LaTeXのフォントの話

私立文系初級ユーザー

2011 年 10 月 31 日

概　要

(La)TeX でのフォントの扱いについての覚え書きです。取り上げるのは本文フォントについてのみで，しかも，欧文フォントについての話が主です。数式フォントについては分かりません（すいません）。和文フォントについては，最後のほうで少しだけ触れています。フォントのインストール方法に関しては既に多くの解説が存在するので，本文書ではもっと全体的な仕組みについてのザックリとした把握を目指します。既存のフォントパッケージに含まれている各種ファイルの役割を理解したり手持ちのフォントを (La)TeX で使うことができるようになるための，基礎知識の確認をします。

1　はじめに

まず，前提として，私の手許の TeX は角藤先生の W32TeX です（したがって，OS は Windows です）。私が LaTeX を使うようになったのはここ数年のことですので，METAFONT で解像度ごとに pk フォントを作って…というような頃の話は全然分かりません。使用するフォントとしては，PostScript か TrueType を考えています。また，最終的にはデータは dvipdfmx（または pdfTeX）で pdf にするものとします。pdf ファイルだと大体誰でも閲覧できるということと，dviout と dvips と dvipdfmx とでは，dvipdfmx の map ファイルの書き方がとても簡単で，フォントの埋め込みも容易だというのが pdf にしている理由です。なお，dvi → ps → pdf のように一旦 PostScript を経由することは本文書では想定していません。

さて，元々 TeX で使うようにパッケージ化されたフォントであれば，それを導入するのはそれほど難しくありませんよね。基本的には同梱されているドキュメントに従えばいいわけですが，入手したアーカイブを展開して，出来たファイル群（tfm，vf，fd，map，sty[1]）を TDS（TeX Directory Structure）に則って（$texmf-local/ 以下に）配置して，あとは（普通は updmapを使って）map ファイルの更新をすれば，インストール完了です。実フォント自体は別に入手する（購入する）必要がある場合も多いですし[2]，実フォントをどこにインストールしておくべきかは texmf.cnf ファイルの設定次第[3]など，多少の注意は必要ですけれど。

パッケージに含まれている tfm ファイル，vf ファイル，fd ファイル，map ファイル，sty ファイルの働きについて理解できるようになれば，それらを改変したり，手持ちのフォント用に必要なファイルを自作したりすることも容易になります（既存の各種解説に従えば，仕組みを理解せずにフォントをインストールすることも実際は可能ではありますが）。

更に，LaTeX 2ε におけるフォントの扱いについて理解するには，NFSS についてや，いわゆる Berry 則についてですとか，エンコーディングについての知識も必要です。

細かな話に入る前に，ごく大まかに整理をすると，TeX におけるフォントの扱いというのは，3 段階に分けることが出来るかと思います。①まず，「TeX にとってのフォント」というのは tfm ファイルのことですよね。したがって，既存の（TeX 用に用意されたのではない）フォントを TeX で使おうという場合には，何らかの方法で当該フォント用の tfm ファイルを作成することになります。②次に，TeX でそのフォントを使うには，原稿ファイルでプリミティブの \font コマンドを使って，当該 tfm ファイルをロードします。これで，TeX レベルでは話が済んで，TeX で原稿を処理すると無事 dvi ファイルが出来ることになります。③最後に，生成した dvi ファイルを閲覧したり印刷したり出来るようにするのは，dviware/driver の役目です。そのためには，dviware/driver は，「dvi ファイルの中に書き込まれている tfm ファイル名」と「実フォントファイル」とを対応付ける（mapping する）必要があって，通常は，dviware/driver ごとの map ファイルでその設定を行います[4]。

---

(1) パッケージに含まれるファイルはこの 5 種類に限られるということではありません。フォントそのものが含まれているパッケージであれば，例えば PostScript フォントなら afm ファイルや pfb ファイルなども同梱されているでしょう。

(2) 既存の欧文フォントを TeX で使うためのパッケージを集めた web page で有名どころとしては，*Walter Schmidt* さんの HP（http://home.vrweb.de/~was/fonts.html）がありますが，そこに挙げられているのは商用の PostScript フォントなので，フォント自体は購入する必要があります。他方 The LaTeX Font Catalogue（http://www.tug.dk/FontCatalogue/）にはフリーのフォントのパッケージが集められています。

(3) 例えば，PostScript フォントの置き場所は “T1FONTS”，TrueType フォントなら “SYSTTF” などの変数に設定されています。

(4) 今述べた概要には，エンコーディングの話も，NFSS の話も出てきませんが，実際には，既存のフォントと TeX とではエンコーディングが異なるのでエンコーディングを変換する必要がありますし，また，プリミティブの \font を直接使うやり方だと使いづらいので，NFSS に沿って tfm ファイルをロード出来るように fd ファイルを作成したり，“\xxxfamily” とか “\textxxx” のようなコマンドが使えるように sty ファイルで設定しておく場合が多いです。map ファイルでの tfm 名と実フォントとの mapping についても，実際には，「vf の参照先の tfm 名」と「enc ファイルで reencoding した実フォント」とを mapping する，というケースがほとんどです。

以下，順番に見ていきますが，私の力不足から，相互に関連する事項をうまく切り分けることができていませんので，あとから触れる事柄が説明なしに出てくる場合が随分とあります。そのようなときには，全体に目を通してからもう一度戻ると，理解がしやすいかも知れません。

2　Berry 則について

いきなり Berry 則から説き起こすのは変に思われるかも知れませんけれど，LaTeX の欧文フォント関係のドキュメントを初めて読む場合に混乱するのが，呪文のような Berry 則と NFSS の記号の関係ではないかと思います（少なくとも私はそうでした）。しかし，ひとたび理解できてしまえばとても便利ですし，本文書での説明でも使おうと思いますので，まずは Berry 則の話から始めます。

- いわゆる Berry 則というのは，フォントファイル（`tfm`，`vf`，`pfb` など）の命名ないし rename の規則で，
- 他方 NFSS の記号は，フォントの属性を 5 つに分類して，フォント指定コマンドでその属性を表わすものです。
- そして，Berry 則で命名された `tfm` ファイルと NFSS の 5 つの属性の組み合わせとを結びつけているのが，（普通は `fd` ファイルの中で用いられている）`\DeclareFontShape` というコマンドです。

サンプルとして，X という会社の Oops という名前の架空の PostScript フォントの設定について考えてみます。Oops ファミリーには 10，11，12 の design size が用意されていて，そのファイル名が `oops___10.pfb`，`oops___11.pfb`，`oops___12.pfb` であるとします[5]。

これらのファイルをそれぞれまず Berry 則にのっとって `xoor8a10.pfb`，`xoor8a11.pfb`，`xoor8a12.pfb` に rename します[6]。

それから何らかの方法でこれらに対応する `tfm` ファイルを作れたとして，その名前を `xoor8t10.tfm`，`xoor8t11.tfm`，`xoor8t12.tfm` にしたと考えます[7]。

これらの `tfm` ファイルを，（仮に `T1` エンコーディングで）NFSS にしたがって LaTeX 2ε でロードするには，普通は “`t1xoo.fd`” という `fd` ファイルの中で，

```
\DeclareFontFamily{T1}{xoo}{}
\DeclareFontShape{T1}{xoo}{m}{n}{<10> xoor8t10
                              <11> xoor8t11
                              <12> xoor8t12}{}
```

のように設定します。ここで赤い字の部分が NFSS による属性の表現で，青い字が Berry 則による `tfm` 名です。

これは，フォントの属性の組み合わせが “`T1/xoo/m/n/10`” のときには `xoor8t10.tfm` をロードして，“`T1/xoo/m/n/11`” なら `xoor8t11.tfm`，“`T1/xoo/m/n/12`” という組み合わせだと `xoor8t12.tfm` をロードする，ということを意味しています。

NFSS については後述しますが，`\DeclareFontShape` の最初の 4 つの引数はそれぞれ `encoding`，`family`，`series`，`shape` という属性を表わし，5 つ目の引数の中で `< >`で囲まれているのが `size` です。今の例では，`encoding` を表わす記号は “`T1`”（TeX extended text, 8bit）で，`family` は “`xoo`”，`series` が “`m`”（medium）で `shape` が “`n`”（normal, i.e., upright, roman），そして `size` が “`10pt`”，“`11pt`”，“`12pt`” ということになります。

NFSS の表現で “`T1`” は Berry 則では “`8t`”（8bit TeX Text）と表わされ，NFSS でのファミリ名 “`xoo`” は Berry 則でも “`xoo`” のまま，NFSS の “`m`” は Berry 則では “`r`”（regular）となり，そして NFSS での “`n`” に相当する記号は Berry 則では省略されています[8]。

どうしてこんなに複雑な名前に命名/rename しなくちゃならないのか不思議ですが，その最大の理由は，TeX に関係するファイルの互換性・可搬性の確保ということのようです。ファイル名が 8 文字に制約されている OS でも，フォントファイル名が同じになるように rename するために Berry 則は作られたらしいです。しかし，そうだとしますと，ファイル名が 8 文字に制限されない OS ならば，Berry 則に拘る必要はないということにもなります。実際，Berry 則に従わないと LaTeX 2ε でフォントを扱えないということはまったくありません[9]。それでも，Berry 則を使うとフォントの管理がしやすい[10]ということもありますし，また，既存のフォントパッケージの多くは Berry 則に従って作られているのでそれらのドキュメントを読むためにも，ユーザー個人としてフォント名に Berry 則を採用するか否かは別にして，おおよその規則は知っておいたほうがいいと思われます。

Berry 則自体は，既存のあらゆるフォント名に対応しようと務めているので，全体では膨大な数の記号がリストアップされていますが，英語とヨーロッパの言語を使う程度で，書体の種類も Roman，Bold，Italic，BoldItalic くらいしか必要ないのであれば，それほど沢山の記号を覚える必要はありません（逆に，ギリシア語やロシア語，東欧の言語等を扱う必要がある方で，Berry 則に従ってフォントを rename した上で自分でインストールしようという方や，fontinst パッケージの能力を最大限に引き出そうなどという方などは，ドキュメント[11]をちゃんと読まないといけません）。

具体例として，PSNFSS バンドルに含まれている Times フォントについて，`tfm` ファイルのファイル名を調べてみましょう。対応する NFSS の記号の組み合わせと比較してみると，Berry 則の命名規則が大体分かるかと思います。

---

(5) PostScript フォントという想定なのに design size ごとにファイルが用意されているというのはおかしいと思われるでしょうけれど，ここでは飽くまで話を分かりやすくするための便法です。

(6) 「Berry 則にのっとって」と言いましたが，実際には（少なくとも本文書執筆時点では）Berry 則には “`x`” という *supplier* 名や “`oo`” という *typeface* 名はありません。

(7) 何の説明もなしにエンコーディングを “`8a`” から “`8t`” に変えていますが，エンコーディングの変換については後述します。

(8) もしも Berry 則や NFSS の表記についてここで初めて目にしたという場合には，かなり困惑されるのではないかと思いますが，NFSS の表現と Berry 則との対応関係の一覧表が *Philipp Lehman*, The Font Installation Guide, 2002–2004 (`CTAN:info/Type1fonts/fontinstallationguide.pdf`) の付録に掲載されています。

(9) Oops フォントについても，例えば，`tfm` 名を “`OopsRoman10.tfm`” としても構わなくて，`\DeclareFontShape` の第 5 引数の中で “`xoor8t10`” の代わりに “`OopsRoman10`” とすれば，属性の組み合わせが “`T1/xoo/m/n/10`” のときに，ちゃんと “`OopsRoman10.tfm`” がロードされます。あとは dviware/driver の `map` ファイルで “`OopsRoman10.tfm`” と “`oops___10.pfb`” とを対応付ければよいので，実フォントのファイル名も実は必ずしも rename する必要はありません。

(10) 例えば Adobe 社の Times-Roman フォントの PostScript ファイル名は，“`tir_____.pfb`” ですので，Berry 則が理解できれば，“`ptmr8a.pfb`” のほうがファイル名に含まれている情報が多くて分かりやすいともいえます。

(11) *Karl Berry*, Fontname, 2009 (`CTAN:info/fontname/fontname.pdf`).

