

JFM ファイルフォーマット

ASCII Corporation & Japanese TeX Development Community

2023 年 2 月 23 日

JFM (Japanese Font Metric) は、pTeX で和文フォントを扱うためのフォントメトリックであり、オリジナルの TeX の TFM (TeX Font Metric) に相当する。pTeX と同じく株式会社アスキーによって開発され、この文書も pTeX に付属していたものであるが、ここでは 2018 年に日本語 TeX 開発コミュニティによって拡張された JFM フォーマットに基づいて説明する。

なお、pTeX の内部コードを Unicode 化した upTeX でも、JFM フォーマットの仕様は全く同じであり、ただ *char_type* テーブルに文字コードを格納するときに JIS コードを用いる (pTeX の場合) か、UCS-4 の下位 3 バイトを用いる (upTeX の場合) かだけが異なる。

目次

1	JFM ファイルの構成	3
1.1	<i>char_type</i> テーブル	3
1.2	<i>char_info</i> テーブル	4
1.3	<i>glue_kern</i> テーブル	4
1.4	<i>glue</i> テーブル	5
1.5	<i>param</i> テーブル	5
1.6	日本語 TeX 開発コミュニティによる新仕様	6
1.7	JFM フォーマットの制約事項	7
2	JPL ファイル	9
2.1	CHARSINTYPE	9
2.2	TYPE	9
2.3	GLUEKERN	10
2.4	DIRECTION	10
3	JFM を扱うプログラム	11
3.1	pPLtoTF, upPLtoTF	11
3.2	pTFtoPL, upTFtoPL	11
3.3	chkdvifont	12
3.4	makejvf	13
3.5	jfmutil	13

4	付録：利用可能な JFM について	14
4.1	ptex-fonts	14
4.2	uptex-fonts	16
4.3	実フォントへの割り当て (ptex-fontmaps)	17

1 JFM ファイルの構成

JFM ファイルのフォーマットは、基本的には TFM ファイルのフォーマットに準拠しており、TFM を拡張した形になっている。ここでは、主にその拡張部分について説明を行い、その他の部分に関しては、`TEX the program` 等の TFM の説明を参照してもらいたい。

JFM ファイル全体の構成は、表 1 (8 ページ) に示すとおりである。ここで TFM と異なるのは次の点である。

1. `char_type` のテーブルが付け加えられたこと。
2. `exten` の代わりに `glue` のテーブルが設けられたこと。
3. 2 に関連して、`lig_kern` から `glue_kern` テーブルへ変更されたこと。
4. これらに伴い、先頭のファイル内の各部分を規定するパラメータ表が変更されている。また、オリジナルの TFM との区別のために `id` を付加しており、先頭の半ワード (2 バイト) が横組用は 11, 縦組用は 9 である^{*1}。

最初の 7 ワードは半ワード (= 2 バイト) ずつに区切られ、JFM ファイルを構成する 14 個の要素のサイズが収められている^{*2}。これらの値は、すべて 2^{15} よりも小さい非負の値で、次の条件を満たしていなければならない：

$$\begin{aligned}bc &= 0 \\0 &\leq ec \leq 255 \\lf &= 7 + nt + lh + (ec - bc + 1) + nw + nh + nd + ni + nl + nk + ng + np\end{aligned}$$

ここで、`bc` と `ec` は最小・最大の文字タイプ番号、`nt` は `char_type` テーブルに登録された文字の数 (文字タイプ 0 も含む)、`nl` と `ng` はそれぞれは `glue_kern` テーブルと `glue` テーブルのサイズであり、その他の値は TFM を踏襲する。

JFM ファイルでも TFM ファイルと同じく、拡張子は `.tfm` が用いられる。

1.1 `char_type` テーブル

p`TEX` では欧文 `TEX` よりはるかに多くの文字を扱う必要があるが、そのほとんどは漢字であり、それらは全て同一の寸法 (全角幅) を持つ。また、括弧や句読点などの約物も種類が増えるが、こちらも幾つかのパターンに分類すれば済む (例えば “ ” と “ ” は同様に扱える)。

そこで、JFM フォーマットでは、同一の文字幅、高さ、前後に挿入されるグルー等、「その文字が持つ属性全てが同じもの」を 1 つの**文字タイプ** (`char_type`) として、欧文フォントの 1 文字と同様にして扱うようにしている。そして、文字コードと文字タイプとの対応付けを、この `char_type` テーブルを使って行う。

このテーブルの各エントリーは 1 ワード (= 4 バイト) で構成され、上位 3 バイトに文字コード (符号位置)、下位 1 バイトに文字タイプを持つ^{*3}。文字コードは、それが 16 進数 24bit

