The dvipdfmx User's Manual

The dvipdfmx project team

May 7, 2017

Contents

Chapter 1

Getting Started

1.1 Introduction

The `dvipdfmx` (formerly `dvipdfm`-cjk) project provides an extended version of the `dvipdfm`, a DVI to PDF translator developed by Mark A. Wicks.

The primary goal of this project is to support multi-byte character encodings and large character sets such as for East Asian languages. This project started as a combined work of the `dvipdfm`-jpn project by Shunsaku Hirata and its modified one, `dvipdfm`-kor, by Jin-Hwan Cho.

Extensions to `dvipdfm` include,

- Support for OpenType and TrueType font, including partial support for OpenType Layout for finding glyphs and vertical writing.
- Support for CJK-LaTeX and HLaTeX with Subfont Definition Files.
- Support for various legacy multi-byte encodings via PostScript CMap Resources.
- Unicode related features: Unicode as an input encoding and auto-creation of ToUnicode CMaps.
- Support for pTeX (a Japanese localized variant of TeX) including vertical writing extension.
- Some extended DVI specials.
- Reduction of output files size with on-the-fly Type1 to CFF (Type1C) conversion and PDF object stream.
- Advanced raster image support including alpha channels, embedded ICC profiles, 16-bit bit-depth colors, and so on.
- Basic PDF password security support for PDF output.

Some important features are still missing:

- Linearization.
- Color Management.

- Resampling of images.
- Selection of compression filters.
- Variable font and OpenType 1.8.
- and many more...

dvipdfmx is now maintained as part of TEX Live. Latest source code can be found at TEX Live SVN repository. For an instruction on accessing the development sources for TEX Live, see,

http://www.tug.org/texlive/svn/

This document, "The dvipdfmx User's Manual", was originally prepared for TEX Live 2017. Current maintainer of this document is Shunsaku Hirata. Latest version and contact information can be found at:

http://github.com/shirat74/dvipdfm-x-doc

Please send quenstions or suggestions.

1.1.1 xdvipdfmx

xdvipdfmx is an extended version of dvipdfmx, and is now incorporated into dvipdfmx.

The xdvipdfmx extensions provides support for the Extended DVI (.xdv) format generated by XETEXwhich includes support for platform-native fonts and the XETEX graphics primitives, as well as Unicode text and OpenType font.

XETEX originally used a Mac-specific program called xdv2pdf as a backend program instead of xdvipdfmx. The xdv2pdf program supported a couple of special effects that are not yet available through xdvipdfmx: The Quartz graphics-based shadow support, AAT "variation" fonts such as Skia, transparency as an attribute of font, and so on. It would be nice if they continue to be supported. Suggestions and help are welcomed.

1.1.2 Legal Notice

1.2 Installation and Usage

Typical usage and installation steps are not different from the original dvipdfm. Please refer documents from dvipdfm distribution for detailed instruction on how to install and how to use dvipdfm. The dvipdfm manual is available from its CTAN site:

http://www.ctan.org/tex-archive/dviware/dvipdfm

Option	Description
`-C` *number*	Specify miscellaneous option flags. See, section of "Incompatible Changes" for details.
`-S`	Enable PDF encryption.
`-K` *number*	Set encryption key length. The default value is 40.
`-P` *number*	Set permission flags for PDF encryption. The *number* is a 32-bit unsigned integer representing permission flags. See, section of "Encryption Support". The default value is `0x003C`.
`-I` *number*	Life of image cache in hours. By specifying value `0` `dvipdfmx` erases cached images, and value `-1` erases all cached images and does not leave newly generated one. The default value is `-2`. (ignore image cache)
`-M`	Process METAPOST generated PostScript file.
`-E`	Always try to embed fonts *regardless of liscensing*.
`-O` *number*	Set maximum depth of open bookmark item.

Table 1.1: Additional command line options recognized by `dvipdfmx`.

The minimal requirements for building `dvipdfmx` is the `kpathsea` library. `zlib` for compression and `libpng` for PNG inclusion are highly recommended. Optionally, the libpaper library may be used to handle paper size.

This document only describes additions and modifications to `dvipdfm`. Please refer the "Dvipdfm User's Manual" available from the CTAN site mentioned above for basic usage.

Some additional command line options recognized by `dvipdfmx` are listed in Table 1.1. Try

```
dvipdfmx --help
```

for the list of command line options and thier explanations.

1.3 Quick Guide

`dvipdfmx` is supposed to be used by users of LaTeX packages for typesetting CJK languages like HLaTeX and CJK-LaTeX, and TeX variants such as XeTeX, pTeX, and upTeX. This section is intended to be a quick guide for each users.

1.3.1 XeTeX

For XeTeX users, most part of this document is irrelevant except section of "Graphics and Image Formats" and "DVI Specials".

1.3.2 pTeX

pTeX users are at least required to install several auxilially files mentioned in the section of "Auxilially Files" and to setup fontmappings. Just install the *adobemap-*

pings and *glyphlist* for auxilially files. (As TEX Live basic intallation requires them, they are probably already installed for TEX Live users.)

Setting up fontmaps can be done easily with help of the *ptex-fontmaps* package. For examples, to use with IPA fonts (contained in the *ipaex* package), run,

```
updmap ipaex
```

Alternatively, just for a quick test of installation, try the following:

```
\documentclass{article}
\begin{document}
\special{pdf:mapline rml H KozMinProVI-Regular}
...Some Japanese text goes here...
\end{document}
```

In this example, PDF viewer which can handle substitute font is required since `dvipdfmx` does not embed fonts.

For using Japanese text in PDF document information and annotations, put the following `special` command,

```
\AtBeginDocument{\special{pdf:tounicode 90ms-RKSJ-UCS2}}
```

in the preamble. The above `special` command instructs `dvipdfmx` to convert text encoded in Shift-JIS to Unicode. For EUC-JP, replace 90ms-RKJK-UCS2 with EUC-UCS2.

1.3.3 upTEX

upTEX users are basically the same as pTEX users but there are two choices for setting fontmaps. Setup fontmaps as mentioned in the above for pTEX, or use keyword `unicode` in the encoding field of fontmap files.

The former case might be easier as auto-creation of fontmap files can be done with the `updmap` program and the *ptex-fontmaps* package. But in this method there are some difficulties when using fonts which employ character collection (glyph repertoire) other than Adobe-Japan1 in the case of PostScript flvavored OpenType fonts. In the later case, the *adobemappings* package is not reuiqred and newer PostScript flavored OpenType fonts which do not employ Adobe-Japan1 can be used too.

Using `unicode` is more simpler and intuitive thus it is recommended to use this method.[1] Typical example fontmap entries for using Adobe's SouceHan fonts look like:

```
urml    unicode  SourceHanSerifJP-Light.otf
```

[1] For TEX Live 2017. Earlier versions have buggy support.

```
urmlv   unicode  SourceHanSerifJP-Light.otf  -w 1
ugbm    unicode  SourceHanSansJP-Medium.otf
ugbmv   unicode  SourceHanSansJP-Medium.otf  -w 1
```

As in pTEX, the following `special` instruction is necessary to PDF document information and annotations to be shown correctly:

```
\AtBeginDocument{\special{pdf:tounicode UTF8-UCS2}}
```

Here, input encodig is assumed to be UTF-8.

1.3.4 CJK-LATEX

CJK-LATEX users are required to have Subfont Definition Files to be installed. They are available as part of the *ttfutils* package.

To use TrueType Ariphic fonts provided by the *arphic-ttf* package:

```
\documentclass{article}
\usepackage{CJKutf8}
...Other packages loaded here...
\AtBeginDocument{%
  \special{pdf:tounicode UTF8-USC2}%
  \special{pdf:mapline bsmiu@Unicode@ unicode bsmi00lp.ttf}%
 }
\begin{document}
\begin{CJK}{UTF8}{bsmi}
...some Chinese text goes here...
\end{CJK}
\end{document}
```

Here, `pdf:mapline` special is used to setup fontmappings.

