The pdfTEX user manual

```
%PDF-1.4
%ÐÔÅØ
3 0 obj <<
/Length 152
>>
stream
BT
/F51 9.9626 Tf 91.925 759.927 Td
[(W)80(elcome)-250(to)-250(pdfT)]TJ
67.818 -2.241 Td [(E)]TJ 4.842
2.241 Td [(X!)]TJ 138.924 -654.747
Td [(1)]TJ
ET
endstream
endobj
2 0 obj <<
/Type /Page
/Contents 3 0 R
/Resources 1 0 R
/MediaBox [0 0 595.276 841.89]
/Parent 5 0 R
>> endobj
1 0 obj <<
/Font << /F51 4 0 R >>
/ProcSet [ /PDF /Text ]
>> endobj
7 0 obj
[333 408 500 500 833 778 333 333
333 500 564 250 333 250 278 500
500 500 500 500 500 500 500 500
500 278 278 564 564 564 444 921
722 667 667 722 611 556 722 722
333 389 722 611 889 722 722 556 722
667 556 611 722 722 944 722 722 611
333 278 333 469 500 333 444 500 444
500 444 333 500 500 278 278 500 278
778 500 500 500 500 333 389 278]
endobj
8 0 obj <<
/Type /FontDescriptor
/FontName /Times-Roman
/Flags 34
/FontBBox [0 -216 1000 678]
/Ascent 678
/CapHeight 651
/Descent -216
/ItalicAngle 0
/StemV 83
/XHeight 450
>> endobj
6 0 obj <<
/Type /Encoding
/Differences [33/exclam 49/one
69/E 84/T 87/W/X 99/c/d/e/f 108/l/m
111/o/p 116/t]
>> endobj
4 0 obj <<
/Type /Font
/Subtype /Type1
/BaseFont /Times-Roman
/FontDescriptor 8 0 R
/FirstChar 33
/LastChar 116
/Widths 7 0 R
/Encoding 6 0 R
>> endobj
5 0 obj <<
/Type /Pages
/Count 1
/Kids [2 0 R]
>> endobj
9 0 obj <<
/Type /Catalog
/Pages 5 0 R
>> endobj
10 0 obj <<
/Producer (pdfTeX-1.40.9)
/Creator (TeX)
/CreationDate (D:20091130223220Z)
/ModDate (D:20091130223220Z)
/Trapped /False
/PTEX.Fullbanner (This is pdfTeX,
Version 3.1415926-1.40.9-2.2 (Web2C
7.5.7) kpathsea version 3.5.7)
>> endobj
xref
0 11
0000000000 65535 f
0000000335 00000 n
0000000224 00000 n
0000000015 00000 n
0000001058 00000 n
0000001210 00000 n
0000000939 00000 n
0000000403 00000 n
0000000756 00000 n
0000001267 00000 n
0000001316 00000 n
trailer
<< /Size 11
/Root 9 0 R
/Info 10 0 R
/ID
[<F1F50AD74E4042CC5CBDE3B79C474D57>
<F1F50AD74E4042CC5CBDE3B79C474D57>]
>>
startxref
1559
%%EOF
```

The pdfTEX user manual

Hàn Thế Thành
Sebastian Rahtz
Hans Hagen
Hartmut Henkel
Paweł Jackowski
Martin Schröder

January 25, 2007

Rev. 1.675

The title page of this manual represents the plain TEX coded text "Welcome to pdfTEX!"

```
\pdfoutput=1
\pdfcompresslevel=0
\font\tenrm=ptmr8r
\tenrm
Welcome to pdf\TeX!
\bye
```

Contents

1 Introduction 1
2 About PDF 2
3 Getting started 3
4 Macro packages supporting PDFTEX . . 9
5 Setting up fonts 10
6 Formal syntax specification 17
7 PDFTEX primitives 21
8 Graphics 41
9 Character translation 42

Abbreviations 42
Examples of HZ and protruding 43
Additional PDF keys 45
Colophon 46
GNU Free Documentation License 47

1 Introduction

The main purpose of the pdfTEX project is to create and maintain an extension of TEX that can produce pdf directly from TEX source files and improve/enhance the result of TEX typesetting with the help of pdf. When pdf output is not selected, pdfTEX produces normal dvi output, otherwise it generates pdf output that looks identical to the dvi output. An important aspect of this project is to investigate alternative justification algorithms (e. g. a font expansion algorithm akin to the hz micro–typography algorithm by Prof. Hermann Zapf), optionally making use of Multiple Master fonts.

pdfTEX is based on the original TEX sources and Web2c, and has been successfully compiled on Unix, Win32 and MS-DOS systems. It is under active development, with new features trickling in. Great care is taken to keep new pdfTEX versions backward compatible with earlier ones.

For some years there has been a 'moderate' successor to TEX available, called ε-TEX. Because mainstream macro packages such as LATEX have started supporting this welcome extension, the ε-TEX functionality has also been integrated into the pdfTEX code. For a while (TEX Live 2004 and 2005) pdfTEX therefore came in two flavours: the ε-TEX enabled pdfeTEX engine and the standard one, pdfTEX. The ability to produce both pdf and dvi output made pdfeTEX the primary TEX engine in these distributions. Since pdfTEX version 1.40 now the ε-TEX extensions are part already of the pdfTEX engine, so there is no need anymore to ship pdfeTEX. The ε-TEX functionality of pdfTEX can be disabled if not required. Other extensions are MLTEX and encTEX; these are also included in the current pdfTEX code.

pdfTEX is maintained by Hàn Thế Thành, Martin Schröder, Hans Hagen, Taco Hoekwater, Hartmut Henkel, and others. The pdfTEX homepage is `http://www.pdftex.org`. Please send pdfTEX comments and bug reports to the mailing list pdftex@tug.org.

We thank all readers who send us corrections and suggestions. We also wish to express the hope that pdfTEX will be of as much use to you as it is to us. Since pdfTEX is still being improved and extended, we suggest you to keep track of updates.

1.1 About this manual

This manual revision (1.675) tries to keep track with the recent pdfTEX development up to version 1.40.0. Main text updates were done regarding the new configuration scheme, font mapping, and new or updated primitives. The primary repository for the manual and its sources is at `http://sarovar.org/projects/pdftex/`. Copies in pdf format can also be found at the CTAN network in directory `ctan:systems/pdftex`.

Thanks to Karl Berry for proof reading and submitting a long changes list. New errors might have slipped in afterwards by the editor. Please send questions or suggestions by email to pdftex@tug.org.

1.2 Legal Notice

2 About PDF

The cover of this manual lists an almost minimal pdf file generated by pdfTEX, with the corresponding source file on the next page. Unless compression is enabled, such a pdf file is rather verbose and readable. The first line specifies the version used; currently pdfTEX produces level 1.4 output by default. pdf viewers are supposed to silently skip over all elements they cannot handle.

A pdf file consists of objects. These objects can be recognized by their number and keywords:

```
9 0 obj << /Type /Catalog /Pages 5 0 R >> endobj
```

Here `9 0 obj ... endobj` is the object capsule. The first number is the object number. The sequence `5 0 R` is an object reference, a pointer to another object (no. 5). The second number (here a zero) is currently not used in pdfTEX; it is the version number of the object. It is for instance used by pdf editors, when they replace objects by new ones.

When a viewer opens a pdf file, it goes right to the end of the file, looking for the keyword `startxref`. The number after `startxref` gives the absolute position (byte offset from the file start) of the so called 'object cross-reference table' that begins with the keyword `xref`. This table in turn tells the byte offsets of all objects that make up the pdf file, providing fast random access to the individual objects (here the `xref` table shows 11 objects, numbered from 0 to 10; the object no. 0 is always unused). The actual starting point of the file's object structure is defined after the `trailer`: The `/Root` entry points to the `/Catalog` object (no. 9). In this object the viewer can find the pointer `/Pages` to the page list object (no. 5). In our example we have only one page. The trailer also holds an `/Info` entry, which points to an object (no. 10) with a bit more about the document. Just follow the thread:

`/Root` ⟶ object 9 ⟶ `/Pages` ⟶ object 5 ⟶ `/Kids` ⟶ object 2 ⟶ `/Contents` ⟶ object 3

As soon as we add annotations, a fancy word for hyperlinks and the like, some more entries will be present in the catalog. We invite users to take a look at the pdf code of this file to get an impression of that.

The page content is a stream of drawing operations. Such a stream can be compressed, where the level of compression can be set with `\pdfcompresslevel` (compression is switched off for the title page). Let's take a closer look at this stream in object 3. Often (but not in our example) there is a transformation matrix, six numbers followed by `cm`. As in PostScript, the operator comes after the operands. Between `BT` and `ET` comes the text. A font is selected by a `Tf` operator, which is given a font resource name `/F..` and the font size. The actual text goes into `()` bracket pairs so that it creates a PostScript string. The numbers inbetween bracket pairs provide horizontal movements like spaces and fine glyph positioning (kerning). When one analyzes a file produced by a less sophisticated typesetting engine, whole sequences of words can be recognized. In pdf files generated by pdfTEX however, many words come out rather fragmented, mainly because a lot of kerning takes place; in our example the `80` moves the text `(elcome)` left towards the letter `(W)` by 80/1000 of the font size. pdf viewers in search mode simply ignore the kerning information in these text streams. When a document is searched, the search engine reconstructs the text from these (string) snippets.

Every `/Page` object points also to a `/Resources` object (no. 1) that gives all ingredients needed to assemble the page. In our example only a `/Font` object (no. 4) is referenced, which in turn tells that the text is typeset in `/Font /Times-Roman`. The `/Font` object points also to a `/Widths` array (object no. 7) that tells for each character by how much the viewer must move forward horizontally after typesetting a glyph. More details about the font can be found in the `/FontDescriptor` object (no. 8); if a font file is embedded, this object points to the font program stream. But as the Times-Roman font used for our example is one of the 14 so–called standard fonts that should always be present in any pdf viewer and therefore need not be embedded in the pdf file, it is left out here for brevity. However, when we use for instance a Computer Modern Roman font, we have to make sure that this font is later available to the pdf viewer, and the best way to do this is to embed the font.

It's highly recommended nowadays to embed even the standard fonts, as modern viewers often don't use the original 14 standard fonts anymore, but instead approximate them by instances of built–in Multiple Master fonts (e. g. the Adobe Reader 7 approximates the Times-Roman variants by the Minion font). So you never really know how it looks exactly at the viewer side unless you embed every font.

In this simple file we don't specify in what way the file should be opened, for instance full screen or clipped. A closer look at the page object no. 2 (`/Type /Page`) shows that a mediabox (`/MediaBox`) is part of the page description. A mediabox acts like the (high-resolution) bounding box in a PostScript file. pdfTEX users can add dictionary stuff to page objects by the `\pdfpageattr` primitive.

Although in most cases macro packages will shield users from these internals, pdfTEX provides access to many of the entries described here, either automatically by translating the TEX data structures into pdf ones, or manually by pushing entries to the catalog, page, info or self created objects. One can for instance create an object by using `\pdfobj` after which `\pdflastobj` returns the number. So

```
\pdfobj{/Type /Catalog /Pages 5 0 R}
```

inserts an object into the pdf file, while `\pdflastobj` returns the number pdfTEX assigned to this object. Unless objects are referenced by others, they will just end up as isolated entities, not doing any real harm but bloating the pdf file. In general this rather direct way of pushing objects in the pdf files by primitives like `\pdfobj` is not very useful, and only makes sense when implementing, say, fill–in field support or annotation content reuse. We will come to that later.

For those who want to learn more about the gory pdf details, the best bet is to read the PDF Reference. As of the time of writing you can download this book as a big pdf file from Adobe's PDF Technology Center, `http://www.adobe.com/devnet/pdf/pdf_reference.html` — or get the heavy paper version.

Those who, after this introduction, feel unsure how to proceed, are advised to read on but skip section 7. Before we come to that section, we will describe how to get started with pdfTEX.

3 Getting started

This section describes the steps needed to get pdfTEX running on a system where pdfTEX is not yet installed. Nowadays virtually all TEX distributions have pdfTEX as a component, such as TEX Live, teTEX, XEmTEX, MikTeX, proTEXt, and CMacTEX. The ready to run TEX Live distribution comes with pdfTEX versions for many Unix, Win32, and Mac OS X systems; more information can be found at `http://www.tug.org/tex-live/`. teTEX by Thomas Esser is a source distribution with an automated compilation process for Unix systems; see `http://www.tug.org/teTeX/`. For Win32 systems there are also three separate distributions that contain pdfTEX, all in `ctan:systems/win32`: XEmTEX by Fabrice Popineau, MikTeX by Christian Schenk, and proTEXt (based on MikTeX) by Thomas Feuerstack. So when you use any of these distributions, you don't need to bother with the pdfTEX installation procedure in the next sections.

If there is no precompiled pdfTEX binary for your system, or the version coming with a distribution is not the current one and you would like to try out a fresh pdfTEX immediately, you will need to build pdfTEX from sources; read on. You should already have a working TEX system, e. g. TEX Live or teTEX, into which the freshly compiled pdfTEX will be integrated. Note that the installation description in this manual is Web2c–specific.

3.1 Getting sources and binaries

The latest sources of pdfTEX are currently distributed for compilation on Unix systems (including Linux), and Win32 systems (Windows 95, 98, NT, 2000, XP). The primary location where one can fetch the latest released code is at the developers' homepage `http://sarovar.org/projects/pdftex/`, where you also find bug tracking information, and the manual sources. Download the pdfTEX archive from there.

The pdfTEX sources can also be found at their canonical place in the CTAN network, `ctan:systems/pdftex`. Separate pdfTEX binaries for various systems might also be available, check out the subdirectories below `ctan:systems`.

3.2 Compiling

The compilation is expected to be easy on Unix–like systems and can be described best by example. Assuming that the file `pdftex.zip` is downloaded to some working directory, e. g. `$HOME/pdftex`, on a Unix system the following steps are needed to compile pdfTEX:

```
cd $HOME/pdftex
unzip pdftex-1.40.0.zip
cd pdftex-1.40.0
./build.sh
```

The binary `pdftex` is then built in the subdirectory `build/texk/web2c`. In the same directory also the corresponding pool file `pdftex.pool` is generated; it's needed for creating the format files.

