writes MTX and PL
 \frompl {<font>}                       % reads from PL, writes MTX

 \reencodefont {<ENC>}{<font>}          % PostScript /ReencodeFont
 \extendfont{<factor>}{<font>}          % PostScript /ExtendFont
 \slantfont {<factor>}{<font>}          % PostScript /SlantFont
```

- Examples of `\transformfont`:

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

Low-level *fontinst* commands: `\installfont`

- Overview of `\installrawfont` and `\installfont`:
  - generates PL or VPL files, which can be converted to TFM or VF files
  - uses target encoding specified in a given ETX file
  - uses glyph metrics and kerns from a list of given MTX files
- Syntax of `\installrawfont` and `\installfont`:

```
\installrawfont{<font>} {<mtx,mtx,...>}  {<etx>} <LaTeX fd param>
\installfont    {<font>} {<mtx,mtx,...>}  {<etx>} <LaTeX fd param>
```

- Examples of `\installrawfont` and `\installfont`:

```
\installrawfont{ptmr8r} {ptmr8r,8r}       {8r}   {8r} {ptm}{m}{n}{}
\installrawfont{ptmri8r}{ptmri8r,8r}      {8r}   {8r} {ptm}{m}{it}{}
\installrawfont{ptmro8r}{ptmro8r,8r}      {8r}   {8r} {ptm}{m}{sl}{}

\installfont    {ptmr7t} {ptmr7r,latin}   {OT1} {OT1}{ptm}{m}{n}{}
\installfont    {ptmr8t} {ptmr8r,latin}   {T1}  {T1} {ptm}{m}{n}{}
\installfont    {ptmr8c} {ptmr8r,textcomp}{TS1} {TS1}{ptm}{m}{n}{}

\installfont    {ptmri7t}{ptmri8r,latin}  {OT1i}{OT1}{ptm}{m}{it}{}
\installfont    {ptmro7t}{ptmro8r,latin}  {OT1} {OT1}{ptm}{m}{sl}{}
\installfont    {ptmrc7t}{ptmr8r,latin}   {OT1c}{OT1}{ptm}{m}{sc}{}
```

Low-level *fontinst* commands: `\installfamily`

- Overview of `\installfamily`:
  - grouped between `\installfonts` and `\endinstallfonts`
  - initializes a token list, in which `*.fd` information is recorded
  - `*.fd` entries are recorded for each `\installfont` command
  - `*.fd` entries are written out when `\endinstallfonts` is processed
- Syntax of `\installfamily`:

```
\installfamily {<enc>}{<family><variant>}{}
```

- Examples of `\installfamily`:

```
\installfamily {8r}{pmn}{}      % standard family
\installfamily {8r}{pmnx}{}     % expertized family
\installfamily {8r}{pmnj}{}     % oldstyle family

\installfamily {8r}{hls}{}      % standard family
\installfamily {8r}{hlst}{}     % variant family
```

High-level *fontinst* commands: `\latinfamily` (I)

- Overview of `\latinfamily`:
  - attempts to do an automatic installation of a given font family
  - supports standard, expertized, or expertized oldstyle installations
  - installs `8r` (or `8y`) raw fonts as well as `8x` expert fonts
  - installs `OT1`, `T1` and `TS1` virtual fonts
  - installs all available font series (weights) for standard font shapes
  - installs faked small caps if real small caps are not available
- Syntax of `\latinfamily`:

```
\latinfamily {<family><variant>}{}
```

- Examples of `\latinfamily`:

```
\latinfamily {pmn}{}             % standard family:   7t, 8t, 8c
\latinfamily {pmnx}{}            % expertized family: 9t, 9e, 9c
\latinfamily {pmnj}{}            % oldstyle family:   9o, 9d, 9c
```

High-level *fontinst* commands: `\latinfamily` (II)

- What's going on inside `\latinfamily`:
  - calls `\installfamily` for desired raw font encoding (`8r` or `8y`)
  - calls `\installfamily` for TEX font encodings (`OT1`, `T1` and `TS1`)
  - processes a list of series (all weights, starting with regular and bold)
  - processes a list of shapes (upright, slanted, italic, small caps)
  - attempts to install fonts for all combinations of series and shape
- What's going on inside font installation attempt?
  - checks if `8a`-encoded base font exists for current series and shape
  - calls `\transformfont` to reencode or transform base fonts to raw fonts
  - calls `\installrawfont` to install `8r`- or `8y`-encoded raw fonts
  - calls `\installfont` to install virtual fonts for `OT1`, `T1` and `TS1` variants

How fonts are installed in `\latinfamily` (I)

- Installation of normal (upright) font shapes:
  - checks if `8a`-encoded base font in *upright* shape exists
  - reencodes and installs `8r`- or `8y`-encoded raw font in *upright* shape
  - installs virtual fonts for *standard* encoding variants (`OT1`, `T1`, `TS1`)
- Installation of *real* italic font shapes:
  - checks if `8a`-encoded base font in *italic* shape exists
  - reencodes and installs `8r`- or `8y`-encoded raw font in *italic* shape
  - installs virtual fonts for *italic* (£ vs. $) encoding variants (`OT1i`, `T1`, `TS1`)
- Installation of *faked* slanted font shapes:
  - checks if `8a`-encoded base font in *upright* shape exists
  - transforms and installs `8r`- or `8y`-encoded raw font to *slanted* shape
  - installs virtual fonts for *standard* encoding variants (`OT1`, `T1`, `TS1`)

How fonts are installed in `\latinfamily` (II)

- Installation of *real* small caps font shapes:
  - checks if `8a`-encoded base font in *small caps* shape exists
  - reencodes and installs `8r`- or `8y`-encoded raw font in *small caps* shape
  - installs virtual fonts for *standard* encoding variants (`OT1`, `T1`)
- Installation of *faked* small caps font shapes:
  - checks if `8a`-encoded base font in *upright* shape exists
  - installs virtual fonts for *small caps* encoding variants (`OT1c`, `T1c`)
  - raw font encodings provide standard glyphs: `/A.../Z, /a.../z`
  - target encodings request small caps glyphs: `/Asmall.../Zsmall`
  - `latin.mtx` contains `\setglyph` commands to fake small caps glyphs

Summary of `\latinfamily` (I)

```
% upright shape
\transformfont  {<font>8r} {\reencodefont{8r}{\fromafm{<font>8a}}}
\installrawfont {<font>8r} {<font>8r,8r}     {8r}  {8r} {<fam>}{<series>}{n}{}

\installfont {<font>7t} {<font>8r,latin}     {OT1} {OT1}{<fam>}{<series>}{n}{}
\installfont {<font>8t} {<font>8r,latin}     {T1}  {T1} {<fam>}{<series>}{n}{}
\installfont {<font>8c} {<font>8r,textcomp}  {TS1} {TS1}{<fam>}{<series>}{n}{}

% italic shape
\transformfont  {<font>i8r}{\reencodefont{8r}{\fromafm{<font>i8a}}}
\installrawfont {<font>i8r}{<font>i8r,8r}    {8r}  {8r} {<fam>}{<series>}{it}{}

\installfont {<font>i7t}{<font>i8r,latin}    {OT1i}{OT1}{<fam>}{<series>}{it}{}
\installfont {<font>i8t}{<font>i8r,latin}    {T1i} {T1} {<fam>}{<series>}{it}{}
\installfont {<font>i8c}{<font>i8r,textcomp} {TS1i}{TS1}{<fam>}{<series>}{it}{}

% slanted shape faked
\transformfont  {<font>o8r}{\slantfont{167}{\frommtxm{<font>8a}}}
\installrawfont {<font>o8r}{<font>o8r,8r}    {8r}  {8r} {<fam>}{<series>}{sl}{}