^{*1} 欧文 TFM の半ワードは `lf` すなわちファイルサイズであり、11 や 9 になることはない。

^{*2} 欧文 TFM では 12 個だが、JFM では `id` と `nt` が増え、`ne` が `ng` に置き換わったため 14 個である。

^{*3} 日本語 `TEX` 開発コミュニティによる新仕様；詳細は第 1.6 節を参照。

(3 バイト) で `0xABcdef` と表されるとき、`char_type` テーブルには `cd ef AB` として格納される。テーブル内にはコードの値の順番に収められていなければならない。またこのテーブルの先頭には、デフォルトのインデックスとして文字コード及び文字タイプの項が 0 のものが、必ず 1 つ存在しなければならない。このテーブルに登録されていない文字は、文字タイプが 0 として扱う。つまり、このデフォルト以外の文字幅、カーン等の属性を持つキャラクタのコードとタイプが 2 番目以降のエントリーとして存在しなければならない。

1.2 `char_info` テーブル

`char_type` をインデックスとしてこのテーブルを参照することにより、各 `char_type` の属性を検索する。各テーブルへのインデックス等の情報を次の順番でパッキングして 1 ワードに収めてある。

`width_index` (8bits) `width_table` へのインデックス

`height_index` (4bits) `height_table` へのインデックス

`depth_index` (4bits) `depth_table` へのインデックス

`italic_index` (6bits) `italic_table` へのインデックス

`tag` (2bits) `remainder` をどのような目的で使うかを示す。

`tag = 0` `remainder` の項は無効であり使用しないことを示す。

`tag = 1` `remainder` の項が `glue_kern` への有効なインデックスであることを示す。

`tag = 2, 3` JFM では使用していない。

`remainder` (8bits)

JFM では `bc` は必ずゼロ^{*4}なので、1 つの JFM に含まれる `char_info` は全部で `ec + 1` ワードになる。

1.3 `glue_kern` テーブル

特定の文字タイプの組み合わせ時に挿入すべき `glue` 又は `kern` を簡単なプログラム言語によって指定する。各命令は、以下の 4 バイトで構成される。

第 1 バイト (`skip_byte`)

- 128 より大きいとき

現在のワードが `char_info` から示された最初のワードである場合は、実際の `glue_kern` プログラムが `glue_kern[256 × op_byte + remainder]` から収められている (すなわち、再配置されている) ことを示す^{*5}。最初のワードでない場合 (すなわち、既に再配置先あるいはプログラムのステップを開始した後のワードである場合) は、その場でプログラムを終了する。

- 128 のとき

このワードを実行してプログラムを終了する。

^{*4} 前節にあるとおり、文字コード及び文字タイプの項が 0 のものが必ず 1 つ存在するため。

^{*5} 日本語 TeX 開発コミュニティによって新たにサポート；詳細は第 1.6 節を参照。

- 128 より小さいとき
このワードを実行した後、次のステップまでスキップするワード数を示す*6.

第 2 バイト (*char_type*)

- 次の文字の文字タイプが、このバイトで示す文字タイプ*7と同じ場合、第 3 バイトの処理を実行し、プログラム終了.
- そうでなければ次のステップへ.

第 3 バイト (*op_byte*)

この値によってグルーを扱うかカーンを扱うかを規定する.

- 127 以下の場合 *glue[remainder × 3]* のグルーを挿入.
- 128 以上の場合 *kern[remainder]* のカーンを挿入.

第 4 バイト (*remainder*)

第 3 バイトにより規定される *glue* または *kern* へのインデックスを示す.

1.4 *glue* テーブル

自然長、伸び長、縮み長の 3 ワードで 1 つのグルーを構成する (したがって、*ng* は必ず 3 の倍数となる). 各値は、 $\text{designsize} \times 2^{-20}$ を単位として表す.

第 1 ワード *width*

第 2 ワード *stretch*

第 3 ワード *shrink*

1.5 *param* テーブル

一応、以下のように定義されている.

param[1] 文字の傾き (*italic slant*).

param[2][3][4] 和文文字間に挿入するグルー (*\kanjiskip*) のデフォルト値.

param[5] pTeX で *zh* で参照される寸法.

param[6] pTeX で *zw* で参照される寸法.

param[7][8][9] 和文文字と欧文文字間に挿入するグルー (*\xkanjiskip*) のデフォルト値.



このように書かれているが、実際には pTeX の *zw* は「文字タイプ 0 の文字の幅」、pTeX の *zh* は「文字タイプ 0 の文字の高さと深さの和」である. 明示的に *\fontdimen* で取得する場合を除くと、JFM の *param* テーブルの値が用いられる状況は限られている.

*6 日本語 TeX 開発コミュニティによって新たにサポート; 詳細は第 1.6 節を参照.

*7 ここに文字タイプが格納されるため、文字タイプの上限は 255 なのである.

