The pdfTeX 0.12+ user manual

Sebastian Rahtz

Hàn Thế Thành
thanh@informatics.muni.cz
s.rahtz@elsevier.co.uk

April 1998

1 Introduction

The main purpose of the pdfTeX project was to create an extension of TeX that could create PDF directly from TeX source files and improve/enhance the result of TeX typesetting with the help of PDF. pdfTeX contains TeX as a subset. When PDF output is not selected, pdfTeX produces normal DVI output, otherwise it 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 Hàn Thế Thành. 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 Getting started

This section describes steps needed to get pdfTeX run on a system where pdfTeX is not installed. Some TeX packages have already contained pdfTeX as a part, such as MikTeX, Web2c for Win32 or teTeX, where we need not to bother with pdfTeX instalation. Note that installation descriptions in this manual are web2c-specific.

2.1 Getting sources and binaries

Latest sources of pdfTeX are distributed together with precompiled binaries of pdfTeX for some platforms, including Linux, SGI IRIX, Sun SPARC Solaris

and DOS (DJGPP). The primary location is ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdftex-testing/latest

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).

2.2 Compiling

If there is no precompiled binary of pdfTEX for our system, we need to build pdfTEX from sources. The compilation is expected to be easy on UNIX-like systems and can be described best by example. Assuming that all needed files are downloaded to $HOME/pdftex, the following steps are needed to compile pdfTEX (based on web2c-7.2) on a UNIX system :

```
cd $HOME/pdftex
gunzip < web-7.2.tar.gz | tar xvf -
gunzip < web2c-7.2.tar.gz | tar xvf -
gunzip < pdftex.tar.gz | tar xvf -
mv pdftexdir web2c-7.2/web2c
cd web2c-7.2
./configure
cd ./web2c
make pdftex
```

Apart from the binary of pdfTEX the compilation also produces several other files which are needed for running pdfTEX:

pdftex.pool – so-called pool file, needed for creating formats, located in web2c-7.2/web2c

texmf.cnf – web2c run-time configuration file, located in web2c-7.2/kpathsea

ttf2afm – an external program to generate AFM file from TrueType fonts, located in web2c-7.2/web2c/pdftexdir

For precompiled binaries these files are included in the zip archive containing the binary (pdftex.zip).

2.3 Getting pdfTEX-specific platform-indenpendent files

Apart from above-mentioned files, there is another zip archive (pdftexlib-0.12.zip) in pdfTEX distribution which contains platform-indenpendent files required for running pdftex: pdftex config file (pdftex.cfg), encoding vectors (*.enc), map files (*.map), macros (*.tex). Unpacking this archive (don't forget -d option when using pkunzip) will create a texmf tree containing pdfTEX-specific files.

2.4 Placing files

The next step is to place the binaries somewhere in PATH. We also need to make a copy (or symbolic link) of pdftex and name it as pdflatex if we want to use LaTeX. texmf.cnf, pdftex.pool and directory texmf/ (created by unpacking pdftexlib-0.12.zip) should be also moved to "appropriate" place (see below).

2.5 Setting search paths

web2c-based programs, including pdfTeX, use a web2c run-time configuration file called texmf.cnf. This file can be found via the user-set environment variable TEXMFCNF or via the compile-time default value if the former is not set. It is strongly recommended to use the first way. Then we need to edit texmf.cnf so pdfTeX can find all necessary files. Usually one has to edit TEXMFS and maybe some next. When runing pdfTeX, some extra search paths are used beyond those normally requested by TeX itself:

VFFONTS — the path where pdfTeX looks for virtual fonts.

T1FONTS — the path where pdfTeX looks for Type1 fonts.

TTFONTS — the path where pdfTeX looks for TrueType fonts.

PKFONTS — the path where pdfTeX looks for PK fonts.

TEXPSHEADERS — the path where pdfTeX looks for the configuration file pdftex.cfg, font mapping files (*.map), encoding files (*.enc), and PNG pictures.

2.6 The pdfTeX configuration file

When pdfTeX starts, apart from web2c configuration file it reads a *pdfTeX configuration file* called pdftex.cfg, searched for in the TEXPSHEADERS path. As web2c systems commonly specify a 'private' tree for pdfTeX where configuration and map files are located, this allows individual users or projects to maintain customized versions of the configuration file, and means that specific TeX input files need not set any pdfTeX-specific macros.