\installfont {<font>o7t}{<font>o8r,latin}    {OT1} {OT1}{<fam>}{<series>}{sl}{}
\installfont {<font>o8t}{<font>o8r,latin}    {T1}  {T1} {<fam>}{<series>}{sl}{}
\installfont {<font>o8c}{<font>o8r,textcomp} {TS1} {TS1}{<fam>}{<series>}{sl}{}
```

Summary of `\latinfamily` (II)

```
% small caps shape using SC+OsF fonts
\transformfont  {<font>c8r}{\reencodefont{8r}{\fromafm{<font>c8a}}}
\installrawfont {<font>c8r}{<font>c8r,8r}    {8r}  {8r} {<fam>}{<series>}{sc}{}

\installfont {<font>c7t}{<font>c8r,latin}    {OT1} {OT1}{<fam>}{<series>}{sc}{}
\installfont {<font>c8t}{<font>c8r,latin}    {T1}  {T1} {<fam>}{<series>}{sc}{}

% small caps shape faked
\installfont {<font>c7t}{<font>8r,latin}     {OT1c}{OT1}{<fam>}{<series>}{sc}{}
\installfont {<font>c8t}{<font>8r,latin}     {T1c} {T1} {<fam>}{<series>}{sc}{}

% small caps shape standard + expert fonts
\installfont {<font>c9t}{<font>8r,<font>8x,latin} {OT1c} {OT1}{<fam>x}{<series>}{sc}{}
\installfont {<font>c9e}{<font>8r,<font>8x,latin} {T1c}  {T1} {<fam>x}{<series>}{sc}{}

% small caps shape standard + expert + oldstyle fonts
\installfont {<font>c9o}{<font>8r,<font>8x,latin} {OT1cj}{OT1}{<fam>j}{<series>}{sc}{}
\installfont {<font>c9d}{<font>8r,<font>8x,latin} {T1cj} {T1} {<fam>j}{<series>}{sc}{}
```

Understanding the details of font installation (I)

- Installation of `8r`-encoded raw fonts:

```
\transformfont  {<font>8r} {\reencodefont{8r}{\fromafm{<font>8a}}}
\installrawfont {<font>8r} {<font>8r,8r} {8r} {8r}{<fam>}{<series>}{n}{}
```

- What's going on:
  - `\fromafm{<font>8a}` creates "raw" `<font>8a.mtx` and `<font>8a.pl`
  - `\transformfont{<font>8r}` creates "raw" `<font>8r.mtx` and `<font>8r.pl`
  - `<font>8a.mtx`: contains glyph metrics and kerns for accessible glyphs
  - `<font>8r.mtx`: contains glyph metrics and kerns for all available glyphs
  - `\installrawfont{<font>8r}` creates "ligfull" raw font `<font>8r.pl`
  - `8r.etx`: adds TEX-specific input ligatures (dashes, quotes, ligatures)
  - `8r.mtx`: adds kern pairs for accented glyphs, inherited from raw glyphs

Understanding the details of font installation (II)

- Installation of `7t`, `8t` and `8c` virtual fonts:

```
\installfont {<font>7t} {<font>8r,latin}    {OT1} {OT1}{<fam>}{<series>}{n}{}
\installfont {<font>8t} {<font>8r,latin}    {T1}  {T1} {<fam>}{<series>}{n}{}
\installfont {<font>8c} {<font>8r,textcomp} {TS1} {TS1}{<fam>}{<series>}{n}{}
```

- What's going on:
  - `\installraw{<font>xx}` creates “ligfull” virtual font `<font>xx.vpl`
  - `<font>8r.mtx`: contains glyph metrics and kerns for all available glyphs
  - `OT1.etx`, `T1.etx`, `TS1.etx`: defines glyphs to install (or fake if unavailable)
  - `latin.mtx`, `textcomp.mtx`: contains commands to fake unavailable glyphs

```
\setglyph{Asmall}
  \movert{\int{smallcapsextraspace}}
  \glyph{A}{\int{smallcapsscale}}
  \movert{\int{smallcapsextraspace}}
\endsetglyph
```

Understanding the details of font installation (III)

- Installing small caps fonts:

```
\installfont {<font>c7t}{<font>c8r,latin} {OT1} {OT1}{<fam>}{<series>}{sc}{}
\installfont {<font>c7t}{<font>8r,latin}  {OT1c}{OT1}{<fam>}{<series>}{sc}{}
```

- real small caps:
  - SC+OsF fonts include small caps and oldstyle figures
  - AFM files for SC+OsF pretend to provide standard glyphs
  - `OT1.etx` references slots for standard glyphs
  - `<font>c8r.mtx` provides metrics for standard glyphs
  - `latin.mtx` glyph commands for small caps are ignored
- faked small caps:
  - `OT1c.etx` references slots for small caps glyphs
  - `<font>8r.mtx` does not provide metrics for small caps
  - `latin.mtx` defines glyph commands to fake small caps

Understanding the details of font installation (IV)

- Installing small caps using SC+OsF fonts:

```
\installfont {<font>c7t}{<font>8r,unsetalf,<font>c8r,latin}
                         {OT1} {OT1}{<fam>}{<series>}{sc}{}
```

- real small caps (non-standard installation):
  - `OT1.etx` references slots for standard glyphs
  - `<font>8r.mtx` provides metrics for standard glyphs
  - `unsetalf.mtx` unsets letters, keeping numerals and symbols
  - `<font>c8r.mtx` provides metrics for standard glyphs (again!)
  - letters (capitals and small caps) are filled in from `<font>c8r.mtx`
  - `latin.mtx` glyph commands for small caps are ignored

Understanding the details of font installation (V)

- Installing small caps using expert fonts:

```
\installfont {<font>c9t}{<font>8r,<font>8x,latin}
                        {OT1j} {OT1}{<fam>x}{<series>}{sc}{}
\installfont {<font>c9o}{<font>8r,<font>8x,latin}
                        {OT1cj}{OT1}{<fam>j}{<series>}{sc}{}
```

- standard + expert:
  - `OT1c.etx` references slots for small caps glyphs
  - `<font>8r.mtx` provides metrics for standard glyphs
  - `<font>8x.mtx` provides metrics for small caps glyphs
  - `latin.mtx` glyph commands for small caps are ignored
- standard + expert + oldstyle:
  - `OT1cj.etx` references slots for small caps and oldstyle figs
  - `<font>8r.mtx` provides metrics for standard glyph
  - `<font>8x.mtx` provides metrics for small caps and oldstyle figs
  - `latin.mtx` glyph commands for small caps are ignored

Understanding the details of font installation (VI)

- Installing small caps using expert *and* SC+OsF fonts:

```
\installfont {<font>c9t}{kernoff,<font>8r,<font>8x,kernon,
                         glyphoff,<font>c8r,glyphon,resetsc,latin}
                        {OT1c} {OT1}{<fam>x}{<series>}{sc}{}
\installfont {<font>c9o}{kernoff,<font>8r,<font>8x,kernon,
                         glyphoff,<font>c8r,glyphon,resetosf,resetsc,latin}
                        {OT1cj}{OT1}{<fam>j}{<series>}{sc}{}
```

- Non-standard installation:
  - `OT1c.etx`, `OT1cj.etx` reference slots for small caps and oldstyle figs
  - `<font>8r.mtx` provides metrics for standard glyphs
  - `<font>8x.mtx` provides metrics for small caps glyphs
  - `<font>c8r.mtx` provides kern pairs between uppercase and small caps
  - `kernoff.mtx`, `kernon.mtx` disables and restores `\setkern`
  - `glyphoff.mtx`, `glpyh.mtx` disables and restores `\setrawglyph`
  - `resetsc.mtx`, `resetosf.mtx` reshuffels metrics to SC+OsF glyph names
  - `latin.mtx` glyph commands for small caps are ignored

Perl front-ends for *fontinst*

- Perl utilities available from `CTAN:fonts/psfonts/tools`
  - `make-fam.pl` generates font metrics for complete typeface families
  - automatically creates temporary TeX files used as *fontinst* control files
  - invokes TeX to run *fontinst* in a temporary directory
  - converts generated PL and VPL files to TFM and VF files
  - generates font map files for `dvips` and `pdftex`
  - installs generated files in CTAN-ready directory structure
- Syntax of `make-fam.pl` and `make-one.pl`:

```
> perl make-fam.pl [-options] [-expert <variant>] <family>
> perl make-one.pl [-options] <font>
```

- Examples of `make-fam.pl` and `make-one.pl`:

```
> perl make-fam.pl pmn                  # standard family
> perl make-fam.pl -expert x pmn        # expertized family
> perl make-fam.pl -expert j pmn        # oldstyle family
> perl make-one.pl pmnrp                # single ornaments font
```

Some little-known *fontinst* tricks

- `\NOFILES` command:
  - turns `\transformfont` and `\installfont` commands into no-ops
  - causes dummy files to be created for all file output commands
  - may be used to diagnose which commands are issued from `\latinfamily`
  - may be used to diagnose which files are created in a normal run
- `fontinst.rc` configuration file:
  - may contain extra commands read at the end of `fontinst.sty`
  - may be used to redefine the raw font encoding: `\def\raw_encoding{8y}`
  - may be used to redefine the list of series and shapes for `\latinfamily`
  - may be used to redefine internals of `\latinfamily`

Font installation step by step (I)

Step 1: Installing and renaming AFM files

- Example: Adobe Palatino (Package #001)

```
pplb8a    Palatino-Bold                   A    001    pob_____
pplbi8a   Palatino-BoldItalic             A    001    pobi____
pplri8a   Palatino-Italic                 A    001    poi_____
pplr8a    Palatino-Roman                  A    001    por_____
```

- Rename distributed `AFM` (and `PFB`) files:

```
POR_____.AFM  ->  pplr8a.afm
POI_____.AFM  ->  pplri8a.afm
POB_____.AFM  ->  pplb8a.afm
POBI____.AFM  ->  pplbi8a.afm
```

- Install renamed `AFM` (and `PFB`) files:

```
> cp *.afm $TEXMF/fonts/afm/adobe/palatino/
> cp *.pfb $TEXMF/fonts/type1/adobe/palatino/
```

- Don't forget to run `texhash` or `mktexlsr` !

Font installation step by step (II)

Step 2: Running *fontinst*

- Manual installation:
  - Create *fontinst* control file:

```
% file: fontppl.tex
\input fontinst.sty
\latinfamily{ppl}{}
\bye
```

  - Run *fontinst* from the command line:

```
> fontinst fontppl.tex
> tex -progname=fontinst fontppl.tex
```

- Automatic installation:
  - Call Perl front-end from the command line:

```
> perl make-fam.pl -outdir $OUTDIR/adobe/palatino ppl
```

Font installation step by step (III)

Step 3: Installing generated font metrics

- Manual installation:
  - Generated PL and VPL must be converted to TFM and VF files:

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

  - Install TFM and VF files:

```
> cp *.tfm $TEXMF/fonts/tfm/adobe/palatino/
> cp *.vf  $TEXMF/fonts/vf/adobe/palatino/
```

  - Don't forget to run `texhash` or `mktexlsr` !
- Automatic installation:
  - Converted TFM and VF files are left in CTAN-ready directory structure:

```
$OUTDIR/adobe/palatino/tfm/*.tfm
$OUTDIR/adobe/palatino/vf/*.vf
```

  - Directories can be moved to TDS directory structure:

```
> mv $OUTDIR/adobe/palatino/tfm/ $TEXMF/fonts/tfm/adobe/palatino/
> mv $OUTDIR/adobe/palatino/vf/  $TEXMF/fonts/vf/adobe/palatino/
```

Font installation step by step (IV)

Step 4: Setting up `dvips` and `pdftex`

- Font map file `psfonts.map` specified in `config.ps` or `pdftex.cnf`
- Entries for `8r`-encoded raw fonts:

```
pplr8r  Palatino-Roman        "TeXBase1Encoding ReEncodeFont" <8r.enc
pplri8r Palatino-Italic       "TeXBase1Encoding ReEncodeFont" <8r.enc
pplb8r  Palatino-Bold         "TeXBase1Encoding ReEncodeFont" <8r.enc
pplbi8r Palatino-BoldItalic   "TeXBase1Encoding ReEncodeFont" <8r.enc
pplro8r Palatino-Roman        ".167 SlantFont TeXBase1Encoding ReEncodeFont" <8r.enc
pplbo8r Palatino-Bold         ".167 SlantFont TeXBase1Encoding ReEncodeFont" <8r.enc
```

- Entries for `8y`-encoded raw fonts:

```
pplr8y  Palatino-Roman        "TeXnANSIEncoding ReEncodeFont" <texnansi.enc
pplri8y Palatino-Italic       "TeXnANSIEncoding ReEncodeFont" <texnansi.enc
pplb8y  Palatino-Bold         "TeXnANSIEncoding ReEncodeFont" <texnansi.enc
pplbi8y Palatino-BoldItalic   "TeXnANSIEncoding ReEncodeFont" <texnansi.enc
pplro8y Palatino-Roman        ".167 SlantFont TeXnANSIEncoding ReEncodeFont" <texnansi.enc
pplbo8y Palatino-Bold         ".167 SlantFont TeXnANSIEncoding ReEncodeFont" <texnansi.enc
```

III Overview of math fonts

- Text fonts vs. math fonts
- Choices of math font sets for TeX
- Why are math fonts so difficult?
- Summary and details of the old 7-bit math font encodings
- Problems of the old 7-bit math font encodings
- Design goals for new 8-bit math font encodings
- Summary and details of new 8-bit math font encodings
- Design goals for new 16-bit math font encodings

Text fonts vs. math fonts

- Text fonts:
  - 7-bit Computer Modern is still used by default
  - switching font families is no problem with LaTeX 2_ε (or ConTeXt)
  - switching encodings (`OT1`/`T1`/`LY1`) is no problem either
  - metrics for common PostScript fonts are available from CTAN
  - metrics for other fonts can be prepared with `fontinst`
  - many thousands of text fonts exist in Type 1 format
- Math fonts:
  - 7-bit Computer Modern is difficult to change
  - very few sets of math fonts are available for use with TeX
  - each math font set uses different encoding variants
  - each math font set requires different macro packages

Choices of math font sets for TEX

- METAFONT font sets:
  - Computer Modern + AMS symbols (also as Type 1 fonts)
  - Concrete + AMS Euler
  - Concrete Math
  - Belleek (MathTime replacement)
- PostScript Type 1 font sets:
  - Lucida Bright + Lucida New Math (Y&Y Inc.)
  - Times + MathTime + Adobe MathPi (Y&Y Inc.)
  - Times + Mathematica
  - TM-Math, HV-Math, IF-Math (MicroPress Inc.)
  - SMF Baskerville
- stop-gap solutions (hacks):
  - `mathptm`: Times + Adobe Symbol + CM
  - `mathppl`: Palatino + CM
  - `mathpple`: Palatino + AMS Euler

Why are math fonts so difficult? (I)

- glyph set / encoding considerations:
  - math fonts include many symbols not available from text fonts
  - math fonts don't include text symbols which do not make sense
  - alignment and spacing of math formulas underlies special rules
  - math fonts include Latin and Greek alphabets in many different styles
  - font styles of math alphabets attach a special meaning to symbols
  - font styles of math alphabets do not depend on typographical context
  - letters in math formulas are set as symbols, not word-components
- design considerations:
  - design and spacing of math italic may be different from text italic
  - alignment of symbols on the math axis requires special care
  - placement of math accents requires special care

Why are math fonts so difficult? (II)

- TEXnical considerations:
  - TEX interprets glyph metrics of math fonts in a peculiar way
  - TFM width denotes position where subscripts are attached
  - italic correction denotes position where superscripts are attached