The obsolescent binary `pdfetex` is still generated for backward compatibility, but since version 1.40 it is just a file copy of the file `pdftex`.

Together with the `pdftex` binary also the `pdftosrc` and `ttf2afm` binaries are generated.

3.3 Placing files

The next step is to put the freshly compiled `pdftex`, `pdftosrc`, and `ttf2afm` binaries and the pool file `pdftex.pool` into their proper places within the tds structure of the TEX system. Put the binary files into the binary directory (e. g. for a typical TEX Live system) `/usr/local/texlive/2006/bin/x86_64-linux`, and the pool file into `/usr/local/texlive/2006/texmf/web2c`.

Don't forget to do a `texconfig-sys␣init` afterwards, so that all formats are regenerated system-wide with the fresh `pdftex` binary.

3.4 Setting search paths

Web2c–based programs, including pdfTEX, use the Web2c run–time configuration file called `texmf.cnf`. The location of this file is the appropriate position within the tds tree relative to the place of the pdfTEX binary; on a TEX Live system, file `texmf.cnf` typically is located either in directory `texmf/web2c` or `texmf-local/web2c`. The path to file `texmf.cnf` can also be set up by the environment variable `TEXMFCNF`.

Next you might need to edit `texmf.cnf` so that pdfTEX can find all necessary files, but the `texmf.cnf` files coming with the major TEX distributions should already be set up for normal use. You might check into the file `texmf.cnf` to see where the various bits and pieces are going.

pdfTEX uses the search path variables shown in table 1.

`TEXMFOUTPUT` Normally, pdfTEX puts its output files in the current directory. If any output file cannot be opened there, it tries to open it in the directory specified in the environment variable `TEXMFOUTPUT`. There is no default value for that variable. For example, if you type `pdftex paper` and the current directory is not writable, if `TEXMFOUTPUT` has

used for	texmf.cnf
output files	`TEXMFOUTPUT`
input files, images	`TEXINPUTS`
format files	`TEXFORMATS`
text pool files	`TEXPOOL`
encoding files	`ENCFONTS`
font map files	`TEXFONTMAPS`
tfm files	`TFMFONTS`
virtual fonts	`VFFONTS`
type1 fonts	`T1FONTS`
TrueType fonts	`TTFONTS`
OpenType fonts	`OPENTYPEFONTS`
pixel fonts	`PKFONTS`

Table 1 The Web2c variables.

the value `/tmp`, pdfTeX attempts to create `/tmp/paper.log` (and `/tmp/paper.pdf`, if any output is produced.)

`TEXINPUTS` This variable specifies where pdfTeX finds its input files. Image files are considered input files and searched for along this path.

`TEXFORMATS` Search path for format (`.fmt`) files.

`TEXPOOL` Search path for pool (`.pool`) files.

`ENCFONTS` Search path for encoding (`.enc`) files.

`TEXFONTMAPS` Search path for font map (`.map`) files.

`TFMFONTS` Search path for font metric (`.tfm`) files.

`VFFONTS` Search path for virtual font (`.vf`) files. Virtual fonts are fonts made up of other fonts. Because pdfTeX produces the final output code, it must consult those files.

`T1FONTS` Search path for Type 1 font files (`.pfa` and `.pfb`). These outline (vector) fonts are to be preferred over bitmap pk fonts. In most cases Type 1 fonts are used and this variable tells pdfTeX where to find them.

`TTFFONTS, OPENTYPEFONTS` Search paths for TrueType (`.ttf`) and OpenType (`.otf`) font files. Like Type 1 fonts, TrueType and OpenType fonts are also outlines.

`PKFONTS` Search path for packed (bitmap) font (`.pk`) files. Unfortunately bitmap fonts are still displayed poorly by some pdf viewers, so when possible one should use outline fonts. When no outline is available, pdfTeX tries to locate a suitable pk font (or invoke a process that generates it).

3.5 The PDFTeX configuration

One has to keep in mind that, as opposed to TeX with its dvi output, the pdfTeX program does not require a separate postprocessing stage to transform the TeX input into a pdf file. As a consequence, all data needed for building a ready pdf page must be available during the pdfTeX run, in particular information on media dimensions and offsets, graphics files for embedding, and font information (font files, encodings).

When TeX builds a page, it places items relative to the top left page corner (the dvi reference point). Separate dvi postprocessors allow specifying the paper size (e. g. ‘A4’ or ‘letter’), so that this reference point is moved to the correct position on the paper, and the text ends up at the right place.

In pdf, the paper dimensions are part of the page definition, and pdfTEX therefore requires that they be defined at the beginning of the pdfTEX run. As with pages described by PostScript, the pdf reference point is in the lower–left corner.

Formerly, these dimensions and other pdfTEX parameters were read in from a configuration file named `pdftex.cfg`, which had a special (non-TEX) format, at the start of processing. Nowadays such a file is ignored by pdfTEX. Instead, the page dimensions and offsets, as well as many other parameters, can be set by pdfTEX primitives during the pdfTEX format building process, so that the settings are dumped into the fresh format and consequently will be used when pdfTEX is later called with that format. All settings from the format can still be overridden during a pdfTEX run by using the same primitives. This new configuration concept is a more unified approach, as it avoids the configuration file with a special format.

A list of pdfTEX primitives relevant for setting up the pdfTEX engine is given in table 2. All primitives are described in detail within later sections. Figure 1 shows a recent configuration file (`pdftexconfig.tex`) in TEX format, using the primitives from table 2, which typically is read in during the format building process. It enables pdf output, sets paper dimensions and the default pixel density for pk font inclusion. The default values are chosen so that pdfTEX often can be used (e. g. in `-ini` mode) even without setting any parameters.

internal name	**type**	**default**	**comment**
`\pdfoutput`	integer	0	dvi
`\pdfadjustspacing`	integer	0	off
`\pdfcompresslevel`	integer	9	best
`\pdfobjcompresslevel`	integer	0	no object streams
`\pdfdecimaldigits`	integer	4	max.
`\pdfimageresolution`	integer	72	dpi
`\pdfpkresolution`	integer	0	72 dpi
`\pdfpkmode`	token reg.	empty	mode set in `mktex.cnf`
`\pdfuniqueresname`	integer	0	
`\pdfprotrudechars`	integer	0	
`\pdfminorversion`	integer	4	pdf 1.4
`\pdfforcepagebox`	integer	0	
`\pdfinclusionerrorlevel`	integer	0	
`\pdfhorigin`	dimension	1 in	
`\pdfvorigin`	dimension	1 in	
`\pdfpagewidth`	dimension	0 pt	
`\pdfpageheight`	dimension	0 pt	
`\pdflinkmargin`	dimension	0 pt	
`\pdfdestmargin`	dimension	0 pt	
`\pdfthreadmargin`	dimension	0 pt	
`\pdfmapfile`	text	`pdftex.map`	not dumped

Table 2 The set of pdfTEX configuration parameters.

```
% Set pdfTeX parameters for pdf mode (replacing pdftex.cfg file).
% Thomas Esser, 2004. public domain.
\pdfoutput=1
\pdfpagewidth=210 true mm
\pdfpageheight=297 true mm
\pdfpkresolution=600
\endinput
```

Figure 1 A typical configuration file (`pdftexconfig.tex`).

Independent of whether such a configuration file is read or not, the first action in a pdfTEX run is that the program reads the global Web2c configuration file (`texmf.cnf`), which is common to all programs in the web2C system. This file mainly defines file search paths, the memory layout (e. g. pool and hash size), and other general parameters.

3.6 Creating format files

```
% Thomas Esser, 1998, 2004. public domain.
\ifx\pdfoutput\undefined
\else
  \ifx\pdfoutput\relax
  \else
    \input pdftexconfig
    \pdfoutput=0
  \fi
\fi
\input etex.src
\dump
\endinput
```

Figure 2 File `etex.ini` for ε-TEX format with dvi output.

```
\ifx\pdfoutput\undefined
\else
  \ifx\pdfoutput\relax
  \else
    \input pdftexconfig
    \pdfoutput=1
  \fi
\fi
\scrollmode
\input latex.ltx
\endinput
```

Figure 3 File `pdflatex.ini` for LATEX format with pdf output.

The pdfTEX engine allow building formats for dvi and pdf output in the same way as the classical TEX engine does for dvi. Format generation is enabled by the `-ini` option. The default mode (dvi or pdf) can be chosen either on the command line by setting the option `-output-format` to `dvi` or `pdf`, or by setting the `\pdfoutput` parameter. The format file then inherits this setting, so that a later call to pdfTEX with this format starts in the preselected mode (which still can be overrun then). A format file can be read in only by the engine that has generated it; a format incompatible with an engine leads to a fatal error.

It is customary to package the configuration and macro file input into a `.ini` file. E. g., the file `etex.ini` in figure 2 is for generating an ε-TEX format with dvi output (it contains a few comparisons to be safe also for TEX engines). A similar file `pdflatex.ini` can be used for generating a LATEX format with pdf output; refer to figure 3. One can see how the primitive `\pdfoutput` is used to override the output mode set by file `pdftexconfig.tex`. The corresponding pdfTEX calls for format generation are:

```
pdftex -ini *etex.ini
pdftex -ini pdflatex.ini
```

These calls produce format files `etex.fmt` and `pdflatex.fmt`, as the default format file name is taken from the input file name. You can overrule this with the `-jobname` option. The asterisk `*` in the first example line below tells the pdfTEX engine to go into extended `-ini` mode (ε-TEX enabled); otherwise it stays in non–extended `-ini` mode. The extended `-ini` mode can also be forced by the `-etex` command line option, as shown in the 2nd line below.

```
pdftex -ini -jobname=pdfelatex *pdflatex.ini
pdftex -ini -jobname=pdfelatex -etex pdflatex.ini
```

In ConTEXt the generation depends on the interface used. A format using the English user interface is generated with

```
pdftex -ini cont-en
```

When properly set up, one can also use the ConTEXt command line interface TEXexec to generate one or more formats, like:

```
texexec --make en
```

for an English format, or

```
texexec --make en de
```

for an English and German one. Most users will simply say:

```
texexec --make --all [--alone]
```

and so generate the TEX and METAPOST related formats that ConTEXt needs. Whatever macro package used, the formats should be placed in the `TEXFORMATS` path.

3.7 Testing the installation

When everything is set up, you can test the installation. In the distribution there is a plain TEX test file `samplepdf.tex` in the `manual/samplepdf/` directory. Process this file by typing:

```
pdftex samplepdf
```

If the installation is ok, this run should produce a file called `samplepdf.pdf`. The file `samplepdf.tex` is also a good place to look for how to use pdfTEX's primitives.

3.8 Common problems

The most common problem with installations is that pdfTEX complains that something cannot be found. In such cases make sure that `TEXMFCNF` is set correctly, so pdfTEX can find `texmf.cnf`. The next best place to look/edit is the file `texmf.cnf`. When still in deep trouble, set `KPATHSEA_DEBUG=255` before running pdfTEX or run pdfTEX with option `-k 255`. This will cause pdfTEX to write a lot of debugging information that can be useful to trace problems. More options can be found in the Web2c documentation.

Variables in `texmf.cnf` can be overwritten by environment variables. Here are some of the most common problems you can encounter when getting started:

- `I can't read pdftex.pool; bad path?`

  `TEXMFCNF` is not set correctly and so pdfTEX cannot find `texmf.cnf`, or `TEXPOOL` in `texmf.cnf` doesn't contain a path to the pool file `pdftex.pool`.

- `You have to increase POOLSIZE.`

  pdfTEX cannot find `texmf.cnf`, or the value of `pool_size` specified in `texmf.cnf` is not large enough and must be increased. If `pool_size` is not specified in `texmf.cnf` then you can add something like

  ```
  pool_size=500000
  ```

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

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

- `---! xx.fmt was written by tex`
  `Fatal format file error; I'm stymied`

  This appears e. g. if you forgot to regenerate the `.fmt` files after installing a new version of the pdfTEX binary and `pdftex.pool`. The first line tells by which engine the offending format was generated.

- `TEX.POOL doesn't match; TANGLE me again!`
  `TEX.POOL doesn't match; TANGLE me again (or fix the path).`

  This might appear if you forgot to install the proper `pdftex.pool` when installing a new version of the pdfTEX binary. E. g. under TEX Live then run `texconfig-sys init` as root.

- pdfTEX cannot find one or more map files (`*.map`), encoding vectors (`*.enc`), virtual fonts, Type 1 fonts, TrueType or OpenType fonts, or some image file.

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

  When you have installed new fonts, and your pdf viewer complains about missing fonts, you should take a look at the log file produced by pdfTEX. Missing fonts, map files, encoding vectors as well as missing characters (glyphs) are reported there.

Normally the page content takes one object. This means that one seldomly finds more than a few hundred objects in a simple file. This pdfTEX manual for instance uses approx. 750 objects. In more complex applications this number can grow quite rapidly, especially when one uses a lot of widget annotations, shared annotations or other shared things. In any case pdfTEX's internal object table size will automatically grow to the required size (the parameter `obj_tab_size` for manual control of the object table size is now obsolete and ignored).

4 Macro packages supporting PDFTEX

As pdfTEX generates the final pdf output without help of a postprocessor, macro packages that take care of these pdf features have to be set up properly. Typical tasks are handling color, graphics, hyperlink support, threading, font–inclusion, as well as page imposition and manipulation. All these pdf–specific tasks can be commanded by pdfTEX's own primitives (a few also by a pdfTEX–specific `\special{pdf: ...}` primitive). Any other `\special{}` commands, like the ones defined for various dvi postprocessors, are simply ignored by pdfTEX when in pdf output mode; a warning is given only for non–empty `\special{}` commands.

