Software and Tools

The pdfTEX user manual

Han The Thanh and Sebastian Rahtz

1 Introduction

The pdfTEX project started in 1996 at the Faculty of Informatics, Masaryk University, Brno, Czech Republic. It forms a primary part of the MSc and PhD research of Han The Thanh, under the supervision of Jiří Zlatuška and Petr Sojka. The main purpose of the project was to create an extension of TEX that could create PDF directly from TEX and improve/enhance the result of TEX typesetting with the help of PDF. pdfTEX contains TEX as a subset. When PDF output is not set, it produces normal DVI output; when PDF output is selected, pdfTEX produces PDF output that looks identical to the DVI output. The next stage of the project, apart from fixing any errors in the program, is to investigate alternative justification algorithms, possibly making use of multiple master fonts.

pdfTEX is based on the original TEX sources and web2c, and has been successfully compiled on Unix, Amiga, Win32 and DOS systems. It is still under beta development and all features are liable to change.

This manual was compiled by Sebastian Rahtz from notes and examples by Han The Thanh. Many thanks are due to members of the pdfTEX mailing list (most notably Hans Hagen), whose questions and answers have contributed much to this manual.

2 Implementation details

The implementation of pdfTEX consists of two parts: the changes to TEX, and the addition of 'driver' features. The TEX-specific part is written as a change file to the original source of TEX and contains:

- a part that generates PDF code visually equivalent to DVI commands;
- virtual font processing;
- implementation of new primitives for PDF control.

The 'driver' part is written as several source files in C, and implement

- font mapping,
- font inclusion and partial downloading,
- font re-encoding,
- PNG picture insertion (using the public domain `libpng` code),
- text compression (using the public domain `zlib` code).

2.1 Compilation

pdfTEX is supplied as a set of additions to the standard Unix Web2c setup, and is a standard part of Web2c 7.2 (from January 1998); compilation should present no problems on most Unix systems. Porting it to other setups will require a little work, because of the requirement to combine the normal TEX Web output, and the parts written in C.

For Win32 systems (Windows 95, Windows NT) there are two packages that contain pdfTEX, both in `CTAN:systems/win32`. Web2c for Win32 is maintained by Fabrice Popineau (`mailto:popineau@ese-metz.fr`), and MikTeX by Christian Schenk (`mailto:cschenk@berlin.snafu.de`).

A binary version of pdfTEX for Amiga is made available (`CTAN:systems/\pdftex/bin/Amiga`) by Andreas Scherer (`mailto:Scherer@Physik.RWTH-Aachen.De`).

To make the situation more complicated, a Win32 binary version of pdfTEX compiled with Cygnus tools will also be made available in `CTAN:systems/pdftex/bin/Win32`. This version will be compiled in the same way as Unix systems.

2.2 Search paths

When runing pdfTEX, some extra search paths need to be set up beyond those normally requested by TEX itself; in web2c these are:

VFFONTS the path where pdfTEX looks for virtual fonts.

T1FONTS the path where pdfTEX looks for Type1 and TrueType fonts.

TEXPSHEADERS the path where pdfTEX looks for the font mapping file (`pdftex.map`), PNG pictures and encoding files (`*.enc`).

3 Fonts

pdfTEX can only work with Type 1 and TrueType fonts at present, and a source must be available for all fonts used in the document, except for the 14 base fonts supplied by Acrobat Reader (Times, Courier, and Symbol). It is *not* possible to use METAFONT-generated fonts in pdfTEX—apart from the technical reasons, the resulting Type 3 fonts render very poorly in Acrobat Reader. Given the free availability of Type 1 versions of all the Computer Modern fonts, and the ability to use standard PostScript fonts without further ado, most existing TEX users should be able to experiment with pdfTEX.

pdfTEX reads a map file called `pdftex.map`. Reencoding and partial downloading for each font is specified in this file. Every font must be listed in this file, each on a separate line. The syntax of each line is similiar to dvips' map files (but may be changed later). Each line contains the following fields (some of them are optional):

`texname basename fontflags fontfile encoding`

`texname`: the name of the TFM file

`basename`: the base font name (PostScript font name)