1.4 Auxiliary Files

This section is mostly for supporting legacy encodings and legacy font format such as PostScript Type1 font. XETEX users may skip this section.

`dvipdfmx` has a capability to handle various input encodings from 7-bit encodings to variable-width multi-byte encodings. It also has some sort of Unicode support. Several auxiliary files which are not common to TEX users are needed to enable those features. This section shortly describes about them.

1.4.1 PostScript CMap Resources

PostScript CMap Resources[2] are required for supporting lagacy encodings such as Shift-JIS, EUC-JP, Big5, and other East Asian encodings. `dvipdfmx` internally identifies glyphs with identifiers (CIDs[3]) represented as an integer ranging from 0 to 65535 in the CID-based glyph access. PostScript CMap Resources describes the mapping between sequences of input character codes and CIDs. `dvipdfmx` has an extensible support for multi-byte encodings via PostScript CMap Resources.

CMap files for standard East Asian encodings, for use with Adobe's character collections, are included in the *adobemapping* package. The latest version of those CMap files maintained by Adobe can be found at Adobe's GitHub Project page:

`http://github.com/adobe-type-tools/cmap-resources`

Those files are mandatory for supporting pTEX. upTEX users may also want to install them but they are not required.

1.4.2 Subfont Definition Files

CJK fonts usually contain several thousands of glyphs. For using such fonts with (original) TEX, which can only handle 8-bit encodings, it is necessary to split fonts into several "subfonts". Subfont Definition File (SFD) specify the way how those fonts are split into subfonts. `dvipdfmx` uses SFD files to convert subfonts back to a single font.

SFD files are not required for use with TEX variants which can handle multi-byte character encodings and large character sets such as pTEX, upTEX, XETEX, and Omega. HLATEX and CJK-LATEX users are required to have those files to be installed. SFD files are available as a part of the *ttfutils* package for TEX Live users.

1.4.3 The Adobe Glyph List and ToUnicode Mappings

The Adobe Glyph List[4] (AGL) describes correspondence between PostScript glyph names (e.g., `AE`, `Aacute`,...) and Unicode character sequences representing them. Some features described in the section "Unicode Support" requires AGL file.

`dvipdfmx` looks for the file `glyphlist.txt` when conversion from PostScript glyph names to Unicode sequences is necessary. This conversion is done in various situations; when creating ToUnicode CMaps for 8-bit encoding fonts, finding glyph descriptions from TrueType and OpenType fonts when the font itself does not provide a mapping from PostScript glyph names to glyph indices (version 2.0 "post" table), and when the encoding `unicode` is specified for Type1 font.

AGL file is included in the *glyphlist* package. The latest version can be found at Adobe's GitHub site:

`http://github.com/adobe-type-tools/agl-aglfn`

ToUnicode Mappings are similar to AGL but they describe correspondence between CID numbers (instead of glyph names) and Unicode values. The content of those files are the same as CMap Resources. They are required when using

[2]See, "Adobe CMap and CIDFont Files Specification"
[3]PostScript terminology "Character IDentifier".
[4]See, "Adobe Glyph List Specification"

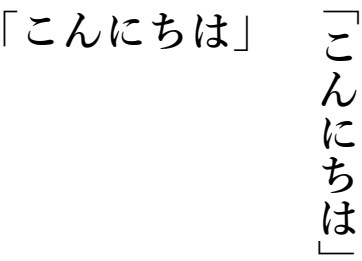


Figure 1.1: An example of horizontal and vertical text; left and right corner brackets are replaced with their vertical counterparts.

TrueType fonts emulated as CID-keyed fonts. They should be found in the same directory as ordinary CMap files.

ToUnicode Mapping files are included in the *adobemapping* package. Those files are not required for XeTeX users.

1.5 Overview of Extensions

This section gives a quick overview of `dvipdfmx`'s extended capabilities.

1.5.1 CJK Support

There are many extensions made for supporting CJK languages. Features described here is mainly for CJK languages but their use is actually not limited to it. Those features are implemented in a generic way so that it can be beneficial to users who are not involved in CJK languages.

Legacy Multi-byte Encodings

`dvipdfmx` has an extensible support for encodings by means of PostScript CMap Resources. Just like `enc` files are written for 8-bit encodings, one can write their own CMap files to support custom encodings. See, Adobe's technical notes for details on PostScript CMap Resources.

Vertical Writing

`dvipdfmx` supports vertical writing extension used by pTeX and upTeX. A DVI instruction to set writing mode is supported. OpenType Layout GSUB Feature is supported for selecting vertical version of glyphs.

1.5.2 Unicode Support

Unicode support here consists of two parts: Supporting Unicode as input encodings and making output PDF files "Unicode aware".

Unicode as Input Encoding

`dvipdfmx` recognizes an additional keyword `unicode` in fontmap files to declare that Unicode values are used in input DVI files. Unicode support is basically limited to the Basic Multilingual Plane (BMP) since there are no support for code ranges that requires more than three bytes in TFM and extended TFM formats.

ToUnicode CMap Support

In PDF,it is often the case that text is not encoded in Unicode. However, modern applications usually want them represented in Unicode to make it usable. ToUnicode CMap is a bridge between PDF text string encodings and Unicode encodings, and they makes it possible to extract text in PDF as Unicode encoded strings. It is important to make resulting PDF search-able and copy-and-past-able. Dvipdfmx supports the auto-creation of ToUnicode CMaps.

It will not work properly for multiply encoded glyphs due to fundamental limitations of Unicode conversion mechanism with ToUnicode CMaps.

1.5.3 Other Extensions

`dvipdfmx` can generate encrypted PDF documents to protect its contents from unauthorized access. It is limited to passorwd-based authentication, and public-key based authentication is not supported. The 256-bit AES encryption is also supported for PDF version 1.7 setting although it may not be supported by PDF viewers.

There are various other improvements over `dvipdfm`. The most notable one is more improved PDF input and output support: The cross-reference stream and object stream introduced in PDF-1.5 are also supported.

Chapter 2

Graphics

2.1 Image Inclusion

The basics of incorporating images into output PDF is the same as in `dvipdfm`. To do this, LaTeX users can simply use the *graphicx* package. (possibly with the driver option `dvipdfmx`) This section is *not* for providing a how-to guide to include images but just for supported graphics and image formats along with supported features.

Graphics support was mostly rewritten in `dvipdfmx`. Support for BMP and JPEG2000 was added. An effort to preserve more information originally found in included images, e.g., embedded ICC Profiles and XMP Metadata, was made.

However, `dvipdfmx` does not support various features common to graphics manipulation programs such as resampling, color conversion, and selection of compression filters. Thus, it is recommended to use other programs specialized in image manipulation for preparation of images.

2.1.1 Supported Graphics File Formats

Supported formats are, PNG, JPEG, JPEG2000, BMP, PDF, and METAPOST generated EPS. All other format images, such as SVG and PostScript, must be converted to PDF before inclusion. The '`-D`' option, as in `dvipdfm`, can be used for filtering images.

Notes on PNG Support

PNG is supported as in `dvipdfm` with many improvements.

PNG support includes most of important features of PNG format such as color palette, transparency, 16-bit bit-depth color, embedded ICC Profiles, calibrated color, and embedded XMP Metadata.

In including PNG images, `dvipdfmx` first decompresses image data and then compresses (if requested) it again. For better compression ratio, a preprocessing filter (Predictor filter) might be applied before compression. `dvipdfmx` supports TIFF Predictor 2 and PNG optimum filter. However, there is yet no way to specify which predictor function is to be used and currently PNG optimum filter is always employed.