The configuration file is used to set default values for the following parameters (all of which can be over-ridden in the TeX source file):

output_format Integer parameter specifying whether the output format should be DVI or PDF. Positive value means PDF output, otherwise DVI output.

compress_level 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.

decimal_digits The number of decimal digits after the decimal point.

page_width, page_height Dimension parameters specifying the page width and page height of PDF output. If not specified then page width is calculated as **width of the box being shiped out** $+ 2 \times (1in +$ `\hoffset`$)$. The page height is similar.

horigin, vorigin Dimension parameters specifying the offset of the TEX output box from the top left corner of the 'paper'

map The name of the font mapping file (similar to those used by many DVI to PostScript drivers); more than one map file can be specified, using multiple `map` lines. If the name of the map file is prefixed with a +, its values are appended to the existing set, otherwise they replace it.

A typical `pdftex.cfg` file looks like this, setting up output for A4 paper size and the standard TEX offset of 1 inch, and loading two map files for fonts:

```
output_format 1
compress_level 0
decimal_digits 2
page_width 210mm
page_height 297mm
horigin 1in
vorigin 1in
map standard.map
map +cm.map
```

2.7 Creating formats

Formats for pdfTEX are created in the same way as for TEX. For plain TEX and LATEX it looks like:

```
pdftex -ini -fmt=pdftex plain \\dump
pdftex -ini -fmt=pdflatex latex.ltx
```

The formats (`pdftex`, `pdflatex` or other formats) should be placed in the `TEXFORMATS` path.

2.8 Testing the installation

Now we can test the installation:

```
pdftex example
```

If the installation is ok, this run should produce a file called `example.pdf`. The file `example.tex` is also a good place to look how pdfTEX new primitives are used.

2.9 Common problems

The most common problem with installation is pdfTEX complains that something cannot be found. In such case the best place to look/edit is the file `texmf.cnf`. Also setting `KPATHSEA_DEBUG=255` before running pdftex or running pdfTEX with option `-k 255` will cause pdfTEX writes a lot of debugging information, which may be useful to find out the troubles. Variables in `texmf.cnf` can be overwritten by enviroment variables. Here are some most common problems with getting started:

`"! I can't read tex.pool; bad path?"`

Probably `TEXMFCNF` is not set correctly, or `TEXPOOL` (in `texmf.cnf`) doesn't contain path to pool file (`pdftex.pool`).

`"! You have to increase POOLSIZE."`

Value of `pool_size` in `texmf.cnf` must be increased. If `pool_size` is not specified in `texmf.cnf` then we can add something like

```
 "pool_size        =  500000"
```

`"I can't find the format file 'pdftex.fmt'!"`

or

`"I can't find the format file 'pdflatex.fmt'!"`

Format is not created (see above how to do that) or is not properly placed. Make sure that `TEXFORMATS` in `texmf.cnf` contains path to `pdftex.fmt` or `pdflatex.fmt`.

pdfTEX cannot find: config file (`pdftex.cfg`), map files (`*.map`), encoding vectors (`*.enc`), virtual fonts, Type1 fonts, TrueType fonts or image files.

Make sure that the required file exists and the correspoding variable in `texmf.cnf` contains path to the file. See above which variables pdfTEX needs apart from the ones TeX uses.

3 Fonts

pdfTEX can 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, Helvetica, Courier, Symbol and Dingbats). It is possible to use METAFONT-generated fonts in pdfTEX—however it is strongly recommended not to use METAFONT-fonts if an equivalent is available in Type 1 or TrueType format, as 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.

3.1 Map files

pdfTEX reads *map files* (specified in the *configuration file* (see above, section 2.6), in which reencoding and partial downloading for each font are specified. Every font needed must be listed, each on a separate line. The syntax of each line is similiar to dvips' map files (but may be changed later), and can contain up to the following (some are optional) fields in fixed order: *texname*, *basename*, *fontflags*, *fontfile* and *encodingfile*.

texname: the name of the TFM file

basename: the base font name (PostScript font name)

fontflags: the flags specifying some chracteristics of the font. If not specified then will be treated as 4.