$TEXMF/tex/latex/psnfss/ フォルダの中にある ot1ptm.fd，t1ptm.fd，8rptm.fd，ts1ptm.fd から，\DeclareFontShape の NFSS の組み合わせと，対応する Berry 則による tfm 名を一覧にしてみました（本来 \DeclareFontShape の設定は，NFSS の属性の組み合わせに対してどの tfm ファイルをロードするか，という指定なのですが，ここでは Berry 則に着目しているので，tfm 名のほうを先に挙げて，それと対応する NFSS の組み合わせをその右に並べています．また，PSNFSS バンドルは，PostScript フォントのパッケージなので design size の設定はなく，すべてのサイズにひとつのフォントを拡大縮小して用いるので，NFSS の 5 つめの属性（=size）の部分は記載していません）。

ot1ptm.fd より

tfm 名	対応する NFSS の組み合わせ
ptmr7t	OT1/ptm/m/n
ptmrc7t	OT1/ptm/m/sc
ptmro7t	OT1/ptm/m/sl
ptmri7t	OT1/ptm/m/it
ptmb7t	OT1/ptm/b/n
ptmbc7t	OT1/ptm/b/sc
ptmbo7t	OT1/ptm/b/sl
ptmbi7t	OT1/ptm/b/it

t1ptm.fd より

tfm 名	対応する NFSS の組み合わせ
ptmr8t	T1/ptm/m/n
ptmrc8t	T1/ptm/m/sc
ptmro8t	T1/ptm/m/sl
ptmri8t	T1/ptm/m/it
ptmb8t	T1/ptm/b/n
ptmbc8t	T1/ptm/b/sc
ptmbo8t	T1/ptm/b/sl
ptmbi8t	T1/ptm/b/it

8rptm.fd より

tfm 名	対応する NFSS の組み合わせ
ptmr8r	8r/ptm/m/n
ptmro8r	8r/ptm/m/sl
ptmri8r	8r/ptm/m/it
ptmb8r	8r/ptm/b/n
ptmbo8r	8r/ptm/b/sl
ptmbi8r	8r/ptm/b/it

ts1ptm.fd より

tfm 名	対応する NFSS の組み合わせ
ptmr8c	TS1/ptm/m/n
ptmro8c	TS1/ptm/m/sl
ptmri8c	TS1/ptm/m/it
ptmb8c	TS1/ptm/b/n
ptmbo8c	TS1/ptm/b/sl
ptmbi8c	TS1/ptm/b/it

Berry 則によるファイル名の最初の 1 文字は *supplier* 名の略語で，“p” は Adobe 社を表わし[12]，次の 2 文字は *typeface* 名の略語なので “tm” は Times の意味，その次の 1 文字は *weights* の略語で，“r” は Regular Roman，“b” は Bold を表わしています。

続く 1 文字は *variants* ですが，これは，*variants* がノーマルで *width* もノーマルの場合には，省略されます（NFSS の 4 つめの属性が “n” の場合に対応する tfm 名を参照）。省略されていない場合，“c” は SmallCaps で，“o” は Oblique (slanted)，“i” が Italic の略語です。

そして最後の 2 文字が *encoding* の略語で，“7t” は TeX Text (7bit)，“8t” は CorkEncoding (8bit)，“8r” は TeXBase1Encoding (8bit, raw) で，“8c” が TeX-TextCompanion (8bit) を表わしています。今の例ではファイル名が *encoding* の略語で終わっていますが，フォントによっては，その後に *width* の略語と *design size* の数字が続きます（上の例では PostScript ファイルのため design size ごとにはフォントは用意されていませんので，どの tfm 名にも *design size* の数字はついていませんし，対応する NFSS の属性の組み合わせにも，5 番目の属性 (=size) の記載をしていません）。

ここでは tfm ファイルのファイル名を見てみましたが，vf ファイルも同様の規則で命名され，実フォントファイルも同じ規則にのっとって rename されます。実フォントの場合，PostScript フォントのエンコーディングは “8a”（AdobeStandardEncoding, 8bit）と表記されます。

以上の例に出て来なかった残りの沢山の Berry 則の記号については，前掲註（11）に挙げたドキュメントや，$TEXMF/fonts/map/ フォルダに収められているファイル群を参照してください。

3　tfm ファイルについて

TEX にとっての「フォント」とは，tfm ファイルのことです。TEX が原稿ファイルを処理する際には，フォントに関しては tfm ファイルがあれば十分で，対応する実フォントの有無は関係ありません（組版結果の dvi ファイルを閲覧したり，印刷したりする際には，当然，tfm ファイルに対応する実フォントファイルが必要になりますが，両者を結び付けるのは dviware/driver の仕事です）。tfm ファイルには，各グリフの幅・高さ・深さや，カーニングの情報，特定のグリフが連続した際に置き替えるリガチャの情報，イタリック補正の情報などが収められています（欧文フォントの場合）。したがって，元々 TEX で使うために作られているわけではないフォントを TEX で使おうという場合には，当該フォントについてこれらの情報を抽出して，tfm ファイルの形式に変換する必要があることになります。

tfm ファイルはバイナリファイルなので，そのままではエディタ等で閲覧したり編集したり出来ませんが，tftopl を使うとテキストファイルにすることが出来ます。そして，中味を見たり，編集したりした後は，pltotf でまたバイナリに戻すことが出来ます。日本語用の tfm ファイルは拡張子は同じ tfm ですが内部の構造が欧文 tfm ファイルとは異なり “jfm” ファイルと呼ばれ，使うツールも ptftopl と ppltotf です。

---

(12) Berry 則で *supplier* の “p” は Adobe 社の略語なのですが，Adobe のフォントは商品ですから，実フォントは購入しないと使えません。そのため，PSNFSS バンドルでは例えば ptmr7t.tfm はまず ptmr7t.vf を介して ptmr8r.tfm へと帰着された後，dviware/driver の map ファイルで，utmr8a.pfb（= Times 相当の URW 社のフリーフォント NimbusRomNo9L-Regu, n0210031.pfb を Berry 則にのっとって rename したもの）へと mapping されています（フォントを埋め込む場合）。

(13) 既存の欧文フォントから，対応する tfm ファイルや vf ファイル，fd ファイルを自動的に作り出すパッケージが fontinst ですが，その複雑な使用例が *Alan Hoenig*, TEX Unbound, Oxford University Press 1998, Chaps. 7–9 に載っています。*Hoenig* さんによる Poetica パッケージ (CTAN:fonts/poetica/) やその解説の “The Poetica Family: Fancy Fonts with TEX and LATEX,” TUGboat, Vol. 16 (1995), No.3 (http://www.tug.org/TUGboat/tb16-3/tb48hoen.pdf) もスゴイです。

欧文の PostScript フォントや TrueType フォントから tfm ファイルを生成するためのツールは既にいろいろと開発されていて，複数のフォントを合成してひとつのフォントにする，みたいな複雑なことをするのでなければ[13]，案外簡単に tfm ファイルを作ることができます。

以下では，既存の欧文フォントから対応する tfm ファイルを作成するツールについて，ごく簡単にまとめておきます（fontinst は複雑なので触れません）。

PostScript フォントの場合

PostScript フォントから tfm ファイルを作るツールとしては dvips 付属の **afm2tfm** があります。afm2tfm の解説は dvips のドキュメント[14]の “6.2 Making a PostScript font available” に書かれています。afm2tfm の具体的な使用例については web に沢山情報があります。

afm ファイルがない場合に，pfb ファイルや pfa ファイルから afm ファイルを生成するツールに Ghostscript 付属の **pf2afm** があります。但し，これで作った afm ファイルにはカーニングの情報が含まれていないようなので，この afm ファイルから作った tfm ファイルによる最終的な組み上がり具合は，いまいちかも知れません。

TrueType フォントの場合

TrueType フォントから tfm ファイルを作り出すツールには **ttf2tfm** があります。ドキュメントは $TEXMF/doc/ttf2pk/ にある ttf2pk.doc というファイルです（拡張子が “doc” となっていますが，中味はテキストファイルです）。ttf2tfm の使い方は，afm2tfm とほとんど同じです。具体的な使用例としては，例えば，The LaTeX Graphics Companion: Supplementary Material[15]の “21.5 Using TrueType fonts with pdfTeX” に説明があります（pdfTeX での使用と書いてありますが，map ファイルの書き方以外の作業は他の dviware/driver の場合でも同じです）。

ttf2tfm を使う他には，**ttf2afm** を使って一旦 afm ファイルを作って，それから PostScript の場合と同様に afm2tfm を使うという経路もあります。この方法についても web に沢山例を見つけることが出来ますが，$TEXMF/doc/pdftex/base/ にある “PDFTEX-W32.txt” という角藤先生によるドキュメントには「pdfTeX での TrueType フォント使用法」という記述があり，そこには ttf2afm，afm2tfm，vptovf を角藤先生がひとつにまとめられた **ttftotfm** というツールの使用法が説明されています。

OpenType フォントの場合

OpenType フォントから tfm ファイルを作り出すツールとしては，LCDF-Typetools（http://www.lcdf.org/type/）による **otftotfm** があり，W32TeX にも収録されています[16]。ドキュメントは，LCDF-Typetools の web page にありますが，$TEXMF/doc/lcdf-typetools/base/ にも otftotfm.pdf として含まれています。

afm2tfm や ttf2tfm が作るのは，(raw.)tfm ファイルと vpl ファイルのみで，vf は直接には作られないので，vptovf を使って vpl から vf を作るのですが，otftotfm の場合には，tfm と vf，それから enc ファイルと map ファイルまでが一度に出来ます。

4 NFSS について

よく，「LaTeX 2ε では，フォントは 5 つの要素を有している」ですとか，「NFSS とは，フォントの属性を 5 つに分けて管理する機構である」みたいな説明がされますが，初めてこの種の記述を読んだときには，「フォントを属性に則して管理するなんて当たり前のことを，なんでそんなに大層なことのように書いてあるんだろう」と不思議に思ったものです。

しかし，plainTeX や LaTeX2.09 の時代には，フォント属性の組み合わせに則して tfm ファイルを切り替えるということが出来なかったらしいので，概説書に書かれている説明はその歴史的経緯を踏まえてのことのようです。それがどういうことなのか，簡単に見てみましょう。

プリミティブの \font コマンドは，

\font ⟨*control sequence*⟩ = ⟨*tfm name*⟩

という形で使うと，そこで ⟨*tfm name*⟩ という tfm ファイルをロードして，以後は ⟨*control sequence*⟩ というコマンドによって，使用する tfm ファイルを ⟨*tfm name*⟩ に切り替えることが出来るというものです（“scaled” や “at” を使って拡大縮小を指定することも出来ます）。

ここでまた Oops フォントにご登場願って，例えば，OopsRoman10.tfm，OopsBold10.tfm，OopsItalic10.tfm という tfm ファイルがあるとしましょう。そして，

```
\font\Roman = OopsRoman10
\font\Xii = OopsRoman10 at 12pt
\font\Bold = OopsBold10
\font\Italic = OopsItalic10
```

と設定したとします。このとき，

```
\Roman xxx \Bold\Italic yyy
```

と原稿ファイルに書くと，“xxx” の部分は OopsRoman10 となりますが，それでは “yyy” の部分はボールドイタリックになるかというと，そうではなくて，まず \Bold で OopsBold10 に切り替わり，続いて \Italic で OopsItalic10 に切り替わるので，結局 “yyy” はイタリックにしかなりません。同様に，

```
\Roman xxx \Bold\Xii yyy
```

とした場合には，“yyy” はボールドの 12pt にはならず，ローマンの 12pt となります。試みに，

```
\Roman xxx \Italic\Bold yyy
\Roman xxx \Xii\Bold yyy
```

と順番を変えてみても，今度は，どちらも「10pt のボールド」にしかなりません。

落ち着いて考えれば，そのように設定しているのですからこのようにしかなり得ないのは当然なのですけれど，書体やサイズの変更を組み合わせて指定することが出来ないというのはとても不便です。しかし，\font コマンドで ⟨*control sequence*⟩ と ⟨*tfm name*⟩ とを 1 対 1 に対応させて，⟨*control sequence*⟩ を直接使っている限りでは，この問題は解決できません。

そこで，複数のフォント属性をそれぞれ独立に変更した上でその属性の名前を一旦保存して，それら属性の組み合わせに応じて tfm ファイルをロードすることが出来れば，書体やサイズの変更を組み合わせて指定することが

---

(14) *Tomas Rokicki*, Dvips: A DVI-to-PostScript Translator, 2009 ($TEXMF/doc/dvips/base/dvips.pdf).