1.6 日本語 T_EX 開発コミュニティによる新仕様

長らく JFM フォーマットは株式会社アスキーが開発した当初仕様のままであったが、2018 年 1 月から 2 月にかけて、日本語 T_EX 開発コミュニティは下記の 3 点につき JFM フォーマットの仕様を拡張した。

1. *char_type* テーブルへの 3 バイトの文字コード格納
2. *glue_kern* テーブルでのスキップ (SKIP) コマンド使用
3. *glue_kern* テーブルでの再配置 (rearrangement)

和文 JFM でこれらの拡張機能が使われている場合は pT_EX p3.8.0 以上が必要である。

1.6.1 *char_type* テーブルへの 3 バイトの文字コード格納

char_type テーブルの各エントリは 1 ワード (= 4 バイト) で構成されるが、オリジナルの仕様では

- 上位半ワードに文字コード (符号位置)、下位半ワードに文字タイプを持つ

であった。pT_EX では内部処理に JIS が用いられ、JFM で扱う文字コードは 2 バイトが上限だったため十分であったが、upT_EX で BMP 超えの 3 バイトの文字を JFM で扱うことができなかった。また、オリジナルの仕様では文字タイプ用に下位半ワードが確保されている一方で、文字タイプの上限は 255 なので実はその上位バイトは常に 00 であり、勿体なかった。

そこで、日本語 T_EX 開発コミュニティの新仕様 (2018 年 1 月) では

- 上位 3 バイトに文字コード (符号位置)、下位 1 バイトに文字タイプを持つ
- 文字コードは、それが 16 進数 24bit (3 バイト) で 0xABCDEF と表されるとき、テーブルには cd ef AB として格納される

とした。オリジナルの仕様で常に 00 だったバイトが「実は文字コードの上位だった」と解釈することにして、3 バイト (U+10000 以上) の文字コードで不足する 1 バイトを確保したのである。これにより、新仕様はオリジナルの仕様の上位互換であることが保証されている。

1.6.2 *glue_kern* テーブルでのスキップ (SKIP) コマンド使用

「スキップ」(SKIP) は、元々アスキーの公式ページ^{*8} に文書化されてはいたが、実際には (p)PLtoTF は GLUEKERN プロパティ内で SKIP 命令を受け付けず、pT_EX もやはり JFM の SKIP 命令をサポートしていなかった。2018 年 2 月の日本語 T_EX 開発コミュニティの改修により、新たにサポートが開始された。

^{*8} <https://asciidwango.github.io/ptex/tfm/jfm.html>

1.6.3 *glue_kern* テーブルでの再配置 (rearrangement)

「再配置」は、サイズが 256 を超える大きな *glue_kern* テーブルを格納するための方策であり、欧文 TFM の *lig_kern* テーブルにおけるそれと同様である。2018 年 2 月に日本語 \TeX 開発コミュニティによって、 $\text{p}\TeX$ 及び pPLtoTF で新たにサポートされた。

1.7 JFM フォーマットの制約事項

冒頭に述べたとおり、文字タイプ (*char_type*) の上限は 255 であり、それに伴い可能な字幅も最大 256 種類しか許されない（さらに同じ字幅でも *glue* 又は *kern* の挿入規則を変える場合はその分減る）。しかし、実際のフォントでは字幅が 256 種類を超えることもあり、そのような時に JFM フォーマットは使用できない。

例として、OTF パッケージ (*japanese-otf*) [4] が提供する「ヒラギノフォントのプロポーショナル仮名を使う `\propshape` 命令」では

- 組版時に使われる `(up)phiraminw3-h.tfm` などの「 $(\text{u})\text{p}\TeX$ 用 JFM」は、約物類とプロポーショナル仮名のみに対応するので、字幅が 256 種類以内
- VF を介して実際のフォントに割り当てられる `hiramin-w3-h.tfm` などの「DVI ドライバ用 JFM」は、Adobe-Japan1-5 の 20,317 文字を全て CID コード順に実際のフォントに即して定義しようとして、字幅が 256 種類を超えてしまう

という状況になっており、後者は JFM フォーマットで定義できない。

一方で、 Ω で使われる OFM フォーマットは最大 65,536 種類の字幅を定義でき、かつ主要な DVI ドライバ (`dvipdfmx`, `dvips`) は JFM と OFM の両方に対応している。そこであくまで回避策であるが、 $(\text{u})\text{p}\TeX$ 用 JFM はそのままに、機能しない DVI ドライバ用 JFM の代わりに「正しい字幅を定義した OFM」を用意して `dvipdfmx` や `dvips` に優先的に使わせるという方法が長らく用いられてきた。OTF パッケージが提供する OFM ファイル群 (`\CID` で使われる `otf-cjmr-h.ofm`^{*9} や `\propshape` で使われる `hiramin-w3-h.ofm` など) はこの目的で用意されている [5].

