

JFM ファイルフォーマット

ASCII Corporation & Japanese TeX Development Community

2020 年 3 月 21 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	2
1.1	<i>char_type</i> テーブル	2
1.2	<i>char_info</i> テーブル	3
1.3	<i>glue_kern</i> テーブル	3
1.4	<i>glue</i> テーブル	4
1.5	<i>param</i> テーブル	4
1.6	日本語 TeX 開発コミュニティによる新仕様	5
2	JPL ファイル	6
3	JFM を扱うプログラム	8
3.1	pPLtoTF, upPLtoTF	8
3.2	pTFtoPL, upTFtoPL	8
3.3	chkdvifont	9
3.4	makejvf	10
3.5	jfmutil	10

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 (7 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

pTeX 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 (符合位置)、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} 前節にある通り、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 TeX 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 TeX 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 pTeX 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 (符合位置)、下位半ワードに文字タイプを持つ

であった。pTeX 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、upTeX 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 TeX 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 (符合位置)、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、pTeX 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 TeX 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、pTeX 及び pPLtoTF で新たにサポートされた。

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。

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.

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^{*9}.

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 `tfootpl` 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.

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

- `tfootpl`: 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 `tfootpl` コマンドが用いられる.
- `utfootpl`: 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.

`tfootpl` 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`). `utfootpl` 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 `tfootpl` と同じ挙動を示す) ことに注意^{*10}.

^{*9} `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.

^{*10} `utfootpl` 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^{*11}。

■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

^{*11} *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。