1 Compatibility

1.1 TeX Font Metric

Not finished...

1.2 Subfont Support

Plain TEX is only capable of handling 7 or 8-bit encodings and all font usable by TEX can not contain more than 256 glyphs. But many languages requires more characters than this limit. Well known examples are Chinese, Japanese, and Korean (CJK) languages. CJK fonts may contain even more than ten thousands of glyphs in single font. Thus, plain TEX has fundamental problem in treating those languages. There are several way to typeset text written in languages with larger character set. One way to do this is to split large font into multiple smaller fonts, *subfonts*, and manage characters by identifier of subfont to which character belong and single-byte code within that subfont. There are several TEX or LATEX macro packages does this process automatically, like CJK-LATEX, HLATEX, and NTT-jTEX.

However, dvipdfmx want single multibyte font rather than multiple subfonts for various reason, and it provide a way to map subfonts originated from a single font to a single *intermediate font* without relying on virtual fonts. This is essentially the inverse process of what macro packages like CJK-LATEX does. Support for CJK-LATEX and HLATEX is realized by this feature. This mapping is done through *subfont definition* (SFD) file. The subfont definition file describes how font is split into a set of subfonts, and is used by `ttf2pk` program to automate things and consintently manage them in font-independend manner. Please refer documents from `ttf2pk` program for detailed information on SFD file format.

To enable this feature, you should use a special name of format

```
tex_font_name@SFD_name@
```

in font mapping file for TEX font name rather than listing font mapping for all subfonts separately. For example, to map subfonts of traditional chinese font with prefix `bsmi` to single intermediate font with Big-5 encoding, write

```
bsmi@Big5@  ETen-B5-H  bsmi00lp
```

The effect of using this font name is that if dvipdfmx encounters fonts with prefix `bsmi` such as `bsmi01` within DVI file and if the font has subfont identifier (digits following `bsmi` in this case) allowed for SFD `Big5`, dvipdfmx maps those subfonts to an intermediate double-byte font `bsmi@Big5@` and re-encode text in Big-5 encoding accroding to the rule for subfont creation described in the SFD file `Big5`. The mapping is done when dvipdfmx is interpreting DVI file but not when it is handling text and font for PDF output.

2 Font and Encoding

Managing font is rather complicated task in TEX and in using DVI drivers. It is also true for dvipdfmx, rather, it is worse than in other DVI drivers. Dvipdfmx supports a variety of font formats and they are sometimes treated differently depending on how they are used, and they are sometimes converted to different font format for embedding to output PDF file.[1] Dvipdfmx tries to absorb difference of font format as much as possible. And indeed there are mostly no distinction between them unless you use multi-byte encodings.

2.1 Supported Font Format

The following font formats are currently supported by dvipdfmx:

- **PK font**

  TEX's PK font format is supported as in dvipdfm with few modification to avoid problems rarely heppens in some PDF viewers. Always embedded as Type 3 (bitmap) font.

- **PostScript Type 1 font**

  PostScript Type 1 fonts are supported as in dvipdfm but it was completely rewritten. They are always converted to CFF (Compact Font Format) which is another representation of PostScript Type 1 font such as PFB and PFA. As PDF support Type 1 font, there are no need to convert them to CFF, however, conversion to CFF makes it easier to fully support Unicode and higher compression ratio is expected since CFF format is free from encryption. PostScript Type 1 font may be embedded as Type 1C (CFF) or CIDFontType 0 CIDFont. Multiple-Master font is not supported yet.

- **TrueType font**

  Both TTF and TTC (TrueType Collection) is supported. There are many enhancement to dvipdfm including subsetting. They may be embedded as TrueType (simple) font or as CIDFontType 2 CIDFont which is basically CID-keyed version of Type 42 font. Dvipdfmx may fail to handle old TrueType font for Mac OS since it depend on extension made for Windows and OS/2.

- **OpenType font**

  OpenType fonts with TrueType outlines (`.ttf`) and with PostScript outlines (`.otf`) is both supported. They may be embedded as simple font (Type 1C or TrueType), CIDFontType 2, or CIDFontType 0 CIDFont depending on outline format and their use.

In addition to the fonts listed above, there are special kind of font supported by dvipdfmx for CJK support, which is described in "Acrobat and Printer Resident Font".

[1]There might be minor quality loss introduced by font format conversion. But there should not be any quality loss when the document is rendered at enough resolution. Dvipdfmx does not do outline format conversion such as TrueType to PostScript conversion. Hinting is preserved whenever possible. The only expection is ghost stem hint in Type 1 font. It is not distinguished from ordinary stem hints in Type 1 to CFF conversion.