`fontflags`: an integer number specifying the most important characteristics of the font. This number is significant only in the case that the font file is *not* included, and Acrobat Reader is asked to simulate missing font using its multiple master defaults. It is a PDF font descriptor (see PDF manual, section 7.9.2) which is a 32-bit integer containing a collection of Boolean attributes, with bit 1 being the least significant. Attributes are true if the corresponding bit is set to 1 in the integer. The meanings of the bits is as follows:

1	Fixed-width font
2	Serif font
3	Symbolic font
4	Script font
5	(reserved)
6	Uses the Adobe Standard Roman Character Set
7	Italic
8–16	(reserved)
17	All-cap font
18	Small-cap font
19	Force bold at small text sizes
20–32	(reserved)

Bit 6 indicates that the font's character set is the Adobe Standard Roman Character Set, or a subset of that, and that it uses the standard names for those characters. About bit 19, the PDF specification says

> ... used to determine whether or not bold characters are drawn with extra pixels even at very small text sizes. Typically, when characters are drawn at small sizes on very low resolution devices such as display screens, features of bold characters may appear only one pixel wide. Because this is the minimum feature width on a pixel-based device, ordinary non-bold characters also appear with one-pixel wide features, and cannot be distinguished from bold characters. If bit 19 is set, features of bold characters may be thickened at small text sizes.

It should be stressed that the font flags provided for many fonts in the currently distributed `pdftex.map` are not correctly derived.

`fontfile`: the name of the font source file. This must be a Type 1 or TrueType font file. If it is preceded by a `<` then the font file will be partly downloaded; if it preceded by a double `<<` then the font file will

be included entirely. Otherwise the font is not included, and only the font parameters are extracted. Note that the font file *must* always be available at runtime, even if it not downloaded.

`encoding`: the encoding vector used for the font. It may be preceded by a `<`, but the effect is the same. The format is identical to that used by dvips.

Here are some sample lines from `pdftex.map`:

- include font entirely without reencoding

  `pplr8r Palatino-Roman 34 <<Palatino-Roman.pfb`

- include font partly without reencoding

  `pplr8r Palatino-Roman 34 <Palatino-Roman.pfb`

- do not include font, or reencode it, but extract parameters from font file

  `pplr8r Palatino-Roman 34 Palatino-Roman.pfb`

- include font entirely and reencode

  `pplr8r Palatino-Roman 34 <<Palatino-Roman.pfb 8r.enc`

- partially include font and reencode

  `pplr8r Palatino-Roman 34 <Palatino-Roman.pfb 8r.enc`

- do not include font but extract parameters from font file and reencode

  `pplr8r Palatino-Roman 34 Palatino-Roman.pfb 8r.enc`

- A TrueType font can be used in the same way as a Type 1 font

  `Cxr10 Cxr10 34 <Cxr10.ttf Cxtext.enc`

- base font can be also reencoded; the name of the font file is simply ignored

  `phvr8r Helvetica 32 <Helvetica.pfb 8r.enc`

4 New primitives

Here is a short description of new primitives added by pdfTEX:

`\pdfoutput`=*n*

Integer parameter specifying whether the output format should be DVI or PDF. Positive value means PDF output, otherwise DVI output. This parameter cannot be specified *after* shipping out the first page.

`\pdfcompresslevel`=*0*

Integer parameter specifying the level of text compression via `zlib`. Zero means no compression, 1 means fastest, 9 means best, 2..8 means something in between.

`\pdfpagewidth`=*dimen*

`\pdfpageheight`=*dimen*

Dimension parameters specifying the page width and page height of PDF output. If not specified then they are calculated as `\hsize` or `\vsize` + 2 × (1*in* + `\hoffset` or `\voffset`), when pdfTEX writes the first page.

`\pdfliteral`{*pdf text*}

Like `\special`, this can be used to insert raw PDF code into the output. This allows support of color, and text transformation.