  - actual glyph width = TFM width + italic correction + sidebearings
  - pseudo kern pairs with `\skewchar` control placement of math accents
  - math fonts are organized into math families (no more than 16!)
  - math fonts must have special `FONTDIMEN` parameters
  - `FONTDIMEN`s control placement of subscripts and superscripts
  - `FONTDIMEN`s control spacing of fractions, radicals and big operators
  - glyph height of radicals determines rule thickness of bar
  - big radicals must be designed to hang below baseline
  - big delimiters and operators are centered on the math axis
  - big delimiters and operators may be designed to be centered
  - however: TFM format imposes limit of 16 heights + 16 depths

Summary of the old 7-bit math font encodings

- **Plain TeX or LaTeX base:** 4 math families
  - Math operators (`OT1`, `7t`, `cmr`, `\fam0`)
  - Math Letters (`OML`, `7m`, `cmmi`, `\fam1`)
  - Math Symbols (`OMS`, `7y`, `cmsy`, `\fam2`)
  - Math eXtension (`OMX`, `7v`, `cmex`, `\fam3`)
- **with LaTeX symbols:** 5 math families
  - LaTeX Symbols (`U`, `lasy`)
- **with AMS symbols:** 6 math families
  - AMS Symbols A (`U`, `msam`)
  - AMS Symbols B (`U`, `msbm`)
  - additional math alphabets (optional)

Details of the old 7-bit math font encodings (I)

- Math operators (`OT1`, `cmr`, `\fam0`)
  - upright digits (used as default digits in math)
  - upright Latin alphabets (`\mathrm`), upright Greek capitals
  - some symbols (‘$+$’, ‘$=$’) and delimiters
- Math Letters (`OML`, `cmmi`, `\fam1`)
  - oldstyle digits (not needed in math)
  - italic Latin alphabets (`\mathnormal`), italic Greek alphabets
  - symbols and punctuation for kerning
- Math Symbols (`OMS`, `cmsy`, `\fam2`)
  - calligraphic letters (`\mathcal`)
  - most symbols and delimiters
- Math eXtension (`OMX`, `cmex`, `\fam3`)
  - extensible delimiters
  - big operators, wide accents

Details of the old 7-bit math font encodings (II)

- LaTeX Symbols (U, `lasy`)
  - LaTeX 2.09 symbol complement
- AMS Symbols A (U, `msam`)
  - AMS symbols and relations
- AMS Symbols B (U, `msbm`)
  - AMS symbols and negated relations
  - Blackboard Bold (`\mathbb`)
- additional math alphabets (optional)
  - Fraktur alphabet (`\mathfrak`)
  - Script alphabet (`\mathscr`)

Problems of the old 7-bit math font encodings

- 7-bit encodings (valuable slots wasted)
- multitude of different encodings
- inter-dependencies between text and math
- mathematical symbols taken from text fonts (e.g. Greek capitals from `OT1`)
- non-mathematical symbols in math fonts (e.g. oldstyle digits in `OML`, '¶','§' in `OMS`)
- some symbols used for multiple purposes (e.g. '$=$' in '$\Longrightarrow$'; '$=$'/'$\Rightarrow$' from `OT1/OMS`)
- building blocks for long arrows split across different encodings (`\joinrel` kerning)
- no kerning between upper/lowercase Greek
- no upright lowercase Greek alphabet available
- TEX-specific symbols (e.g. lowered '$\surd$' in `OMS`) may cause problems with non-TEX software

Design goals for new 8-bit math font encodings

- 8-bit encodings (for conventional 8-bit TEX)
- use one consistent encoding for all font sets
- compatibility with LATEX or AMS within 4 or 6 families
- maybe add some frequently-requested new symbols
- if possible, add slots for multiple uses of symbols
- if possible, add slots for constructed symbols
- separate geometric and Humanist (‘shapy’) symbols
- keep all letter-like symbols together (design similarity)
- keep symbols of similar design or similar type together
- keep TEX-specific symbols together (technical requirements)