`-e` *`number`*	Set horizontal scaling of font.
`-s` *`number`*	Slant the font.
`-r`	Obsolete...
`-b` *`number`*	Specify boldness parameter of fake bold font.
`-l` *`string`*	...

2.2 Font Mapping

Not finished yet.

2.3 Accessing Glyphs in Font

To deal with problems releted to font you may encounter when using dvipdfmx, you should know that there are several different way to use font in dvipdfmx. They are classified according as how dvipdfmx access glyphs contained in a font, and is determined from what is specified in encoding field of font mapping record. Dvipdfmx basically uses *PostScript based glyph access* to get glyph descriptions (glyph metrics or outline data describing detailed shape of glyph) for when *encoding file* or *CMap PostScript Resource* is used as encoding. *Unicode based glyph access* is chosen when keyword `unicode` is specified as encoding.

PostScript based glyph access PostScript Type 1 font traditionally uses glyph names which is a string representing them, such as `AE` and `quoteleft`, to uniquely identify individual glyphs contained in a font. Glyph descriptions are stored with those string keys and accessed with it. In CID font format, which is relatively newer format in PostScript, non-negative integers called CID (Character Identifier) are used instead of strings for this purpose. CID-keyed font is originally developed for supporting large character set such as Chinese, Japanese, and Korean in PostScript. In those languages, using strings is quite inefficient and it is difficult to give consice names to all glyphs and to manage it since there are more than several ten thousands of glyphs actually used in publishing. Glyphs are uniquely identified by specifying both CID and *character collection* to which the glyphs belongs.

Unicode based glyph access Glyphs are accessed through Unicode and tags representing variant forms. This is experimental feature and is discussed in a separate section. See the section "Unicode Support".

Both the above method is supported for all font format[2] except PK font and non-embedded CIDFont mentioned in "Acrobat and Printer Resident Font". However, as natural way to access glyphs depend on font format, dvipdfmx may require auxiliary file(s) to absorb the difference of font format. Those files are described in "Adobe Glyph List and CID-To-Code Mapping" in details. In some situations, dvipdfmx mixes those method to increase reliability on finding glyphs. Thus, you should install at least Adobe Glyph List file. And CJK TrueType font users must install CID-To-Code Mapping files adequate to each languages to enable PostScript based glyph access.

Each method have their own mirit and demirit: PostScript based glyph access is, in general, more flexible and reliable in that users can re-encode font rather arbitrary by supplying encoding file or CMap, and all glyphs are accessible in this way for PostScript fonts. It is well suited for using PostScript and PostScript flavored OpenType fonts, but it may not work as reliably for TrueType because conversions are sometimes involved several times to enable access to glyph with PostScript glyph names or CIDs. Using Unicode is much simpler in some cases but tend to lack flexibility and reliability in many cases mostly due to limitations in dvipdfmx, and several problems are never resolved in dvipdfmx as a DVI driver.

[2]That is not exactly true: CIDs can not be used for Type 1 and glyph names can not be used for OpenType CIDFont. TrueType fonts must have Unicode cmap for Unicode access.

As long as you stay on 8-bit encodings, the only choice is to use PostScript based glyph access and there are essentialy no difference with dvips on using font. Just specify the name of encoding file (`.enc` file) which defines correspondence between input codes in DVI and glyphs in actual font to which TEX font is mapped. For CJK support and other multi-byte character support, CMap is used in place of encoding file to translate input codes to CIDs in PostScript based glyph access. Dvipdfmx can work relatively nicely with TrueType and OpenType fonts (not different than using Type 1 font). For CJK-LATEX and Omega users, Unicode is reasonable choice but dvipdfmx can't fulfill all requirement for high-quality publishing. If you have good quiality OpenType font with PostScript outline and want to use all of glyphs contained within them, PostScript based glyph access might be suited for your purposes.

2.4 CMap and Encoding File

Nothing yet...

2.5 Adobe Glyph List and CID-To-Code Mapping

As mentioned earlier on this section, there are several method to access glyphs in a font. However, since each font formats uses different way to identify glyphs, some font format may not work well for some situation. To minimize restrictions and differences on using font, dvipdfmx tries to compensate difference of font format using auxiliary files, *Adobe Glyph List* and *CID-To-Code Mapping* files.

Adobe Glyph List This resource associate PostScript glyph names with it's corresponding Unicode characters.

CID-To-Code Mapping This resource describes the mapping from CIDs in a given character collection to character codes in some character encoding (inverse mapping of usual CMap). When the target encoding is Unicode encodings, the term *ToUnicode CMap* may be used to denote this resource in this document.