`\pdfinfo` *`author`*{*author*} *`title`*{*title*} *`subject`*{*subject*} *`keywords`*{*keywords*}

Specify document information. All keys are optional; if this information is provided, it can be seen in Acrobat Reader with the menu option Document Information ⇝ General

`\pdfcatalog` *pagemode*`{`*pagemode*`}` *uri*`{`*uri*`}`

Document display information. Both keys are optional. *URI* provides the base URL of the document, and *pagemode* determines how Acrobat displays the document on startup. The possibilities are:

/UseNone Open document with neither outline nor thumbnails visible.

/UseOutlines Open document with outline visible.

/UseThumbs Open document with thumbnails visible.

/FullScreen Open document in full-screen mode. In full-screen mode, there is no menu bar, window controls, nor any other window present.

The default is **/UseNone**.

`\pdfimage` *height height width width filename*

Insert a bitmap image in PNG format, optionally changing width, height or both. Dimensions which are not specified will be treated as zero. If both height and width are zero then the box takes the natural size of the image. If one of them (width or height) is zero and the second is not, then the first one (the zero one) will be set to a value proportional to the second one so to make the box have the same proportion of width and height as the image natural size. If both width and height are positive then the image will be scaled to fit these dimensions.

`\pdfform` *num*

Write out the TEX box *num* as a Form object to PDF output.

`\pdflastform`

Returns the object number of the last Form written to the PDF file

`\pdfrefform \name`

Put in a reference to the PDF Form called `\name`.

These macros support a feature called "object reuse" in pdfTEX. The idea is to create a Form object in PDF. The content of the Form object corresponds to the content of a TEX box, which can contain pictures and references to other Form objects as well. After that the Form can be used by simply referring to its object number. This feature can be useful for large document with a lot of similiar elements, as it can reduce the duplication of identical objects.

`\pdfannottext` *width width height height depth depth attr*`{`*attr*`}` `{`*text*`}`

Attach a text annotation at the current point in the text. The attributes can be used to specify the default appearance of the annotation (e.g., `/Open true` or `/Open false`), as well as many other features (see Table 6.8 of the PDF manual).

`\pdfdest` *num num name*`{`*name*`}` *appearance*

Establish a destination for links and bookmark outlines; the link must be identified by either a number or a symbolic name, and the way Acrobat is to display the page must be specified; *appearance* must be one of

fit	fit whole page in window
fith	fit whole width of page
fitv	fit whole height of page
fitb	fit whole 'Bounding Box' page
fitbh	fit whole width of 'Bounding Box' of page
fitbv	fit whole height of 'Bounding Box; of page

`\pdfannotlink` *height height depth depth attr*`{`*attr*`}` *action*

Start a hypertext link; if the optional dimensions are not specified, they will be calculated from the box containing the link. The *attributes* (explained in great detail in section 6.6 of the PDF manual) determine the appearance of the link. Typically, this is used to specify the color and thickness of any border around the link. This `/C [0.9 0 0] /Border [0 0 2]` specifies a color (in RGB) of dark red, and a border thickness of 2 points.

The *action* can do many things; some possibilities are

goto num *n*	
goto name {*refname*}	Jump to a point established as *name*with `\pdfdest`
goto file {*filename*}	Open a local file; this can be used *with* a *name* or *num* specification, to point to a specific location on the file. Thus `goto file{foo.pdf} name{intro}`
thread num {*n*}	
thread name {*refname*}	Jump to thread identified by *n* or *refname*
user {*spec*}	Perform user-specified action. Section 6.9 of the PDF manual explains the possibilities. A typical use of this is to specify a URL, as `/S /URI /URI (http://www.tug.org/)`.

`\pdfendlink`

Ends link; all text between `\pdfannotlink` and `\pdfendlink` will be treated as part of this link. pdfTEX may break the result across lines (or pages), in which case it will make several links with the same content.

`\pdfoutline` *action* **`count`** *count* {*text*}

Create a outline (or bookmark) entry. The first parameter specificies the action to be taken, and is the same as that allowed for `\pdfannotlink` (see above, though note that the `Page` key does not work properly at present). The *count* specifies the number of direct subentries under this entry, 0 if this entry has no subentries (in this case it may be omitted). If the number is negative, then all subentries will be closed and the absolute value of this number specifies the number of subentries. The *text* is what will be shown in the outline window (note that this is limited to characters in the PDFEncoding vector).