^{*9} 実はこちらは字幅が 256 種類以内であり、JFM フォーマットで十分対応できるが。

表 1 JFM ファイルの構成

<i>id</i>	<i>nt</i>
<i>lf</i>	<i>lh</i>
<i>bc</i>	<i>ec</i>
<i>nw</i>	<i>nh</i>
<i>nd</i>	<i>ni</i>
<i>nl</i>	<i>nk</i>
<i>ng</i>	<i>np</i>
<i>header</i>	
<i>char_type</i>	
<i>char_info</i>	
<i>width</i>	
<i>height</i>	
<i>depth</i>	
<i>italic</i>	
<i>glue_kern</i>	
<i>kern</i>	
<i>glue</i>	
<i>param</i>	

id = JFM_ID number. (= 11 for yoko, 9 for tate)

nt = number of words in the character type table.

lf = length of the entire file, in words.

lh = length of the header data, in words.

bc = smallest character type in the font. (= 0 for JFM)

ec = largest character type in the font.

nw = number of words in the width table.

nh = number of words in the height table.

nd = number of words in the depth table.

ni = number of words in the italic correction table.

nl = number of words in the glue/kern table.

nk = number of words in the kern table.

ng = number of words in the glue table.

np = number of font parameter words.

2 JPL ファイル

TFM はバイナリ形式であるが、これをプロパティ（特性）という概念を使ってテキスト形式で視覚化したものが PL (Property List) ファイルである。同様に、JFM をテキスト形式で視覚化したものが **JPL (Japanese Property List)** ファイルである。JPL ファイルでも PL ファイルと同じく、拡張子は .pl が用いられる。

和文用 JPL に特有のプロパティとして、以下が存在する。

2.1 CHARSINTYPE

CHARSINTYPE <1 バイト長の整数> <任意個数の和文文字> の形式で、文字タイプ *char_type* 及びそれに属する文字コードを設定する。以下に示すのは、開き括弧類を *char_type* が 1 のグループに、閉じ括弧類を *char_type* が 2 のグループに指定する例である（最初の 0 1 と 0 2 はそれぞれ 8 進数の 1, 2）。

```
(CHARSINTYPE 0 1
  ( { [ { < 《 「 『 【
)
(CHARSINTYPE 0 2
) } ] } > 》 」 』 】
)
```

なお、<任意個数の和文文字> は直接入力のほか 16 進数コード値で与えることもできる。JIS コードの場合は J で始め、UCS コード (upTeX 専用) の場合は U で始める。以下の指定は上の例と等価である（例えば () は JIS 0x214A）。

```
(CHARSINTYPE 0 1
  J 214A J 214C J 214E J 2150 J 2152 J 2154 J 2156 J 2158 J 215A
)
(CHARSINTYPE 0 2
  J 214B J 214D J 214F J 2151 J 2153 J 2155 J 2157 J 2159 J 215B
)
```

2.2 TYPE

TYPE <1 バイト長の整数> <プロパティリスト> の形式で、文字タイプ *char_type* に含まれる文字の性質を定義する。欧文用 PL の CHARACTER プロパティに相当するが、利用可能な下位のプロパティは

- CHARWD <実数値>：文字の幅
- CHARHT <実数値>：文字の高さ

- CHARDP 〈実数値〉：文字の深さ
- CHARIC 〈実数値〉：文字のイタリック補正值

に限られる (NEXTLARGER と VARCHAR は使えない).

以下の例は, pTeX の既定で使用している `min10.tfm` のソース `min10.pl` からの抜粋である. `min10.tfm` のデザインサイズは 10.0pt であるので, これは文字タイプ 0 に含まれる文字の幅を 9.62216pt, 高さを 7.77588pt, 深さを 1.38855pt という定義である.

```
(TYPE 0 0
  (CHARWD R 0.962216)
  (CHARHT R 0.777588)
  (CHARDP R 0.138855)
)
```

2.3 GLUEKERN

JFM の `glue_kern` テーブルに収めるプログラムを記述する. 欧文用 PL の LIGTABLE と似ているが, LIG の代わりに GLUE を用いるところが異なる.

2.4 DIRECTION

DIRECTION 〈文字列〉の形式で, 日本語フォントの組方向を指定する. 〈文字列〉の一文字目が Y であれば横組, T であれば縦組として扱う. 未指定の場合はデフォルトで横組として扱う. (u)pTeX と一緒に標準で配布しているものは, 横組用に

```
(DIRECTION YOKO)
```

を, 縦組用に

```
(DIRECTION TATE)
```

を指定している.

3 JFM を扱うプログラム

pTeX と upTeX, あるいはそれらが生成した DVI を扱うプログラムが JFM を扱うのは当然であるが, ここでは JFM および関連するフォントフォーマットを扱うことに特化したプログラムの主なものを挙げる.