(15) *Michel Goossens*, The LaTeX Graphics Companion: Supplementary Material, Version February 17, 2008 (http://xml.web.cern.ch/XML/lgc2/).

(16) ここで挙げたツール類はすべて，W32TeX には Windows で使える状態にしたものが収録されています。W32TeX を使っていますと，web を探し回らなくても必要なツール群のかなりの部分が既に含まれているので，本当にありがたいです。

出来ることになります。それを実現したのが NFSS（New Font Selection Scheme）というわけです[17]。

NFSS では encoding，family，series，shape，size という 5 つの属性でフォントを管理していて，それぞれを個別に，以下のコマンドで変更することができます。

```
\fontencoding{⟨encoding⟩}\selectfont
\fontfamily{⟨family⟩}\selectfont
\fontseries{⟨series⟩}\selectfont
\fontshape{⟨shape⟩}\selectfont
\fontsize{⟨size⟩}{⟨baselineskip⟩}\selectfont
```

\usefont コマンドを使うと，⟨size⟩ 以外の 4 つの属性をまとめて指定することが出来て，\selectfont も省略出来ますよね。

```
\usefont{⟨encoding⟩}{⟨family⟩}{⟨series⟩}{⟨shape⟩}
```

なお，⟨size⟩ 以外のフォント属性については，予め宣言済みの記号でないと指定は出来ません。⟨encoding⟩ は \DeclareFontEncoding コマンドによって，⟨family⟩ は \DeclareFontfamily で，そして ⟨series⟩ と ⟨shape⟩ については \DeclareFontShape を使って，予め宣言されている記号が，上記のコマンドで属性として指定することが出来ます（逆に言えば，予め宣言さえされていれば，どんな記号でも NFSS の記号として指定可能です）。

通常，\DeclareFontEncoding は “⟨encoding⟩enc.def” というファイルの中で使用され，\DeclareFontFamily と \DeclareFontShape は “⟨encoding⟩⟨family⟩.fd” ファイルの中で使用されます。

“⟨encoding⟩enc.def” ファイルは，原稿ファイルに直接 \input するか，又は fontenc パッケージで読み込みますが，一般のユーザーが全く新しいエンコーディングを宣言する必要はほとんどないでしょうから，\DeclareFontEncoding コマンドや def ファイルの中味についてはここでは割愛します（“U” (= Unknown or Unclassified) というエンコーディングが予め宣言されていて，それが fmt ファイル作成段階で読み込まれているので，デフォルトで def ファイルが用意されてないようなエンコーディングについては，この “U” を使うことでひとまずしのげます）。

使用フォントについての \DeclareFontFamily コマンドが原稿ファイルのプリアンブルに存在しないと，LaTeX は “⟨encoding⟩⟨family⟩.fd” というファイルを探すので，普通はこの fd ファイルの中で，\DeclareFontFamily と \DeclareFontShape とを使って，⟨family⟩，⟨series⟩，⟨shape⟩ が宣言されています。

新たな ⟨family⟩ は，宣言済みの ⟨encoding⟩ と組にして，

```
\DeclareFontFamily {⟨encoding⟩} {⟨family⟩}
                                {⟨loading-settings⟩}
```

で宣言され，⟨loading-settings⟩ は，大抵は空のママです。

そして，“encoding/family/series/shape/size” の組み合わせに対してどの tfm ファイルをどういうサイズでロードするかを決めているのが \DeclareFontShape コマンドということになります。

```
\DeclareFontShape {⟨encoding⟩} {⟨family⟩} {⟨series⟩}
          {⟨shape⟩} {⟨loading-info⟩} {⟨loading-settings⟩}
```

こちらも普通は，⟨loading-settings⟩ は空のママです。⟨loading-info⟩ の部分で，⟨size⟩ と tfm ファイル名とを指定しますが，サイズ指定を “<->” とすると，すべてのサイズに対して当該 tfm ファイルが使用されます（逆に，“< >” で特定のサイズを指定してしまうと，\fontsize コマンドによるサイズ指定が効かなくなってしまいます）[18]。

\DeclareFontShape の ⟨loading-info⟩ の部分ではかなり細かな設定が出来るのですが[19]，ここではフォントの代替についてのみ取り上げます。例として t1ptm.fd ファイルの最後の 4 行を見てみましょう（ここでは改行しているために 8 行になってしまいました）。

```
\DeclareFontShape{T1}{ptm}{bx}{n}
                              {<->ssub * ptm/b/n}{}
\DeclareFontShape{T1}{ptm}{bx}{sc}
                              {<->ssub * ptm/b/sc}{}
\DeclareFontShape{T1}{ptm}{bx}{sl}
                              {<->ssub * ptm/b/sl}{}
\DeclareFontShape{T1}{ptm}{bx}{it}
                              {<->ssub * ptm/b/it}{}
```

⟨loading-info⟩ の中の “<->ssub *” の部分は，当該 family/series/shape の組み合わせについてはすべてのサイズに関して，“*” の右側の family/series/shape の組み合わせで代替する，という意味です（silent substitution; “silent” というのは TeX での処理時に warning を表示しないということ）。つまりこの場合は，いずれも series が “bx” であるような属性の組み合わせについては，series が “b” の組み合わせで代替するという設定になっています[20]。

NFSS の記号としてよく使われるのは，⟨encoding⟩ については，“OT1” (TeX text)，“T1” (TeX extended Text; CorkEncoding)，“TS1” (TextCompanionEncoding)，“LY1” (Y&YEncoding; texnansi)，“8r”(TeXBase1Endocing) など，⟨series⟩ については，“m” (medium)，“b” (bold)，“bx” (bold extended)，そして ⟨shape⟩ については，“n” (normal; upright, roman)，“it” (italic)，“sl” (slanted; oblique)，“sc” (small caps) くらいでしょうか。

さて，またしても Oops フォントを例に考えてみましょう。まず，プリアンブルで

```
\usepackage[T1]{fontenc}
\DeclareFontFamily{T1}{xoo}{}
\DeclareFontShape{T1}{xoo}{m}{n}{<->OopsRoman}{}
\DeclareFontShape{T1}{xoo}{b}{n}{<->OopsBold}{}
\DeclareFontShape{T1}{xoo}{m}{it}
                                {<->OopsItalic}{}
\DeclareFontShape{T1}{xoo}{b}{it}
                              {<->OopsBoldItalic}{}
```

---

[17] NFSS1 が 1989 年，NFSS2 が 1993 年にリリースされて，LaTeX 2ε は NFSS2 を取り込んだ，と *LaTeX3 Project Team*, LaTeX 2ε font selection, 1995–2005 (fntguide.pdf) に書いてありますけれど，NFSS1 と NFSS2 の違いについては調べていません。

[18] LaTeX のデフォルトの欧文フォントは OT1 エンコーディングの cmr ファミリですが，ot1cmr.fd の中で OT1/cmr/m/n のサイズが <5><6><7><8><9><10><10.95><12><14.4><17.28><20.74><24.88> と指定されているので，\fontsize コマンドで他のサイズを指定してもそのサイズにはならず，この 12 種類のサイズしか使えません。これは元々 plainTeX の時代に 10pt より大きいフォントサイズについては，その 1.2 倍，またその 1.2 倍…と定数倍されたもの（10.95 のみ $\sqrt{1.2}$ 倍）が使われており，LaTeX2.09 でもそれが踏襲された後，互換性を大事にする LaTeX 2ε もそれを引き継いだということのようです。

[19] 詳細については前掲註（17）のドキュメントの “4.4 Size functions” や *Frank Mittelbach and Michel Goossens*, The LaTeX Companion 2nd edition, Addison-Wesley 2004 の “7.10.3 Declaring new font families and font shape groups” などをご覧ください。

[20] \bfseries や \textbf が参照している \bfdefault が “bx” と設定されているので，そのための手当てです。したがって，\bfdefault を “b” に \renewcommand すれば，代替をしなくても済みます。尤も，代替をしなかった場合には，⟨series⟩ に明示的に “bx” が指定されたときに困りますけれど（いえ，その場合でも \DeclareErrorFont の設定によってフォントは代替されます）。

のように設定をしてから，原稿本文には次のように書いたとします。

```
\usefont{T1}{xoo}{m}{n}
xxx
\fontseries{b}\selectfont
yyy
\fontshape{it}\selectfont
zzz
```

まず，\usefont{T1}{xoo}{m}{n} によって，最初のフォント属性の組み合わせは

T1/xoo/m/n/

なので，“xxx” は OopsRoman で組まれます。次に \fontseries{b} によって，⟨*series*⟩ が “m” から “b” に変更されます：

T1/xoo/b/n/

したがって，“yyy” は OopsBold になります。次いで \fontshape{it} によって，⟨*shape*⟩ が “n” から “it” になります（⟨*series*⟩ は “b” のままです）：

T1/xoo/b/it/

これで，属性の組み合わせが T1/xoo/b/it となったので，\DeclareFontShape の設定により，“zzz” はめでたく OopsBoldItalic で組まれることになります。

“\fontseries{b}\selectfont” や “\fontshape{it}\selectfont” のようなよく使われるコマンドについては，もっとユーザー寄りのコマンドが用意されていますよね。つまり，\bfseries，\itshape ですとか，\textbf，\textit 等々のことです。これらは ltfssini.dtx と ltfntcmd.dtx でそれぞれ，

```
\DeclareRobustCommand\bfseries
      {\not@math@alphabet\bfseries\mathbf
       \fontseries\bfdefault\selectfont}
```

```
\DeclareTextFontCommand{\textbf}{\bfseries}
```

のように定義されていますので，これを真似すれば，新たに “\xxxfamily” とか “\textxxx” みたいなコマンドを作ることが出来ます[21]。例えば，Oops フォントを例にとりますと，

```
\DeclareRobustCommand{\oopsfamily}
                     {\fontfamily{xoo}\selectfont}
\DeclareTextFontCommand{\textoops}{\oopsfamily}
```

と定義すれば，\oopsfamily と \textoops というコマンドが無事使えるようになります。

なお，\bfseries の定義をよく見てみますと，

\fontseries{b}\selectfont

ではなく，

\fontseries\bfdefault\selectfont

となっていることに気付かれると思います。このように，よく使われる ⟨*family*⟩，⟨*series*⟩，⟨*shape*⟩ については，デフォルトが定義されています。⟨*family*⟩ については，\rmdefault，\sfdefault，\ttdefault が定義されており，⟨*series*⟩ については \bfdefault と \mddefault が，そして ⟨*shape*⟩ については \itdefault，\sldefault，\scdefault，\updefault が，既に定義されています。その値は fontdef.dtx（→ fonttext.ltx）において，

```
\newcommand\rmdefault{cmr}
\newcommand\sfdefault{cmss}
\newcommand\ttdefault{cmtt}
\newcommand\bfdefault{bx}
\newcommand\mddefault{m}
\newcommand\itdefault{it}
\newcommand\sldefault{sl}
\newcommand\scdefault{sc}
\newcommand\updefault{n}
```

と定義されていますので，これらを変更する場合には \renewcommand を使います。ここで若干注意が必要な点は，\bfdefault の値が “bx” になっているということです（ですから，実は，デフォルトでは \bfseires は “\fontseries{b}” ではなく “\fontseries{bx}” と同値です）。そのため，新しい Bold フォントを導入する際には，\DeclareFontShape の設定でちゃんと “bx” という ⟨*series*⟩ を定義するか，それとも属性の組み合わせに “bx” が入る場合を “b” で代替しておくか，または \bfdefault を “b” に \renewcommand するかしないと，クラスファイルで \bfseries が指定されている部分が当該 Bold フォントにはならなくなってしまいます。