- take availability of symbols in different font sets into account
- don’t go too far beyond symbols available in existing font sets
- avoid problems with dumb software, reserve special slots

Summary of proposed new 8-bit math font encodings

- **LaTeX compatibility:** 4 math families
  - Math operators (`T1`, `\fam0`)
  - Math Core (`MC`, `\fam2`)
  - Math Symbols Primary (`MSP`, `\fam1`)
  - Math eXtension Primary (`MXP`, `\fam3`)
- **AMS compatibility:** 6 math families
  - Math Symbols 1 (`MS1`)
  - Math Symbols 2 (`MS2`)
- **additional features (optional):**
  - Math eXtension 1 (`MX1`)

Details of proposed new 8-bit math font encodings (I)

- Math operators (`T1`, `\fam0`)
  - upright Latin alphabets (`\mathrm`)
- Math Core (`MC`, `\fam2`)
  - upright digits (default)
  - italic Latin alphabets (`\mathnormal`)
  - upright and italic Greek alphabets
  - delimiters and punctuation for kerning
  - Humanist symbols, Hebrew letters, etc.
- Math Symbols Primary (`MSP`, `\fam1`)
  - Calligraphic or Script alphabets
  - geometric symbols (`OT1`, `OML`, `OMS`)
  - LaTeX symbols + selected AMS symbols
- Math eXtension Primary (`MXP`, `\fam3`)
  - symbols with special properties
  - extensible symbols (`OMX`, `OMS`)

Details of proposed new 8-bit math font encodings (II)

- Math Symbols 1 (`MS1`)
  - Blackboard Bold (`\mathbb`)
  - remaining AMS symbols
- Math Symbols 2 (`MS2`)
  - Fraktur letters (`\mathfrak`)
  - arrow construction kit (experimental)
- Math eXtension 1 (`MX1`)
  - new extensible symbols
  - variable area for additional sizes

Design goals for new 16-bit math font encodings

- 16-bit encodings, designed for use with MathML / Unicode
- should include *all* symbols collected by the STIX project
- requires extended TEX engine: Omega, ee-TEX, maybe NTS
- design should not be limited by artificial TEX constraints
- design should be orthogonal as much as possible
- building blocks of 8-bit code pages organized by type of symbols
- building blocks may be used in virtual fonts for 8-bit math fonts
- implementation of 16-bit math fonts still work in progress
- encodings for 8-bit math fonts may be decided afterwards

References (I)

- TEX-FONTS mailing list:
  mailto:tex-fonts-requests@math.utah.edu
- *fontinst* mailing list:
  mailto:fontinst-request@tex.ac.uk
- *fontinst* Homepage:
  http://www.tug.org/applications/fontinst/
- Berry font naming scheme:
  http://www.ctan.org/tex-archive/info/fontname/
- *fontinst* distribution (v 1.8xx):
  http://www.ctan.org/tex-archive/fonts/utilities/fontinst/
- *fontinst* pre-release (v 1.9xx):
  http://www.ctan.org/tex-archive/fonts/utilities/fontinst-prerelease/
- PostScript font tools (Perl front-end for *fontinst*):
  http://www.ctan.org/tex-archive/fonts/psfonts/tools/
- PostScript font metrics (generated using *fontinst*):
  http://www.ctan.org/tex-archive/fonts/psfonts/

References (II)

- Adobe Type Homepage:
  `http://www.adobe.com/type/main.html`
- Adobe Type Browser:
  `http://www.adobe.com/type/browser/main.html`
- Adobe Technical Notes for Developers:
  `http://partners.adobe.com/asn/developer/technotes.html`
- PostScript Language Reference Manual:
  `http://partners.adobe.com/asn/developer/PDFS/TN/PLRM.pdf`
- Portable Document Format (PDF) Reference Manual:
  `http://partners.adobe.com/asn/developer/PDFS/TN/PDFSPEC.pdf`
- Adobe Type 1 Font Format:
  `http://partners.adobe.com/asn/developer/PDFS/TN/T1_SPEC.pdf`
- Microsoft Typography Homepage:
  `http://www.microsoft.com/typography/default.asp`
- TrueType Font Specification:
  `http://www.microsoft.com/typography/tt/tt.htm`