The Adobe Glyph List file is required for supporting PostScript based glyph access in TrueType or OpenType font with TrueType outlines if font itself does not provide information for associating PostScript glyph names with glyphs contained in font. This information is stored in an auxiliary TrueType table called `post` table (version 2.0)[3] and most of TrueType fonts has this table for compatibility with PostScript. However, TrueType fonts often has incomplete table and OpenType font may not have this table at all. In this case, Adobe Glyph List file may be used to find glyphs through Unicode if font supports Unicode encoding. It is also required to enable Unicode based glyph access in PostScript Type 1 font. Installation of this resource is recommended since other features relies on it too.

The file name of this resource must be `glyphlist.txt` and must be located in dvipdfmx's search path for "other text files". The content of this file should look like:

```
# comment
A;0041
AE;00C6
```

Lines starting with `#` is a comment. All other lines actually describes the mapping, the first field is PostScript glyph name, a semicolon delimiting each fields follows, and the next field is corresponding Unicode values (separated by spaces if the glyphs is mapped to a sequence of multiple Unicode characters) in uppercase hexadecimal notion.

For CID fonts (CJK) support, CID-To-Code Mapping resource is required for supporting PostScript based glyph access in TrueType and OpenType font with TrueType outlines. You should at least install ToUnicode CMap appropriate for language you use. Dvipdfmx also support other type of CID-To-Code mapping than ToUnicode for several CJK encodings to support older TrueType fonts not supporting Unicode. Table 1 lists CID-To-Code Mappings required for each encodings supported by dvipdfmx and for Adobe's character collections. Those files are available from Adobe. You may already have those files

[3]Apple seems to have more sophisticated extension to manage glyphs but dvipdfmx not supporting this yet.

Table 1: CID-to-Code mapping file for each TrueType encodings.

Encoding	**Platform**	**Character Collection**	**ToCode Mapping**
Unicode		Adobe-GB1	Adobe-GB1-UCS2
		Adobe-CNS2	Adobe-CNS1-UCS2
		Adobe-Japan1	Adobe-Japan1-UCS2
		Adobe-Korea1	Adobe-Korea1-UCS2
RPC	Windows	Adobe-GB1	Adobe-GB1-GBK-EUC
	Mac OS		Adobe-GB1-GBpc-EUC
Big5	Windows	Adobe-CNS1	Adobe-CNS1-ETen-B5
	Mac OS		Adobe-CNS1-B5pc
SJIS	Windows	Adobe-Japan1	Adobe-Japan1-90ms-RKSJ
	Mac OS		Adobe-Japan1-90pv-RKSJ
Wansung	Windows	Adobe-Korea1	Adobe-Korea1-KSCms-UHC
	Mac OS		Adobe-Korea1-KSCpc-EUC

somewhere on your hard-disk if you have installed localize versions of Adobe (Acrobat) Reader (or with Acrobat Reader Asian Font Packs) or GhostScript. As this resource file uses same syntax as ordinary CMap PostScript resources, they should be installed in dvipdfmx's search path for "cmap files".

There is known problem in ToUnicode CMaps: Apple and Microsoft using different Unicode assignment for several CJK symbols, and Adobe seems following Apple's one. This will cause problems when using TrueType fonts for Microsoft Windows platform with PostScript based glyph access. Dvipdfmx automatically adjust few problematic symbols frequently used such as horizontal ellipsis (three-dot leader), wave dash, and double vertical line, but it will not work perfectly. Another format may be introduced to resolve this issue.

Table 2: Acrobat and PostScript printer resident font supported by dvipdfmx.

Language	**Character Collection**	**PostScript Font Name**
Chinese (trad.)	Adobe-CNS1-0	MHei-Medium-Acro
		MSung-Light-Acro
	Adobe-CNS1-4	AdobeMingStd-Light-Acro
Chinese (simpl.)	Adobe-GB1-2	STSong-Light-Acro
	Adobe-GB1-4	AdobeSongStd-Light-Acro
Japanese	Adobe-Japan1-2	HeiseiMin-W3-Acro
		HeiseiKakuGo-W5-Acro
		Ryumin-Light
		GothicBBB-Medium
	Adobe-Japan1-4	KozMinPro-Regular-Acro
		KozGoPro-Medium-Acro
Korean	Adobe-Korea1-0	HYGoThic-Medium-Acro
		HYSMyeongJo-Medium-Acro
	Adobe-Korea1-2	AdobeMyungjoStd-Medium-Acro

2.6 Acrobat and Printer Resident Font

As CJK fonts contains too many glyphs, it takes quite long time and cost much to create font with enough quality. Due to this fact, there are not so many CJK fonts freely available there and sometimes they tend to be rather expensive. Furthermore, file size become very large when they are embedded into PDF.