更に，⟨*encoding*⟩，⟨*family*⟩，⟨*series*⟩，⟨*shape*⟩ 自体のデフォルトもあり，やはり fontdef.dtx で，それぞれ次のように定義されています。

```
\newcommand\encodingdefault{OT1}
\newcommand\familydefault{\rmdefault}
\newcommand\seriesdefault{\mddefault}
\newcommand\shapedefault{\updefault}
```

⟨*encoding*⟩ 以外のデフォルトは，値を直接指定してあるのではなく，\rmdefault，\mddefault，\updefault が参照されていますので，さきほどの Oops フォントの例の冒頭の

\usefont{T1}{xoo}{m}{n}

の部分は，

\renewcommand{\rmdefault}{xoo}

とするだけでよいということも分かります（⟨*encoding*⟩ についてはプリアンブルにおいて fontenc パッケージで T1 に変更してありました）。

以上の諸設定については，LaTeX 2ε の慣例に倣うならば，\DeclareFontFamily と \DeclareFontShape の設定は “t1xoo.fd” というファイルにまとめ，\oopsfamily コマンドや \textoops コマンド，そして \renewcommand{\rmdefault}{xoo} は，“sty” ファイルにまとめることになります（なお，上の例は T1 エンコーディング用の設定なので，\requirepackage[T1]{fontenc} も sty に含めてしまってもいいかも知れません）。特に汎用性は必要ない場合や，ちょっと試してみるというような場合には，上で見ましたように，みなプリアンブルに入れておいても大丈夫です。

---

(21) ここで \DeclareRobustCommand は \newcommand と大体同じですが，定義されたコマンドが \section の引数のような「動く引数」の中で使われる場合でも \protect を前置しなくてもいいようにしてくれます。また，青い字の部分の “\not@math@alphabet” は数式モード内で当該コマンドが使われた際の警告用なので，数式を使わないなら省略しても大丈夫です（または，“\mathbf” に当たる部分を “\relax” としておきます）。

5　エンコーディングについて

既に何度も「OT1（7t）エンコーディング」ですとか「T1（8t）エンコーディング」というような用語を使ってしまっていますが，エンコーディングというのは，文字コード（スロット）とグリフとの対応関係のセットのことです。

欧文の場合，(LA)TEX では 7 ビットか 8 ビットのエンコーディングを使用します。7 ビットのエンコーディングでは 0 番から 127 番までの 128 個（$= 2^7$）のスロットがあり，8 ビットエンコーディングでは，0 番から 255 番までの 256 個（$= 2^8$）のスロットが使えます。

通常の欧文 PostScript フォントは 8 ビットエンコーディングです（ややこしいですが，PostScript フォントに収録され得るグリフ数が 256 個までということではありません．コードを割り当てられるのが 0 から 255 までということです）。TrueType フォントや OpenType フォントでは，256 個よりもずっと多くのグリフが収録されていますので，その分 (LA)TEX で使うには工夫が必要となります。

エンコーディングの種類はとても沢山ありますが[22]，ここでは以下のエンコーディングについてだけ触れることにします。

- TEX Text Encoding (NFSS: OT1; Berry: 7t)
- Cork Encoding (NFSS: T1; Berry: 8t)
- Y&Y 256 Glyph Encoding (NFSS: LY1; Berry: 8y)
- Text Companion Encoding (NFSS: TS1; Berry: 8c)
- Adobe StandardEncoding (NFSS: 8a; Berry: 8a)
- TEX Base 1 Encoding (NFSS: 8r; Berry: 8r)

エンコーディングの表は web 等でもすぐ見つかりますし，$TEXMF/doc/fontinst/base/talks/et99-font-tables.pdf などでも参照することができますのでここには掲載しません。

異なるエンコーディングを比較してみますと，当然，同じ番号のコードに別のグリフが割り当てられている場合がありますが，32 番から 126 番までのスロットについては，英語のアルファベットや数字，記号類が収められていて，どのエンコーディングでもほぼ同じ配置になっているので，英語しか必要ないような場合には，エンコーディングの差に気付かずに使っていることもあるかも知れません。しかし例えば，“ß” というグリフは，OT1 では 25 番ですが，T1 では 255 番で，LY1 と 8r だと 223 番，8a では 251 番ですし，その他，発音区別符号（diacritical mark）付きグリフのコードは，エンコーディングによって異なり得ます。

TEX でデフォルトの OT1 エンコーディングでは，diacritical mark 付きのグリフについては，通常のグリフと diacritical mark を合成することで実現していましたが，その場合にはハイフネーションが出来なくなったり pdf にした際に不都合が生じるため，初めから diacritical mark 付きのグリフを含む T1 エンコーディングや LY1 エンコーディングが作られました。T1 や LY1 はともに PostScript フォントや TrueType フォントを LATEX で使用することを念頭に作られたエンコーディングですが，配置が若干異なります（T1 や LY1 に収録されているグリフが必ず PostScript フォントや TrueType フォントに含まれるというわけでもありません）。TS1 は記号類を収めたエンコーディングです（8bit ですけれど，空きスロットが沢山あります）。

PostScript フォントのエンコーディングである Adobe StandardEncoding（8a）は，256 あるスロットのうち，なぜか 149 スロットにしかグリフが割り当てられていません（スペースを含む）。それでは，PostScript フォントには 149 グリフしか収録されていないのかというと，実はそうではありません。一般的な欧文 PostScript フォントの場合，229 個のグリフが収められています。

例として，Adobe 社の Times-Roman フォントの afm ファイル（“tir_____.afm”）の中味を覗いてみましょう。

```
...
StartCharMetrics 229
C 32 ; WX 250 ; N space ; B 0 0 0 0 ;
C 33 ; WX 333 ; N exclam ; B 130 -9 238 676 ;
C 34 ; WX 408 ; N quotedbl ; B 77 431 331 676 ;
...
C 125 ; WX 480 ; N braceright ; B 130 -181 380 680 ;
C 126 ; WX 541 ; N asciitilde ; B 40 183 502 323 ;
C 161 ; WX 333 ; N exclamdown ; B 97 -218 205 467 ;
C 162 ; WX 500 ; N cent ; B 53 -138 448 579 ;
...
C 249 ; WX 500 ; N oslash ; B 29 -112 470 551 ;
C 250 ; WX 722 ; N oe ; B 30 -10 690 460 ;
C 251 ; WX 500 ; N germandbls ; B 12 -9 468 683 ;
C -1 ; WX 300 ; N onesuperior ; B 47 270 249 676 ;
C -1 ; WX 564 ; N minus ; B 30 220 534 286 ;
C -1 ; WX 400 ; N degree ; B 57 390 343 676 ;
...
C -1 ; WX 722 ; N Udieresis ; B 14 -14 705 835 ;
C -1 ; WX 444 ; N ecircumflex ; B 25 -10 424 674 ;
EndCharMetrics
...
```

文字コードが 32 番の space から始まって，exclam，quotedbl，…と続いていきますが，126 番の asciitilde の次が 161 番の exclamdown となっていて，その間の 35 個のスロットにはグリフの割り当てがありません。その後も一部空のスロットがあって，コードは 251 番の germandbls で終わってしまい，252 番から 255 番のスロットも空きとなっています。そして，続く onesperior から ecircumflex までの 80 個のグリフでは，コードの部分が “-1” と記載されています。

(LA)TEX では，フォントのグリフにアクセスするにはコードを指定しますが，PostScript フォントは，グリフの PostScript 名を介してグリフにアクセスすることが出来ます。TrueType フォントは，内部的には glyph index と cmap で管理されているらしいですが，ちゃんとした TrueType フォントの場合には，グリフの PostScript 名のテーブルも内部に持っているので，やはり PostScript グリフ名を介してグリフにアクセスすることが出来ます[23]。

そこで，PostScript フォントで元々 “-1” というコードが振られていたグリフまで含めた全部のグリフにコードを振り直して，PostScript フォントに収録されている全グリフに LATEX でアクセス出来るようにしたのが，TEX Base 1 Encoding（8r）です（一般的な PostScript フォント

[22] *Frank Mittelbach, Robin Fairbairns, Werner Lemberg and LATEX3 Project Team*, LATEX font encodings, 2006 (encguide.pdf) に載っているテキスト用フォントのエンコーディングだけでも，OT1 から OT6，T1 から T7，X2，LY1，LV1，LGR などがあります。

[23] グリフの PostScript 名が含まれていない TrueType フォントを LATEX で使う方法については，*Hàn Thế Thành* さんが “Using TrueType fonts with pdfTeX” (http://support.river-valley.com/wiki/index.php?title=Using_TrueType_fonts_with_pdfTeX や “A closer look at TrueType fonts and pdfTEX,” TUGboat, Volume 30 (2009), No. 1 (http://www.tug.org/TUGboat/tb30-1/tb94thanh.pdf) で解説をされています。

には含まれていないグリフも若干追加されているため，8r エンコーディングに収録されているグリフ数は 229 個よりも多くなっています)。

この TEX Base 1 Encoding (8r) は，普通は直接組版に使うものではなくて，エンコーディング間の変換をする際の, 中間エンコーディングとして利用されます。PostScript フォントや TrueType フォントを使う場合に，実フォントは dviware/driver の map ファイルで 8r エンコーディングに並べ直して，他方 TEX 側の OT1，T1，TS1 エンコーディングの tfm ファイルは vf を利用して 8r エンコーディングの tfm に帰着させる，ということがよく行われます。

エンコーディングの変換には，tfm ファイルや vf ファイルを作成する際に “enc” ファイルを使い，後述する dviware/driver の map ファイルでその enc ファイルを指定します。enc ファイルの中では，グリフの PostScript 名が当該エンコーディングのコード順に並んでいます。$TEXMF/fonts/enc/dvips/base/ には，例えば，以下のファイルが用意されています。

- OT1: “7t.enc”
- T1: “cork.enc”
- LY1: “texnansi.enc”
- 8r: “8r.enc”

また Oops フォントを例にしますが，oops___.pfb という PostScript フォントがあったとして，慣例によりこれを xoor8a.pfb に rename します。この実フォントから，何らかの方法で，エンコーディングを TEX 側と揃えた tfm ファイルを作れたとしましょう。例えば，7t.enc を使って xoor7t.tfm ですとか，cork.enc を使って xoor8t.tfm とかを作ったとします。

xoor7t.tfm が作れたのであれば，map ファイルでは，7t.enc を介して，xoor7t.tfm と xoor8a.pfb とを対応させることが出来ます（map ファイルの記述法は dviware/driver ごとに異なりますが，ここでは飽くまで考え方のみを示すために，簡略化しています)。

```
xoor7t      7t.enc      xoor8a.pfb
```

xoor8t.tfm を作れたのであれば，TEX 側では fontenc パッケージを使ってエンコーディングを T1 にして，map ファイルでは，

```
xoor8t      cork.enc      xoor8a.pfb
```

のように対応させることが出来ます。

そして，8r.enc を使って作った xoor8r.tfm が用意してあり，xoor7t.tfm や xoor8t.tfm を，vf を使って xoor8r.tfm に帰着させている場合には，map ファイルでの指定は，

```
xoor8r      8r.enc      xoor8a.pfb
```

のようになります。

なお，TrueType フォント内のグリフの PostScript 名は，通常の PostScript グリフ名とは一部異なっているらしく，それで TrueType フォント内に実際に含まれているグリフの PostScript 名[24]を Cork Encoding の順番に並べた

- “T1-WGL4.enc”

というファイルが $TEXMF/fonts/enc/ttf2pk/ にあります。TrueType フォントを使う際には，cork.enc ではなくこの T1-WGL4.enc を用いることが多いようです。