3.1 pPLtoTF, upPLtoTF

テキスト形式の JPL ファイルをバイナリ形式の JFM ファイルに変換する. いずれも欧文 TeX 用の `pltotf` の上位互換であり, 入力ファイルが欧文用の PL ファイルであれば欧文用の TFM を生成し, 和文用の JPL ファイルであれば和文用の JFM を生成する.

`ppltotf` と `uppltotf` の違いは, エンコーディングである.

- `ppltotf`: 常に **JIS** コードでエンコードされた JFM を生成するため, pTeX 用の JFM 生成には多くの場合 `ppltotf` コマンドが用いられる.
- `uppltotf`: デフォルトでは **Unicode** (UCS-4 の下位 3 バイト) でエンコードされた JFM を生成するため, 主に upTeX 用の JFM 生成に用いられる.

`ppltotf` においては, `-kanji` オプションで入力 JPL ファイルの文字コードを指定できる (有効な値は `eut`, `jis`, `sjis`, `utf8`). `uppltotf` でも `-kanji` オプションが同じく使えるが, 同時に JFM のエンコードも JIS になる (従って `ppltotf` と同じ挙動を示す) ことに注意^{*10}.

3.2 pTFtoPL, upTFtoPL

バイナリ形式の JFM ファイルをテキスト形式の JPL ファイルに変換する. いずれも欧文 TeX 用の `tftopl` の上位互換であり, 入力ファイルが欧文用の TFM であれば欧文用の PL ファイルを生成し, 和文用の JFM であれば和文用の JPL ファイルを生成する.

`ptftopl` と `uptftopl` の違いは, やはりエンコーディングである.

- `ptftopl`: 入力 JFM ファイルを常に **JIS** コードで解釈するため, pTeX 用の JFM デコードには多くの場合 `ptftopl` コマンドが用いられる.
- `uptftopl`: 入力 JFM ファイルをデフォルトでは **Unicode** で解釈するため, 主に upTeX 用の JFM デコードに用いられる.

`ptftopl` においては, `-kanji` オプションで出力 JPL ファイルの文字コードを指定できる (有効な値は `eut`, `jis`, `sjis`, `utf8`). `uptftopl` でも `-kanji` オプションが同じく使えるが, 同時に JFM も JIS コードで解釈される (従って `ptftopl` と同じ挙動を示す) ことに注意^{*11}.

^{*10} `uppltotf` における規定値は `uptex` であり, この場合は JFM が Unicode でエンコードされる.

^{*11} `uptftopl` における規定値は `uptex` であり, この場合は JFM が Unicode で解釈される.

3.3 chkdvifont

TeX Live 2019 で追加された比較的新しいコマンドであり、TFM/JFM ファイルの簡単な情報を表示する機能を持つ (Ω 用の OFM ファイルにも対応).

実行例を示す (注意: ファイル名の拡張子は省略不可. また, ファイルが現在のディレクトリにない場合は, フルパスの指定が必要).

■欧文 TFM の場合

```
$ chkdvifont cmr10.tfm
"cmr10" is a tfm file : 0 -> 127
checksum           = 4BF16079
design size        = 10485760 2^{-20} points = 10 points
```

一行目の表示から, 欧文 TFM であることと $bc = 0$, $ec = 127$ であることが読み取れる.

■和文横組用 JFM の場合

```
$ chkdvifont jis.tfm
"jis" is a jfm file : 0 -> 5
checksum           = 00000000
design size        = 10485760 2^{-20} points = 10 points
```

和文 (横組用) JFM であることと $bc = 0$, $ec = 5$ であることが読み取れる.

■和文縦組用 JFM の場合

```
$ chkdvifont upjisr-v.tfm
"upjisr-v" is a jfm(tate) file : 0 -> 5
checksum           = 00000000
design size        = 10485760 2^{-20} points = 10 points
```

和文縦組用 JFM であることと $bc = 0$, $ec = 5$ であることが読み取れる^{*12}.

■和文 JFM の拡張機能が使われている場合

第 1.6 節で述べたとおり, 日本語 TeX 開発コミュニティによって下記の 3 点につき JFM フォーマットの仕様が拡張されている.

1. *char_type* テーブルへの 3 バイトの文字コード格納
2. *glue_kern* テーブルでのスキップ (SKIP) コマンド使用
3. *glue_kern* テーブルでの再配置 (rearrangement)

和文 JFM でこれらの拡張機能が使われている場合は pTeX p3.8.0 以上が必要であるが, この

^{*12} *upjisr-v.tfm* は upTeX 用 JFM であるが, 原理的に pTeX 用と upTeX 用の JFM は区別できない.