To deal with those problems, dvipdfmx supports several fonts that act like "PDF Standard Font for CJK" listed in table 2. For those fonts, minimal font information usually required by PDF viewers are available from the dvipdfmx's built-in data. They does not contain any glyph (outline or bitmap) data required to draw actual shape of each glyphs, hence, PDF viewers must replace those fonts with suitable one. This means that the reproducibility of document layout, when the document is opened on the recipient's system, is not guaranteed at all, however, it works quite well for CJK text if you do not use special glyphs in your document. Please use those fonts if you are sure that all peoples who receives your document have suitable font installed on their system and if your interest is only the file size.

This feature is provided only for convenience and please do not expect you can always obtain correct result since font substitution is dependent on the ability of PDF viewers. Note that proportional glyphs are not supported for those fonts. Basically, only glyphs of which widths are determined solely from their

CID and character collection but do not differ among different font are supported; i.e., full-, half-, quarter-, and third-width forms.

All of the above fonts containing "-Acro" in their PostScript font name are available from Adobe as part of Acrobat Reader Asian Font Packs for use with Acrobat Reader, other font may found in PostScript printers.

2.7 Notes on OpenType CIDFont Support

There are few differences in OpenType CIDFont support between dvipdfmx and other software regarding the treatment of "OpenType" fonts: In dvipdfmx generated PDF file, embedded CID fonts inherit all glyph metric information from the original OpenType font's glyph metrics tables in their CIDFont dictionary entries `W` (horizontal metrics) and `W2` (vertical metrics). This difference especially affects when you are using pre-rotated forms of proportional glyphs; The `W2` metrics of CIDFont may contain proportional vertical displacement in the case of dvipdfmx, while other converter like Distiller may treat them like fixed-pitch font, and place each glyphs with position adjustment for compensating the difference from OpenType vertical metrics. You should prepare TeX font metrics using information from `vmtx` and `VORG` for vertical typesetting.

For the position vector (it describes displacement from the origin used for horizontal writing to the origin used for vertical writing), the vertical component of position vector is taken from `VORG` table whenever available, otherwise, all vertical component of the position vector is set to `sTypoAscender` value in `OS/2` table, and the horizontal component is set to a half of horizontal advance width. Currently, dvipdfmx does not make use of `BASE` table.

2.8 Using TrueType Font as CID-keyed Font

When a valid CMap is specified in the encoding field of font mapping and font is mapped to TrueType font, dvipdfmx will treat that TrueType font as CIDFontType 2 CIDFont. However, as TrueType font is not CIDFont, they does not supports accessing glyphs with CIDs. Hence, dvipdfmx requires auxiliary resource, CID-To-Code Mapping, to achieve compatibility between PostScript flavored font (CID-keyed font) and TrueType font. Please refer "Adobe Glyph List and CID-To-Code Mapping" for description about this resource.

In addition to this resource, the name of character collection to be used for this font should be specified in font mapping record. This tells dvipdfmx information about glyph set covered by the font and ordering of them, and is specified by appending a `/` and a string denoting character collection immediately after the font name as follows:

```
    mincho    UniJIS-UCS2-H  ttmincho/AJ14
```

In the above example, `ttmincho` is converted to CIDFontType 2 CIDFont with Adobe-Japan1-4 character collection. This can be implicit if you are not using Identity CMaps; dvipdfmx will use the information available from CMaps applied to TrueType font: The following font mapping record

```
    mincho    UniJIS-UCS2-H  ttmincho
```

implies Adobe-Japan1-4 since the CMap `UniJIS-UCS2-H` is a mapping from character codes in UCS-2 to CIDs in Adobe-Japan1-4 character collection. However,

```
mincho    Identity-H     ttmincho
```

does not suggest any useful information since Identity CMap is generic CMap that does not implies any specific character collection. In this case, dvipdfmx will use font's internal glyph ordering. If you meant the font `mincho` using Adobe-Japan1 ordering, you should explicitly specify this as

```
mincho    Identity-H     ttmincho/AJ12
```

Abbreviation AK1, AC1, AG1, and AJ1 can be used in character collection field for Adobe's character collection Adobe-Korea1, Adobe-CNS1, Adobe-GB1, and Adobe-Japan1 respectively. If you want to use other character collection, you must specify it with full name, e.g., for Adobe-Japan2-0,

```
min-hk-h  Identity-H     ttmincho/Adobe-Japan2-0
```

There are several limitations and problems in using TrueType fonts in this way. The reason for this is, of course, because TrueType font is not CIDFont. For CJK fonts, you should be careful about use of proportional glyphs.

3 Unicode Support

Since the version 200408XX, there is some support for Unicode in dvipdfmx. This feature is highly experimental, especially features described in the section "Selecting Glyph Variants" and "Discretionary Ligatures" is provided only for testing purpose and to investigate what is required for basic Unicode support. Please be aware that format of configuration files and options mentioned in this section is subject to futrue change.