Feature	PDF Version Required
16-bit Color Depth	Version 1.5
Transparency	Full support for alpha channel requires PDF version 1.4. Color key masking (a specific color is treated as fully transparent) requires 1.3.
XMP Metadata	Version 1.4
ICC Profile	Version 1.3

Table 2.1: PNG features and corresponding PDF versions required.

Predictor filter is preprocessing filter to image data for improving compression. Using a predictor filter usually gives better compression but in many cases compression speed becomes significantly slower. Try '`-C 0x20`' command line option to disable predictor filters and to check if slowness is due to filtering.

For PNG optimum filter, the "minimum sum of absolute differences" heuristic approach is employed for finding the most optimal filter to be used. See, discussion in the PNG Specification "Filter selection":

`http://www.w3.org/TR/2003/REC-PNG-20031110/#12Filter-selection`

As built-in support for the sRGB color space is absent in PDF, the sRGB color can only be represented precisely by means of sRGB ICC Profile. However, for sRGB color PNG images, `dvipdfmx` uses an approximate calibrated RGB color space instead. For approximating the sRGB color, the gamma and CIE 1931 chromaticity values mentioned in the PNG Specification is used. See, the following page for more information:

`http://www.w3.org/TR/2003/REC-PNG-20031110/#11sRGB`

`dvipdfmx` also supports calibrated color with PNG `gAMA` and `cHRM` chunk. Those PNG chunks carry information on more accurate color representation. Some software programs, however, write only `cHRM` but do not record the gamma value although the PNG specification recommends to do so. Gamma value 2.2 is assumed if only `cHRM` is present but `gAMA` is not.

Some PNG features are unavailable depending on output PDF version setting. Please refer Table 2.1 for more details.

JPEG and JPEG2000

JPEG is also supported as in `dvipdfm`. In addition, JPEG2000 is also supported.

JPEG and JPEG2000 image inclusion is basically "pass-through", that is, image data is not decompressed. So, although these formats are lossy, there should be no quality loss when including images.

JPEG is relatively well supported. `dvipdfmx` supports embedded ICC Profiles and CMYK color. Embedded XMP metadata is also preserved. JFIF or Exif data is used to determin image's physical size.

As PDF specification does not require information irrelevant to displaying images to be embedded, `dvipdfmx` does not embedd whole data. Especially, not all application specific data is retained. Application specific data such as JFIF, Exif, and `APP14` Adobe markers are preserved. Please note that XMP and Exif data which may contain location information where the photo was taken is always preserved in the output PDF file.

JPEG2000 is also supported. It is restricted to JP2 and JPX baseline subset as required by the PDF specification. It is not well supported and still in an experimental stage. J2C format and transparency are not supported.

PDF Support

PDF inclusion is supported as in `dvipdfm`, with various important enhancement over `dvipdfm` for more robust inclusion. `dvipdfmx` can handle cross-reference streams and object streams introduced in PDF-1.5. `dvipdfmx` also supports inclusion of PDF pages other than the first page. However, tagged PDF may cause problems and annotations are not kept.

As there are no clear way to determin the natural extent of the graphics content to be clipped, `dvipdfmx` preferably try to find the *crop box* to decide image size. If there are no crop box *explicitly* specified,[1] then it tries to refer other boundary boxes such as the *art box*. If there are no possible boundaries of the graphics content explicitly specified, the *media box*, which is the boundaries of the physical medium on which the page is to be printed, is used as the last resort.

The `pdf:image` special supports additional keys, "`page`" and "`pagebox`". The `page` key takes an integer value representing the page number of PDF page to be included, and the `pagebox` takes one of the keywords `mediabox`, `cropbox`, `artbox`, `bleedbox`, or `trimbox` for selecting page's boundary box to be used. For examples,

```
\special{pdf:image pagebox artbox page 3 (foo.pdf)}
```

includes 3rd page of 'foo.pdf' with the boundary box set to art box.

Additional Supported Formats

For METAPOST generated PostScript files, multi-byte encoding support is added. `dvipdfmx` also supports "METAPOST mode". When `dvipdfmx` is invoked with '`-M`' option, it enters in METAPOST mode and processes a METAPOST generated EPS file as input.[2]

BMP support is added. It is limited to uncompressed or RLE-compressed raster images. Extensions are not (won't be) supported.

Inclusion of Other Format Images

As in `dvipdfm`, the `-D` option can be used to convert images to PDF format before inclusion. In `dvipdfmx`, the letter `v` in the `-D` option argument is expanded to the output PDF version.

2.1.2 Image Cache

Caching of images generated via filtering command specified with '`-D`' option is supported. It solves the problems that image inclusion becomes very slow when

[1] There are some accusations by Japanese TEX users that "`dvipdfmx` violates the PDF spec." regarding this point. However, what we are talking here is about how to guess the *natural* or *intended* size of images but not about the default value of the PDF `CropBox` itself.

[2] `prologue` should be set to 2.

external filtering program such as GhostScript is invoked each time images are included.

Use '`-I`' option to enable this feature:

```
-I 24
```

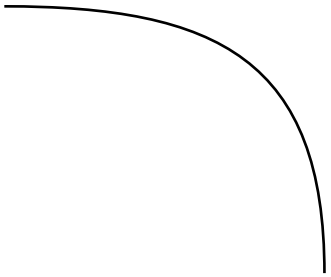
where the integer represents the life of cache files, 24 hours in this example. Zero and negative values have a special meaning. Value 0 for "erase old cached images while leaving newly created one", -1 for "erase all cached images", and -2 for "ignore image cache". Default value is set to -2.

2.2 Graphics Drawing

`dvipdfmx` does not offer a high level interface to draw graphics objects. A possible way to draw grahpics is to write raw PDF graphics drawing codes and then to insert them into the output via the `special` command. To show an example, the following code:

```
\special{pdf:content
  1 0 0 1 0 0 cm
  0 100 m
  80 100 120 80 120 0 c
  S
}
```

draws a Bézier curve,



The `pdf:content` special is used here which is useful for inserting an isolated graphics object.

This illustrates a typical example of PDF graphics drawing. It consists of three parts; setting graphics state, constructing a path, and painting a path. A graphics object is specified as a sequence of operators and thier operands using *postfix notation*. *Graphics state operators* comes first, `cm` in this example sets the current transformation matrix (CTM). Then, *path construction* operators follow; move to position $(0, 100)$, append a Bézier curve from current point to $(120, 0)$ with control points $(80, 100)$ and $(120, 80)$. Finally, a *path painting* operator comes to draw the constructed path. In this example the stroking operator `S` is used.

2.2.1 The `pdf:content` Special

The `pdf:content` special can be used for drawing an *isolated* graphics object. It sets the origin of graphics drawing operators supplied to this command to the position where it is inserted. The whole content is enclosed by a pair of graphics state save-restore operators. So for examples, a color change made within a `pdf:content` command takes an effect only within the content of this special.

2.2.2 Guide to PDF Graphics Operators

PDF employs essentially the same imaging model as PostScript. So, it is easy to learn about PDF graphics drawing for people who are well accustomed to PostScript. This section is intended to be a short guide for PDF graphics operators.

Graphics State Operators

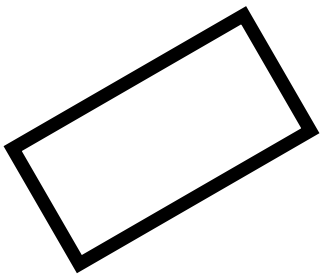
The `cm` operator modifies CTM by concatinating the specified matrix. Operands given to this operators are six numbers each representing transformation matrix elements: translation represented as $[1, 0, 0, 1, t_x, t_y]$, scaling $[s_x, 0, 0, s_y, 0, 0]$, rotation $[\cos\theta, \sin\theta, -\sin\theta, \cos\theta, 0, 0]$.