情報も表示される（下の例は再配置あり）：

```
$ chkdvifont upphiraminw3-h.tfm
"upphiraminw3-h" is a jfm file : 0 -> 146
New features in pTeX p3.8.0 / JFM 2.0:
+ rearrangement in glue_kern
checksum          = 00000000
design size       = 10485760 2^{-20} points = 10 points
```

3.4 makejvf

JFM ファイルを基にして、VF (virtual font) を生成するプログラムである。makejvf が生成する和文 VF の目的は以下のとおりである：

- pTeX や upTeX で使われる多くの JFM では、約物類（かっこ、句読点など）の文字幅を半角幅として定義し、見た目の空白をグルーやカーンの挿入によって実現している。例えば“（”のような左に空きがある括弧類は、左半分は文字の一部として扱わず、「グルーによる半角分の右シフト」と「半角幅の(」として扱っている。
- 一方、DVI から PostScript や PDF へ変換時に使われる実際のフォントでは、約物類も全角幅でデザインされている。そのため、DVI に配置された“（”を実際のフォントの“（”に安直に置き換えると、想定よりも右にずれた位置に出力されてしまう。
- この位置ずれを補正するため、欧文フォントの合成や置換に実用されている VF（仮想フォント）という仕組みを和文フォントにも応用する。例えば VF 中に「“（”は左に半角分ずらして置き換える」という記述を追加することで、DVI ドライバがそれを解釈して位置補正できるようにする。

詳細はマニュアル makejvf.1（英語版）を参照してほしい。

3.5 jfmutil

JFM および和文 VF を操作する種々の機能を提供する Perl スクリプトである。主な機能は以下のとおり：

- 和文の仮想フォント（VF と JFM の組）に対応する独自仕様のテキスト形式（ZVP 形式）と、仮想フォントとの間の相互変換。欧文の仮想フォント（VF と TFM の組）とテキスト形式（VPL 形式）との間を相互変換する vftovp/vptovf の和文版に相当する。
- 和文・欧文問わず、VF それ単独に対応する独自仕様のテキスト形式（ZVP0 形式；ZVP 形式のサブセット）と、VF との間の相互変換。
- 和文の仮想フォント（VF と JFM の組）を別の名前複製する機能。VF 中に記録された参照先の JFM 名も適切に変更される。多書体化などに有用。

詳細は公式ドキュメントを参照してほしい。

4 付録：利用可能な JFM について

pTeX/upTeX と一緒に標準で配布している JFM について、簡単に説明する。

4.1 ptex-fonts

コミュニティによる配布場所は <https://github.com/texjporg/ptex-fonts> であり、内容物は以下のとおり。なお、pTeX および pLaTeX のフォーマットでは、既定で横組用に min*.tfm と goth*.tfm を、縦組用に tmin*.tfm と tgoth*.tfm を使用している。

- 株式会社アスキーによるもの
 - min*.tfm, goth*.tfm, nmin*.tfm, ngoth*.tfm (以上、横組用)
 - tmin*.tfm, tgoth*.tfm (以上、縦組用)
- 東京書籍印刷、現リーブルテックの小林肇氏によるもの
 - jis*.tfm (JIS フォントメトリック)

■株式会社アスキーによる JFM

しばしば「min10 系」と呼ばれる。歴史が非常に古く、開発当時のドキュメントは残念ながらほとんど残っていない。以下は、株式会社アスキーによる「日本語 TeX version j1.7」[1] に付属していたドキュメント README.KANJI からの抜粋である。

■漢字フォント

ディレクトリ jfms 内の漢字フォント用 JFM (TFM) ファイルは、以下に示すサイズ (ドット数) のフォントを想定して作成されています (日本語 version j0.3 の配布時のものと若干異なります。注意して下さい)。尚、(株)アスキーより販売されている「日本語 Micro TeX」においても、これと全く同じものを使用しています。

TeX での呼び名	実寸 (pt)	300dpi	480dpi	240dpi	118dpi
5pt	4.58222	19	30	15	7
6pt	5.49866	23	36	18	9
7pt	6.4151	27	42	21	10
8pt	7.33154	30	49	24	12
9pt	8.24799	34	55	27	13
10pt	9.16443	38	61	30	15
10pt * 1.2 ^{1/2}	10.0391	42	67	33	16
10pt * 1.2 ¹	10.9973	46	73	36	18
10pt * 1.2 ²	13.1968	55	88	44	22
10pt * 1.2 ³	15.8361	66	105	52	26
10pt * 1.2 ⁴	19.0034	79	126	63	31
10pt * 1.3 ⁵	22.804	95	151	75	37