When a macro package already written for classical TEX with dvi output is to be modified for use with pdfTEX, it is very helpful to get some insight to what extent pdfTEX–specific support is needed. This info can be gathered e. g. by outputting the various `\special` commands as `\message`. Simply type

```
\pdfoutput=1 \let\special\message
```

or, if this leads to confusion,

```
\pdfoutput=1 \def\special#1{\write16{special: #1}}
```

and see what happens. As soon as one 'special' message turns up, one knows for sure that some kind of pdfTEX specific support is needed, and often the message itself gives a indication of what is needed.

Currently all mainstream macro packages offer pdfTEX support, with automatic detection of pdfTEX as engine. So there is normally no need to turn on pdfTEX support explicitly.

- For LATEX users, Sebastian Rahtz' and Heiko Oberdiek's `hyperref` package has substantial support for pdfTEX and provides access to most of its features. In the simplest and most common case, the user merely needs to load `hyperref`, and all cross–references will be converted to pdf hypertext links.

pdf output is automatically selected, compression is turned on, and the page size is set up correctly. Bookmarks are created to match the table of contents.

- The standard LATEX `graphics`, `graphicx`, and `color` packages also have automatic pdfTEX support, which allow use of color, text rotation, and graphics inclusion commands.
- The ConTEXt macro package by Hans Hagen has very full support for pdfTEX in its generalized hypertext features. Support for pdfTEX is implemented as a special driver, and is invoked by typing `\setupoutput[pdftex]` or feeding TEXexec with the `--pdf` option.
- pdf from Texinfo documents can be created by running pdfTEX on the Texinfo file, instead of TEX. Alternatively, run the shell command `texi2pdf` instead of `texi2dvi`.
- A small modification of `webmac.tex`, called `pdfwebmac.tex`, allows production of hyperlinked pdf versions of the program code written in web.

Some nice samples of pdfTEX output can be found at `http://www.pdftex.org`, `http://www.pragma-ade.com`, and `http://www.tug.org/texshowcase`.

5 Setting up fonts

pdfTEX can work with Type 1 and TrueType fonts (and to some extent also with OpenType fonts). Font files should be available and embedded for all fonts used in the document. It is possible to use METAFONT–generated fonts in pdfTEX — but it is strongly recommended not to use these fonts if an equivalent is available in Type 1 or TrueType format, if only because bitmap Type 3 fonts render very poorly in (older versions of) Adobe Reader. Given the free availability of Type 1 versions of all the Computer Modern fonts, and the ability to use standard PostScript fonts, there is rarely a need to use bitmap fonts in pdfTEX.

5.1 Map files

Font map files provide the connection between TEX tfm font files and the outline font file names. They contain also information about re–encoding arrays, partial font embedding ("subsetting"), and character transformation parameters (like SlantFont and ExtendFont). Those map files were first created for dvi postprocessors. But, as pdfTEX in pdf output mode includes all pdf processing steps, it also needs to know about font mapping, and therefore reads in one or more map files. Map files are not read in when pdfTEX is in dvi mode. Pixel fonts can be used without being listed in the map file.

By default, pdfTEX reads the map file `pdftex.map`. In Web2c, map files are searched for using the `TEXFONTMAPS` config file value and environment variable. By default, the current directory and various system directories are searched.

Within the map file, each font is listed on an individual line. The syntax of each line is upward–compatible with `dvips` map files and can contain the following fields (some are optional; explanations follow):

tfmname basename fontflags special encodingfile fontfile

It is mandatory that *tfmname* is the first field. If a *basename* is given, it must be the second field. Similarly if *fontflags* is given it must be the third field (if *basename* is present) or the second field (if *basename* is left out). It is possible to mix the positions of *special*, *encodingfile*, and *fontfile*, however the first three fields must be given in fixed order.

tfmname

sets the name of the tfm file for a font — the name TEX sees. This name must always be given.

basename

sets the (PostScript) base font name, which has two uses:

First, when a pdf file is embedded by `\pdfximage`, the `/BaseFont` names in the font dictionaries of Type 1 and Type 1C (CFF) fonts from the embedded pdf file are checked against this *basename* field. If names match, the glyphs of that font will not be copied from the embedded pdf file, but instead a local font is opened, and all needed glyphs will be taken from the Type 1 font file that is mentioned in the map line (see *fontfile* below). By this collecting mechanism Type 1 glyphs can be shared between several embedded pdf files and with text that is typeset by pdfTEX, which helps keeping the resulting pdf file size small, if many files with similar Type 1(C) fonts are embedded. Replacing Type1 fonts from embedded pdf files requires that also a Type1 font file name is in the *fontfile* field (see below).

Second, if a font file is not to be embedded into the pdf output (*fontfile* field missing), then the *basename* field will be copied to the `/BaseFont` and `/FontName` dictionary entries in the pdf file, so that the PostScript font name will be known to the consumer application (e. g. viewer).

It is highly recommended to always use the *basename* field (but strictly speaking it's optional).

fontflags

specify some characteristics of the font. The following description of these flags is taken, with slight modification, from the PDF Reference (the section on font descriptor flags). Viewers can adapt their rendering to these flags, especially when they substitute a non-embedded font by some own approximation.

The value of the flags key in a font descriptor is a 32–bit integer that contains a collection of boolean attributes. These attributes are true if the corresponding bit is set to 1. Table 3 specifies the meanings of the bits, with bit 1 being the least significant. Reserved bits must be set to zero.

bit position	semantics
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

Table 3 The meaning of flags in the font descriptor.

All characters in a *fixed–width* font have the same width, while characters in a proportional font have different widths. Characters in a *serif font* have short strokes drawn at an angle on the top and bottom of character stems, while sans serif fonts do not have such strokes. A *symbolic font* contains symbols rather than letters and numbers. Characters in a *script font* resemble cursive handwriting. An *all–cap* font, which is typically used for display purposes such as titles or headlines, contains no lowercase letters. It differs from a *small–cap* font in that characters in the latter, while also capital letters, have been sized and their proportions adjusted so that they have the same size and stroke weight as lowercase characters in the same typeface family.

Bit 6 in the flags field indicates that the font's character set conforms to the Adobe Standard Roman Character Set, or a subset of that, and that it uses the standard names for those characters.

Finally, bit 19 is 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 thus cannot be distinguished from bold characters. If bit 19 is set, features of bold characters may be thickened at small text sizes.

If the *fontflags* field is not given, pdfTEX treats it as being 4, a symbolic font. If you do not know the correct value, it is best not to specify it at all, as specifying a bad value of font flags may cause troubles in viewers. On the other hand this option is not absolutely useless because it provides backward compatibility with older map files (see the *fontfile* description below).

special

instructions can be used to manipulate fonts similar to the way `dvips` does. Currently only the keywords `SlantFont` and `ExtendFont` are interpreted, other instructions (as `ReEncodeFont` with parameters, see *encoding* below) are just ignored. The permitted `SlantFont` range is −1..1; for `ExtendFont` it's −2..2. The block of *special* instruction must be enclosed by double quotes `"`.

encodingfile

specifies the name of the file containing the external encoding vector to be used for the font. The encoding file must have name extension `.enc`, and the full file name including this extension must be given with preceding `<` character. 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. The *encodingfile* field may be omitted if you are sure that the font resource has the correct built–in encoding. In general this option is highly recommended, and it is *required* when subsetting a TrueType font.

fontfile

sets the name of the font file to be embedded into the pdf output for a given TEX font (the *tfm-name* ⟷ *fontfile* mapping is the most prominent use of the `pdftex.map` file). The font file name must belong to a Type 1 or TrueType font file. If the *fontfile* field is missing, no font embedding can take place; in case the *basename* field does not contain one of the 14 standard font names also a warning will be given. Not embedding a font into a pdf file might be troublesome, as it requires that the font or some similar looking replacement font is available within the pdf viewer, so that it can render the glyphs with its own font version.

The font file name should be preceded by one or two special characters, which tells how to handle the font file:

- If the font file name is preceded by a `<` character, the font file will be only partially embedded into the pdf file ("subsetted"), meaning that only used glyphs are going into the pdf file. This is the most common use and is *strongly recommended* for any font, as it ensures the portability and reduces the size of the pdf output. Subsetted fonts are included in such a way that name and cache clashes are minimized.
- If the font file name is preceded by a double `<<`, the font file will be included entirely — all glyphs of the font are embedded, including even the ones that are not used in the document. Apart from causing large size pdf output, this option may cause troubles with TrueType fonts, so it is normally not recommended for Type1 or TrueType fonts. But this is currently the only mode that allows the use of OpenType fonts. This mode might also be useful in case the font is atypical and can not be subsetted well by pdfTEX. *Beware: some font vendors forbid full font inclusion.*
- The case that no special character precedes the font file name is deprecated since pdfTEX version 1.40.0. These font files are now completely ignored, and a corresponding warning is given. You achieve exactly the same pdf result if you just remove the font file name from the map entry. Then the glyph widths that go into the pdf file are extracted from the tfm file, and a font descriptor object is created that contains approximations of the font metrics for the selected font.

  This option is useful only as fallback when you do not want to embed the font (e. g. due to font license restrictions), but wish to use the font metrics and let the pdf viewer generate instances that look close to the used font in case the font resource is not installed on the system where the pdf output will be viewed or printed. To use this feature, the font flags *must* be specified, and it must have the bit 6 set on, which means that only fonts with the Adobe Standard Roman Character Set can be simulated. The only exception is the case of a Symbolic font, which is not very useful.

When one suffers from invalid lookups, for instance when pdfTEX tries to open a `.pfa` file instead of a `.pfb` one, one can add the suffix to the filename. In this respect, pdfTEX completely relies on the `kpathsea` libraries.

If a used font is not present in the map files, first pdfTEX will look for a source with suffix `.pgc`, which is a so–called pgc source (pdf Glyph Container)[1]. If no pgc source is available, pdfTEX will try to use pk fonts as dvi drivers do, creating pk fonts on–the–fly if needed.

Lines containing nothing apart from *tfmname* stand for scalable Type 3 fonts. For scalable fonts as Type 1, TrueType and scalable Type 3 font, all the fonts loaded from a tfm at various sizes will be included only once in the pdf output. Thus if a font, let's say `csr10`, is described in one of the map files, then it will be treated as scalable. As a result the font source for csr10 will be included only once for `csr10`, `csr10 at 12pt` etc. So pdfTEX tries to do its best to avoid multiple embedding of identical font sources. Thus vector pgc fonts should be specified as scalable Type 3 in map files like:

```
csr10
```

It doesn't hurt much if a scalable Type 3 font is not given in map files, except that the font source will be embedded into the pdf file multiple times for various sizes, which causes a much larger pdf output. On the other hand if a font in the map files is defined as scalable Type 3 font and its pgc source is not scalable or not available, pdfTEX will use pk fonts instead; the pdf output is still valid but some fonts may look ugly because of the scaled bitmap.

To summarize this rather confusing story, we include some example lines. The most common way is to embed only a glyph subset from a font like this, with re–encoding:

```
ptmri8r Times-Italic <8r.enc <ptmri8a.pfb
```

Without re–encoding it looks like this:

```
cmr10 CMR10 <cmr10.pfb
```

A SlantFont is specified similarly as for `dvips`. The `SlantFont` or `ExtendFont` entries work only with embedded Type1 fonts:

```
psyro   StandardSymL ".167 SlantFont"         <usyr.pfb
pcrr8rn Courier      ".85 ExtendFont" <8r.enc <pcrr8a.pfb
```

Entirely embed a font into the pdf file without and with re–encoding:

```
fmvr8x MarVoSym         <<marvosym.pfb
pgsr8r GillSans <8r.enc <<pgsr8a.pfb
```

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

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

Now follow a few cases with non-embedded fonts. If the fontfile is missing, the viewer application will have to use its own approximation of the missing font (with and without re–encoding):

```
ptmr8r Times-Roman <8r.enc
psyr Symbol
```

[1] This is a text file containing a pdf Type 3 font, created by METAPOST using some utilities by Hans Hagen. In general pgc files can contain whatever is allowed in a pdf page description, which may be used to support fonts that are not available in METAFONT. pgc fonts are not widely useful, as vector Type 3 fonts are not displayed very well in older versions of Acrobat Reader, but may be more useful when better Type 3 font handling is more common.

In the next example the numerical font flags give some rough hint what general characteristics the GillSans font has, so e. g. the Adobe Reader might try an approximation, if it doesn't have the font resource nor any clue how a font named GillSans should look like:

```
pgsr8r GillSans 32 <8r.enc
```

Not embedding fonts is rather risky and should generally be avoided. If in doubt, always embed all fonts, even the 14 standard ones.

5.2 Helper tools for TrueType fonts

As mentioned above, pdfTeX can work with TrueType fonts. Defining TrueType fonts is similar to Type 1. The only extra thing to do with TrueType is to create a tfm file. There is a program called `ttf2afm` in the pdfTeX distribution which can be used to extract afm from TrueType fonts (another conversion program is `ttf2pt1`). Usage of `ttf2afm` is simple:

```
ttf2afm -e <encoding vector> -o <afm outputfile> <ttf input file>
```

A TrueType file can be recognized by its suffix `ttf`. The optional *encoding* specifies the encoding, which is the same as the 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. If we need to know which glyphs are available in the font, we can run `ttf2afm` without encoding to get all glyph names. The resulting afm file can be used to generate a tfm one by applying `afm2tfm`.

To use a new TrueType font the minimal steps may look like below. We suppose that `test.map` is used.

```
ttf2afm -e 8r.enc -o times.afm times.ttf
afm2tfm times.afm -T 8r.enc
echo "times TimesNewRomanPSMT <8r.enc <times.ttf" >>test.map
```

There are a few limitations with TrueType fonts in comparison with Type 1 fonts:

a. The special effects SlantFont/ExtendFont cannot be used. (Update: this is no longer true since version 1.40.0).
b. To subset a TrueType font, the font must be specified as re–encoded, therefore an encoding vector must be given.
c. TrueType fonts coming with embedded pdf files are kept untouched; they are not replaced by local ones.