To uniformly scale the object, just use

```
2.0 0 0 2.0 0 0 cm
```

The `w` operator sets the line width, e.g., '`2 w`' sets the line width to 2. Here is an example of drawing a rotated rectangle:

```
0.866 0.5 -0.5 0.866 30 2 cm 5 w 0 0 100 50 re S
```



Transformations can be sequentially applied; for the above example,

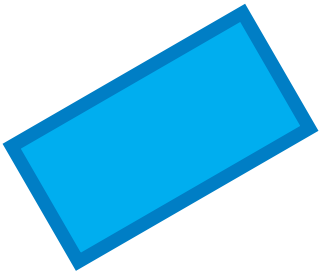
```
1 0 0 1 30 2 cm 0.866 0.5 -0.5 0.866 0 0 cm
5 w 0 0 100 50 re S
```

gives the same result.

To specify colors, use `RG`, `rg`, `K`, and `k` operators, for RGB or CMYK color for stroking (upper-case) and nonstroking (lower-case).

```
0.866 0.5 -0.5 0.866 30 2 cm 5 w
1 0.4 0 0 K 1 0 0 0 k
0 0 100 50 re B
```

where the `B` operator fill and then stroke the path.



A dash pattern can be specified with `d` operator. Operands for this operator are the *dash array* and the *dash phase*. For examples, to specify a dash pattern with 6-on 4-off starting with phase 0:

```
[6 4] 0 d 2 w 0 0 m 320 0 l S
```

draws the following dashed line:

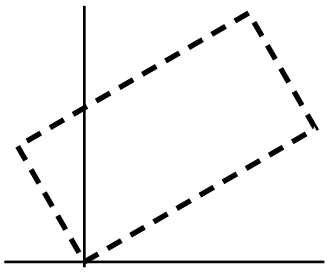
Other important operators are `q` and `Q`, which saves and restores the graphics state.

```
1 0 0 1 30 2 cm
q
0.866 0.5 -0.5 0.866 0 0 cm
[6 4] 0 d 2 w 0 0 100 50 re S
Q
-30 0 m 90 0 l S
0 -2 m 0 96 l S
```

In the above example, `d`, `w`, and rotation only take effect within the `q`-`Q` block. The portion drawing two straight lines is unaffected.

Operands	Operator	Description
—	`q`	Save the current graphics state.
—	`Q`	Restore the previously saved graphics state.
a b c d e f	`cm`	Modify the current transformation matrix by concatinating the specified matrix.
width	`w`	Set the line width.
array phase	`d`	Set the line dash pattern.
r g b	`RG`	Set the stroking color space to RGB and set the stroking color as specified.
r g b	`rg`	Set the nonstroking color space to RGB and set the nonstroking color as specified.
c m y k	`K`	Set the stroking color space to CMYK and set the stroking color as specified.
c m y k	`k`	Set the nonstroking color space to CMYK and set the nonstroking color as specified.

Table 2.2: A few examples of graphics state operators and color operators.



For a (incomplete) list of graphics state operators, see Talbe 2.2.

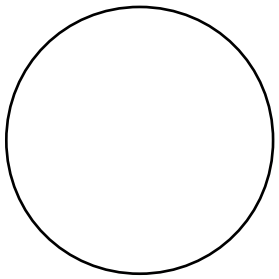
Path Construction Operators

Path construction normally start with a `m` operator which moves the current point to a specified position and then a sequence of other path construction operators follows. The path currently under construction is called the *current path*. A sequence of path construction operators append segments of path to the current path and then move the *current point* to the end point of appended path. A typical sequence of path construction looks like,

```
100 50 m
100 78 78 100 50 100 c
22 100 0 78 0 50 c
0 22 22 0 50 0 c
78 0 100 22 100 50 c
S
```

Operands	Operator	Description
$x\ y$	`m`	Begin a new path by moving the current point specified by given operands.
$x\ y$	`l`	Append a line segment from the current point to the point specified.
$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$	`c`	Append a Bézier curve to the current path. Two Control points and the end point given as operands.
$x_2\ y_2\ x_3\ y_3$	`v`	Append a Bézier curve to the current path. Using the current point and first two operand as the Bézier control points.
$x_1\ y_1\ x_3\ y_3$	`y`	Append a Bézier curve to the current path. The second control point coincides with the end point.
—	`h`	Close the current path by appending a straight line segment from the current point to the starting point of the path.
x y width height	`re`	Append a rectangle. First two operands for the position of lower-left corner, third and forth operand representing width and height.

Table 2.3: List of path construction operators. All operators move the current point to the end point of appended path.



This example is an approximated circle drawn by four Bézier curves.

Table 2.3 shows a list of path construction operators. Those who are accustomed to the PostScript language should note that in PDF the current path is not part of the graphics state, and hece is *not* saved and restored along with the other graphics state parameters.

Path Painting Operators

There are basically four kind of path painting operators: `S`, `f`, `B`, and `n`. The first three for "stroke", "fill", and "fill then stroke" operators respectively, and the last one `n` paints nothing but end the path object. For filling operators, there are two kind of operators depending on how "insideness" of points are determined: the *non-zero winding number rule* and the *even-odd rule*. The even-odd rule operators are `f*` and `B*`.

The following example illustrates the difference:

```
0 0 100 100 re 20 20 60 60 re f
1 0 0 1 120 0 cm
0 0 100 100 re 20 20 60 60 re f*
```

The “interior” of the “inner” square has a non-zero even winding number. (In this example, counter-clockwise direction is assumed for both of two `re` operators.)

Chapter 3

Specials

3.1 PDF Specials

Several `special` commnads are added for more flexible PDF generation: creation of arbitrary stream object, controlling `dvipdfmx` behavior, and some specials which might be useful for graphics drawing.

3.1.1 Additions to PDF Specials

PDF object manipulation is a key feature of PDF specials. The `pdf:fstream` special is added in `dvipdfmx` which enables creation of PDF stream object from an existing file. The syntax of this special is,

```
pdf:fstream @identifier (filename) <<dictionary>>
```

where identifier and filename (specified as a PDF string object) are mandatory and a dictionary object, following the filename, which is to be added to the stream dictionary is optional.

For examples, to incorporate a XMP Metadata,

```
\special{pdf:fstream @xmp (test.xmp) <<
    /Type    /Metadata
    /Subtype /XML
>>}
\special{pdf:put @catalog << /Metadata @xmp >>}
```

Similary, `pdf:stream` special can be used to create a PDF stream object from a PDF string instead of a file.

```
pdf:stream @identifier (stream contents) <<dictionary>>
```

`pdf:mapline` and `pdf:mapfile` specials can be used to append a fontmap entry or to load a fontmap file:

```
pdf:mapline foo unicode bar
pdf:mapfile foo.map
```

`pdf:majorversion` and `pdf:minorversion` specials can be used to specify major and minor version of output PDF.

```
pdf:minorversion 3
```

To protect output PDF with encryption, use `pdf:encrypt` special

```
pdf:encrypt userpw (foo) ownerpw (bar) length 128 perm 20
```

where user-password (`userpw`) and owner-password (`ownerpw`) must be specified as PDF string object. (which can be empty) Numbers specifying key-length and permission flags here are decimal numbers. See, section "Encryption Support" for a brief description of permission flags.

Other notable extensions are `code`, `bcontent`, and `econtent`. The `code` special can be used to insert raw PDF graphics instructions into the output. It is different from `dvipdfm`'s `content` special in that it does not enclose contents with a `q` and `Q` (save-restore of graphics state) pair. A typical usage of this special is:

```
\special{pdf:code q 1 Tr}
...some text goes here...
\special{pdf:code Q}
```

which changes text rendering mode to 1, as shown in Figure 3.1.