3.1 Unicode Support in Dvipdfmx

Dvipdfmx accepts encoding keyword `unicode`, in addition to `default` and `none`, in the font mapping file for all font formats supported by dvipdfmx except PK font and non-embedded CID font.

TrueType and OpenType font must have Windows UCS-2 format 4 cmap (character mapping) subtable or Windows UCS-4 format 12 cmap subtable. For Type 1 font support, you must install Adobe Glyph List (AGL) file[4] in the dvipdfmx search path for "other text files". AGL file describes mapping from PostScript glyph names to the corresponding Unicode values. All features related to Unicode requires this file for PostScript Type 1 font (and sometimes for TrueType font) support. Type 1 font support is not fully functional yet: Glyph substitution and PUA assignment of ligatures described later are not supported for Type 1 font.

If you have single big Unicode Omega font metric (OFM) file, simply specify `unicode` in the encoding field of fontmap file:

```
cyberbit        unicode  cyberbit
```

for Omega and Aleph, or if you are using CJK-LaTeX package,

```
cyberb@Unicode@ unicode  cyberbit
```

should work. In the previous versions of dvipdfmx, some tricks are used to support Unicode encoding as input DVI encoding. If your fontmap file contains the following line,

```
cyberb@Unicode@ Identity-H cyberbit/UCS
```

you should modify it as the above example.

As there are no TeX font metric format supporting four-byte character code, Unicode support in dvipdfmx is limited to a single Unicode code block of the same group-plane per single TeX font. When font is embedded into PDF, they are converted to CIDFont format. So, if Unicode encoding is used, the resulting PDF file may not print correctly when it is sent to PostScript (clone) printer after conversion to PostScript. Please refer Adobe's web site[5] for PostScript interpreter versions supporting CID-keyed font.

[4] `http://partners.adobe.com/asn/tech/type/unicodegn.jsp`
[5] Adobe Solutions Network, Type Technology - CID-Keyed Fonts.

3.2 Selecting Glyph Variants

There are several problems in using Unicode with TeX. If you switch from legacy TeX's way to Unicode, you will soon be troubled about how to pick up desired glyphs from a font: Recently developed fonts in "intelligent" font format such as OpenType tend to have all variant forms of single character in single font file, and they offer a way to select a specific instance of characters from multiple variants rather than providing multiple font with different styles: You can not choose desired glyph simply by changing font file in this case. Dvipdfmx can partially resolve this issue.

In OpenType font format, application program may use data contained in OpenType Layout (OTL) `GSUB` (glyph substitution) table to choose a glyph from multiple variants representing same character. For example, if user request to replace numerals 0123456789 in font's default style with oldstyle 0123456789, the program looks for OTL table if font supports glyph substitution for oldstyle numerals. In dvipdfmx, how OTL glyph substitution is done is controled by a file with suffix `.otl` (not zero-t-one). The content of this file looks like:

```
script    latn;
language  *;

@numeral = [zero-nine];
required {
    substitute @numeral by @numeral.onum;
};
```

In the above example, script that this file is supposed to control is `latn` (Latin script) and all features belong to languages that uses this script may be used. The line starting with "`@`" declares a class of Unicode characters, `@numeral` represents `zero`, `one`, ..., `nine` here. This is the same as writing

```
@numeral = [uni0030 uni0031 uni0032 uni0033 uni0034
            uni0035 uni0036 uni0037 uni0038 uni0039];
```

where all characters are explicitely listed by their Unicode values. The block enclosed by braces actually specifies the rule of glyph substitution: All glyphs corresponding to `@numeral` in a font is replaced with the resulting glyphs after OTL feature `onum` (oldstyle figures) is applied. The keyword `required` forces dvipdfmx to abort if one or more of glyphs for `@numeral` is missing in font or if they do not have oldstyle forms. Dvipdfmx currently supports `prefered` to warn all missing glyphs and `optional` to silently ignore them.

After you have finished writing configuration file, and suppose that you have saved it as `oldstyle.otl`,[6] you can turn on OTL `onum` feature with the option `-l` (lowercase L):

```
lplru-onum   unicode  pala  -l oldstyle
```

As there are several features common to several languages with same script, it is desirable to place them into single file. For this purpose, dvipdfmx supports `option` in configuration file which is enabled by specifying tag identifies that option in dvipdfmx font mapping file.

[6]This file must be located under dvipdfmx's search path for format "other text files" (This usually includes current working directory).