`\pdfthread` ***num*** *num* ***name***{*name*}

Start an article thread; the corresponding `\pdfendthread` must be in the box in the same depth as the box containing `\pdfthread`. All boxes in this depth level will be treated as part of this thread. An identifier (*n* or *refname*) must be specified; threads with same identifier will be joined together.

`\pdfendthread`

Finish the current thread.

`\pdfthreadhoffset`=*dimen*

`\pdfthreadvoffset`=*dimen*

Specify thread margins.

One way to learn more about how to use these primitives is to have a look at the file `example.tex` in pdfTEX distribution.

5 Graphics and color

Probably the biggest single usage problem with pdfTEX at the present time is the inclusion of graphics. The program only directly supports graphic inclusion in one bitmap format, PNG (Portable Network Graphics).

Two commonly-used techniques are not available — the inclusion of Encapsulated PostScript figures, and the inclusion of raw PostScript commands (the techique utilized by the pstricks package). Although PDF is a direct descendant of PostScript, it lacks any programming language commands, and cannot deal with arbitrary PostScript. There are two ways to proceed with existing EPS files: firstly, convert them to

PNG (using programs like Image Magick, Image Alchemy, or Ghostscript); or secondly, try converting them to simple PDF. If the picture has no special fonts, the chances are quite good that Ghostscript's pdf writer will produce a file containing a single PDF object, which can be included using `\pdfliteral` commands (this is managed by the standard LaTeX graphics package).

Other alternatives for graphics in pdfTeX are:

LaTeX picture mode Since this is implemented simply in terms of font characters, it works in exactly the same way as usual;

Xy-pic If the PostScript backend is not requested, Xy-pic uses its own Type 1 fonts, and needs no special attention;

tpic The 'tpic' `\special` commands (used in some macro packages) can be redefined to produce literal PDF, using macros by Hans Hagen;

MetaPost Although the output of MetaPost is PostScript, it is in a highly simplified form, and a MetaPost to PDF conversion (written by Hans Hagen and Tanmoy Bhattacharya) is implemented as a set of macros which read MetaPost output and support all of its features;

It is possible to insert a "pure" PDF file (PDF that has only one page without fonts, bitmap and other resources) with a macro package that reads the external PDF file line by line.

The two latter macro packages are part of ConTeXt (`supp-pdf.tex` and `supp-mis.tex`), but also work with LaTeX and may be distributed separately.

For new work, the MetaPost route is highly recommended. For the future, Adobe has announced that they will define a specification for 'encapsulated PDF', and this should solve some of the present difficulties.

6 Macro packages supporting pdfTeX

- For LaTeX users, Sebastian Rahtz' hyperref package has substantial support for pdfTeX, and provides access to most of its features. In the simplest case, the user merely needs to load hyperref with a 'pdftex' option, and all cross-references will be converted to PDF hypertext links. PDF output is automatically selected, text compression turned on, and the page size is set up correctly. Bookmarks are created to match the table of contents.
- The standard LaTeX graphics and color packages have `pdftex` options, which allow use of normal color, text rotation, and graphics inclusion commands. Only PNG and MetaPost files can be included.
- The ConTeXt macro package by Hans Hagen (`mailto:pragma@pi.net`) has very full support for pdfTeX in its generalized hypertext features.
- Hypertexted PDF from `texinfo` documents can be created with `pdftexinfo.tex`, which is a slight modification of the standard `texinfo` macros. This is part of the pdfTeX distribution.
- A similiar modification of the `webmac`, called `pdfwebmac.tex`, allows production of hypertexted PDF versions of programs written in WEB. This is part of the pdfTeX distribution.

Some nice samples of pdfTeX output can be found on the TUG Web server, at `http://www.tug.org/applications/pdftex/`.

◇ Han The Thanh
Faculty of Informatics
Masaryk University
Brno, Czech Republic
`thanh@informatics.muni.cz`

◇ Sebastian Rahtz
7 Stratfield Road
Oxford OX2 7BG
United Kingdom
`s.rahtz@elsevier.co.uk`