Be careful on using this special as it is very easy to generate broken PDF files. The `bcontent` and `econtent` pair is somewhat fragile and might be incompatible to other groups of special commands. It may not always be guaranteed to work as 'expected'.

3.1.2 Summary of PDF Specials

Please refer to the "Dvipdfm User's Manual".[1]

3.1.3 PDF Special Examples

This section is intended to be a hint for advanced users. Uninterested users can safely skip this section.

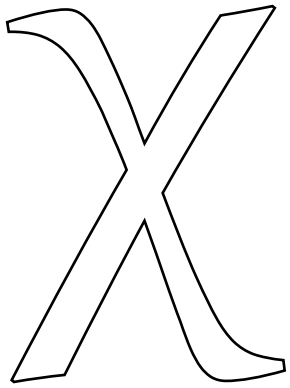


Figure 3.1: A character drawn in the PDF text rendering mode 1.

Annotations

In this section, some useful special commands for creating *annotations* are explained. Note that viewer support is required for annotations to be shown correctly.

First start with a very simple *Text Annotation* for attaching a comment. This feature is supported by many PDF viewer applications.

```
\special{pdf:ann width 20bp height 20bp
  <<
     /Type     /Annot
     /Subtype  /Text
     /Name     /Comment
     /T        (Author of This Document)
     /Subj     (An Example of Text Annotations)
     /Contents (A Quick Brown Fox Jumped Over The Lazy Dog.)
  >>
}
```

`pdf:ann` special is used to create an annotation.

Likewise, *Rubber Stamp Annotation*,

```
\special{pdf:ann width 150bp height 50bp
  <<
     /Type     /Annot
     /Subtype  /Stamp
     /Name     /Approved
```

```
  >>
}
```

Other keywords such as `Expired`, `Final`, `Draft`, and so on, can be used in place of `Approved`.

One can create thier own stamps. For this purpose, specials `pdf:bxobj` and `pdf:exobj` can be used for designing stamps. Those specials "capture" all typset material enclosed by them into a *Form XObject*, which is a reusable graphics object like included images.

For a simple example,

```
\special{pdf:bxobj @MyStamp
            width 280pt height 0pt depth 40pt}
\addfontfeature{Scale=4,Color=FF9933}My Own Stamp
\special{pdf:exobj}
```

It captures typeset material "My Own Stamp" (this example uses `fontspec` package's command for changing font size and text color) into an object `MyStamp` for later reuse. Then, `AP` (*appearance dicrionary*) entry for controling the appearance of annotations is used in the annotation dictionary:

```
\special{pdf:ann width 280pt height 40pt
  <<
    /Type    /Annot
    /Subtype /Stamp
    /AP      << /N @MyStamp >>
  >>
}
```

Text stored in `@MyStamp` is used as "Normal" (`AP` entry `N`) appearance. (`R` for "Rollover" and `D` for 'Down" can be used.)

The result is:

My Own Stamp

With the following code, `dvipdfmx` reads source file and creates a stream object named `SourceFile`, and then creates file attachment annotation.

```
\special{pdf:fstream @SourceFile (\jobname.tex)}%
\special{pdf:ann width 10bp height 20bp
   <<
      /Type /Annot
```

```
        /Subtype /FileAttachment
        /FS <<
              /Type /Filespec
              /F    (\jobname.tex)
              /EF   << /F @SourceFile >>
            >>
        /Name     /PushPin
        /C        [0.8 0.2 0.2]
        /T        (The dvipdfmx project team)
        /Subj     (The Dvipdfmx User's Manual)
        /Contents (XeLaTeX source file of the manual.)
    >>
}
```

A push-pin image must be shown in the margin if viewer supports this kind of annotation. PDF viewer applications are required to provide predefined icon appearances at least for the following standard icons: `Graph`, `PushPin`, `PaperClip`, and `Tag`.

Special Color Space

This section gives various examples of using *Special color spaces*. Examples in this section have a common structure. They consist of essentially three parts. The first part is for defininining color space itself. PDF object creation commands like `pdf:obj` and `pdf:stream` are used for this purpose. Next is for registering color space resources in the page's *Resource Dicrionary*. It can be done via `pdf:put` command as,

```
\special{pdf:put @resource <<
  /Category << ...key-value pairs... >>
>>}
```

where `@resource` is a special keyword representing current page's Resource Dictionary and `Category` (to be replaced by actual category name) is category name such as `ColorSpace`. Finally, graphics objects are placed, with or with a combination of text and, PDF drawing operators inserted by `pdf:code` or `pdf:contents` specials.

The first example is *Separation* color space:

Orange and Green

```
\special{pdf:stream @TintTransform1
  ({0 exch dup 0.62 mul exch 0})
  <<  /FunctionType 4
      /Domain [ 0.0 1.0 ]
```

```
      /Range  [ 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 ]
  >>
}
\special{pdf:stream @TintTransform2
  ({dup 0.78 mul exch dup 0.05 mul exch 0.71 mul 0})
  <<  /FunctionType 4
      /Domain [ 0.0 1.0 ]
      /Range  [ 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 ]
  >>
}
\special{pdf:obj @Orange [
    /Separation /Orange /DeviceCMYK @TintTransform1
  ]
}
\special{pdf:obj @Green [
    /Separation /Green /DeviceCMYK @TintTransform2
  ]
}
\mbox{%
  \special{pdf:put @resources <<
      /ColorSpace << /CS01 @Orange /CS02 @Green >>
    >>
  }%
  \special{pdf:code q /CS01 cs 1.0 scn}
  Orange
  \special{pdf:code Q}
  and
  \special{pdf:code q /CS02 cs 1.0 scn}
  Green
  \special{pdf:code Q}
}
```

`TintTransform`'s defined here are functions for transforming *tint* values into approximate colors in the *alternate color space* (`DeviceCMYK` in this example). PostScript calculator functions are used for converting a single component value representing "Orange" or "Green" into four component CMYK values to approximate those colors. The "Orange" color v is approximated as $(0, 0.62v, v, 0)$ in CMYK color space for alternate display here.

The `cs` operator for selecting color space and the `scn` operator for color values are used in `pdf:code` special. Be sure that `pdf:put` command, which puts color space resources into the current page's Resource Dictionary, goes into the same page as subsequent drawing commands.

`dvipdfmx` currently does not have an easy interface for using various color space families such as CIE-Based color spaces (e.g., calibrated colors and color space with an ICC profile) and Special color spaces (e.g., indexed, separation, and shading and patterns).

Another example is *shading pattern*:

```
\special{pdf:put @resources <<
  /Shading <<
    /SH01 <<
      /ShadingType 2
      /ColorSpace @Orange
      /Coords     [0 0 320 20]
      /Extend     [true true]
      /Function   << /FunctionType 2 /Domain [0 1] /N 1.0 >>
    >>
  >>
>>}
\special{pdf:content 0 0 320 20 re W n /SH01 sh}
```

where the "Orange" separation color space defined before is used again. This example shows an axial shading (`ShadingType` 2) pattern.

Type 2 (Exponential Interpolation) *Function* is used for mapping coordinate values into color values. The above example, with the exponent $N = 1$, is just a simple linear-gradient.

The final examples is a *tiling pattern*.