このサイズ以外のフォントを使用する場合は、プリンタドライバで疑似的にこれらのサイズに合わせるか、そのフォントに合わせた JFM (TFM) ファイルを作成するかのどちらかになります。互換性を考慮する場合は、前者の方法を取るべきです。しかし、それではどうしても気に入らないというのであれば、jtex/TeXware に JFM 用の tftopl および pltotf がありますから、これを使用して新しい JFM ファイルを作成して下さい。これらのユーティリティのドキュメントは、株式会社アスキー発行の「日本語 TeX テクニカルブック I」を参照して下さい。(中略)

jfms 内には、min*.tfm、goth*.tfm の他に nmin*.tfm と ngoth*.tfm が含まれています。前者は、句読点、ピリオド、カンマといくつかの仮名文字との間でカーニング処理を行うようになっていますが、後者ではこの処理を行っていません。両者間では、これ以外の違いはありません。ただしマクロ中では、前者のみが定義されており、後者を使用する場合は自分で定義する必要があります。

通常の和文フォントは正方形の仮想ボディに収まるようにデザインされるが、例えば公称 10 ポイントの min10 は「幅 9.62216pt, 高さ 7.77588pt, 深さ 1.38855pt」の扁平な矩形でデザインされている (zw = 9.62216pt と zh = 9.16443pt が一致しない)。また、例えば「ちょっと」が「ちよつと」ではなく「ちよつと」と詰まるなどの不自然な挙動も知られている [2]。ただし、互換性維持のため、コミュニティが修正を行う予定はない。

■ JIS フォントメトリック

jis*.tfm は、JIS X 4051-1995 「日本語文書の行組版方法」になるべく即したメトリックとして開発された [3]。min10 系を「そのまま置き換えて使ってもらおう」ことを意図しており、そのため、min10 系の字詰まりの不具合等は直しつつ、zw, zh の値を仮定した既存スタイルに影響を与えないように min10 の zw・zh の寸法は温存した設計になっている。なお、jsclasses は横組用の jis.tfm と jisg.tfm を標準で使用している。

なお、jis.tfm (n 無し) と jisn.tfm (n 有り) の違いは

中黒〈・〉, コロン〈:〉, セミコロン〈;〉の組み方を「半角幅+前後グルー」とするか「全角幅」とするか

だけである。

■対応する VF と実フォント

第 3.4 節で述べたとおり、pTeX で使用する JFM と実際のフォントでは一部の字幅が異なり、その位置ずれを補正するために VF を用いている。対応は以下のとおり (左が pTeX 用 JFM, 右が DVI ドライバ用 JFM である^{*13}。実際の OpenType/TrueType フォントの割り当ては、map ファイルに依存する; 後述)。

- min*.tfm → min*.vf → rml.tfm (→明朝体フォント, 横組)
- goth*.tfm → goth*.vf → gbm.tfm (→ゴシック体フォント, 横組)

^{*13} DVI ドライバ用 JFM は全ての文字を全角扱いとしており、グルー挿入などは一切定義していない。当然、pTeX の組版時には使用すべきでない。

- `tmin*.tfm` → `tmin*.vf` → `rmlv.tfm` (→明朝体フォント, 縦組)
- `tgoth*.tfm` → `tgoth*.vf` → `gbmv.tfm` (→ゴシック体フォント, 縦組)
- `jis*.tfm` → `jis*.vf` → `rml.tfm` (→明朝体フォント, 横組)
- `jisg*.tfm` → `jisg*.vf` → `gbm.tfm` (→ゴシック体フォント, 横組)
- `jis*-v.tfm` → `jis*-v.vf` → `rmlv.tfm` (→明朝体フォント, 縦組)
- `jisg*-v.tfm` → `jisg*-v.vf` → `gbmv.tfm` (→ゴシック体フォント, 縦組)


また, `tmin*.vf` 及び `tgoth*.vf` では「縦書き時にソース中にクオート (') < ") を入力すると, ミニユート (') < ") を加工した文字に置き換える」という処置も加えてある (`makejvf` の `-m` オプションの機能による). 一方, `jis*-v.vf` 及び `jisg*-v.vf` ではそのような処置を行わない.

4.2 uptex-fonts

配布場所は <https://github.com/texjporg/uptex-fonts> であり, 内容物は以下のとおり. なお, `upTeX` および `upLATEX` のフォーマットでは, 既定で横組用に `upjisr-h.tfm` と `upjisg-h.tfm` を, 縦組用に `upjisr-v.tfm` と `upjisg-v.tfm` を使用している.

■日本語用 (upjis 系)

JIS フォントメトリックを基に, JIS X 0208 → Unicode で追加された記号類を各文字タイプに追加した日本語用メトリックである. `upjis*` と `upjpn*` の JFM の中身は同じであり, 対応する VF も (2022 年現在は) 実質的に同等である.

 両者の違いは歴史的経緯にすぎない. かつては DVI ドライバが `set3` 命令すなわち「3 バイト以上の文字出力」に非対応のケースが多かったため, 以下の仕様を設けていた.