5.3 A closer look at TrueType fonts and PDFTeX

The most common outline format for TeX is Type 1. The TrueType format is slightly different from Type 1, and getting it right requires some extra work. In particular, it is important to understand how TrueType handles encoding and glyph names (or more precisely, glyph identity).

We start with Type 1, since most TeX users are more familiar with it. In the Type 1 format glyphs are referred to by names (such as '/A', '/comma', and so on). Each glyph is identified by its name; so, given a glyph name, it is easy to tell whether or not a Type 1 font contains that glyph. Encoding with Type 1 is therefore simple: for each number n in the range 0 to 255, an encoding tells us the name of the glyph that should be used to render (or display) the charcode n.

With TrueType the situation is not that simple, since TrueType does not use names to refer to glyphs, but uses indices instead. This means that each glyph is identified by its index, not its name. The indices are numbers that differ from font to font. The TrueType format handles encodings by a mechanism called "cmap", which (roughly) consists of tables mapping from character codes to glyph indices. A TrueType font can contain one or more such tables (each corresponding to an encoding).

Since glyph names are not strictly necessary for TrueType, they are not always available inside a TrueType font. Given a TrueType font, one of the following cases may arise.

a. The font contains correct names for all its glyphs. This is the ideal situation and is often the case for high-quality latin fonts.
b. The font contains wrong name for all or most of its glyphs. This is the worst situation that often happens with poor-quality fonts, or fonts converted from other formats.
c. The font contains no glyph names at all. Newer versions of Palatino fonts by Linotype (v1.40, coming with Windows XP) are examples of this.
d. the font contains correct names for most glyphs, and no names or wrong names for a few glyphs. This happens from time to time.

One may wonder why things can be so complex with glyph names in TrueType. The reason is that Type 1 fonts rely on correct names to work properly. If a glyph has a wrong name, it gets noticed immediately. As mentioned before, TrueType does not use names for encoding. So, if glyph names in a TrueType font are wrong or missing, it is usually not a big deal and often goes unnoticed.

The potential problem with using TrueType in pdfTEX is that we are so used to the Type 1 encoding convention, which relies on correct glyph names. Furthermore, most font tools rely on this convention and all encoding files (.enc files) use glyph names. But, as explained above, glyph names in TrueType are not very reliable. If we encounter a font that does not have correct names for its glyphs, we need to do some more work.

If glyph names are not correct, we need a better way to refer to a glyph in TrueType fonts, rather than using names. The most reliable way seems to be via Unicode: most TrueType fonts provide a correct mapping from Unicode value to glyph index. This is something we can count on, since it is required for a TrueType font to be usable.

From version 1.21a pdfTEX supports the naming convention 'uniXXXX' in encoding (.enc) files. This only makes sense with TrueType fonts, of course. When pdfTEX sees for example '/uni12AB', it will

a. read the table <unicode> -> <glyph-index> from the font,
b. look up the value '12AB' in the table, and if found then pick the relevant glyph index.

ttf2afm also does the same lookup when it sees names like 'uni12AB'.

Now let us review the minimal steps to get a TrueType font working with pdfTEX:

a. generate an afm from TrueType using ttf2afm. Example:

```
ttf2afm -e 8r.enc -o times.afm times.ttf
```

b. convert afm to tfm using whatever tool suitable: afm2tfm, fontinst, afm2pl, etc. Example:

```
afm2tfm times.afm -T 8r.enc
```

c. create the needed map entry for the font. Example:

```
\pdfmapline{+times TimesNewRomanPSMT <8r.enc <times.ttf}
\font\f=times \f Hello this is Times.
```

The above deals with the easiest case: when glyph names are correct. Now let us consider a font where we cannot rely on glyph names: Palatino by Linotype version 1.40, for example. Let us assume that we want to use the T1 encoding with this font. So we put pala.ttf and ec.enc in the current directory before proceeding further.

The first attempt would be:

```
ttf2afm -e ec.enc -o pala.afm pala.ttf
```

However, since the names in ec.enc are not available in pala.ttf (in fact there are no names inside the font), we get a bunch of warnings:

```
Warning: ttf2afm (file pala.ttf): no names available in 'post' table, print
glyph names as indices
```

```
Warning: ttf2afm (file pala.ttf): glyph ‘grave’ not found
...
```

and the output pala.afm will contain no names at all. Instead of glyph names in ec.enc, we get weird things like ‘index123’. And glyphs are not encoded:

```
C -1 ; WX 832 ; N index10 ; B 24 -3 807 689 ;
...
```

We try again, this time without giving an encoding:

```
ttf2afm -o pala.afm pala.ttf
```

Since this time we did not ask ttf2afm to re-encode the output afm, we get fewer warnings:

```
Warning: ttf2afm (file pala.ttf): no names available in ‘post’ table, print
glyph names as indices
```

and the afm output is the same as in the previous attempt. This is not very useful, since there is little we can do with names like ‘index123’.

So we try to go with Unicode:

```
ttf2afm -u -o pala.afm pala.ttf
```

This time we get a different bunch of warnings, for instance:

```
Warning: ttf2afm (file pala.ttf): glyph 108 have multiple encodings (the
first one being used): uni0162 uni021A
```

At first sight it is hard to understand what tfm2afm is telling us with this message. So let us recap the connection between glyph name, glyph index and Unicode value:

a. TrueType glyphs are identified internally by index.
b. <glyph-name> -> <glyph-index> is optional, and not always reliable. Likewise <glyph-index> -> <glyph-name>.
c. <unicode> -> <glyph-index> is (almost) always present and reliable.
d. <glyph-index> -> <unicode> is not always reliable, and need not even be a mapping, since there can be more than one Unicode value mapping to a given glyph index. Given a glyph index, there may be no corresponding Unicode value, or there may be more than one. If there is none, the glyph index will be used (‘index123’, for example). Now suppose that there are more than one, as in the case above (where 0162 and 021A are both mapped to glyph index 108). We have asked ttf2afm to print glyphs by Unicode, and ttf2afm cannot know for sure which value to print in this case. Hence it outputs the first Unicode value and issues the warning.

Now if all we want is to use pala.ttf with T1 encoding, probably the easiest way is to create a new enc file ec-uni.enc from ec.enc, with all glyph names replaced by Unicode values. (This simple approach won’t handle ligatures; see below.) This can be done easily for example by a script that reads the AGL (Adobe Glyph List, available at http://www.adobe.com/devnet/opentype/archives/glyphlist.txt) and converts all glyph names to Unicode. Assuming that we have ec-uni.enc, the steps needed to create the tfm are as follows.

```
ttf2afm -u -e ec-uni.enc -o pala-t1.afm pala.ttf
afm2pl pala-t1.afm
pltotf pala-t1.pl
```

We could then use the font as follows.

```
\pdfmapline{+pala-t1 <ec-uni.enc <pala.ttf}
\font\f=pala-t1\f
This is a test of font Palatino Regular in T1 encoding.
```

If we want to do more than just using pala.ttf with T1 encoding, for example processing the afm output with fontinst for a more complex font setup, then we must proceed slightly differently. Having an afm file where all glyph names are converted to 'uniXXXX' form is not very useful for fontinst. Instead, we need an afm file with AGL names to use with fontinst. We can do this as follows.

a. Generate an afm with glyph names in form 'uniXXXX'.

```
ttf2afm -u -o pala.afm pala.ttf
```

b. Convert pala.afm to pala-agl.afm, so that pala-agl.afm contains AGL names only. Again, a simple script can do that.
c. Process pala-agl.afm by fontinst as needed.
d. In the final stage, when we already have the tfm's from fontinst and friends, plus the map entries (generated by fontinst, or created manually), we need to replace the encoding by its counterpart with 'uniXXXX' names. For example, if fontinst tell us to add a line saying

```
pala-agl-8r <8r.enc <pala.ttf
```

to our map file, then we need to change that line to

```
pala-agl-8r <8r-uni.enc <pala.ttf
```

where 8r-uni.enc is derived from 8r.enc by converting all glyph names to the 'uniXXXX' form.

The encoding files coming with TEX Gyre fonts cover almost everything a typical TEX user might need. Those encodings have been converted to the 'uniXXXX' form for your convenience and are available at http://tug.org/fontname and are named like 'q-ec-uni.enc' etc.

Another problem that happens from time to time is the case when we are totally sure that a glyph exists inside a font but we don't get that glyph in the output of pdfTEX. The likely reason of this problem is that the glyph is referenced by different names at various places during the process of creating support for the font, like tfm, vf, enc or map files. For example the names 'dcroat', 'dbar', 'dslash' and 'dmacron' can all refer to the same glyph in a TrueType font. The origin of a glyph name can come from several sources:

a. the name comes from the font itself.
b. the name comes from a predefined scheme called "the standard Macintosh ordering of glyphs". Unfortunately the TrueType specifications by various companies (Apple, Microsoft and Adobe) are not consistent in this scheme and there are small differences; one example is 'dmacron' vs 'dslash'.
c. the name comes out after conversion <unicode> -> <glyph-name> according to AGL.

In such situation, probably the easiest and safest way to get the glyph we want is to use a font editor like FontForge, look into the font to find out the Unicode for the glyph and then use the 'uniXXXX' form to instruct ttf2afm and pdfTEX to pick up that glyph.

Another way to get a problematic TrueType font to work with pdfTEX is simply to convert the font to Type 1 format using FontForge. While it sounds like a quick hack, it can be a simple and effective workaround.

6 Formal syntax specification

This section formally specifies the pdfTEX specific extensions to the TEX macro programming language. All primitives are prefixed by `pdf` except for `\efcode`, `\lpcode`, `\rpcode`, `\leftmarginkern`, and `\rightmarginkern`. The general definitions and syntax rules follow after the list of primitives.

Integer registers

`\pdfoutput` (integer)
`\pdfminorversion` (integer)
`\pdfcompresslevel` (integer)

\pdfobjcompresslevel (integer)
\pdfdecimaldigits (integer)
\pdfimageresolution (integer)
\pdfpkresolution (integer)
\pdftracingfonts (integer)
\pdfuniqueresname (integer)
\pdfadjustspacing (integer)
\pdfprotrudechars (integer)
\efcode ⟨font⟩ ⟨8-bit number⟩ (integer)
\lpcode ⟨font⟩ ⟨8-bit number⟩ (integer)
\rpcode ⟨font⟩ ⟨8-bit number⟩ (integer)
\pdfforcepagebox (integer)
\pdfoptionalwaysusepdfpagebox (integer)
\pdfinclusionerrorlevel (integer)
\pdfoptionpdfinclusionerrorlevel (integer)
\pdfimagehicolor (integer)
\pdfimageapplygamma (integer)
\pdfgamma (integer)
\pdfimagegamma (integer)
\pdfdraftmode (integer)

Dimen registers

\pdfhorigin (dimen)
\pdfvorigin (dimen)
\pdfpagewidth (dimen)
\pdfpageheight (dimen)
\pdflinkmargin (dimen)
\pdfdestmargin (dimen)
\pdfthreadmargin (dimen)
\pdfpxdimen (dimen)

Token registers

\pdfpagesattr (tokens)
\pdfpageattr (tokens)
\pdfpageresources (tokens)
\pdfpkmode (tokens)

Expandable commands

\pdftexrevision (expandable)
\pdftexbanner (expandable)
\pdfcreationdate (expandable)
\pdfpageref ⟨page number⟩ (expandable)
\pdfxformname ⟨object number⟩ (expandable)
\pdffontname ⟨font⟩ (expandable)
\pdffontobjnum ⟨font⟩ (expandable)
\pdffontsize ⟨font⟩ (expandable)
\pdfincludechars ⟨font⟩ ⟨general text⟩ (expandable)
\leftmarginkern ⟨box number⟩ (expandable)

\rightmarginkern ⟨box number⟩ (expandable)
\pdfescapestring ⟨general text⟩ (expandable)
\pdfescapename ⟨general text⟩ (expandable)
\pdfescapehex ⟨general text⟩ (expandable)
\pdfunescapehex ⟨general text⟩ (expandable)
\ifpdfprimitive ⟨control sequence⟩ (expandable)
\ifpdfabsnum ⟨number⟩ (expandable)
\ifpdfabsdim ⟨dimen⟩ (expandable)
\pdfuniformdeviate ⟨number⟩ (expandable)
\pdfnormaldeviate (expandable)
\pdfmdfivesum [file] ⟨general text⟩ (expandable)
\pdffilemoddate ⟨general text⟩ (expandable)
\pdffilesize ⟨general text⟩ (expandable)
\pdffiledump [offset ⟨number⟩] [length ⟨number⟩] ⟨general text⟩ (expandable)
\pdfcolorstackinit [page] [direct] ⟨general text⟩ (expandable)

Read–only integers

\pdftexversion (read–only integer)
\pdflastobj (read–only integer)
\pdflastxform (read–only integer)
\pdflastximage (read–only integer)
\pdflastximagepages (read–only integer)
\pdflastannot (read–only integer)
\pdflastlink (read–only integer)
\pdflastxpos (read–only integer)
\pdflastypos (read–only integer)
\pdflastdemerits (read–only integer)
\pdfelapsedtime (read–only integer)
\pdfrandomseed (read–only integer)
\pdfshellescape (read–only integer)

General commands

\pdfobj ⟨object type spec⟩ (h, v, m)
\pdfrefobj ⟨object number⟩ (h, v, m)
\pdfxform [⟨xform attr spec⟩] ⟨box number⟩ (h, v, m)
\pdfrefxform ⟨object number⟩ (h, v, m)
\pdfximage [⟨image attr spec⟩] ⟨general text⟩ (h, v, m)
\pdfrefximage ⟨object number⟩ (h, v, m)
\pdfannot ⟨annot type spec⟩ (h, v, m)
\pdfstartlink [⟨rule spec⟩] [⟨attr spec⟩] ⟨action spec⟩ (h, m)
\pdfendlink (h, m)
\pdfoutline ⟨outline spec⟩ (h, v, m)
\pdfdest ⟨dest spec⟩ (h, v, m)
\pdfthread ⟨thread spec⟩ (h, v, m)
\pdfstartthread ⟨thread spec⟩ (v, m)
\pdfendthread (v, m)
\pdfsavepos (h, v, m)
\pdfinfo ⟨general text⟩
\pdfcatalog ⟨general text⟩ [⟨open-action spec⟩]