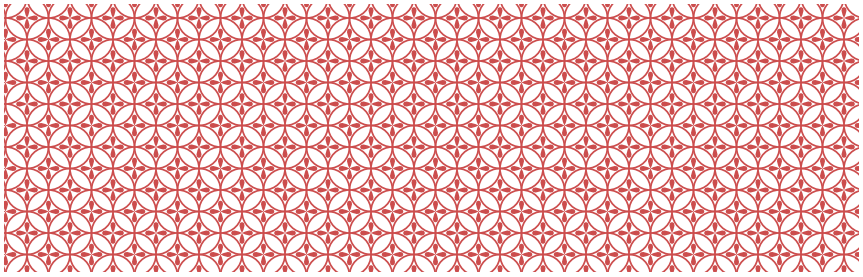
```
\special{pdf:stream @MyPattern
    (0.16 0 0 0.16 0 0 cm 4 w
     50 0 m 50 28 28 50 0 50 c S 100 50
     m 72 50 50 28 50 0 c S
     50 100 m 50 72 72 50 100 50 c S
     0 50 m 28 50 50 72 50 100 c S
     100 50 m 100 78 78 100 50 100 c 22 100 0 78 0 50 c
     0 22 22 0 50 0 c 78 0 100 22 100 50 c S
     0 0 m 20 10 25 5 25 0 c f 0 0 m 10 20 5 25 0 25 c f
     100 0 m 80 10 75 5 75 0 c f
     100 0 m 90 20 95 25 100 25 c f
     100 100 m 80 90 75 95 75 100 c f
     100 100 m 90 80 95 75 100 75 c f
     0 100 m 20 90 25 95 25 100 c f
     0 100 m 10 80 5 75 0 75 c f
     50 50 m 70 60 75 55 75 50 c 75 45 70 40 50 50 c f
     50 50 m 60 70 55 75 50 75 c 45 75 40 70 50 50 c f
     50 50 m 30 60 25 55 25 50 c
     25 45 30 40 50 50 c f
     50 50 m 60 30 55 25 50 25 c 45 25 40 30 50 50 c f)
    <<
        /BBox [0 0 16 16]
        /PaintType 2
        /PatternType 1
        /Resources <<
```

```
            /ProcSet [/PDF]
        >>
        /TilingType 3
        /Type /Pattern
        /XStep 16
        /YStep 16
    >>
}
```

The above example defines a tiling pattern. The content stream containing painting operators, `m` for "move-to", `c` for "curve-to", `f` for "fill", and `S` for "stroke", defines the appearance of the *pattern cell* for this tiling pattern. With the following code,

```
\special{pdf:put @resources
    <<
        /ColorSpace << /CS01 [/Pattern /DeviceRGB] >>
        /Pattern << /PT01 @MyPattern >>
    >>
}
\special{pdf:content
    q 0.8 0.3 0.3 RG /CS01 cs 0.8 0.3 0.3 /PT01 scn
    0 0 320 100 re f
}
```

A box filled with the tiling pattern defined above is drawn.



Transparency

XeTeX's transparency feature is currently lost in `xdvipdfmx`, but the same effect can be achieved by setting graphics state parameters with `ExtGState` resources and `gs` operator. Here is a simple transparency example:

```
\special{pdf:obj @GS01 <<
  /Type /ExtGState /CA 0.5 /ca 0.5 /AIS false
>>}%
\mbox{%
```

```
\special{pdf:put @resources <<
  /ExtGState << /GS01 @GS01 >>
>>}%
\special{pdf:code q /GS01 gs 1.0 0.8 0.2 rg}%
α%
\special{pdf:code 0.4 0.8 0.4 rg}%
\hspace{-0.3em}%
β%
\hspace{-0.3em}\raisebox{0.5ex}{%
  \special{pdf:code 0.4 0.4 0.8 rg}%
  π%
}%
\special{pdf:code 1.0 0.2 0.4 rg}%
\hspace{-0.2em}%
γ%
\special{pdf:code Q}%
}
```

where values for `CA` and `ca` represent opacity of stroke and fill color respectively. Again, `pdf:put` command must go into the same page as subsequent grahics and text drawing operators.

3.2 Dvipdfmx Extensions

A new special `dvipdfmx:config` was introduced in TEXLive 2016 which makes it possible to invoke a command line option. Several single letter command line options except 'D' are supported. For examples,

Addition in TEX Live 2016

Classification	Operators
Arithmetic & Math	`add sub mul div neg truncate`
Stack Operation	`clear pop exch`
Graphis State	`gsave grestore setlinewidth setdash setlinecap setlinejoin setmiterlimit setgray setrgbcolor setcmykcolor`
Coordinate System	`concat scale translate rotate idtransform dtransform`
Path Construction	`currentpoint newpath closepath moveto rmoveto lineto rlineto curveto rcurveto arc arcn clip eoclip`
Painting	`stroke fill`
Glyph & Font	`show findfont scalefont setfont currentfont stringwidth`

Table 3.1: List of PostScript operators recognized by `dvipdfmx`.

```
dvipdfmx:config C 0x10
```

sets the `dvipdfmx`'s compatibility flags. See, the section "Incompatible Changes" for an explanation of compatibility flags.

3.3 PS Specials

PS (PostScript) specials can be used to insert a raw PostScript code for drawing graphics objects and transforming subsequent text and graphics. Please note that support for PostScript operators in `dvipdfmx` is very limited. It is just enough for supporting METAPOST output. Only basic set of operators for arithmetic and math, stack operation and manupilation, graphics state, path construction and painting, glyph and font, are supported. See, Table 3.1 for the list of recognized PostScript operators.

It might be enough for the purpose of basic graphics drawings but as there are no support for conditionals and controls it is not enough for complicated tasks, espacially, the PSTricks package is not supported.

In `dvipdfmx`, text handling is extended to support CJK text. The following code draws Japanese text like shown in Figure 1.1:

```
\special{pdf:mapline uprml UniJIS-UTF8-H yumindb.ttf}
\special{ps: uprml findfont 16 scalefont setfont
  currentpoint moveto
  (...some Japanese text goes here...) show
}
```

Chapter 4

Fonts

4.1 Font Mapping

Syntax of fontmap file is basically the same as `dvipdfm`. There are few extensions in `dvipdfmx`. In addition to 8-bit `enc` files and keyword `builtin` and `none`, `dvipdfmx` accepts CMap name and the keyword `unicode` in the encoding field.

This section is completely irrevant to XeTeX users.

4.1.1 Extended Syntax and Options

Few options are available in `dvipdfmx` in addition to the original dvipdfm's one. Please note that all features which makes `dvipdfmx` to use non-embedded fonts are deprecated, as by doing so it makes `dvipdfmx` to create PDF files which can be non-compliant to the ISO standards.

SFD Specification

For bundling up a font split into multiple subfonts via SFD back into a single font, dvipdfmx supports extended sytax of the form

```
tfm_name@SFD@  encoding  filename  options
```

A typical example looks like:

```
gbsn@EUC@  GB-EUC-H  gbsn00lp
```

where TFMs `gbsn00`, `gbsn01`, `gbsn02`... are mapped into a single font named `gbsn00lp` via the rule described in SFD file `EUC`.

TrueType Collection Index

TrueType Collection index number can be specified with `:n:` in front of the TrueType font name:

```
min10  H  :1:mincho
```

In this example, the option `:1:` tells `dvipdfmx` to select first TrueType font from the TTC font `mincho.ttc`. Alternatively, the '`-i`' option can be used in the option field to specify TTC index:

```
min10  H  mincho  -i 1
```

Non-embedding Switch

Use of this option is deprecated.

The character '`!`' in front of the font name can be used to indicate that the font shall not be embedded. This feature greatly reduces the size of the final PDF output, but the PDF file may not be viewed exactly the same in other systems on which appropriate fonts are not installed.

NOTE: `dvipdfmx` always converts input encodings to CIDs and then uses Identity CMaps[1] in the output PDF. However, ISO 32000-1:2008 describes as

> *The Identity-H and Identity-V CMaps shall not be used with a non-embedded font. Only standardized character sets may be used.*

which had never appeared in Adobe's PDF References. This makes all PDF files generated by `dvipdfmx` with non-embedded CID-keyed fonts non-compliant to the ISO standards.

'Standard' CJK Fonts

This feature is deprecated.