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. If it is preceded by a !, the font is not read at all, and is assumed to be available on the users Acrobat platform. Note that the standard 14 fonts are never downloaded, even they are specified to be downloaded in some map file.

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

If no encoding is specified, the font's built-in default encoding is used.

Here are some sample lines:

Include font entirely without reencoding:

```
pgsr8r GillSans <<pgsr8a.pfb
```

Include font partly without reencoding:

```
pgsr8r GillSans <pgsr8a.pfb
```

Do not read the font at all, or reencode it from font file —

```
pgsr8r GillSans !pgsr8a.pfb
```

Include font entirely and reencode:

```
pgsr8r GillSans <<pgsr8a.pfb 8r.enc
```

Partially include font and reencode:

```
pgsr8r GillSans <pgsr8a.pfb 8r.enc
```

Do not include font but extract parameters from font file and reencode — only works for font with Adobe Standard Encoding. The font flags says how this font looks like so AcroReader can generate similiar instance if the font resource is not available on the target system.

```
pgsr8r GillSans 32 pgsr8a.pfb 8r.enc
```

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

```
verdana8r Verdana  <verdana.ttf 8r.enc
```

A base font can also be reencoded:

```
phvr8r Helvetica 8r.enc
```

3.2 TrueType fonts

As mentioned above, pdfTEX can work with TrueType fonts. Adding TrueType into map files is similar to Type 1 font. The only extra thing to do with TrueType is to create TFM. There is a program ttf2afm in pdfTEX distribution which can be used to extract AFM from TrueType fonts. Usage is simple:

```
ttf2afm <ttf> [<encoding>]
```

ttf is the TrueType font file, the optinal *encoding* specifies the encoding, which is the same as encoding vector used in map files for pdfTEX and dvips. If the encoding is not given, all the glyphs of the AFM output will be mapped to `/.notdef`. ttf2afm writes the output AFM to standard output. The rest is easy, as we have AFM already. If we need to know which glyphs are available in the font, we can run ttf2afm without encoding to get all glyph names.

4 New primitives

There follows a short description of new primitives added by pdfTEX. One way to learn more about how to use these primitives is to have a look at the file `example.tex` in the pdfTEX distribution.

4.1 Document setup

`\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. In other words, this parameter must be set before pdfTEX ships out the first page if we want PDF output. This is the only one parameter that must be set to produce PDF output. All others are optional.

`\pdfcompresslevel`$=n$

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. A value out of this range will be adjusted to the nearest meaningfull value.

`\pdfpagewidth`$=dimen$

`\pdfpageheight`$=dimen$

Dimension parameters specifying the page width and page height of PDF output. If not given then the page dimensions will be calculated as mentioned above.

`\pdfpagesattr{`$text$`}`

Token list parameter specifying optional attributes common for all pages of PDF output file. These attributes can be `MediaBox` (rectangle specifying the natural size of the page), `CropBox` (rectangle specifying the region of the page being displayed and printed), `Rotate` (number of degrees the page should be rotated clockwise when it is displayed or printed; must be 0 or a multiple of 90).

`\pdfpageattr{`$text$`}`

this is similiar to `\pdfpageattr`, but it takes priority to the former one. It can be used to overwrite any attributes given by `\pdfpagesattr` for individual pages.

4.2 The document info and catalog

`\pdfinfo{`$info\ keys$`}`

This allows the user to add information to the document info section; if this is provided, it can be seen in Acrobat Reader with the menu option Document Information, General. The $info\ keys$ parameter is a set of data pairs, a key and a value. The key names are preceded by a `/`, and the values are in parentheses; all keys are optional. The possible keys are `/Author`, `/CreationDate` (defaults to current date), `/ModDate`, `/Creator` (defaults to TeX), `/Producer` (defaults to pdfTeX), `/Title`, `/Subject`, and `/Keywords`.

`/CreationDate` and `/ModDate` are expressed in the form `D:YYYYMMDDhhmmss`), where YYYY is the year, MM is the month, DD is the day, hh is the hour, mm is the minutes, and ss is the seconds.