For example, to put substitution rules for oldstyle and small caps in single configuration file, use `option` as:

```
script    latn;
language  *;

@numeral   = [zero-nine];
@lowercase = [a-z];
@lcasesupp = [agrave-odieresis oslash-ydieresis];
option oldstyle {
    required {
        substitute @numeral by @numeral.onum;
    };
};
option smallcap {
    required {
        substitute @lowercase by @lowercase.smcp;
    };
    prefered {
        substitute @lcasesupp by @lcasesupp.smcp;
    };
};
```

If you save this file as `latin.otl`, then you can select `smallcap` option by

```
lplrcu       unicode  pala  -l latin:smallcap
```

to replace lowercase letters with SMALL CAPS.

An another quick example that illustrate the use of this feature is selection of simplified forms and traditional forms for Chinese.

```
script    hani;
language  (dflt|ZHT_|ZHS_);

@hanideo = [uni4E00-uni9FAF];

# Rules common to smpl and trad may appear here.

option smpl {
    optional {
        substitute @hanideo by @hanideo.smpl;
    };
};
option trad {
    optional {
        substitute @hanideo by @hanideo.trad;
    };
};
```

Here language tag `ZHT_` represents traditional Chinese and `ZHS_` is simplified Chinese, and `dflt` is the default language for script `hani` (CJK Ideograph) which may differ among fonts. All features with script `hani` and language `dflt`, `ZHT_`, or `ZHS_` can be controlled within this configuration.

3.3 Discretionary Ligatures

The glyph substitution feature described in the previous section is limited to single glyph to single glyph substitution. However, OpenType font may contain ligatures not frequently used and not found in Unicode. To use those ligatures, you must assign code point to them. To do this, use `assign` command in OTL configuration file as follows.

```
optional { # Ignore if missing
    assign uniE00B to Q_u.dlig;
    assign uniE018 to s_p.dlig;
};
```

The above example assign Unicode code point `U+E00B` in Private Use Area (PUA) to ligature "Qu" of letter "Q" and "u", and `U+E018` to "sp" ligature of "s" and "p". As those ligatures are not frequently used, they may be described in OTL `GSUB` feature `dlig` (discretionary ligatures) but not in `liga` which is for standard ligatures.

The last argument to this command is of the form

`input_char_sequence.otl_tag`

where the *`input_char_sequence`* is the sequence of input Unicode character represented as character name or `uniXXXX` form. The charcter names is the same as PostScript glyph names listed in AGL file, but they are restricted only to the names corresponding to single Unicode character, and names like `Asmall` (small cap A) should not be used here. They are joined with underscores to represent ligature. For examples, if there are ligature of "T" and slashed-l, the input character sequence can be `T_lslash`, `T_uni0142`, `uni0054_lslash`, and so on. The order that ligature component appears in the character sequence should be in writing order. It may not be left-to-right order in some script. The number of ligature components in single input character sequence is currently limited up to 16 Unicode characters.

Unicode value to which the ligature glyph is mapped is single Unicode value in the PUA range. As far as concerned with ligatures, PUA assignment can be font specific, since users usually do not directly input character codes representing ligatures when preparing document. However, it is recommended to assign unique code point among all fonts as much as possible.

For list of registered OpenType Layout tag and it's brief description, please refer OpenType Specification, "OpenType Layout tag registry".[7] Dvipdfmx currently only supports `GSUB` substitution with lookup type 1 (single) and type 4 (ligature), and lookup type 2 (multiple) will never be supported. It might be possible to support lookup type 3 (alternate) substitution but there are problems in how to identify glyph variants of same kind among other fonts (there may be problem even in same font with different versions). Glyph substitution is basically limited to one-to-one or many-to-one substitution. The substituted glyph with many-to-one substitution must be accessed through PUA; dvipdfmx will not replace multiple characters in DVI file with single glyph and contextual substitution will never be done. TeX is responsible for doing such things. What dvipdfmx will offer is to allow access to a glyph by some way.

[7] `http://www.microsoft.com/typography/otspec/ttoreg.htm`

3.4 ToUnicode CMap Support

In PDF, Unicode appears whenever exchange of text information between PDF viewers and other applications happens, or when PDF viewers interact with users. Dvipdfmx try to create ToUnicode CMap, which explicitly tell the mapping from character code to Unicode, when doing so is preferable and is possible. This is important to ensure that PDF viewers correctly handle text strings within PDF document for extraction of text information. If Unicode mapping can not be done, copy-and-pasting text, searching text, and conversion from PDF to other format such as plain-text or HTML, will not work. This feature is not available for PK font.

There are nothing to be done to enable ToUnicode support if you are using `unicode` encoding. However, dvipdfmx requires AGL file for Type1 and TrueType font if single-byte encoding is used; it read mapping data necessary for converting PostScript glyph names to Unicode from AGL file. For this feature to work correctly, both font and encoding file must use proper glyph names.