\pdfnames ⟨general text⟩
\pdfmapfile ⟨map spec⟩
\pdfmapline ⟨map spec⟩
\pdffontattr ⟨font⟩ ⟨general text⟩
\pdftrailer ⟨general text⟩
\pdffontexpand ⟨font⟩ ⟨expand spec⟩
\vadjust [⟨pre spec⟩] ⟨filler⟩ { ⟨vertical mode material⟩ } (h, m)
\quitvmode
\pdfliteral [⟨pdfliteral spec⟩] ⟨general text⟩ (h, v, m)
\special ⟨pdfspecial spec⟩
\pdfresettimer
\pdfsetrandomseed ⟨number⟩
\pdfnoligatures ⟨font⟩
\pdfprimitive ⟨control sequence⟩
\pdfcolorstack ⟨stack number⟩ ⟨stack action⟩ ⟨general text⟩
\pdfsetmatrix
\pdfsave
\pdfrestore

General definitions and syntax rules

⟨general text⟩ → { ⟨balanced text⟩ }
⟨attr spec⟩ → attr ⟨general text⟩
⟨resources spec⟩ → resources ⟨general text⟩
⟨rule spec⟩ → (width | height | depth) ⟨dimension⟩ [⟨rule spec⟩]
⟨object type spec⟩ → reserveobjnum |
    [useobjnum ⟨number⟩]
    [stream [⟨attr spec⟩]] ⟨object contents⟩
⟨annot type spec⟩ → reserveobjnum |
    [useobjnum ⟨number⟩] [⟨rule spec⟩] ⟨general text⟩
⟨object contents⟩ → ⟨file spec⟩ | ⟨general text⟩
⟨xform attr spec⟩ → [⟨attr spec⟩] [⟨resources spec⟩]
⟨image attr spec⟩ → [⟨rule spec⟩] [⟨attr spec⟩] [⟨page spec⟩] [⟨colorspace spec⟩] [⟨pdf box spec⟩]
⟨outline spec⟩ → [⟨attr spec⟩] ⟨action spec⟩ [count ⟨number⟩] ⟨general text⟩
⟨action spec⟩ → user ⟨user-action spec⟩ | goto ⟨goto-action spec⟩ |
    thread ⟨thread-action spec⟩
⟨user-action spec⟩ → ⟨general text⟩
⟨goto-action spec⟩ → ⟨numid⟩ |
    [⟨file spec⟩] ⟨nameid⟩ |
    [⟨file spec⟩] [⟨page spec⟩] ⟨general text⟩ |
    ⟨file spec⟩ ⟨nameid⟩ ⟨newwindow spec⟩ |
    ⟨file spec⟩ [⟨page spec⟩] ⟨general text⟩ ⟨newwindow spec⟩
⟨thread-action spec⟩ → [⟨file spec⟩] ⟨numid⟩ | [⟨file spec⟩] ⟨nameid⟩
⟨open-action spec⟩ → openaction ⟨action spec⟩
⟨colorspace spec⟩ → colorspace ⟨number⟩
⟨pdf box spec⟩ → mediabox | cropbox | bleedbox | trimbox | artbox
⟨map spec⟩ → { [⟨map modifier⟩] ⟨balanced text⟩ }
⟨map modifier⟩ → + | = | -
⟨numid⟩ → num ⟨number⟩
⟨nameid⟩ → name ⟨general text⟩
⟨newwindow spec⟩ → newwindow | nonewwindow

⟨dest spec⟩ → ⟨numid⟩ ⟨dest type⟩ | ⟨nameid⟩ ⟨dest type⟩
⟨dest type⟩ → `xyz` [`zoom` ⟨number⟩] | `fitr` ⟨rule spec⟩ |
    `fitbh` | `fitbv` | `fitb` | `fith` | `fitv` | `fit`
⟨thread spec⟩ → [⟨rule spec⟩] [⟨attr spec⟩] ⟨id spec⟩
⟨id spec⟩ → ⟨numid⟩ | ⟨nameid⟩
⟨file spec⟩ → `file` ⟨general text⟩
⟨page spec⟩ → `page` ⟨number⟩
⟨expand spec⟩ → ⟨stretch⟩ ⟨shrink⟩ ⟨step⟩ [`autoexpand`]
⟨stretch⟩ → ⟨number⟩
⟨shrink⟩ → ⟨number⟩
⟨step⟩ → ⟨number⟩
⟨pre spec⟩ → `pre`
⟨pdfliteral spec⟩ → `direct` | `page`
⟨pdfspecial spec⟩ → `{` [⟨pdfspecial id⟩ [⟨pdfspecial modifier⟩]] ⟨balanced text⟩ `}`
⟨pdfspecial id⟩ → `pdf:` | `PDF:`
⟨pdfspecial modifier⟩ → `direct:`
⟨stack action⟩ → `set` | `push` | `pop` | `current`

A ⟨general text⟩ is expanded immediately, like `\special` in traditional TEX, unless explicitly mentioned otherwise.

Some of the object and image related primitives can be prefixed by `\immediate`. More about that in the next sections.

7 PDFTEX primitives

Here follows a short description of the primitives added by pdfTEX to the original TEX engine (other extensions by MLTEX and encTEX are not listed). One way to learn more about how to use these new primitives is to have a look at the file `samplepdf.tex` in the pdfTEX distribution.

Note that if the output is dvi then the pdfTEX specific dimension parameters are not used at all. However some pdfTEX integer parameters can affect the dvi as well as pdf output (currently `\pdfoutput` and `\pdfadjustspacing`).

General warning: many of these new primitives, for example `\pdfdest` and `\pdfoutline`, write their arguments directly to the pdf output file (when producing pdf), as pdf string constants. This means that *you* (or, more likely, the macros you write) must escape characters as necessary (namely `\`, `(`, and `)`. Otherwise, an invalid pdf file may result. The `hyperref` and Texinfo packages have code which may serve as a starting point for implementing this, although it will certainly need to be adapted to any particular situation.

7.1 Document setup

► `\pdfoutput` (integer)

This parameter specifies whether the output format should be dvi or pdf. A positive value means pdf output, otherwise (default 0) one gets dvi output. This primitive is the only one that must be set to produce pdf output (unless the commandline option `-output-format=pdf` is used); all other primitives are optional. This parameter cannot be specified *after* shipping out the first page. In other words, if we want pdf output, we have to set `\pdfoutput` before pdfTEX ships out the first page.

When pdfTEX starts complaining about specials, one can be rather sure that a macro package is not aware of the pdf mode. A simple way of making macros aware of pdfTEX in pdf or dvi mode is:

```
\ifx\pdfoutput\undefined \csname newcount\endcsname\pdfoutput \fi
\ifcase\pdfoutput DVI CODE \else PDF CODE \fi
```

Using the `ifpdf.sty` file, which works with both LATEX and plain TEX, is a cleaner way of doing this. Historically, the simple test `\ifx\pdfoutput\undefined` was defined; but nowadays, the pdfTEX engine is used in distributions also for non-pdf formats (e. g. LATEX), so `\pdfoutput` may be defined even when the output format is dvi.

► `\pdfminorversion` (integer)

This primitive sets the pdf version of the generated file and the latest allowed pdf version of included pdfs. E. g., `\pdfminorversion=3` tells pdfTEX to set the pdf version to 1.3 and allows only included pdf files with versions numbers up to 1.3. The default for `\pdfminorversion` is 4, producing files with pdf version 1.4. If specified, this primitive must appear before any data is to be written to the generated pdf file, so you should put it at the very start of your files. The command has been introduced in pdfTEX 1.30.0 as a shortened synonym of `\pdfoptionpdfminorversion` command, that is obsolete by now.

► `\pdfcompresslevel` (integer)

This integer parameter specifies the level of *stream* compression (text, in–line graphics, and embedded png images (only if they are un- and re–compressed during the embedding process); all done by the `zlib` library). Zero means no compression, 1 means fastest, 9 means best, 2..8 means something in between. A value outside this range will be adjusted to the nearest meaningful value. This parameter is read each time pdfTEX starts a stream. Setting `\pdfcompresslevel=0` is great for pdf stream debugging.

► `\pdfobjcompresslevel` (integer)

This integer parameter controls the compression of *non-stream* objects. In the pdf-1.4 specification these objects still had to go into the pdf file as clear text, uncompressed. The pdf-1.5 specification now allows to collect non-stream objects as "compressed objects" into "object stream" objects (`/Type /ObjStm`, see pdf Ref. 5th ed., sect. 3.4.6). At the pdf file end instead of the object table then an `/XRef` cross-reference stream is written out. This results in considerably smaller pdf files, particularly if lots of annotations and links are used. The primitive was introduced in pdfTEX 1.40.0.

The writing of compressed objects is enabled by setting `\pdfobjcompresslevel` to a value between 1 and 3; it's disabled by value 0 (default). Enabling requires that also `\pdfminorversion` > 4. If `\pdfobjcompresslevel` > 0, but `\pdfminorversion` < 5, a warning is given and object stream writing is disabled. The `\pdfobjcompresslevel` value is clipped to the range 0..3. Using values outside this range is not recommended (for future extension).

The `\pdfobjcompresslevel` settings have the following effects: When set to 0, no object streams are generated at all. When set to 1, all non-stream objects are compressed with the exception of any objects coming with embedded pdf files ("paranoid" mode, to avoid yet unknown problems), and also the `/Info` dictionary is not compressed for clear-text legibility. When set to 2, also all non-stream objects coming with embedded pdf files are compressed, but the `/Info` dictionary is still not compressed. Finally, when set to 3, all non-stream objects are compressed, including the `/Info` dictionary (this means that the `/Info` can't be read as clear text any more). If object streams are to be used, currently `\pdfobjcompresslevel=2` is recommended,

Caveat: pdf files generated with object streams enabled can't be read with (old) pdf viewers that don't understand pdf-1.5 files. For widest distribution and unknown audience, don't activate object stream writing. The pdf-1.5 standard describes also a hybrid object compression mode that gives some backward compatibility, but this is currently not implemented, as pdf-1.5 seems to be rather quickly adopted by modern pdf viewers. Also not implemented is the optional `/Extends` key.

► `\pdfdecimaldigits` (integer)

This integer parameter specifies the numeric accuracy of real coordinates as written to the pdf file. It gives the maximal number of decimal digits after the decimal point. Valid values are in range 0..4. A higher value means more precise output, but also results in a larger file size and more time to display or print. In most cases the optimal value is 2. This parameter does not influence the precision of numbers used in raw pdf code, like that used in `\pdfliteral` and annotation action specifications; also multiplication items (e. g. scaling factors) are not affected and are always output with best precision. This parameter is read when pdfTEX writes a real number to the pdf output.

When including huge METAPOST images using `supp-pdf.tex`, one can limit the accuracy to two digits by typing: `\twodigitMPoutput`.

- `\pdfhorigin` (dimension)

This parameter can be used to set the horizontal offset the output box from the top left corner of the page. A value of 1 inch corresponds to the normal TEX offset. This parameter is read when pdfTEX starts shipping out a page to the pdf output.

For standard purposes, this parameter should always be kept at 1 true inch. If you want to shift text on the page, use TEX's own `\hoffset` primitive. To avoid surprises, after global magnification has been changed by the `\mag` primitive, the `\pdfhorigin` parameter should still be 1 true inch, e. g. by typing `\pdfhorigin=1 true in` after issuing the `\mag` command. Or, you can preadjust the `\pdfhorigin` value before typing `\mag`, so that its value after the `\mag` command ends up at 1 true inch again.

- `\pdfvorigin` (dimension)

This parameter is the vertical companion of `\pdfhorigin`, and the notes above regarding `\mag` and true dimensions apply. Also keep in mind that the TEX coordinate system starts in the top left corner (downward), while pdf coordinates start at the bottom left corner (upward).

- `\pdfpagewidth` (dimension)

This dimension parameter specifies the page width of the pdf output (the screen, the paper, etc.). pdfTEX reads this parameter when it starts shipping out a page. After magnification has been changed by the `\mag` primitive, check that this parameter reflects the wished true page width.

If the value is set to zero, the page width is calculated as $w_{\text{box being shipped out}} + 2 \times (\text{horigin} + \texttt{\textbackslash hoffset})$. When part of the page falls off the paper or screen, you can be rather sure that this parameter is set wrong.

- `\pdfpageheight` (dimension)

Similar to the previous item, this dimension parameter specifies the page height of the pdf output. If set to zero, the page height will be calculated analogously to the above. After magnification has been changed by the `\mag` primitive, check that this parameter reflects the wished true page height.

7.2 The document info and catalog

- `\pdfinfo` ⟨general text⟩

This primitive allows the user to add information to the document info section; if this information is provided, it can be extracted, e. g. by the pdfinfo program, or by the Adobe Reader (version 7.0: menu option *File* ⟶ *Document Properties*). The ⟨general text⟩ is a collection of key–value–pairs. The key names are preceded by a `/`, and the values, being strings, are given between parentheses. All keys are optional. Possible keys are `/Author`, `/CreationDate` (defaults to current date including time zone info), `/ModDate`, `/Creator` (defaults to `TeX`), `/Producer` (defaults to `pdfTeX-1.40.0`), `/Title`, `/Subject`, and `/Keywords`.

`/CreationDate` and `/ModDate` are expressed in the form `D:YYYYMMDDhhmmssTZ..`, where `YYYY` is the year, `MM` is the month, `DD` is the day, hh is the hour, `mm` is the minutes, `ss` is the seconds, and `TZ..` is an optional string denoting the time zone. An example of this format is shown below. For details please refer to the PDF Reference.