- 標準フォント `upjis*` … BMP 内のみ出力可 (VF は `set2` まで)
- オプションの `upjpn*` … BMP 外も出力可 (VF は `set3` も使用)

しかし, 2018 年以降は DVI ドライバの `set3` 対応が進んだことから, 標準フォント `upjis*` でも「Adobe-Japan1 で定義された BMP 外の文字」を出力できるように, VF に `set3` 命令も使用することとした [6]. さらに, `TeX Live 2021` 以降は各種 DVI ドライバによる JFM 由来の VF の解釈が以下のように拡張された:

MAPFONT で指示されている font ID の一番若いものが JFM であり,
要求されたコードポイントが明言されていない場合は,
それが最小の font ID に属すとみなし, そのコードポイントそのものを出力する.

この解釈拡張により, VF に個々の文字を (`set3` 命令を使うなどして) 定義することなく, 省略されている場合でも文字出力が可能となった [7].

■中国語簡体字用 (upsch 系)・中国語繁体字用 (uptch 系)・韓国語用 (upkor 系)

日本語と中国語・韓国語では, 実際のフォントにおける記号類のデザイン (仮想ボディの中での配置) や字詰めの方式が異なる点が多い. そこで, 実験的ではあるが, 日本語用 (`upjis` 系) の組み方では明らかに不自然な点を調整した JFM をそれぞれ用意した. 詳細は [8] を参照されたい.

■対応する VF と実フォント

対応は以下のとおり（左が upTeX 用 JFM, 右が DVI ドライバ用 JFM である。実際の OpenType/TrueType フォントの割り当ては, map ファイルに依存する; 後述)。

- `upjisr-*.tfm` → `upjisr-*.vf` → `uprml-*.tfm` (→日本語・明朝体)
- `upjisg-*.tfm` → `upjisg-*.vf` → `upgbm-*.tfm` (→日本語・ゴシック体)
- `upjpnrm-*.tfm` → `upjpnrm-*.vf` → `uprml-*.tfm` (→日本語・明朝体)
- `upjpngt-*.tfm` → `upjpngt-*.vf` → `upgbm-*.tfm` (→日本語・ゴシック体)
- `upschrm-*.tfm` → `upschrm-*.vf` → `upstsl-*.tfm` (→簡体中国語・宋体)
- `upschgt-*.tfm` → `upschgt-*.vf` → `upstht-*.tfm` (→簡体中国語・黒体)
- `uptchrm-*.tfm` → `uptchrm-*.vf` → `upmsl-*.tfm` (→繁体中国語・宋体)
- `uptchgt-*.tfm` → `uptchgt-*.vf` → `upmhm-*.tfm` (→繁体中国語・黒体)
- `upkorrm-*.tfm` → `upkorrm-*.vf` → `uphysmjm-*.tfm` (→韓国語・バタン体)
- `upkorgt-*.tfm` → `upkorgt-*.vf` → `uphygt-*.tfm` (→韓国語・ドトゥム体)

[TODO] クオート記号の割り当て

4.3 実フォントへの割り当て (ptex-fontmaps)

先述のとおり, 「DVI ドライバ用 JFM」に実際の OpenType/TrueType フォントを割り当てるのは map ファイルの仕事である。TeX Live では `ptex-fontmaps` として複数のプリセットを用意しており, それらを `kanji-config-updmap(-sys)` というコマンドで切り替えることができる。詳細は公式ドキュメント及び <https://github.com/texjporg/ptex-fontmaps> を参照のこと。

参考文献

- [1] 株式会社アスキー, 「アスキー日本語 TeX 配布テープ 1992.02.24 (release 1.12)」
- [2] 乙部徹己, 「min10 フォントについて」, 2000/12/12,
<http://argent.shinshu-u.ac.jp/~otobe/tex/files/min10.pdf>
- [3] 奥村晴彦, 「pLaTeX2e 新ドキュメントクラス」, 2016/07/07,
<https://oku.edu.mie-u.ac.jp/~okumura/jsclasses/>
- [4] Shuzaburo Saito, 「Open Type Font 用 VF」, 2019/04/01,
<https://psitau.kitunebi.com/otf.html>
- [5] Hironobu Yamashita, 「otf-cjXX-X.ofm は何のため?」, 2019/05/31,
<https://github.com/texjporg/japanese-otf-mirror/issues/15>
- [6] Takuji Tanaka, 「uptex-fonts で提供するフォントの文字範囲など」, 2017/07/22,
<https://github.com/texjporg/uptex-fonts/issues/3>
- [7] Takuji Tanaka, 「和文 vf の fallback を dviware で」, 2020/03/11,
<https://github.com/texjporg/tex-jp-build/issues/99>
- [8] Hironobu Yamashita, 「中韓フォントの JFM」, 2017/07/01,

<https://github.com/texjporg/uptex-fonts/issues/2>