6　バーチャルフォントについて

前節で「vf を利用して 8r エンコーディングの tfm に帰着させる」なんてサラリと言っちゃいましたが，次に，バーチャルフォントの働きについて見てみましょう。

vf ファイルで出来ることはいろいろあるのですが[25]，多くの場合 vf は，

- エンコーディングの変換
- フォントの合成

のために用いられていると思いますので，ここではこの 2 点に絞って眺めてみます（vf を利用してグリフ自体を合成することも可能らしいのですけれど，ここでの「フォントの合成」は，複数のフォントから単純に個々のグリフをもらって来てひとつのフォントと看做す場合のみを考えます)。

vf ファイルはバイナリなので，tfm 同様そのままではエディタで見たり編集したり出来ませんが，欧文の vf であれば，vftovp を使うと，vpl というテキストファイルにすることが出来ます。編集後は vptovf でまたバイナリに戻せます。日本語用の (j)vf ファイルを vpl にしたりまた vf に戻したりするのは簡単には出来ないみたいですので，本文書では触れません。

vf が使われる場合のファイルの参照関係についてまず確認をしておきましょう。例えば，

- foo.tfm
- foo.vf （内部で bar.tfm を参照）
- bar.tfm

という tfm ファイルと vf ファイルがあるとし，foo.vf の中では bar.tfm が参照されていることにします。

ここで TEX が使う tfm ファイルは foo.tfm です。そして TEX で処理して生成した dvi ファイルにはその使われた tfm ファイルの名前 “foo” が書き込まれています。

この dvi ファイルを閲覧したり印刷したりする dviware/driver は，まず，“foo” という文字列から foo.vf を探します。見つかると，foo.vf に書かれている指示に従って，dvi ファイルを処理します。foo.vf の中では，bar.tfm が参照されているため，dviware/driver が利用する tfm ファイルは bar.tfm に帰着されます。したがって，当該 dviware/driver の map ファイルには，vf の参照先の bar.tfm と実フォントとの対応関係を示しておくことになります[26]（もしも，foo.vf が見つからないときには，dviware/driver は単純に foo.tfm が使われたものと考えますので，その場合には foo.tfm と実フォントとの対応関係が map ファイルには書かれていないといけません)。

以上のような仕組みにおいて，foo.tfm と bar.tfm が同一フォントの別エンコーディングのファイルであれば，

---

[24] Windows Glyph List 4 については，*Microsoft Corporation*,“Character Set Specifications: WGL4, Win31, UGL, and Macintosh,” TrueType 1.0 Font Files, Revision 1.66, August 1995 (http://www.microsoft.com/typography/tt/ttf_spec/ttch04a.doc) 参照（この “doc” ファイルは MS-WORD のファイルです)。

[25] バーチャルフォントについての詳しい説明は，*Donald Knuth*, “Virtual Fonts: More Fun for Grand Wizards,” TUGboat, Volume 11 (1990), No. 1 (http://www.tug.org/TUGboat/tb11-1/tb27knut.pdf) にあります。

[26] というのが原則のはずなのですが，$TEXMF/doc/dvipdfm/base/ にある “DVIPDFM-W32.txt” というドキュメントで角藤先生が dvipdfmx について，「マップファイルで TeX フォント名（tfm 名）が解決した場合には，vf や ovf はたとえ存在しても読みに行きません。」と説明してらっしゃいますので，dvipdfmx の場合には，foo.vf が存在している場合に，map ファイルに参照先の bar.tfm と実フォントの対応関係が書いてなくとも，foo.tfm と実フォントの対応が書いてあれば，エラーになりません。

これはエンコーディングの変換に利用することが出来，また，foo.vf の中で，bar.tfm だけでなく例えば baz.tfm も参照しているような場合には，foo.tfm は，bar.tfm と baz.tfm とを合成したフォントであると考えることが出来ます。具体例を少し見てみましょう。

エンコーディングの変換の例

PSNFSS バンドルに入っている Times フォントの T1 エンコーディングの場合について取り上げます。上述の例に相当するのは，例えば，以下の 3 ファイルです：

- ptmr8t.tfm
- ptmr8t.vf
- ptmr8r.tfm

まず，ptmr8t.tfm は，いうなれば，「普通の」tfm ファイルです。TEX はこの tfm ファイルを使って組版をします。tfm ファイルには，カーニングやリガチャについてのテーブルに続いて，グリフのメトリック情報が文字コード順に記載されています。ここでは例として “a” というグリフ（T1 で 97 番）と “Ž” というグリフ（T1 で 154 番）について見てみることにします。該当するコードの部分は，次のようになっています（pl ファイルに変換しています）：

```
(CHARACTER C a
   (CHARWD R 0.4439945)
   (CHARHT R 0.458496)
   (CHARDP R 0.009992)
   (COMMENT
      (KRN C w R -0.01499)
      (KRN C v R -0.019995)
      )
   )
```

```
(CHARACTER O 232
   (CHARWD R 0.610999)
   (CHARHT R 0.8984995)
   )
```

ここで “C” は *Character*（文字），“R” は *Real*（実数），“O” は *Octal*（8 進表記）の意味です。“a” については，幅・高さ・深さと，カーニングについてのコメントとが記載されおり，“Ž”（T1 で 154 番ですが，8 進表記だと 232 番）については幅と高さが書かれていますね。

TEX から見る限りは 97 番と 154 番のコードについての情報は上記のようになっているわけですが，実は同名の vf ファイルには別の指示が書かれています。ptmr8t.vf では，まず，冒頭の部分に，参照先の tfm 名が指定されています（vpl に変換しています）：

```
(MAPFONT D 0
   (FONTNAME ptmr8r)
   (FONTCHECKSUM O 4767720433)
   (FONTAT R 1.0)
   (FONTDSIZE R 10.0)
   )
```

“D” は *Decimal*（10 進表記）の意味で，“MAPFONT” によって，ptmr8r に 0 という番号を振って，参照できるようにしています（参照先フォントを選択する場合には “SELECTFONT” を使って番号を指定しますが，ここでは参照先は 0 番ひとつのみで，且つ，仮に参照先が複数ある場合でも最初の “MAPFONT” の番号がデフォルトの参照先となるので，明示的に 0 番を選択せずとも全コードが 0 番の ptmr8r を参照しています）。そして，“a” と “Ž” に当たる部分は次のようになっています：

```
(CHARACTER C a
   (CHARWD R 0.4439945)
   (CHARHT R 0.458496)
   (CHARDP R 0.009992)
   (COMMENT
      (KRN C w R -0.01499)
      (KRN C v R -0.019995)
      )
   (MAP
      (SETCHAR C a)
      )
   )
```

```
(CHARACTER O 232
   (CHARWD R 0.610999)
   (CHARHT R 0.8984995)
   (MAP
      (SETCHAR O 16)
      )
   )
```

赤字の “MAP” という部分でいろいろと操作が出来るのですけれど，ここでは “SETCHAR” を使って，“C a” を “C a” に置き替え，“O 232” については “O 16” に置き替えるように指示されています（8r での “Ž” のコードが “O 16” です．また，ptmr8r の “C a” を ptmr8r の “C a” に置き替えても変わりはないので，今の例では “(MAP (SETCHAR C a))” という操作は，本当はなくても大丈夫です）[27]。

最後に，参照先の ptmr8r.tfm の該当箇所は，次のようになっています（pl に変換しています）：

```
(CHARACTER C a
   (CHARWD R 0.4439945)
   (CHARHT R 0.458496)
   (CHARDP R 0.007001)
   (COMMENT
      (KRN C w R -0.01499)
      (KRN C v R -0.019995)
      )
   )
```

```
(CHARACTER O 16
   (CHARWD R 0.610999)
   (CHARHT R 0.891998)
   )
```

ファイル変換の際の丸め誤差のせいか，若干数値が異なっていますが，ほぼ ptmr8t.tfm，ptmr8t.vf のメトリック情報と合致しているのが見てとれます。

以上から，T1 エンコーディングの ptmr8t.tfm が，ptmr8t.vf を介して，8r エンコーディングの ptmr8r.tfm に帰着されていることが分かると思います。

フォントの合成の例

vf を使って複数の tfm を参照するようにすれば，それは，フォントを合成しているといえます。今度は PSNFSS バンドルの mathpazo パッケージで “osf” オプションを

[27] この ptmr8t.vf では丁寧にすべてのコードについて MAP を使って参照先の ptmr8r.tfm の番号が指定されていますが，普通は，当該グリフの参照元のエンコーディングの文字コードと，参照先のエンコーディングの文字コードが同じ場合には，わざわざ MAP を使う必要はありません。コードが異なるグリフについて，MAP の部分で SETCHAR を使って参照先のコードを指定することで，エンコーディングの変換を実現しているわけです。

指定した場合を例にしましょう。mathpazo では osf を指定すると，数字がデフォルトで oldstyle になります。エンコーディングが T1 の場合の一例として，以下のファイルを調べてみます（t1pplj.fd から辿りました．“9d” は Berry 則で “expert + oldstyle + Cork” の意味です）。

- pplr9d.tfm
- pplr9d.vf
- pplr8r.tfm，fplmr.tfm，pplrc8r.tfm

この例では，pplr8r.tfm，fplmr.tfm，pplrc8r.tfm という 3 つの tfm ファイルを，vf を介してひとつの pplr9d.tfm として扱っていることになります。

ここでは pplr9d.vf の中だけを見れば十分だと思います。まず，冒頭には，参照先の tfm が列挙されています。

```
(MAPFONT D 0
   (FONTNAME pplr8r)
   (FONTCHECKSUM O 36571141413)
   (FONTAT R 1.0)
   (FONTDSIZE R 10.0)
   )
(MAPFONT D 1
   (FONTNAME fplmr)
   (FONTCHECKSUM O 24030505107)
   (FONTAT R 1.0)
   (FONTDSIZE R 10.0)
   )
(MAPFONT D 2
   (FONTNAME pplrc8r)
   (FONTCHECKSUM O 16710664207)
   (FONTAT R 1.0)
   (FONTDSIZE R 10.0)
   )
```

pplr8r.tfm を 0 番，fplmr.tfm を 1 番，pplrc8r.tfm を 2 番，と番号を振って，参照出来るようにしています。0 番の pplr8r がデフォルトの参照先で，SELECTFONT が使われていない MAP でもこの 0 番が選択されたことになります。1 番の fplmr は “dotlessj” をもらって来るためのもののようです（pplr8r には dotlessi は含まれていますのに，なぜか dotlessj は入っていません）。2 番の pplrc8r からは，oldstyle の数字をもらって来ています。というわけで，“j”（T1 で 26 番，8 進表記だと 32 番）と数字の “1”（大抵のエンコーディングで 49 番）と，それからついでにまた “a” のコードを見てみましょう。該当部分は，以下のようになっています。

```
(CHARACTER O 32
   (CHARWD R 0.234)
   (CHARHT R 0.4845)
   (CHARDP R 0.283)
   (MAP
      (SELECTFONT D 1)
      (SETCHAR O 342)
      )
   )
```

```
(CHARACTER C 1
   (CHARWD R 0.5)
   (CHARHT R 0.4845)
   (CHARDP R 0.0035)
   (MAP
      (SELECTFONT D 2)
      (SETCHAR C 1)
      )
   )
```

```
(CHARACTER C a
   (CHARWD R 0.5)
   (CHARHT R 0.4845)
   (CHARDP R 0.0035)
   (COMMENT
      (KRN O 270 R -0.039)
      (KRN O 375 R -0.039)
      (KRN C y R -0.039)
      (KRN C w R -0.036)
      (KRN C v R -0.036)
      (KRN C T R -0.06)
      (KRN O 224 R -0.06)
      (KRN O 225 R -0.06)
      )
   (MAP
      (SETCHAR C a)
      )
   )
```

もう予想がつくと思いますが，1 番の fplmr の “O 342” は “j” で，2 番の pplrc8r の “C 1” は oldstyle の “1”，そして 0 番の pplr8r の “C a” はもちろんそのまま “a” です。

このようにして，vf を使うと，参照先の tfm に番号を振って，それらの tfm から必要なグリフのメトリック情報をもらって来ることが出来ます。あとは，dviware/driver の map ファイルで，参照先である pplr8r，fplmr，pplrc8r をきちんと実フォントと対応付ければ，出力が出来るということになるわけです。