Multiple appearances of `\pdfinfo` will be concatenated. In general, if a key is given more than once, one may expect that the first appearance will be used. Be aware however, that this behaviour is viewer dependent. Except expansion, pdfTEX does not perform any further operations on ⟨general text⟩ provided by the user.

An example of the use of `\pdfinfo` is:

```
\pdfinfo {
    /Title          (example.pdf)
    /Creator        (TeX)
    /Producer       (pdfTeX 1.40.0)
    /Author         (Tom and Jerry)
    /CreationDate (D:20061226154343+01'00')
    /ModDate        (D:20061226155343+01'00')
    /Subject        (Example)
    /Keywords       (mouse, cat) }
```

► `\pdfcatalog` ⟨general text⟩ [⟨open-action spec⟩]

Similar to the document info section is the document catalog, where keys are `/URI`, which provides the base url of the document, and `/PageMode`, which determines how the pdf viewer displays the document on startup. The possibilities for the latter are explained in Table 4:

value	**meaning**
`/UseNone`	neither outline nor thumbnails visible
`/UseOutlines`	outline visible
`/UseThumbs`	thumbnails visible
`/FullScreen`	full–screen mode

Table 4 Supported `/PageMode` values.

In full–screen mode, there is no menu bar, window controls, nor any other window present. The default setting is `/UseNone`.

The ⟨openaction⟩ is the action provided when opening the document and is specified in the same way as internal links, see section 7.9. Instead of using this method, one can also write the open action directly into the catalog.

► `\pdfnames` ⟨general text⟩

This primitive inserts the ⟨general text⟩ to the `/Names` array. The text must conform to the specifications as laid down in the PDF Reference, otherwise the document can be invalid.

► `\pdftrailer` ⟨general text⟩

This command puts its argument text verbatim into the file trailer dictionary. The primitive was introduced in pdfTEX 1.11a.

7.3 Fonts

► `\pdfpkresolution` (integer)

This integer parameter specifies the default resolution of embedded pk fonts and is read when pdfTEX embeds a pk font during finishing the pdf output. As bitmap fonts are still rendered poorly by some pdf viewers, it is best to use Type 1 fonts when available.

► `\pdffontexpand` ⟨font⟩ ⟨stretch⟩ ⟨shrink⟩ ⟨step⟩ [`autoexpand`]

This extension to TEX's font definitions controls a pdfTEX automatism called *font expansion*. We describe this by an example:

```
\font\somefont=sometfm at 10pt
\pdffontexpand\somefont 30 20 10 autoexpand
\pdfadjustspacing=2
```

The `30 20 10` means this: "hey TEX, when line breaking is going badly, you may stretch the glyphs from this font as much as 3 % or shrink them as much as 2 %". For practical reasons pdfTEX uses discrete expansion steps, in this example, 1 %. Roughly spoken, the trick is as follows. Consider a text typeset in triple column mode. When TEX cannot break a line in the appropriate way, the unbreakable parts of the word will stick into the margin. When pdfTEX notes this, it will try to scale (shrink) the glyphs in that line using fixed steps, until the line fits. When lines are too spacy, the opposite happens: pdfTEX starts scaling (stretching) the glyphs until the white space gaps is acceptable. This glyph stretching and shrinking is called *font expansion*. To enable font expansion, don't forget to set `\pdfadjustspacing` to a value greater than zero.

There are two different modes for font expansion:

First, if the `autoexpand` option is there — which is the recommended mode — only a single map entry is needed for all expanded font versions, using the name of the unexpanded tfm file (*tfmname*). No expanded *tfmname* versions need to mentioned (they are ignored), as pdfTEX generates expanded copies of the unexpanded tfm data structures and keeps them in its memory. Since pdfTEX 1.40.0 the `autoexpand` mode happens within the page stream by modification of the text matrix (pdf operator "`Tm`"), and not anymore on font file level, giving the advantage that it now works not only with Type1 but also with TrueType and OpenType fonts (and even without embedding a font file; but that's not recommended). In this mode pdfTEX requires only unexpanded font files.

Second, if the `autoexpand` option is missing, setting up font expansion gets more tedious, as there must be map entries for tfm files in all required expansion values. The expanded *tfmname* variants are constructed by adding the font expansion value to the *tfmname* of the base font, e. g. there must be a map entry with *tfmname* `sometfm+10` for 1 % stretch or `sometfm-15` for 1.5 % shrink. This also means, that for each expanded font variant a tfm file with properly expanded metrics must exist. Having several map entries for the various expansion values of a font requires to provide for each expansion value an individually crafted font file with expanded glyphs. Depending on how these glyphs are generated, this might give slightly better glyph forms than the rather simple glyph stretching used in `autoexpand mode`. The drawback is that several font files will be embedded in the pdf output for each expanded font, leading to significantly larger pdf files than in `autoexpand` mode. For moderate expansion values going without `autoexpand` mode is not worth the trouble.

The font expansion mechanism is inspired by an optimization first introduced by Prof. Hermann Zapf, which in itself goes back to optimizations used in the early days of typesetting: use different glyphs to optimize the grayness of a page. So, there are many, slightly different a's, e's, etc. For practical reasons pdfTEX does not use such huge glyph collections; it uses horizontal scaling instead. This is sub–optimal, and for many fonts, possibly offensive to the design. But, when using pdf, it's not illogical: pdf viewers use so–called Multiple Master fonts when no fonts are embedded and/or can be found on the target system. Such fonts are designed to adapt their design to the different scaling parameters. It is up to the user to determine to what extent mixing slightly remastered fonts can be used without violating the design. Think of an O: when geometrically stretched, the vertical part of the glyph becomes thicker, and looks incompatible with an unscaled original. With a Multiple Master situation, one can stretch while keeping this thickness compatible.

► `\pdfadjustspacing` (integer)

This primitive provides a switch for enabling font expansion. By default, `\pdfadjustspacing` is set to 0; then font expansion is disabled, so that the pdfTEX output is identical to that from the original TEX engine.

Font expansion can be activated in two modes. When `\pdfadjustspacing` is set to 1, font expansion is applied *after* TEX's normal paragraph breaking routines have broken the paragraph into lines. In this case, line breaks are identical to standard TEX behaviour.

When set to 2, the width changes that are the result of stretching and shrinking are taken into account *while* the paragraph is broken into lines. In this case, line breaks are likely to be different from those of standard TEX. In fact, paragraphs may even become longer or shorter.

Both alternatives require a collection of tfm files that are related to the ⟨stretch⟩ and ⟨shrink⟩ settings for the `\pdffontexpand` primitive, unless this is given with the `autoexpand` option.

► `\efcode` ⟨font⟩ (integer)

We didn't yet tell the whole story. One can imagine that some glyphs are visually more sensitive to stretching or shrinking than others. Then the `\efcode` primitive can be used to influence the expandability of individual glyphs within a given font, as a factor to the expansion setting from the `\pdffontexpand` primitive. The syntax is similar to `\sfcode` (but with the ⟨font⟩ required), and it defaults to 1000, meaning 100 % expandability. The given integer value is clipped to the range 0..1000, corresponding to a usable expandability range of 0..100 %. Example:

```
\efcode\somefont`A=800
\efcode\somefont`O=0
```

Here an A may stretch or shrink only by 80 % of the current expansion value for that font, and expansion for the O is disabled. The actual expansion is still bound to the steps as defined by `\pdffontexpand` primitive, otherwise one would end up with more possible font inclusions than would be comfortable.

► `\pdfprotrudechars` (integer)

Yet another way of optimizing paragraph breaking is to let certain characters move into the margin ('character protrusion'). When `\pdfprotrudechars=1`, the glyphs qualified as such will make this move when applicable, without changing the line-breaking. When `\pdfprotrudechars=2` (or greater), character protrusion will be taken into account while considering breakpoints, so line-breaking might be changed. This qualification and the amount of shift are set by the primitives `\rpcode` and `\lpcode`. Character protrusion is disabled when `\pdfprotrudechars=0` (or negative).

If you want to protrude some item other than a character (e. g. a `\hbox`), you can do so by padding the item with an invisible zero–width character, for which protrusion is activated.

► `\rpcode` ⟨font⟩ (integer)

The amount that a character from a given font may shift into the right margin ('character protrusion') is set by the primitive `\rpcode`. The protrusion distance is the integer value given to `\rpcode`, multiplied with 0.001 em from the current font. The given integer value is clipped to the range −1000..1000, corresponding to a usable protrusion range of −1 em..1 em. Example:

```
\rpcode\somefont`,=200
\rpcode\somefont`-=150
```

Here the comma may shift by 0.2 em into the margin and the hyphen by 0.15 em. All these small bits and pieces will help pdfTEX to give you better paragraphs (use `\rpcode` judiciously; don't overdo it).

Remark: old versions of pdfTEX use the character width as measure. This was changed to a proportion of the em-width after Hàn Thế Thành finished his master's thesis.

► `\lpcode` ⟨font⟩ (integer)

This is similar to `\rpcode`, but affects the amount by which characters may protrude into the left margin. Also here the given integer value is clipped to the range −1000..1000.

► `\leftmarginkern` ⟨box number⟩ (expandable)

The `\leftmarginkern` ⟨box number⟩ primitive expands to the width of the margin kern at the left side of the horizontal list stored in `\box` ⟨box number⟩ . The expansion string includes the unit `pt`. E. g., when the left margin kern of `\box0` amounts to −10 pt, `\leftmarginkern0` will expand to `-10pt`. A similar primitive `\rightmarginkern` exists for the right margin. The primitive was introduced in pdfTEX 1.30.0.

These are auxiliary primitives to make character protrusion more versatile. When using the TEX primitive `\unhbox` or `\unhcopy`, the margin kerns at either end of the unpackaged hbox will be removed (e. g. to avoid weird effects if several hboxes are unpackaged behind each other into the same horizontal list). These `\unhbox` or `\unhcopy` are often used together with `\vsplit` for dis- and re–assembling of paragraphs, e. g. to add line numbers. Paragraphs treated like this do not show character protrusion by default, as the margin kerns have been removed during the unpackaging process.

The `\leftmarginkern` and `\rightmarginkern` primitives allow to access the margin kerns and store them away before unpackaging the hbox. E. g. the following code snipplet restores margin kerning of a horizontal list stored in `\box\testline`, resulting in a hbox with proper margin kerning (which is then done by ordinary kerns).

```
\dimen0=\leftmarginkern\testline
\dimen1=\rightmarginkern\testline
\hbox to\hsize{\kern\dimen0\unhcopy\testline\kern\dimen1}
```

► `\rightmarginkern` ⟨box number⟩ (expandable)

The `\rightmarginkern` ⟨box number⟩ primitive expands to the width of the margin kern at the right side of the horizontal list stored in `\box` ⟨box number⟩ . See `\leftmarginkern` for more details. The primitive was introduced in pdfTEX 1.30.0.

► `\pdffontattr` ⟨font⟩ ⟨general text⟩

This primitive inserts the ⟨general text⟩ to the `/Font` dictionary. The text must conform to the specifications as laid down in the PDF Reference, otherwise the document can be invalid.

► `\pdffontname` ⟨font⟩ (expandable)

In pdf files produced by pdfTEX one can recognize a font resource by the prefix `/F` followed by a number, for instance `/F12` or `/F54`. For a given TEX ⟨font⟩ , this primitive expands to the number from the corresponding font resource name. E. g., if `/F12` corresponds to some TEX font `\foo`, the `\pdffontname\foo` expands to the number `12`.

In the current implementation, when `\pdfuniqueresname` (see below) is set to a positive value, the `\pdffontname` still returns only the number from the font resource name, but not the appended random string.

► `\pdffontobjnum` ⟨font⟩ (expandable)

This command is similar to `\pdffontname`, but it returns the pdf object number of the font dictionary instead of the number from the font resource name. E. g., if the font dictionary (`/Type /Font`) in pdf object 3 corresponds to some TEX font `\foo`, the `\pdffontobjnum\foo` gives the number `3`.

Use of `\pdffontname` and `\pdffontobjnum` allows users full access to all the font resources used in the document.

► `\pdffontsize` ⟨font⟩ (expandable)

This primitive expands to the font size of the given font, with unit `pt`. E. g., when using the plain TEX macro package, the call `\pdffontsize\tenrm` expands to `10.0pt`.

► `\pdfincludechars` ⟨font⟩ ⟨general text⟩

This command causes pdfTEX to treat the characters in ⟨general text⟩ as if they were used with ⟨font⟩, which means that the corresponding glyphs will be embedded into the font resources in the pdf output. Nothing is appended to the list being built.

- `\pdfuniqueresname` (integer)

  When this primitive is assigned a positive number, pdf resource names will be made reasonably unique by appending a random string consisting of six ascii characters.

- `\pdfmapfile` ⟨map spec⟩

  This primitive is used for reading a font map file consisting of one or more font map lines. The name of the map file is given in the ⟨map spec⟩ together with an optional leading modifier character. If no `\pdfmapfile` primitive is given, the default map file `pdftex.map` will be read by pdfTeX. There is a companion primitive `\pdfmapline` that allows to scan single map lines; its map line argument has the same syntax as the map lines from a map file. Both primitives can be used concurrently. The `\pdfmapfile` primitive is fast for reading external bulk font map information (many map lines collected in a map file), whereas the `\pdfmapline` allows to put the font map information for individual TeX fonts right into the TeX source or a style file. In any case the map line information is scanned by pdfTeX, and in the most common case the data are put into a fresh internal map entry data structure, which is then consulted once a font is called.

  Normally there is no need for the pdfTeX user to bother about the `\pdfmapfile` or `\pdfmapline` primitives, as the main TeX distributions provide nice helper tools that automatically assemble the default font map file. Prominent tool examples are the scripts `updmap` and `updmap-sys` coming with TeX Live and teTeX. If your map file isn't in the current directory (or a standard system directory), you will need to set the `TEXFONTMAPS` variable (in Web2c) or give an explicit path so that it will be found.