Use of this feature shall be avoided for new documents. It is described here since it might still be useful for some situations.

`dvipdfmx` recognizes several 'Standard' CJK fonts although there are no such notion in PDF. In older days where there were not so many freely available CJK fonts, it was sometimes useful to create PDF files without embedded fonts and let PDF viewers or printers to use substitute fonts (tend to be higher quality) installed in their systems. `dvipdfmx` 'knows' several fonts which might be available in PostScript printers and PDF applications such as Acrobat Reader, and uses them without actually having it. See, Table 4.1, for the list of available 'Standard' CJK fonts.

Only fixed-pitch glyphs (i.e., quarter, third, half, and full widths) are supported for those fonts.

Stylistic Variants

Use of this option is deprecated.

Keywords `,Bold`, `,Italic`, and `,BoldItalic` can be used to create synthetic bold, italic, and bolditalic style variants from other font using PDF viewer's (or OS's) function.

[1] Predefined CMaps `Identity-H` and `Identity-V` for the identity mapping.

Character Collection	Font Family	Description
Adobe-Japan1	Ryumin-Light	PS printers
	GothicBBB-Medium	
Adobe-CNS1	MHei-Medium-Acro	Acrobat Reader 4
	MSung-Light-Acro	
Adobe-GB1	STSong-Light-Acro	
	STHeiti-Regular-Acro	
Adobe-Japan1	HeiseiMin-W3-Acro	
	HeiseiKakuGO-W5-Acro	
Adobe-Korea1	HYGoThic-Medium-Acro	
	HYSMyeongJo-Medium-Acro	
Adobe-CNS1	MSungStd-Light-Acro	Acrobat Reader 5
Adobe-GB1	STSongStd-Light-Acro	
Adobe-Korea1	HYSMyeongJoStd-Medium-Acro	
Adobe-CNS1	AdobeMingStd-Light-Acro	Adobe Reader 6
Adobe-GB1	AdobeSongStd-Light-Acro	
Adobe-Japan1	KozMinPro-Regular-Acro	
	KozGoPro-Medium-Acro	
Adobe-Korea1	AdobeMyungjoStd-Medium-Acro	
Adobe-CNS1	AdobeHeitiStd-Regular	Adobe Reader 7
Adobe-Japan1	KozMinProVI-Regular	Adobe Reader 8

Table 4.1: List of available 'Standard' CJK font. Most of them are available as a part of Adobe Asian Font Packs for each versions of Adobe or Acrobat Reader.

```
jbtmo@UKS@  UniKSCms-UCS2-H  :0:!batang,Italic
```

Availability of this feature highly depends on the implementation of PDF viewers. This feature is usually not supported for embedded fonts. Notice that this option automatically disables font embedding thus use of it is deprecated.

4.1.2 Specifying Unicode Plane

As there are no existing 3-bytes or 4-bytes TFM formats, the only way to use Unicode characters other than the BMP is to map code range 0-65535 to different planes via (e.g., to plane 1) '`-p 1`' fontmap option. This option is available only when `unicode` is specified in the encoding field.

4.1.3 OpentType Layout Feature

OpenType Layout Feature fontmap options mentioned below are only meaningful when `unicode` is specified in the encoding field.

With the '`-w`' option, writing mode can be specified. '`-w 1`' denotes the font is for vertical writing. It automatically enables an OpentType Layout Feature related to vertical writing, namely, `vert` or `vrt2`, to choose proper glyphs for vertical text.

The '`-l`' (lower case el) option can be used to enable various OpenType Layout GSUB Features. For examples, '`-l jp04`' enables `jp04` feature to select JIS2004

Addition in TeX Live 2017.

Figure 4.1: JIS2004 vs. JIS1990 form.

forms for Kanjis. Features can be specified as a ":" separated list of OpenType Layout Feature tags like '`-l vkna:jp04`'. Script and language may be additionally specified as '`-l kana.JAN.ruby`'.

An example can be

```
uprml-v  unicode  SourceHanSerifJP-Light.otf  -w 1 -l jp90
```

which declares that font should be treated as for vertical writing and use JIS1990 form for Kanjis. (See, Figure 4.1 for an example)

This feature is limited to the single substitution, there are no way to select a glyph from multiple candidates, such as in `aalt`, and specifying general many-to-many glyph substitutions does not take effect.

4.2 Other Improvements

This section briefly describes other improvements made for `dvipdfmx`. There is an extension to glyph name handling in `enc` files for seamless support of both PostScript Type1 and TrueType fonts. PostScript Type1 font support is enhanced although this format might be considered obsolete.

4.2.1 Extended Glyph Name Syntax

`dvipdfmx` accepts the following syntax for glyph names in `enc` files: `uni0130`, `zero.onum` and `T_h.liga`. Each represents a glyph accessed with Unicode value `U+0130`, oldstyle number for zero and "Th" ligature accessed via OpenType Layout GSUB Feature `onum` and `liga`, respectively. Note that `dvipdfmx` does not understand glyph names which directly use a glyph index such as `index0102` or `gid2104`.

When `dvipdfmx` encounters a glyph name, e.g., `T_h.liga`, it first looks for OpenType `post` table if such glyph name exists; if it exists, then `dvipdfmx` simply uses `post` table and maps the glyph name to glyph index; if not, `dvipdfmx` tries to convert `T_h` to Unicode sequence (U+0054 U+0068 in this example) via the AGL mapping; then, OpenType `cmap` table is used to further convert resulting Unicode sequence to the sequence of glyph indices; finally, OpenType Layout Feature `liga` is applied to get the desired glyph.

A glyph name of the form `a.swsh2` can be specified to denote 2nd swash variant form of letter 'a'.

4.2.2 CFF Conversion

dvipdfmx supports on-the-fly PostScript Type1 to CFF (Type1C) conversion which greatly reduces size of the resulting PDF file when using Type1 fonts. Conversion is essentially 'lossless' and there should not be any quality loss. However, due to differences in the ability of rasterizers, there might be noticeable differences on the rendering result.

When (older) Type1 fonts are used, dvipdfmx may give the following warning message:

```
Obsolete four arguments of "endchar" will be used for Type1
"seac" operator.
```

This happens when there is an accented character made from composite glyphs. This warning message is given as conversion can't be done without the use of deprecated endchar operator for composite glyphs. However, as mentioned in "Appendix C Compatibility and Deprecated Operators" of Adobe Technical Note #5177, "Type 2 Charstring Format", PDF applications should support this operator and hence this warning message can be ignored.

Use of Type1 font should be avoided as much as possible. Please consider using OpenType version.

4.3 Font Licensing and Embedding

In OpenType font format, information regarding how a font should be treated when creating a document can be recorded.[2] dvipdfmx uses this information to decide whether font embedding is permitted.

This font embedding information is indicated by the flag called fsType recorded in OpenType font files; each bit representing different restrictions on font embedding. If multiple flag bits are set in fsType, the least restrictive license granted takes precedence in dvipdfmx. The fsType flag bit recognized by dvipdfmx is as follows:

- Installable embedding
- Editable embedding
- Embedding for Preview & Print only

dvipdfmx gives the following warning message for fonts with 'Preview & Print only' setting:

```
This document contains 'Preview & Print' only licensed font
```

For fonts with this type of licensing, font embedding is allowed solely for the purpose of (on-screen) viewing and/or printing; further editing of the document or extracting embedded font data for other purposes are not allowed. One way to ensure this condition is to protect your document with a non-empty password.

[2] See, " OpenType Sepcification: OS/2 – OS/2 and Windows Metrics Table".

All other flags are treated as more restrictive license than any of the above flags and treated as "No embedding allowed"; e.g., if both of the editable-embedding flag and unrecognized license flag is set, the font is treated as editable-embedding allowed, however, if only unrecognized flags are set, the font is not embedded.