7　map ファイルについて

map ファイルの設定というのは，フォントのインストールに慣れていない方にとっては意外と落とし穴かも知れません。TEX 用にパッケージ化されているフォントをどこからかダウンロードして来た場合，そのアーカイブを展開して，各種ファイルを決まった場所に配置すれば，とりあえず TEX による処理は出来てしまいます。しかし，生成した dvi ファイルを閲覧したり印刷したりするには，dviware/driver が必要で，dviware/driver は，dvi ファイルに書き込まれている tfm 名と実フォントとを対応付ける map ファイルを必要とします。この map ファイルの追加や更新を忘れていると，当然，組版結果の閲覧や印刷は出来ません。また，フォントの埋め込みの有無も map ファイルで設定します。

TEX で使うエンコーディングと，実フォントのエンコーディングとが同じであるならば，map ファイルにおいても両者を直接対応させるだけで済むかも知れませんが，実際には，TEX 側の欧文のエンコーディングはデフォルトでは OT1 (7t) で，ヨーロッパの言語を扱うなら T1 (8t) エンコーディングか，もしかすると LY1 (8y) あたりで，他方，実フォントのエンコーディングは，PostScript フォントであれば 8a で，TrueType フォントなら Unicode でしょうか。

dvips や dvipdfmx といった dviware/driver や，pdfTEX では，それぞれの map ファイルで “enc” ファイルを指定することで，実フォントのエンコーディングを変換することが出来るようになっています。「5 エンコーディングについて」の末尾で見ましたように，TEX 側が OT1 なら，“7t.enc” を使って実フォントを並べ替えることが出来，T1 なら “cork.enc” や “T1-WGL4.enc” で，そして LY1 の場合には “texnansi.enc” を使って，実フォントを reencoding 出来ます。

また，これも既に見ましたが，vf を使って OT1，T1，TS1 を 8r に帰着させているようなケースでは，map ファイルで “8r.enc” を指定して実フォントを再配置します。

以下では，PSNFSS バンドルの Times フォントから ptmr8r を例に，読み込む map ファイルの指定の仕方と map ファイルの記述例とを見てみましょう[28]。

dviout の場合

よく分からないので省略しますが[29]，後述の updmap との関係で，“pspksupp.map” 内の ptmr8r のエントリだけチラッと見ておきます。

```
ptmr8r    utmr8a.pfb    0    1    8r.enc
```

dvips の場合[30]

dvips がデフォルトで読み込む map ファイルは，“config.ps” という設定ファイルに “p +myfonts.map” のように指定することになっています（このサンプル行は通常はコメントアウトされています）。

また，dvips の実行時に，“-P” オプションで拡張子を指定すると，指定された拡張子を持つ “config” ファイルが読み込まれます。例えば，“-P dl” なら “config.dl” が，“-P pdf” だと “config.pdf” が読み込まれます。

そして，config.dl や config.pdf というファイルの中に，“p +psnfss-dl.map” とか “p +dvipsfnt.map” という行があるので，“-P dl” オプションや “-P pdf” オプションを指定すると，これらを介して “psnfss-dl.map” や “dvipsfnt.map” が読み込まれるということになります。

以上のようなファイルの依存関係があるので，dvips で任意の map ファイルを読み込むには，

- myfonts.map みたいなファイルを作ってその中にエントリを追加し，config.ps に “p +myfonts.map” と書いておくか，
- またはやはり myfonts.map を作って，config.dl や config.pdf の中に “p +myfonts.map” と書いておいて，dvips 実行時に “-P dl” や “-P pdf” とオプションを指定するか，
- それとも，config.dl や config.pdf から読み込まれる “dvipsfnt.map” にエントリを追加しておいて，同様に dvips 実行時にオプション “-P dl” や “-P pdf” を指定するか，

のいずれかの方法をとることになります。後述の updmap は最後のやり方をとっています（dvipsfnt.map にエントリを追加していきます）。

“psnfss-dl.map” から ptmr8r のためのエントリを抜き出してみますと，

```
ptmr8r NimbusRomNo9L-Regu "TeXBase1Encoding
                  ReEncodeFont" <8r.enc <utmr8a.pfb
```

となっています（ここではスペースの関係上 2 行にしていますが，実際は 1 行で書かないといけません．以下の例についてもすべて同様です）。エントリは，

> “tfm 名，フォントの PostScript 名，ReEncodeFont の指示，enc ファイル名，実フォントファイル名”

の順に並んでいますが，本来 ptmr に相当する実フォントは Adobe 社の商品なので，ここでは代わりに URW 社のフリーフォントが当てられていています（“NimbusRomNo9L-Regu” という名前のフォントで，ファイル名は “n0210031.pfb” を Berry 則に従って “utmr8a.pfb” へと rename したもの）。

Base14 フォントを埋め込まない場合の map ファイルは “psnfss.map” で，該当するエントリは，

```
ptmr8r Times-Roman
          "TeXBase1Encoding ReEncodeFont" <8r.enc
```

となっています。

dvipdfmx の場合

dvipdfmx でデフォルトで読み込む map ファイルを指定するには，“dvipdfmx.cfg” において，“f myfont.map” という風に指定します。dvipdfmx.cfg には既に，

- “f psbase14.map”
- “f pdfmfnt.map”
- “f cid-x.map”

の 3 行が書き込まれています。後述の updmap はこのうちの “pdfmfnt.map” にエントリを追加していきます。

また，dvipdfmx の実行時に，“-f” というオプションを使って map ファイルを指定することも出来ます。

“psbase14.map” ファイルの ptmr8r のエントリは，

```
ptmr8r    8r    Times-Roman
```

となっているので，デフォルトではフォントは埋め込まれません。Base14 フォントを埋め込むには，dvipdfmx 実行時に “-f dlbase14.map” として，“dlbase14.map” を読み込みます。“dlbase14.map” の ptmr8r エントリは次のようになっています。

```
ptmr8r    8r    utmr8a
```

ここでも，埋め込むフォントは URW の utmr8a.pfb です。

pdfTEX の場合

pdfTEX がデフォルトで読み込む map ファイルは “pdftex.map” で，これは fmt ファイル作成時に既に決まってしまっています。したがって，pdfTEX の場合には，map ファイルのエントリを追加するには，“pdftex.map” にエントリを追加することになります。後述の updmap も，“pdftex.map” にエントリを追加していきます。

pdftex.map ファイルでは base14 フォントを埋め込む設定になっており，ptmr8r のエントリは，

```
ptmr8r "TeXBase1Encoding
                  ReEncodeFont" <8r.enc <utmr8a.pfb
```

と書かれています。埋め込まない設定は noembedbase14-base.map に，以下のように書かれています。

---

[28] dviout の設定は私には難し過ぎ，また普段私は dvips は使わないので，この両者についての説明は，すごく適当です。ここでは，どの dviware/driver もそれぞれに map ファイルが必要なんです，ということを示すためだけに，取り上げています。

[29] 本文書冒頭でも述べましたように，私は METAFONT や pk ファイルについては全然分からないのですが，dviout で “Cannot resolve Fonts” となった際に，まず METAFONT でフォント作成にトライして，それが失敗すると “I try ps2pk -> gsftopk -> ttf2pk -> hbf2gf.” と表示されますよね。これは mktexpk というプログラムが順に ps2pk，gsftopk，ttf2pk，hbf2gf を呼んでいるのですが，ps2pk は “pspksupp.map” を，gsftopk は “psfonts.map” を，そして ttf2pk と hbf2gf は “ttfonts.map” を参照しているらしいです。後述の updmap は，pspksupp.map を更新します。

[30] 私の手許の W32TeX は 2010 年 4 月 1 日にインストールしたものなので，dvips については，ファイルの依存関係が現在の構成とは違っている部分があるかも知れません（すいません）。

```
ptmr8r Times-Roman "TeXBase1Encoding
                              ReEncodeFont" <8r.enc
```

なお，pdfTEX の場合には，\pdfmapfile と \pdfmapline というプリミティブがあるので，原稿ファイル内でこれらを使って，map ファイルを追加したり，エントリを追加したりすることができます。pdfTEX のマニュアルからそのまま転記しますが，例えば，

```
\pdfmapfile{+myfont.map}
\pdfmapline{+ptmri8r Times-Italic
                              <8r.enc <ptmri8a.pfb}
```

のように使うことが出来ます（ここではいずれも “+” を使っていますが，“=” や “-” という指定もあります）。

さて，以上 dviout，dvips，dvipdfmx，pdfTEX の map ファイルの設定例を見てみましたが，フォントを追加する度に，記述法も異なるこれら全部の map ファイルを更新するなんていうのは，結構な手間です。

そこで，updmap というツールが用意されています。ドキュメントは $TEXMF/doc/updmap/ にある “updmap.txt” です。dvips 用の map ファイルがあれば，updmap を使うと，

- pspksupp.map
- dvipsfnt.map
- pdfmfnt.map
- pdftex.map

の 4 ファイルを一度に更新してくれます。

既存のフォントパッケージの場合には，大抵 dvips 用の map ファイルが同梱されているので，updmap を使えば map ファイルの更新も簡単ですが，自分でまったく新たにフォントをインストールする場合には，自分で何とかしないといけません。しかし，上述の各種 map ファイルの記述例を比較してみますと，dvipdfmx でフォントを埋め込む場合の記述法がとても簡単なことに気が付きます。また，8r に帰着させるようなことさえしなければ，“TeXBase1Encoding ReEncodeFont” の類いも要らないので，pdfTEX の map ファイルの記述についても，直接手入力をしても，そんなに大変ではないです。

なお，updmap で map ファイルを更新するのではないときには，上記の 4 ファイルは直接編集したりしないほうがいいです（updmap による更新で消えてしまうので）。その場合には，適当な名前の map ファイルを作って，dvipdfmx なら実行時に “-f” オプションで読み込んだり，pdfTEX の場合には原稿ファイルで \pdfmapfile を使って読み込むようにするのが，いいと思います。

8　実例に即して

これまで縷々述べてきたことを踏まえて，実際に欧文の TrueType フォントのインストールをして dvipdfmx と pdfTEX で使えるようにしてみましょう。Windows 付属の欧文 TrueType フォントを TEX で使うためのパッケージは既にいくつかありますが[31]，ここでは，ttf2tfm を使って tfm ファイルを作成してみます。

本文書冒頭で述べましたように，インストールの手順としては，まず，tfm ファイルを作って，それから fd ファイルに相当する設定をして，そして，dviware/driver の map ファイルの設定をする，ということになります。ttf2tfm で tfm を作る過程が理解出来れば，map ファイルをどのように書けばよいのかも分かります（ttf2tfm がヒントを表示してもくれますが）。

ttf2tfm を使うと，TrueType フォントから，“(raw.)tfm” ファイルと “vpl” ファイルを作ることが出来ます。vpl ファイルは，vptovf を使って vf ファイルと tfm ファイルに変換します。

「6 バーチャルフォントについて」の節で例に挙げた “foo.tfm”，“foo.vf”，“bar.tfm” の関係に当てはめますと，参照先の bar.tfm に当たるのが，(raw.)tfm で，vpl から作られるファイルが foo.vf と foo.tfm に相当します。

ttf2tfm から直接作った (raw.)tfm ファイルには，カーニングやリガチャの情報が含まれていません。ttf2tfm（や afm2tfm）では，カーニングとかリガチャの情報は vpl に反映されることになっているので，vpl から変換した tfm には，ちゃんとカーニングの情報もリガチャの情報も含まれています[32]。

わざわざ (raw.)tfm と vpl のエンコーディングを違うものにして，vf を介してエンコーディングの変換をしたり，(raw.)tfm を参照したりする必要はないと思われます。ttf2tfm（や afm2tfm）を利用する場合，vpl を経由するのはカーニングやリガチャの情報を取り込むためです（ですから，以下の作業では vf も出来ますけれど，実際には使いません）。