  When the `\pdfmapfile` or `\pdfmapline` primitive is read by pdfTeX, the argument (map file or map line) will be processed *immediately*, and the internal map entry database is updated. The operation mode of the `\pdfmapfile` and `\pdfmapline` primitives is selected by an optional modifier character (`+`, `=`, `-`) in front of the *tfmname* field. This modifier defines how the individual map lines are going to be handled, and how a collision between an already registered map entry and a newer one is resolved; either ignoring a later entry, or replacing or deleting an existing entry. But in any case, map entries of fonts already in use are kept untouched. Here are two examples:

  ```
  \pdfmapfile{+myfont.map}
  \pdfmapline{+ptmri8r Times-Italic <8r.enc <ptmri8a.pfb}
  ```

  When no modifier character is given (e. g. `\pdfmapfile{foo.map}` or `\pdfmapline{phvr8r Helvetica}`) and there hasn't already been any call of one of these primitives before, then the default map file `pdftex.map` will *not* be read in. Apart from this the given map file will be processed similarly as with a `+` modifier: duplicate later map entries within the file are ignored and a warning is issued. This means, that you can block reading of the default map file also by an empty `\pdfmapfile{}` or `\pdfmapline{}` early in the TeX file. When the default map file is large but you don't need it anyway, these command variants might considerably speed up the pdfTeX startup process.

  If a modifier is given, the mechanism is so that before reading the items given as arguments to `\pdfmapfile` or `\pdfmapline` the default map file will be read first — if this hasn't already been done or been prevented by the above blocking cases. This should be mostly compatible with the traditional behaviour. If you want to add support for a new font through an additional font map file while keeping all the existing mappings, don't use the primitive versions without modifier, but instead type either `\pdfmapfile{+myfont.map}` or `\pdfmapfile{=myfont.map}`, as described below.

  `\pdfmapfile {+foo.map}` reads the file `foo.map`; duplicate later map entries within the file are ignored and a warning is issued.

  `\pdfmapfile {=foo.map}` reads the file `foo.map`; matching map entries in the database are replaced by new entries from `foo.map` (if they aren't already in use).

  `\pdfmapfile {-foo.map}` reads the file `foo.map`; matching map entries are deleted from the database (if not yet in use).

If you want to use a base map file name other than `pdftex.map`, or change its processing options through a pdfTEX format, you can do this by appending the `\pdfmapfile` command to the `\everyjob{}` token list for the `-ini` run, e. g.:

```
\everyjob\expandafter{\the\everyjob\pdfmapfile{+myspecial.map}}
\dump
```

This would always read the file `myspecial.map` after the default `pdftex.map` file.

► `\pdfmapline` ⟨map spec⟩

Similar to `\pdfmapfile`, but here you can give a single map line (like the ones in map files) as an argument. The optional modifiers (`+-=`) have the same effect as with `\pdfmapfile`; see also the description above. Example:

```
\pdfmapline{+ptmri8r Times-Italic <8r.enc <ptmri8a.pfb}
```

This primitive (especially the `\pdfmapline{=...}` variant) is useful for temporary quick checks of a new font map entry during development, before finally putting it into a map file.

`\pdfmapline {}` like `\pdfmapfile {}` blocks reading of the default map file, if it comes early enough in the TEX input. The primitive was introduced in pdfTEX 1.20a.

► `\pdftracingfonts` (integer)

This integer parameter specifies the level of verbosity for info about expanded fonts given in the log, e. g. when `\tracingoutput=1`. If `\pdftracingfonts=0`, which is the default, the log shows the actual non-zero signed expansion value for each expanded letter within brackets, e. g.:

```
...\xivtt (+20) t
```

If `\pdftracingfonts=1`, also the name of the tfm file is listed, together with the font size, e. g.:

```
...\xivtt (cmtt10+20@14.0pt) t
```

Setting `\pdftracingfonts` to a value other than 0 or 1 is not recommended, to allow future extensions. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfmovechars` (integer)

Since pdfTEX version 1.30.0 the primitive `\pdfmovechars` is obsolete, and its use merely leads to a warning. (This primitive specified whether pdfTEX should try to move characters in range 0..31 to higher slots; its sole purpose was to remedy certain bugs of early pdf viewers.)

► `\pdfpkmode` (tokens)

The `\pdfpkmode` is a token register that sets the METAFONT mode for pixel font generation. The contents of this register is dumped into the format, so one can (optionally) preset it e. g. in `pdftexconfig.tex`. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfnoligatures` ⟨font⟩

This diables all ligatures in the loaded font ⟨font⟩ . The primitive was introduced in pdfTEX 1.30.0.

7.4 PDF objects

► `\pdfobj` ⟨object type spec⟩

This command creates a raw pdf object that is written to the pdf file as `1 0 obj ... endobj`. The object is written to pdf output as provided by the user. When ⟨object type spec⟩ is not given, pdfTEX does not any longer create a dictionary object with contents ⟨general text⟩ , as it was in the past.

When however ⟨object type spec⟩ is given as ⟨attr spec⟩ `stream` , the object will be created as a stream with contents ⟨general text⟩ and additional attributes in ⟨attr spec⟩.

When ⟨object type spec⟩ is given as ⟨attr spec⟩ `file` , then the ⟨general text⟩ will be treated as a file name and its contents will be copied into the stream contents.

When ⟨object type spec⟩ is given as `reserveobjnum`, just a new object number is reserved. The number of the reserved object is accessible via `\pdflastobj`. The object can later be filled with contents by `\pdfobj useobjnum` ⟨number⟩ `{` ⟨balanced text⟩ `}`. But the reserved object number can already be be used before by other objects, which provides a forward–referencing mechanism.

The object is kept in memory and will be written to the pdf output only when its number is referred to by `\pdfrefobj` or when `\pdfobj` is preceded by `\immediate`. Nothing is appended to the list being built. The number of the most recently created object is accessible via `\pdflastobj`.

► `\pdflastobj` (read–only integer)

This command returns the object number of the last object created by `\pdfobj`.

► `\pdfrefobj` ⟨object number⟩

This command appends a whatsit node to the list being built. When the whatsit node is searched at shipout time, pdfTEX will write the object ⟨object number⟩ to the pdf output if it has not been written yet.

7.5 Page and pages objects

► `\pdfpagesattr` (tokens)

pdfTEX expands this token list when it finishes the pdf output and adds the resulting character stream to the root `Pages` object. When defined, these are applied to all pages in the document. Some examples of attributes are `/MediaBox`, the rectangle specifying the natural size of the page, `/CropBox`, the rectangle specifying the region of the page being displayed and printed, and `/Rotate`, the number of degrees (in multiples of 90) the page should be rotated clockwise when it is displayed or printed.

```
\pdfpagesattr
  { /Rotate 90                 % rotate all pages by 90 degrees
    /CropBox [0 0 612 792] }  % the crop size of all pages (in bp)
```

► `\pdfpageattr` (tokens)

This is similar to `\pdfpagesattr`, but has priority over it. It can be used to override any attribute given by `\pdfpagesattr` for individual pages. The token list is expanded when pdfTEX ships out a page. The contents are added to the attributes of the current page.

► `\pdfpageref` ⟨page number⟩ (expandable)

This primitive expands to the number of the page object that contains the dictionary for page ⟨page number⟩ . If the page ⟨page number⟩ does not exist, a warning will be issued, a fresh unused pdf object will be generated, and `\pdfpageref` will expand to that object number.

E. g., if the dictionary for page 5 of the TEX document is contained in pdf object no. 18, `\pdfpageref5` expands to the number 18.

7.6 Form XObjects

The next three primitives support a pdf feature called 'object reuse' in pdfTEX. The idea is first to create a 'form XObject' in pdf. The content of this object corresponds to the content of a TEX box; it can contain pictures and references to other form XObjects as well. After creation, the form XObject can be used multiple times by simply referring to its object number. This feature can be useful for large documents with many similar elements, as it can reduce the duplication of identical objects.

These commands behave similarly to `\pdfobj`, `\pdfrefobj` and `\pdflastobj`, but instead of taking raw pdf code, they handle text typeset by TEX.

- `\pdfxform` [⟨attr spec⟩] [⟨resources spec⟩] ⟨box number⟩

  This command creates a form XObject corresponding to the contents of the box ⟨box number⟩ . The box can contain other raw objects, form XObjects, or images as well. It can however *not* contain annotations because they are laid out on a separate layer, are positioned absolutely, and have dedicated housekeeping. `\pdfxform` makes the box void, as `\box` does.

  When ⟨attr spec⟩ is given, the text will be written as additional attribute into the form XObject dictionary. The ⟨resources spec⟩ is similar, but the text will be added to the resources dictionary of the form XObject. The text given by ⟨attr spec⟩ or ⟨resources spec⟩ is written before other entries of the form dictionary and/or the resources dictionary and takes priority over later ones.

- `\pdfrefxform` ⟨object number⟩

  The form XObject is kept in memory and will be written to the pdf output only when its object number is referred to by `\pdfrefxform` or when `\pdfxform` is preceded by `\immediate`. Nothing is appended to the list being built. The number of the most recently created form XObject is accessible via `\pdflastxform`.

  When issued, `\pdfrefxform` appends a whatsit node to the list being built. When the whatsit node is searched at shipout time, pdfTEX will write the form ⟨object number⟩ to the pdf output if it is not written yet.

- `\pdflastxform` (read–only integer)

  The object number of the most recently created form XObject is accessible via `\pdflastxform`.

  As said, this feature can be used for reusing information. This mechanism also plays a role in typesetting fill–in forms. Such widgets sometimes depends on visuals that show up on user request, but are hidden otherwise.

- `\pdfxformname` ⟨object number⟩ (expandable)

  In pdf files produced by pdfTEX one can recognize a form Xobject by the prefix `/Fm` followed by a number, for instance `/Fm2`. For a given form XObject number, this primitive expands to the number in the corresponding form XObject name. E. g., if `/Fm2` corresponds to some form XObject with object number 7, the `\pdfxformname7` expands to the number 2. The primitive was introduced in pdfTEX 1.30.0.

7.7 Graphics inclusion

pdf provides a mechanism for embedding graphic and textual objects: form XObjects. In pdfTEX this mechanism is accessed by means of `\pdfxform`, `\pdflastxform` and `\pdfrefxform`. A special kind of XObjects are bitmap graphics and for manipulating them similar commands are provided.

- `\pdfximage` [⟨rule spec⟩] [⟨attr spec⟩] [⟨page spec⟩] [⟨colorspace spec⟩] [⟨pdf box spec⟩] ⟨general text⟩

  This command creates an image object. The dimensions of the image can be controlled via ⟨rule spec⟩. The default values are zero for depth and 'running' for height and width. If all of them are given, the image will be scaled to fit the specified values. If some (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 original image size, where depth is treated as zero. If none are given then the image will take its natural size.

  An image inserted at its natural size often has a resolution of `\pdfimageresolution` (see below) given in dots per inch in the output file, but some images may contain data specifying the image resolution, and in such a case the image will be scaled to the correct resolution. The dimensions of an image can be accessed by enclosing the `\pdfrefximage` command to a box and checking the dimensions of the box:

  ```
  \setbox0=\hbox{\pdfximage{somefile.png}\pdfrefximage\pdflastximage}
  ```

Now we can use `\wd0` and `\ht0` to question the natural size of the image as determined by pdfTEX. When dimensions are specified before the `{somefile.png}`, the graphic is scaled to fit these. Note that, unlike the e. g. `\input` primitive, the filename is supplied between braces.

The image type is specified by the extension of the given file name: `.png` stands for png image, `.jpg` (or `.jpeg`) for jpeg, `.jbig2` (preferred, but `.jb2` works also) for jbig2, and `.pdf` for pdf file. But once pdfTEX has opened the file, it checks the file type first by looking to the magic number at the file start, which gets precedence over the file name extension. This gives a certain degree of fault tolerance, if the file name extension is stated wrongly.

Similarly to `\pdfxform`, the optional text given by ⟨attr spec⟩ will be written as additional attributes of the image before other keys of the image dictionary. One should be aware, that slightly different type of pdf object is created while including png, jpeg, or jbig2 bitmaps and pdf images.

While working with pdf or jbig2 images, ⟨page spec⟩ allows to decide which page of the document is to be included; the ⟨page spec⟩ is irrelevant for the other two image formats. Starting from pdfTEX 1.11 one may also decide in the pdf image case, which page box of the image is to be treated as a final bounding box. If ⟨pdf box spec⟩ is present, it overrides the default behaviour specified by the `\pdfforcepagebox` parameter. This option is irrelevant for non–pdf inclusions.

Starting from pdfTEX 1.21, `\pdfximage` command supports `colorspace` keyword followed by an object number (user–defined colorspace for the image being included). This feature works for jpeg images only. pngs are rgb palettes, jbig2 s are bitonal, and pdf images have always self–contained color space information.

► `\pdfrefximage` ⟨integer⟩

The image is kept in memory and will be written to the pdf output only when its number is referred to by `\pdfrefximage` or `\pdfximage` is preceded by `\immediate`. Nothing is appended to the list being built.

`\pdfrefximage` appends a whatsit node to the list being built. When the whatsit node is searched at shipout time, pdfTEX will write the image with number ⟨integer⟩ to the pdf output if it has not been written yet.

► `\pdflastximage` (read–only integer)

The number of the most recently created XObject image is accessible via `\pdflastximage`.

► `\pdflastximagepages` (read–only integer)

This read–only register returns the highest page number from a file previously accessed via the `\pdfximage` command. This is useful only for pdf files; it always returns `1` for png, jpeg, or jbig2 files.

► `\pdfimageresolution` (integer)

The integer `\pdfimageresolution` parameter (unit: dots per inch, dpi) is a last resort value, used only for bitmap (jpeg, png, jbig2) images, but not for pdfs. The priorities are as follows: Often one image dimension (`width` or `height`) is stated explicitely in the TEX file. Then the image is properly scaled so that the aspect ratio is kept. If both image dimensions are given, the image will be stretched accordingly, whereby the aspect ratio might get distorted. Only if no image dimension is given in the TEX file, the image size will be calculated from its width and height in pixels, using the x and y resolution values normally contained in the image file. If one of these resolution values is missing or weird (either < 0 or > 65535), the `\pdfimageresolution` value will be used for both x and y resolution, when calculating the image size. And if the `\pdfimageresolution` is zero, finally a default resolution of 72 dpi would be taken. The `\pdfimageresolution` is read when pdfTEX creates an image via `\pdfximage`. The given value is clipped to the range 0..65535 [dpi].