Embedding flags are preserved in the embedded font if they are embedded as a TrueType font or a CIDFontType2 CID-keyed font. For all font embedded as a PostScript font (Type1C and CIDFontType0 CID-keyed font), they are not preserved. Only `Copyright` and `Notice` in the FontInfo dictionary are preserved in this case.

Some font vendors put different embedding restrictions for different condition; e.g., font embedding might not be permitted for commercial use unless you acquire "commercial license" separately. Please read EULA carefully before making decision on the font usage.

See, for examples, Adobe's site on font embedding permissions for fonts in the Adobe Type Library. Microsoft also has a FAQ page on Font Redistribution.

For Japanese font in general, embedding permission tend to be somewhat restrictive. Japanese users should read the statement regarding font embedding from Japan Typography Association (in Japanese):

`http://www.typography.or.jp/act/morals/moral4.html`

`dvipdfmx` does not support full embedding. Only subset embedding is supported.

Chapter 5

Encryption

5.1 Encryption Support

dvipdfmx offers basic PDF password security support including 256-bits AES encryption. Only "Standard" security handler is supported and Public-key security handler is not supported.

Encryption is enabled by '-S' command line option.

```
dvipdfmx -S -K 128 -P 0x14
```

where -K and -P options are used to specify encryption key length and permission flags respectively, and are briefly explained in Table 5.1.

When dvipdfmx is invoked with encryption via -S option, passwords will be asked. However, in some circumstances, it might be desirable to avoid being prompted for passwords. In that case, use the pdf:encrypt special to supply passwords in the TEX file, as,

```
\special{pdf:encrypt userpw (foo) ownerpw (bar) length 128
   perm 20}
```

Here, user and owner password are supplied as PDF string objects (foo and bar in the example above) which can be empty.

Option	Description
-S	Enable PDF encryption.
-K *number*	Set encryption key length. The default value is 40.
-P *number*	Set permission flags for PDF encryption. The *number* is a 32-bit unsigned integer representing permission flags. See, Table 5.2. The default value is 0x003C.

Table 5.1: Command line options for encryption.

Bit Position	Meaning
3	(Revision 2) Print the document. (Revision 3 or greater) Print the document. Print quality depending on bit 12.
4	Modify the contents of the documentby operations other than those controlled by bits 6, 9, and 11.
5	Copy or extract text and graphics from the document.
6	Add or modify text annotations, fill in interactive form fields. Creation and modification of interactive form field is also allowed if bit 4 is set.
9	(Revision 3 or greater) Fill in existing interactive form fields (including signature fields), even if bit 6 is clear.
10	*Deprecated in PDF 2.0* (Revision 3 or greater) Extract text and graphics (in support of accessibility to users with disabilities or for other purposes).
11	(Revision 3 or greater) Assemble the document (insert, rotate, or delete pages and create document outline items or thumbnail images), even if bit 4 is clear.
12	(Revision 3 or greater) High-quality printing. When this bit is clear (and bit 3 is set), printing shall be limited to a low-level, possibly of degraded quality.

Table 5.2: Flag bits and their short explaination. Revision 2 is used when encryption key length is 40 bits or when PDF output version is less than 1.5. Otherwise, Rivision 3 or greater is used.

Up to two passwords can be specified for a document – an owner password and a user password. If a user attempts to open an encrypted document with user password being set, PDF application should prompt for a password. Users are allowed to access the contents of the document only when either password is correctly supplied. Depending on which password (user or owner) was supplied, additional operations allowed for the opened document is determined; full access for users who opened with the correct owner password or additional operations controlled by permission flags for users who opened with the correct user password.

Although PDF specification allows various character encodings other than `US-ASCII` for entering password, `dvipdfmx` is unable to handle it properly. Thus it must be assumed that `US-ASCII` is used for password strings.

Access permision flags can be specified via '`-P`' command-line option. Each bits of (32-bit unsigned) integer number given to this option represents user access permissions; e.g., bit position 3 for allowing "print", 4 for "modify", 5 for "copy or extract", and so on. See, Table 5.2. For examples, `-P 0x34` allows printing, copying and extraction of text, and adding and modifying text annotation and filling in interactive form fields (but disallows modification of the contents of the document).

The '`-K`' option can be used to specify the encryption key length. The key length must be multiple of 8 in the range 40 to 128, or 256 (for PDF version 1.7 plus Adobe Extension or PDF version 2.0). Please note that when key length 256

is specified for PDF version 1.7 output, it requires Adobe's Extension to PDF-1.7 and hence PDF applications may not support it. PDF version 1.4 is required for key length more than 40 bits. AES encryption algorithm requires PDF version 1.6.

To show some examples:
128-bit AES encryption with print-only (high-quality) setting,

```
dvipdfmx -V 5 -S -K 128 -P 0x804 input.dvi
```

256-bit AES encryption with print (low-quality), adding and modifying text annotations allowed,

```
dvipdfmx -V 2.0 -S -K 256 -P 0x24 input.dvi
```

The default values for '`K`' is 40 and for '`-P`' is `0x003C0` (all bits from bit-position 3 to 6 set).

Chapter 6

Compatibility

6.1 Incompatible Changes

There are various minor incompatible changes to `dvipdfm`.

The '`-C`' command line option may be used for compatibility to `dvipdfm` or older versions of `dvipdfmx`. The '`-C`' option takes flags meaning

- bit position 2: Use semi-transparent filling for tpic shading command, instead of opaque gray color. (requires PDF 1.4)
- bit position 3: Treat all CID-keyed font as fixed-pitch font. This is only for compatibility.
- bit position 4: Do not replace duplicate fontmap entries. `dvipdfm` behavior.
- bit position 5: Do not optimize PDF destinations. Use this if you want to refer from other files to destinations in the current file.
- bit position 6: Do not use predictor filter for Flate compression.
- bit position 7: Do not use object stream.

The remap option '`-r`' in fontmaps is no longer supported and is silently ignored. The command line option '`-e`' to disable partial (subset) font embedding is not supported.

6.2 Important Changes

Here is a list of important changes since the TeX Live 2016 release:

- Changes to make PDF/A creation easier: Always write CIDSet and CharSet for embedded fonts. Do not compress XMP metadata.
- Merge from libdpx for pTeX-ng by Clerk Ma.
- Addition of `STHeiti-Regular-Acro` for CJK 'Standard' fonts.
- Command line option '`-p`' takes precedence over `papersize` and `pagesize` specials.

- Fixed serious bugs in supporting 'unicode' encoding: OpenType Layout Feature vert and vrt2 was not enabled. Support for format 2 CFF charsets was broken.
- Added simplified version of OpenType Layout support: The "-l" option in fontmaps.

The full ChangeLog entries can be viewed via the web interface of TEX Live SVN repository:

http://www.tug.org/svn/texlive/trunk/Build/source/texk/dvipdfm-x

There was an undocumented feature for supporting OpenType Layout but it was dropped. Simplified support for OpenType Layout was intorduced instead.

Further Reading

[1] ”Dvipdfm User’s Manual” written by Mark A. Wicks.

[2] Adobe’s PDF References and a free copy of ISO 32000-1:2008 standard are available from “PDF Technology Center” on Adobe Developer Connection.

[3] The OpenType Specification is available from Microsoft’s site: “OpenType Specification”.

[4] ”Portable Network Graphics (PNG) Specification (Second Edition)”.

[5] An article regarding DVI specials: Jin-Hwan Cho, “DVI specials for PDF generation”, TUGboat, 30(1):6-11, 2009.

Appendix A

GNU Free Documentation License

This document is distributed under the term of the GNU Free Documentation License. See, the attached file for copying conditions.

Or, in case that PDF viewers can not extract attached files, please visit the following site:

http://www.gnu.org/licenses/fdl.html