しかし，(raw.)tfm のエンコーディングもオプションで指定出来ますので，敢えて vf を介在させてエンコーディングを変換してみることも，もちろん可能です。

手許の Windows7 付属の “Times New Roman” を例にします。「標準」「太字」「斜体」「太字斜体」の 4 種類が入っており，それぞれのファイル名は，“times.ttf”，“timesbd.ttf”，“timesi.ttf”，“timesbi.ttf” です。このうち，roman, bold, italic の 3 書体を LATEX で使えるようにしてみます。作業はどれも同じですから，times.ttf から tfm ファイルを作る過程について，少し詳しく見てみましょう。まず，コマンドプロンプトで，

```
ttf2tfm times.ttf -q raw-wTimesRoman.tfm
```

とすると，times.ttf から “raw-wTimesRoman.tfm” が作られます（Berry 則は使わないことにします．Windows の Times ということで，“wTimes” の “Roman”，“Bold”，“Italic” という名前の tfm にしましょう）。

“-q” というオプションでスクリーンに出る情報を抑制していますが，“-q” を使わないと，当該 TrueType フォントに含まれるグリフの情報と，raw-wTimesRoman.tfm に適用されるエンコーディングとが表示されます。times.ttf の場合，3,000 個近いグリフが含まれているようですので[33]，“-q” を指定しないと，何千行もの情報がディスプ

---

[31] *Paul Pichaureau* さんの “winfonts” パッケージ (CTAN:fonts/winfonts/) や，*Christophe Caignaert* さんの “WindowsFonts” (http://c.caignaert.free.fr/WindowsFonts.zip)，また dviout に同梱されている乙部厳己先生による “TTNFSS” バンドルなど。なお，私の手許では winfonts パッケージを使うには付属の map ファイルの内容を変更しなければなりませんでしたが，LATEX 2ε におけるフォントの扱いの仕組みが分かっていれば，修正は難しくありません。

[32] enc ファイルでリガチャの設定を書き足すことも出来ます。説明は dvips のマニュアル〔前掲註（14）〕の “6.3.1.5 Encoding file format” の部分にあります。

[33] えっ，3,000 個?!。てっきり Times New Roman は TrueType フォントだと思っていたのですが，よく見たら OpenType でした…。でも，ttf2tfm を使って tfm ファイルが作れちゃったので，このまま Times New Roman で話を続けます。

レイ上を流れていきます。しかし，せっかくの情報をまったく抑制してしまうのももったいないので，例えば，

```
ttf2tfm times.ttf raw-wTimesRoman.tfm > ttf.log
```

のようにリダイレクトして保存すると，参考になるかと思います。今の場合，ttf.log の中味は次のようになっています（スペースの関係上，Glyph，Code，Glyph Name の欄しか載せていませんが，実際には，この右側には Width，llx，lly，urx，ury という欄もあります）。

```
 This is ttf2tfm version 1.5

 Glyph  Code   Glyph Name
-------------------------------------------
     3  00020  space
     4  00021  exclam
...
  1020  0fefc  .c0xfefc
   863  0fffc  .c0xfffc
     1         .g0x1
 *           Germandbls

Using the first 256 glyphs in the following
input encoding:

  0x00    space
  0x01    exclam
...
  0xfe    Ldot
  0xff    ldot

raw-wTimesRoman   times.ttf
```

times.ttf に収録されているグリフが，16 進表記で Unicode 順に “space” から（ttf2tfm が追加した）“Germandbls” まで，列挙されています。今は (raw.)tfm についてエンコーディングを指定しなかったため，raw-TimesRoman.tfm に適用されるエンコーディングは，times.ttf の最初の 256 グリフをただ並べたものになっています（つまり，“space” を 0 番として番号を振り直して，255 番が “ldot”）。そして最後に，map ファイルに書くべき情報として “raw-wTimesRoman times.ttf” との記載があります。

ここで，“-p” というオプションで enc ファイルを指定すると，(raw.)tfm のエンコーディングを変更することが出来ます[34]。

```
ttf2tfm times.ttf -p 7t.enc
                    raw-wTimesRoman.tfm > ttf.log
```

とすると，ディスプレイには，

```
ttf2tfm: WARNING: Cannot find character ‘Sigma’
         specified in input encoding.
ttf2tfm: WARNING: Cannot find character ‘ff’
         specified in input encoding.
ttf2tfm: WARNING: Cannot find character ‘ffi’
         specified in input encoding.
ttf2tfm: WARNING: Cannot find character ‘ffl’
         specified in input encoding.
ttf2tfm: WARNING: Cannot find character ‘dotlessj’
         specified in input encoding.
```

と表示され，今度は ttf.log ファイルの末尾には，

```
Using 7t.enc as input encoding.

raw-wTimesRoman times.ttf Encoding=7t.enc
```

と記載されています。

表示された Warning は，7t.enc に書かれているグリフ名のうち，‘Sigma’，‘ff’，‘ffi’，‘ffl’，‘dotlessj’ というグリフ（というか PostScript 名）は times.ttf には含まれていないということを言っています[35]。

今作った raw-wTimesRoman.tfm は，“-p 7t.enc” によりエンコーディングが OT1 になりましたが，カーニングやリガチャの情報が含まれていません（pl ファイルに変換して中を見ればすぐ確認出来ます．まさに “raw” な tfm です）。カーニングやリガチャの情報込みの tfm を得るには，vpl ファイルを介します。

vpl ファイルを作るには，“-v” というオプションでファイル名を指定します。

```
ttf2tfm times.ttf -v wTimesRoman.vpl
          -p 7t.enc raw-wTimesRoman.tfm > ttf.log
```

とすると，raw-wTimesRoman.tfm と共に wTimesRoman.vpl が出来ます。

wTimesRoman.vpl のエンコーディングはデフォルトで OT1 となりますが，「7t.enc の OT1」ではなく，「Computer Modern Typewriter の OT1」のため，ディスプレイに表示される，Warning は，上掲の “-p 7t.enc” の場合よりも多くなります[36]。

この wTimesRoman.vpl は，内部で raw-wTimesRoman.tfm を参照しています。また，両者のエンコーディングは共に OT1 なので，本来 MAP でエンコーディングの変換をする必要はないはずなのですが，「7t.enc の OT1」と「Computer Modern Typewriter の OT1」の差があるため，一部のコードで MAP と SETCHAR を使ってコードの置き替えが行われています（vpl の中を覗くと確認出来ます）。

ここで，“-t” オプションで enc ファイルを指定すれば，vpl のエンコーディングを変えることも出来ます。

```
ttf2tfm times.ttf -t cork.enc -v wTimesRoman.vpl
          -p 7t.enc raw-wTimesRoman.tfm > ttf.log
```

こうすると，OT1 の “raw-wTimesRoman.tfm” と T1 の “wTimesRoman.vpl” とが出来ます。そして，vptovf を使って，

```
vptovf wTimesRoman
```

とすると，wTimesRoman.vpl から，“wTimesRoman.vf” と “wTimesRoman.tfm” とが出来ます。

---

[34] ttf2tfm のオプションの名前は afm2tfm のオプション名を踏襲しているようです。“-v” オプションが，vpl を作る指定で，“-t” オプションが TEX 側で使う tfm のエンコーディングの指定というのは，ttf2tfm でも不自然には感じませんが，この “-p” オプションは，元々 afm2tfm で，実フォント（PostScript）側の reencoding の指定を意味していたのでしょう。

[35] ttf.log にリダイレクトしても，“-q” で表示を抑制しても，Warning はディスプレイに表示されます。Warning までリダイレクトするには，“> ttf.log” の部分を “> ttf.log 2>&1” とします。

[36] 具体的には，‘Sigma’，‘arrowup’，‘arrowdown’，‘quotesingle’，‘dotlessj’，‘quotedbl’，‘less’，‘greater’，‘backslash’，‘underscore’，‘braceleft’，‘bar’，‘braceright’ がない，と表示されます（ん?，なんで quotedbl や quotesingle がないんだ?）。

ttf.log の末尾には “raw-wTimesRoman times.ttf Encoding=7t.enc” と書いてありますが，これは，飽くまで vf を使う場合についての ttf2tfm からの指示です。つまり，wTimesRoman.vf を使うのであれば，wTimesRoman.tfm は raw-wTimesRomanl.tfm に帰着されるので，map ファイルには，

```
raw-wTimesRoman 7t.enc times.ttf
```

と書くことになります（dvipdfmx の場合）。しかし，wTimesRoman.tfm は，T1 エンコーディングで，カーニングやリガチャも含んだ，「普通の」tfm です。ですから，wTimesRoman.vf を TEX から見えないところに置くか削除してしまえば，map ファイルには，

```
wTimesRoman cork.enc times.ttf
```

と書いて構わないことになります。

以上では説明のため，(raw.)tfm と vpl のエンコーディングを敢えて変えてみましたが，両者のエンコーディングを揃える “-T” というオプションがあります[37]。また，TrueType フォントの場合に T1 にするには，cork.enc ではなく，T1-WGL4.enc を使うことのほうが多いようです。これまでのところをまとめますと，

```
ttf2tfm times.ttf -T T1-WGL4.enc
  -v wTimesRoman.vpl raw-wTimesRoman.tfm > ttf.log
vptovf wTimesRoman
```

のように入力すると，

- raw-wTimesRoman.tfm
- wTimesRoman.vf
- wTimesRoman.tfm

という 3 ファイルを得ることが出来ます（いずれも T1 エンコーディング）。これでようやく，必要な tfm ファイル（wTimesRoman.tfm）が出来ました。raw-wTimesRoman.tfm と wTimesRoman.vf は，せっかくですが使わないことにして，map ファイルには，

```
wTimesRoman T1-WGL4.enc times.ttf
```

と書くことにしましょう。

よく見かける解説では，最後の ttf2tfm と vptovf の行がいきなり書いてありますけれど（普通は “> log” ではなく “-q” オプションを使っています），ここでは，各オプションが何をしているのかを，丁寧に追ってみました。

timesbd.ttf と timesi.ttf からも同様にして，wTimesBold.tfm と wTimesItalic.tfm を作ります。

tfm ファイルが出来れば，あとは NFSS に沿ってこれらの tfm ファイルをロードするための fd ファイルと，dvi-ware/driver 用の map ファイルを作ることになりますが，ここではみな原稿ファイルのプリアンブルにまとめてしまいましょう：

```
\usepackage[T1]{fontenc}

% --- instead of ``t1wintimes.fd'' ---
\DeclareFontFamily{T1}{WinTimes}{}
\DeclareFontShape{T1}{WinTimes}{m}{n} {<-> wTimesRoman}{}
\DeclareFontShape{T1}{WinTimes}{b}{n} {<-> wTimesBold}{}
\DeclareFontShape{T1}{WinTimes}{m}{it}{<-> wTimesItalic}{}
% substitution of b/n for bx/n:
\DeclareFontShape{T1}{WinTimes}{bx}{n}{<-> ssub * WinTimes/b/n}{}
% another alternative:
% \renewcommand{\bfdefault}{b}

% --- instead of ``WinTimes.sty'' ---
\renewcommand{\rmdefault}{WinTimes}
\DeclareRobustCommand{\WinTimesfamily}{\fontfamily{WinTimes}\selectfont}
\DeclareTextFontCommand{\textWinTimes}{\WinTimesfamily}

% --- mapping info for dvipdfmx ---
\AtBeginDvi{
   \special{pdf:mapline wTimesRoman T1-WGL4.enc times.ttf}
   \special{pdf:mapline wTimesBold T1-WGL4.enc timesbd.ttf}
   \special{pdf:mapline wTimesItalic T1-WGL4.enc timesi.ttf}
}

% --- mapping info for pdftex ---
% \pdfmapline{+wTimesRoman <T1-WGL4.enc <times.ttf}
% \pdfmapline{+wTimesBold <T1-WGL4.enc <timesbd.ttf}
% \pdfmapline{+wTimesItalic <T1-WGL4.enc <timesi.ttf}
```

[37] いえ，別に (raw.)tfm と vpl のエンコーディングを揃えなければならない必然性はありません。大事なのは，TEX 側で使う予定のエンコーディングを “-t” オプションで指定することです。T1 で使うのならば cork.enc や T1-WGL4.enc にして，LY1 で使おうと思っているなら texnansi.enc を指定し，他方 TEX 側では fontenc パッケージを使って，T1 や LY1 にします。なお，“-T” オプションで (raw.)tfm と vpl のエンコーディングを揃えた場合には，vf ファイルにはもう，エンコーディング変換の役割さえないですよね。