ToUnicode CMap is created for each encodings and if 10% of glyph names can not be converted to Unicode, ToUnicode CMap is not embedded for that encoding. Please use "`-v`" option to find out which encodings dvipdfmx failed creating or embedding ToUnicode CMap, or option "`-vv`" to see all glyphs with no Unicode mapping available.

If dvipdfmx reports as no Unicode mapping available for some glyph names. And if you know the correspondence between glyph names and Unicode for those glyph names. You can append entries to AGL file. For example, CMMI (Computer Modern Italic) font contains "sharp" and Unicode code point for this glyph is `U+266F`, but AGL file from Adobe lacks entry for this glyph name. Hence, You should add the following lines

```
# Addition for CMMI
sharp;266F
```

to the file `glyphlist.txt`. All lines starting with `#` is treated as comment and is simply ignored.

Dvipdfmx adjust few glyph names like "oneoldstyle", "summationtext", and "parenleftbig" as suitable; those glyph names are converted to glyph names with suffix like "one.oldstyle", "summation.text", and "parenleft.big" when the original glyph name does not have valid Unicode code point assigned or is mapped to Private Use Area; and then Unicode values for glyph names without suffix is used, e.g., Unicode value for "oneoldstyle" is not Adobe's corporate use mapping `U+F731` but `U+0031`. (Glyph names written in the embedded font remains unmodified. This conversion is done temporarily only for the purpose of ToUnicode CMap creation.)

For CID font using character collection different from the one published by Adobe,[8] dvipdfmx looks for ToUnicode CMap file and copy them to output PDF file. The name of this resource file must be in the form *registry-ordering*-UCS2 where *registry* and *ordering* is the registry and ordering of the character collection used by the font. When preparing ToUnicode CMap, please do not use `usecmap` operator as dvipdfmx can not handle them correctly yet.

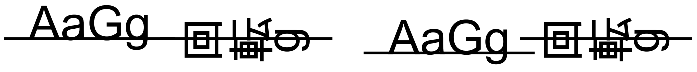
[8]Excluding Adobe-Japan2 which contains glyphs from JIS X 0212:1990 (Hojyo Kanji).

4 Vertical Writing

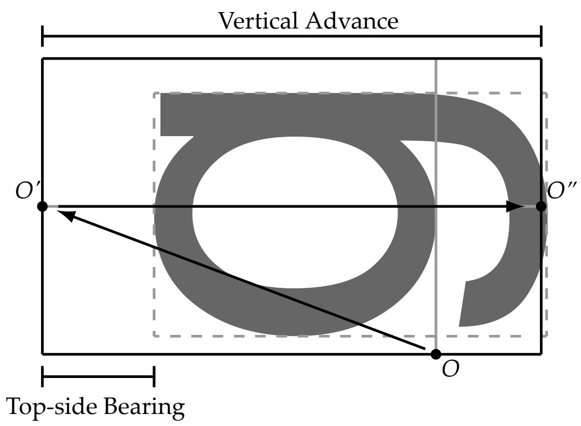
4.1 Vertical Writing Support in Dvipdfmx

In dvipdfmx, all font have an attribute *writing mode* which takes value 0 or 1 (horizontal or vertical) and all text is set either in *horizontal composition mode* or in *vertical composition mode*. The text composition mode is controlled by a special DVI command and only supported by some extension to TEX, and font's writing mode changes according as the `WMode` (writing mode) of current CMap used for that font, or in the case that `unicode` is used as encoding, it should be explicitly specified with `-w` option in font mapping record (the default is horizontal).

Glyphs in vertical font (writing mode 1) is rotated 90 degrees in the counter clock-wise direction in horizontal composition and is aligned at vertical center. The baseline should be adjusted by TEX if both horizontal and vertical text are used in the document.



The font should provide enough information to place glyphs in vertical text correctly: The position vector which describes relation between current text position (O' in the example below) to the origin in glyph coordinate system (O) and the vertical displacement vector which is displacement from current text position to next text position (O'') is necessary.



The example on left is *wrong* in that top-side bearing and vertical advance is not set properly. OpenType font should have `vmtx` table to set those values and if the font uses PostScript outline they should also have `VORG` in addition to that. If those tables are not found and if the font is not designed especially suitable for vertical writing, correct result may not be obtained.

The vertical component of vertical displacement vector (vertical advance) is obtained from `vmtx` table (horizontal component is always zero). The vertical component of position vector is obtained from `VORG` table for OpenType font with PostScript outline, and is determined from top-side bearing in `vmtx` and glyph's bounding box for TrueType. The horizontal component of position vector is set to a half of horizontal width. The default value of advance height is sum of typographic ascent and descent distance, and for vertical component of position vector, the default is set to ascent. Those default values should work for CJK font, however, it is not adequate to latin glyphs in general.