Multiple apperances of `\pdfinfo` will be concatenated to the only one. If a key is given more than once, then the first appearance will take priority. An example of use of `\pdfinfo` may look like:

```
\pdfinfo{
    /Title (example.pdf)
    /Creator (TeX)
    /Producer (pdfTeX)
    /Author (Tom and Jerry)
    /CreationDate (D:19980212201000)
    /ModDate (D:19980212201000)
    /Subject (Example)
    /Keywords (pdfTeX)
}
```

\pdfcatalog{*catalog keys*}*openaction goto* ⟨*num num* — *name*{*name*}⟩ *appearance*

Similar to the document info section is the document catalog, where the available keys are `/URI`, which provides the base URL of the document, and */PageMode* determines how Acrobat displays the document on startup. The possibilities for the latter 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`.

The *openaction* is the action provided when opening the document and is specified in the same way as for internal links (see below, section 4.6), e.g. `goto page 3 {/Fit}`.

4.3 Graphics inclusion

\pdfimage *width width height height depth depth filename*

Insert a bitmap image in PNG format, optionally changing width, height, depth or any combination of them. If all of them are given, the image will be scaled to fit the specified dimensions. If some of them (but not all) are given, the rest will be set to a value corresponding to the remaining ones so as to make the image size to yield the same proportion of $width : (height + depth)$ as the natural image size, where depth is treated as zero. If non of them is given then the image will take the natural size of it. An image inserted at natural size often has resolution 72 DPI in output file, but some images may contain data specifying image resolution, and in such a case the image will be scaled to the

intended resolution. The filename of the image must appear after the optional dimension parameters. The dimension of the image can be accessed by enclosing the `\pdfimage` command to a box and checking the dimensions of the box.

4.4 XObject Forms

`\pdfform` *num*

Write out the TEX box *num* as a XObject Form to the PDF file.

`\pdflastform`

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

`\pdfrefform \name`

Put in a reference to the XObject 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 XObject Form object corresponds to the content of a TEX box, which can also contain pictures and references to other XObject Form objects as well. After that the XObject 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.

4.5 Annotations

`\pdfannot` *width width* `height` *height* `depth` *depth* {*text*}

Attach an annotation at the current point in the text. The text is inserted as raw PDF code to the contents of annotation.

`\pdflastannot`

Returns the object number of last annotation created by `\pdfannot`. These two primitives allow user to create any annotation that cannot be created by `\pdfannotlink` (see below).

4.6 Destinations and links

`\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
`xyz`	keep current zoom factor

`xyz` can optionally be followed by `zoom` *factor* to provide a fixed zoom-in. The *factor* is like TEX magnification, ie 1000 is the 'normal' page view.

`\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. Thus `/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

page n	Jump to page n
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, e.g. `/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.

4.7 Bookmarks

`\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. 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).

4.8 Article threads

\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.

4.9 Miscellaneous

\pdfliteral{*pdf text*}

Like \special in normal TEX, this command inserts raw PDF code into the output. This allows support of color, and text transformation.

\pdfobj *stream* {*text*}

Similar to \pdfliteral, but the text is inserted as contents of an object. If the optional keyword *stream* is given then the contents will be inserted as a stream.

\pdflastobj

Returns the object number of the last object created by `\pdfobj`. These primitives provide a mechanism allowing inserting a user-defined object to PDF output.

`\pdfversion`

Returns the version of pdfTeX multiple by 100, e.g. for version `0.12x` it returns 12.

`\pdfrevision`

Returns the revision of pdfTeX, e.g. for version `0.12x` it returns `x`.

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:

1. LaTeX picture mode: since this is implemented simply in terms of font characters, it works in exactly the same way as usual;
2. Xy-pic: If the PostScript backend is not requested, Xy-pic uses its own Type 1 fonts, and needs no special attention;
3. tpic: The 'tpic' `\special` commands (used in some macro packages) can be redefined to produce literal PDF, using macros by Hans Hagen;
4. 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;

5. It is possible to insert a "pure" PDF file (PDF that has only one page without fonts, bitmaps or other resources) using 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 are distributed separately.

For new work, the MetaPost route is highly recommended. For the future, Adobe have 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 program 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/`.