Windows の Times New Roman について roman，bold，italic の 3 書体を使うための最小限の設定例です。

“T1” の tfm を作ったので，1 行目で TEX 側でも fontenc パッケージで “T1” にしています。

4 行目から 9 行目までは，普通は “fd” ファイルにまとめる内容です。4 行目で “WinTimes” というファミリを宣言して，5〜7 行目で，T1/WinTimes/m/n，T1/WinTimes/b/n，T1/WinTimes/m/it に対応してロードする tfm ファイルを指定しています。9 行目では，“bx/n” の組み合わせを “b/n” で代替しています。11 行目でコメントアウトしているように \bfdefault を “b” にしてもいいと思います。クラスファイル内で \bfseries が使われているのに備えて，最低限この代替が必要となりますが，より丁寧に設定するなら，“sl” や “sc” についても代替をすることになるでしょう（または，dviware/driver の map ファイルで “sl” に対応したり，擬似的な “sc” 用の vf や tfm を作るとか）。

14 行目から 16 行目は，sty ファイルにまとめるような内容です。\rmdefault を WinTimes にしてしまったら，\WinTimesfamily とか \textWinTimes なんてコマンドを使う機会はあまりなさそうですけれど。

19 行目から 23 行目は，dvipdfmx 用の map ファイルの設定を “\special” を使って無理やりプリアンブルに入れたものです。この中味についても，map ファイルにまとめて，dvipdfmx 実行時に “-f” オプションで読み込むのが普通かも知れません。

26 行目から 28 行目は，pdfTEX 用の map ファイルの設定です。pdfTEX を使うときには，19〜23 行目をコメントアウトして，こちらのコメントをはずします[38]。

ここでは，tfm ファイルは作業をしているカレントフォルダに置いて，fd ファイルや map ファイルに相当する内容もプリアンブルにまとめてしまいましたが，ちゃんとするなら，それぞれをファイルに分けて，且つ，TDS に則って配置することになります。

9　和文フォントについて

最後に，和文フォントについてもほんのちょっとだけ触れることにします。取り上げる dviware/driver は dvipdfmx だけです。

和文フォントの扱いについても，基本的な仕組みは欧文の場合と同じです。つまり，LATEX で和文フォントを使うためには，当該和文フォントに対応した jfm ファイルが必要で，その jfm ファイルを NFSS に則ってロードできるように fd ファイルを設定し，それから，dviware/driver 用の map ファイルを用意することになります。

欧文フォントの場合には，フォントごとにメトリック情報が異なりますが，（プロポーショナルではない）和文フォントであれば，どのフォントでも同じメトリック情報で構わないようです[39]。

ひとつ注意が必要なのは，実フォントでは句読点などもみな正方形のボディを持っているのに対して，jfm では，句読点や約物が半角幅となっているらしいということです。そのため，jfm と実フォントをそのまま対応させると，句読点などの前後に余分なアキが出来てしまいます。この jfm と実フォントとのズレは，jvf を利用して修正するそうです（「6 バーチャルフォントについて」の MAP の部分では，“SELECTFONT” と “SETCHAR” しか出てきませんでしたが，“MOVERIGHT” とか “MOVELEFT” という命令もあるので，vf を使うとグリフを左右に移動させることも出来るわけです）。

$TEXMF/fonts/tfm/ptex/ フォルダを見てみますと，“(t)min5|6|7|8|9|10.tfm”，“(t)goth5|6|7|8|9|10.tfm”，“jis(-v).tfm”，“jisg(-v).tfm” などの jfm ファイルがありますが[40]，これらは，同名の jvf ファイルを介して，それぞれ，

- rml.tfm
- gbm.tfm
- rmlv.tfm
- gbmv.tfm

を参照しているらしいです（“t” や “v” は「縦組み」の意味ですね）。

そのため，dviware/driver の map ファイルでは，rml，gbm，rmlv，gbmv という tfm 名に，実フォントが mapping されています。

dvipdfmx の場合，和文フォント関係の設定がまとめられている map ファイルは “cid-x.map” です。「7 map ファイルについて」で見ましたように，dvipdfmx.cfg に “f cid-x.map” という行があるので，dvipdfmx の実行時にはデフォルトでこの map ファイルが読み込まれます。cid-x.map の中では，上述の 4 つの名前についてのエントリは，次のようになっていると思います。

```
rml   H Ryumin-Light
gbm   H GothicBBB-Medium
rmlv  V Ryumin-Light
gbmv  V GothicBBB-Medium
```

欧文の場合と同様，「jfm 名，エンコーディング名，実フォント名」の順に並んでいます[41]。ここで “Ryumin-Light” と “GothicBBB-Medium” は，いうなれば明朝体とゴシック体の一般名詞形というか代表形なので，この設定で作られた pdf が開かれる環境にこの名前のフォントがなければ（普通はありません），それぞれ，その viewer のデフォルトの明朝体とゴシック体で表示されます。例えば，Adobe Reader の場合なら両者は “KozMinPr6N-Regular” と “KozGoPr6N-Medium” とかになります。

Ryumin-Light 等の代わりに手許のシステムの実フォントのファイル名を指定すると，そのフォントが埋め込まれます。実フォント名の前に “!” を指定すると，Ryumin-Light 等の場合と同様，名前だけが入ってフォント自体は埋め込まれません。pdf を開く側のシステムにそのフォントがあれば当該フォントで表示され，ない場合にはまた，KozMinPr6N-Regular 等のフォントで表示されます。

“ttc” フォントの場合には，フォントファイル名の前に “:” で挟んだ “index no.” を書きます。例えば，“msembed.map” を見てみますと，

---

[38] これでうまくいくと思ったのですが，なぜか pdfTEX では，“pdfTeX warning: pdflatex.exe (file timesi.ttf): glyph ‘fi’ not found” との Warning が出て，italic では，“*fi*” と “*fl*” のリガチャが表示出来ませんでした。dvipdfmx で作った pdf だとちゃんと表示されています。同じ設定なのに両者で挙動が異なる理由については私には分かりません…（すいません）。

[39] TEX に限らず，私は和文のプロポーショナルフォントを使いたいと思ったことはないのですけれど，dviout には “propw” というツールが同梱されていますので，関連するドキュメント（“propw0.txt” や “newjfm.txt”，“addjfonts.txt”，“tume.pdf” 等々）とか乙部先生の web page（http://argent.shinshu-u.ac.jp/~otobe/tex/packages/fontproj.html）等をご覧になると，和文のプロポーショナルフォントが使えるようになるのかも知れません。

[40] $TEXMF/fonts/tfm/ptex/ には，nmin とか ngoth，jisn，jisgn，jisn-v，jisgn-v という “n” が付いた tfm も入っているのですが，これらが “n” なしのものとどう違うのかは調べていません。

[41] cid-x.map の書式は，cid-x.map の冒頭に，コメントの形で書かれています。

```
rml   H   :0:msmincho
gbm   H   :0:msgothic
rmlv  V   :0:msmincho
gbmv  V   :0:msgothic
```

のようになっています（msmincho.ttc と msgothic.ttc の共に 0 番が，等幅のＭＳ明朝とＭＳゴシックです）。ここで“:0:!msmincho”と変更すると，ＭＳ明朝で表示はされますが，埋め込まれなくなります。

以上から，2 フォント使用の枠内であれば，map ファイルの設定だけで，手許のシステムにある任意の和文フォントが使えることが分かります。極端な話，rml に或るゴシック系のフォントを対応させ，gbm に別のゴシック系のフォントを対応させて，ゴシック 2 フォントで文書を作成するなんてことも可能なわけです。

第 3 の和文フォントを導入するには，まず，そのフォント用の jfm ファイルを用意します。min10.tfm か jis.tfm を別名でコピーして使えば大丈夫です。ここではそれをまた“Oops.tfm”としましょう。

この jfm ファイルを元に実フォントとのズレ調整のための jvf を作ってくれるツールに makejvf があります。ドキュメントは $TEXMF/doc/jvf/ にある“README.txt”です。コマンドプロンプトで，例えば，

```
makejvf Oops.tfm fix-Oops
```

とすると，Oops.tfm から，“Oops.vf”と“fix-Oops.tfm”とが生成されます（“fix”というプレフィクスは「等幅（fixed）」のつもり．多分，makejvf は，Oops.vf が fix-Oops.tfm を参照し且つ句読点等の位置調整をするようにしていて，また rml.tfm と同様の fix-Oops.tfm を作っているのだと思います）。

つまり，この場合，TEX が使うのは，Oops.tfm で，これが Oops.vf を介して fix-Oops.tfm に帰着されます。

あとは，Oops.tfm を NFSS に則ってロードして，dvi-ware/driver の map ファイルで fix-Oops.tfm と実フォントとを mapping することになります。またプリアンブルにまとめてしまいます：

```
\DeclareKanjiFamily{JY1}{jOops}{}
\DeclareFontShape{JY1}{jOops}{m}{n}{<-> Oops}{}
\DeclareRobustCommand
    {\jOopsfamily}{\kanjifamily{jOops}\selectfont}
\DeclareTextFontCommand{\textjOops}{\jOopsfamily}
\AtBeginDvi{
   \special{pdf:mapline fix-Oops H yourfont.ttf}
}
```

青い字の部分が欧文の場合と異なりますが，意味はお分かりになると思います。“JY1”というのは，NFSS で和文の横組みのエンコーディングを表わします。縦組みは“JT1”です[42]。ファミリ名は“jOops”としました。赤字の“yourfont.ttf”と書いたところに，手持ちの和文 TrueType フォントのファイル名を書けば，\jOopsfamily や \textjOops で，フォントが当該 TrueType フォントになるわけです（なお，jOops ファミリは m/n の設定しかしてませんので，m/n でしか使えません。ここではフォントを代替する設定も全くしていませんから，m/n 以外の属性の組み合わせが指定されてしまうと Warning が出て，結局 m/n になります）。

こんな風にして，必要な数の jfm ファイルと jvf ファイルを用意して，それぞれに設定をすれば，いわゆる「多書体化」が実現出来ることになります。

ちなみに，makejvf には，非漢字部分の jfm ファイルを指定する“-K”というオプションが用意されています。これを使いますと，例えば，

```
makejvf -K kana-Oops Oops.tfm fix-Oops
```

とすると，Oops.tfm から，“Oops.vf”，“kana-Oops.tfm”，“fix-Oops.tfm”が出来ます。

この場合には，Oops.tfm は，Oops.vf を介して，fix-Oops.tfm と kana-Oops.tfm とを参照していることになります。そこで，map ファイルの設定で，

```
fix-Oops H yourKanji.ttf
kana-Oops H yourHiragana.ttf
```

のように，それぞれを実フォントに対応させれば，LATEX 側の設定はさきほどのまま Oops.tfm ひとつをロードするだけで，漢字部分は yourKanji.ttf で組まれ，非漢字部分が yourHiragana.ttf で組まれるということになります。

jvf の中を覗いたことはないのですが，欧文と同じ仕組みだとすると，恐らく Oops.vf の冒頭には，

```
(MAPFONT D 0
   (FONTNAME fix-Oops)
   (FONTCHECKSUM O 36571141413)
   (FONTAT R 0.962216)
   (FONTDSIZE R 10.0)
   )
(MAPFONT D 1
   (FONTNAME kana-Oops)
   (FONTCHECKSUM O 24030505107)
   (FONTAT R 0.962216)
   (FONTDSIZE R 10.0)
   )
```

みたいなことが書いてあるんだろうな，ということが推測されます（数値は適当です）。

LATEX のフォントの話

目 次

おわり

[42] pLATEX では，横組みのクラスファイルを使っているときでも，縦組みのエンコーディングも一緒にチェックされるようなので，今の場合のように JY1 しか設定していないと，“LaTeX Font Info: No file JT1jOops.fd. on input line xxx.”とか“LaTeX Font Warning: Font shape ‘JT1/jOops/m/n’ undefined...”といった Warning が出ます。