The "width" in TEX font metric (TFM) is interpreted as the vertical advance for vertical font, and must be consistent with vertical metrics of actual font. The "height" ("depth") is the distance from centerline to "right" ("left") -most side of box surrounding glyph (those values affect only when dvipdfmx is calculating bounding box of text, e.g., when creating link annotation).

OpenType font may use OpenType Layout `GPOS` table to adjust placement of several glyphs or to modify glyph metrics for vertical writing. However, dvipdfmx does not support them. All glyphs should have fixed metric both for horizonal and vertical composition and they never change depending on the context in dvipdfmx. Virtual font might be enough to get similar effect as OTL `GPOS` for simple task such as reducing excess white space on top-side but may not always work as intended.

In preparing document, it is recommended to use extended version of TEX natively supporting vertical writing (i.e., not relying on glyph rotation) whenever possible: Text selection in PDF viewer might not work correctly for "vertical" text with glyphs rotated (for example text on the previous page), and it may seriously affect searching text if glyph rotation is done by `\special` command since it result in glyph-by-glyph rotation, furthermore output is very wasteful as transfomation matrix is set every time glyph is placed; dvipdfmx does not do optimization for that. Things are a bit better with dvipdfmx's glyph rotation feature since it treat text as a text as much as possible and rotate whole text rather than rotating each glyphs.

There are several (not so many) features useful for vertical wrting in PDF. To rotate whole page 90 degrees in the clock-wise direction, you can use `pdf:` special command

```
\special{pdf:put @pages << /Rotate 90 >>}
```

PDF viewer will rotate pages if they support that. This is usefull for pseudo-vertical writing using glyph rotation. The other thing is `Direction` entry in PDF document's viewer preferences dictionary. You can set this to `R2L` via

```
\special{pdf:docview <<
   /ViewerPreferences << /Direction /R2L >>
>>}
```

This does not affect rendering of page contents but PDF applications (such as n-up program) that recognize this dictionary entry may use it. For example, Acrobat arranges pages in the correct order when the document is opened in view mode "Continuous - Facing" (first page on right and second page on left). This preference dictionary entry merely declares that the natural direction to which text or lines proceeds is from right to left.

4.2 Vertical Composition

Undocumented yet.

5 Test

MS Arial Unicode font using Japanese forms 骨草 by default. To select Simplified Chinese forms 骨草, you must use OTL feature `smpl`, and `trad` feature for Traditional Chinese forms 骨草.

ABCDEFGabcdefgαβγδεζη
ABCDEFGabcdefgαβγδεζη
ABCDEFGabcdefgαβγδεζη
ABCDEFGabcdefgαβγδεζη
ABCDEFGabcdefgαβγδεζη
ABCDEFGabcdefgαβγδεζη

The Quantum spitz attack Kafka check special fact
fjord craft fb fh fft ffb ffh ffk SS
fi fl fb fh fk ffh ffi ffl
The Quantum spitz attack Kafka check special fact
fjord craft fb fh fft ffb ffh ffk SS
fi fl fb fh fk ffh ffi ffl
The Quantum spitz attack Kafka check special fact
fjord craft fb fh fft ffb ffh ffk SS
fi fl fb fh fk ffh ffi ffl
The Quantum spitz attack Kafka check special fact
fjord craft fb fh fft ffb ffh ffk SS
fi fl fb fh fk ffh ffi ffl

6 Related Files

There are three attached files in this PDF document:

- The attached file `dpxunicode.tex` is Lambda source of this document.
- The attached file `dpxunicode-res.zip` is zip archive containing OFM, .fd, and dvipdfmx fontmap files used to produce this document.
- The attached file `glyphlist-cm-add.txt` is plain text file. This file describes additional PS glyph name to Unicode mapping entries for glyphs contained in Computer Modern font (but not in the original AGL file).

If your PDF viewer does not support extracting embedded files, please use `pdftosrc` program from pdfTEX project[9] to extract embedded files. To do this, first search PDF file (open with binary editor, not with PDF viewer) by keyword `/Filespec`, you should find data sections looks like:

```
58 0 obj
<<
/Type /Filespec
/F (dpxunicode.tex)
/EF <<
/F 55 0 R
>>
>>
```

Record two numbers between `/F` and `R` whithin the block `/EF << >>` (55 and 0 in this example) and run `pdftosrc` program as

```
pdftosrc dpxunicode.pdf 55 0
```

on console. Then you'll get `dpxunicode.55` which is extracted embedded file. The appropriate file name for this data is recorded in `/F (...)`.

[9] http://www.tug.org/applications/pdftex/