Currently this parameter is used particularly for calculating the dimensions of jpeg images in exif format (unless at least one dimension is stated explicitely); the resolution values coming with exif files are currently ignored.

► `\pdfforcepagebox` (integer)

When pdf files are included, the command `\pdfximage` allows the selection of which pdf page box to use in the optional field ⟨image attr spec⟩. The integer primitive `\pdfforcepagebox` allows to globally override this choice by giving them one of the following values: (1) media box, (2) crop box, (3) bleed box, (4) trim box, and (5) artbox. The command is available starting from pdfTEX 1.30.0, as a shortened synonym of obsolete `\pdfoptionalwaysusepdfpagebox` instruction.

► `\pdfinclusionerrorlevel` (integer)

This controls the behaviour of pdfTEX when a pdf file is included that has a newer version than the one specified by this primitive: If it is set to 0, pdfTEX gives only a warning; if it's 1, pdfTEX raises an error. The command has been introduced in pdfTEX 1.30.0 as a shortened synonym of `\pdfoptionpdfinclu-sionerrorlevel`, that is now obsolete.

► `\pdfimagehicolor` (integer)

This primitive, when set to 1, enables embedding of png images with 16 bit wide color channels at their full color resolution. As such an embedding mode is defined only from pdf version 1.5 onwards, the `\pdfimagehicolor` functionality is automatically disabled in pdfTEX if `\pdfminorversion` < 5; then each 16 bit color channel is reduced to a width of 8 bit by stripping the lower 8 bits before embedding. The same stripping happens when `\pdfimagehicolor` is set to 0. For `\pdfminorversion` ≥ 5 the default value of `\pdfimagehicolor` is 1. If specified, the parameter must appear before any data is written to the pdf output. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfimageapplygamma` (integer)

This primitive, when set to 1, enables gamma correction while embedding png images, taking the values of the primitives `\pdfgamma` as well as the gamma value embedded in the png image into account. When `\pdfimageapplygamma` is set to 0, no gamma correction is performed. If specified, the parameter must appear before any data is written to the pdf output. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfgamma` (integer)

This primitive defines the 'device gamma' for pdfTEX. Values are in promilles (same as for `\mag`). The default value of this primitive is 1000, defining a device gamma value of 1.0.

When `\pdfimageapplygamma` is set to 1, then whenever a png image is included, pdfTEX applies a gamma correction. This correction is based on the value of the `\pdfgamma` primitive and the 'assumed device gamma' that is derived from the value embedded in the actual image. If no embedded value can be found in the png image, then the value of `\pdfimagegamma` is used instead. If specified, the parameter must appear before any data is written to the pdf output. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfimagegamma` (integer)

This primitive gives a default 'assumed gamma' value for png images. Values are in promilles (same as for `\pdfamma`). The default value of this primitive is 2200, implying an assumed gamma value of 2.2.

When pdfTEX is applying gamma corrections, images that do not have an embedded 'assumed gamma' value are assumed to have been created for a device with a gamma of 2.2. Experiments show that this default setting is correct for a large number of images; however, if your images come out too dark, you probably want to set `\pdfimagegamma` to a lower value, like 1000. If specified, the parameter must appear before any data is written to the pdf output. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfpxdimen` (dimen)

While working with bitmap graphics or typesetting electronic documents, it might be convenient to base dimensions on pixels rather than TEX's standard units like `pt` or `em`. For this purpose, pdfTEX provides an extra unit called px that takes the dimension given to the `\pdfpxdimen` primitive. In example, to make the unit px corresponding to 96 dpi pixel density (then 1 px = 72/96 bp), one can do the following calculation:

```
\pdfpxdimen=1in % 1 dpi
\divide\pdfpxdimen by 96 % 96 dpi
\hsize=1200px
```

Then `\hsize` amounts to 1200 pixels in 96 dpi, which is exactly 903.375 pt (but TEX rounds it to 903.36914 pt). The default value of `\pdfpxdimen` is 1 bp, corresponding to a pixel density of 72 dpi. This primitive is completely independent from the `\pdfimageresolution` and `\pdfpkresolution` parameters. The primitive was introduced in pdfTEX 1.30.0. It used to be an integer register that gave the dimension 1 px as number of scaled points, defaulting to 65536 (1 px equal to 65536 sp = 1 pt). Starting from pdfTEX 1.40.0, `\pdfpxdimen` is now a real dimension parameter.

7.8 Annotations

pdf 1.4 provides four basic kinds of annotations:

- hyperlinks, general navigation
- text clips (notes)
- movies
- sound fragments

The first type differs from the other three in that there is a designated area involved on which one can click, or when moved over some action occurs. pdfTEX is able to calculate this area, as we will see later. All annotations can be supported using the next two general annotation primitives.

► `\pdfannot` ⟨annot type spec⟩

This command appends a whatsit node corresponding to an annotation to the list being built. The dimensions of the annotation can be controlled via the ⟨rule spec⟩ . The default values are running for all width, height and depth. When an annotation is written out, running dimensions will take the corresponding values from the box containing the whatsit node representing the annotation. The ⟨general text⟩ is inserted as raw pdf code to the contents of annotation. The annotation is written out only if the corresponding whatsit node is searched at shipout time.

► `\pdflastannot` (read–only integer)

This primitive returns the object number of the last annotation created by `\pdfannot`. These two primitives allow users to create any annotation that cannot be created by `\pdfstartlink` (see below).

7.9 Destinations and links

The first type of annotation, mentioned above, is implemented by three primitives. The first one is used to define a specific location as being referred to. This location is tied to the page, not the exact location on the page. The main reason for this is that pdf maintains a dedicated list of these annotations —and some more when optimized— for the sole purpose of speed.

► `\pdfdest` ⟨dest spec⟩

This primitive appends a whatsit node which establishes a destination for links and bookmark outlines; the link is identified by either a number or a symbolic name, and the way the viewer is to display the page must be specified in ⟨dest type⟩, which must be one of those mentioned in table 5.

The specification `xyz` can optionally be followed by `zoom` ⟨integer⟩ to provide a fixed zoom–in. The ⟨integer⟩ is processed like TEX magnification, i. e. 1000 is the normal page view. When `zoom` ⟨integer⟩ is given, the zoom factor changes to 0.001 of the ⟨integer⟩ value, otherwise the current zoom factor is kept unchanged.

The destination is written out only if the corresponding whatsit node is searched at shipout time.

keyword	meaning
`fit`	fit the page in the window
`fith`	fit the width of the page
`fitv`	fit the height of the page
`fitb`	fit the 'Bounding Box' of the page
`fitbh`	fit the width of 'Bounding Box' of the page
`fitbv`	fit the height of 'Bounding Box' of the page
`xyz`	goto the current position (see below)

Table 5 Options for display of outline and destinations.

► `\pdfstartlink` [⟨rule spec⟩] [⟨attr spec⟩] ⟨action spec⟩

This primitive is used along with `\pdfendlink` and appends a whatsit node corresponding to the start of a hyperlink. The whatsit node representing the end of the hyperlink is created by `\pdfendlink`. The dimensions of the link are handled in the similar way as in `\pdfannot`. Both `\pdfstartlink` and `\pdfendlink` must be in the same level of box nesting. A hyperlink with running width can be multi–line or even multi–page, in which case all horizontal boxes with the same nesting level as the boxes containing `\pdfstartlink` and `\pdfendlink` will be treated as part of the hyperlink. The hyperlink is written out only if the corresponding whatsit node is searched at shipout time.

Additional attributes, which are explained in great detail in the PDF Reference, can be given via ⟨attr spec⟩. Typically, the attributes 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.

While all graphics and text in a pdf document have relative positions, annotations have internally hard–coded absolute positions. Again this is for the sake of speed optimization. The main disadvantage is that these annotations do *not* obey transformations issued by `\pdfliteral`'s.

The ⟨action spec⟩ specifies the action that should be performed when the hyperlink is activated while the ⟨user-action spec⟩ performs a user–defined action. A typical use of the latter is to specify a url, like `/S /URI /URI (http://www.tug.org/)`, or a named action like `/S /Named /N /NextPage`.

A ⟨goto-action spec⟩ performs a GoTo action. Here ⟨numid⟩ and ⟨nameid⟩ specify the destination identifier (see below). The ⟨page spec⟩ specifies the page number of the destination, in this case the zoom factor is given by ⟨general text⟩. A destination can be performed in another pdf file by specifying ⟨file spec⟩, in which case ⟨newwindow spec⟩ specifies whether the file should be opened in a new window. A ⟨file spec⟩ can be either a `(string)` or a `dictionary`. The default behaviour of the ⟨newwindow spec⟩ depends on the browser setting.

A ⟨thread-action spec⟩ performs an article thread reading. The thread identifier is similar to the destination identifier. A thread can be performed in another pdf file by specifying a ⟨file spec⟩.

► `\pdfendlink`

This primitive ends a link started with `\pdfstartlink`. All text between `\pdfstartlink` 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.

► `\pdflastlink` (read–only integer)

This primitive returns the object number of the last link created by `\pdfstartlink` (analogous to `\pdflastannot`). The primitive was introduced in pdfTEX 1.40.0.

► `\pdflinkmargin` (dimension)

This dimension parameter specifies the margin of the box representing a hyperlink and is read when a page containing hyperlinks is shipped out.

► `\pdfdestmargin` (dimension)

Margin added to the dimensions of the rectangle around the destinations.

7.10 Bookmarks

► `\pdfoutline` [⟨attr spec⟩] ⟨action spec⟩ [`count` ⟨integer⟩] ⟨general text⟩

This primitive creates an outline (or bookmark) entry. The first parameter specifies the action to be taken, and is the same as that allowed for `\pdfstartlink`. The ⟨count⟩ specifies the number of direct subentries under this entry; specify 0 or omit it if this entry has no subentries. 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 pdf Document Encoding vector. The outline is written to the pdf output immediately.

7.11 Article threads

► `\pdfthread` [⟨rule spec⟩] [⟨attr spec⟩] ⟨id spec⟩

Defines a bead within an article thread. Thread beads with same identifiers (spread across the document) will be joined together.

► `\pdfstarthread` [⟨rule spec⟩] [⟨attr spec⟩] ⟨id spec⟩

This uses the same syntax as `\pdfthread`, apart that it must be followed by a `\pdfendthread`. `\pdfstartthread` and the corresponding `\pdfendthread` must end up in vboxes with the same nesting level; all vboxes between them will be added into the thread. Note that during output runtime if there are other newly created boxes which have the same nesting level as the vbox/vboxes containing `\pdfstartthread` and `\pdfendthread`, they will be also added into the thread, which is probably not what you want. To avoid such unconsidered behaviour, it's often enough to wrap boxes that shouldn't belong to the thread by a box to change their box nesting level.

► `\pdfendthread`

This ends an article thread started before by `\pdfstartthread`.

► `\pdfthreadmargin` (dimension)

Specifies a margin to be added to the dimensions of a bead within an article thread.

7.12 Literals and specials

► `\pdfliteral` [⟨pdfliteral spec⟩] ⟨general text⟩

Like `\special` in normal TEX, this command inserts raw pdf code into the output. This allows support of color and text transformation. This primitive is heavily used in the METAPOST inclusion macros. Normally pdfTEX ends a text section in the pdf output and sets the transformation matrix to the current location on the page before inserting ⟨general text⟩, however this can be turned off by giving the optional keyword `direct` . This command appends a whatsit node to the list being built. ⟨general text⟩ is expanded when the whatsit node is created and not when it is shipped out, as with `\special`.

Starting from version 1.30.0, pdfTEX allows to use a new keyword `page` instead of `direct`. Both modify the default behaviour of `\pdfliteral`, avoiding translation of the coordinates space before inserting the literal code. The difference is that the `page` keyword instructs pdfTEX to close a `BT ET` text block before inserting anything. It means that the literal code inserted refers to the origin (lower–left corner of the page) and can be safely enclosed with `q Q`. Note, that in most cases using `q Q` operators inside `\pdfliteral` with `direct` keyword will produce corrupted pdf output, as the pdf standard doesn't allow to do anything like this within a `BT ET` block.

► `\special {pdf:` ⟨text⟩ `}`

This is equivalent to `\pdfliteral {` ⟨text⟩ `}`.

► `\special {pdf:direct:` ⟨text⟩ `}`

This is equivalent to `\pdfliteral direct {` ⟨text⟩ `}`.

► `\special {pdf:page:` ⟨text⟩ `}`

This is equivalent to `\pdfliteral page {` ⟨text⟩ `}`.

7.13 Strings

► `\pdfescapestring` ⟨general text⟩

Starting from version 1.30.0, pdfTEX provides a mechanism for converting a general text into pdf string. Many characters that may be needed inside such a text (especially parenthesis), have a special meaning inside a pdf string object and thus, can't be used literally. The primitive replaces each special pdf character by its literal representation by inserting a backslash before that character. Some characters (e. g. space) are also converted into 3–digit octal number. In example, `\pdfescapestring{Text (1)}` will be expanded to `Text\040\(1\)`. This ensures a literal interpretation of the text by the pdf viewer. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfescapename` ⟨general text⟩

In analogy to `\pdfescapestring`, `\pdfescapename` replaces each special pdf character inside the general text by its hexadecimal representation preceded by `#` character. This ensures the proper interpretation of the text if used as a pdf name object. In example, `Text (1)` will be replaced by `Text#20#281#29`. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfescapehex` ⟨general text⟩

This command converts each character of ⟨general text⟩ into its hexadecimal representation. Each character of the argument becomes a pair of hexadecimal digits. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfunescapehex` ⟨general text⟩

This command treats each character pair of ⟨general text⟩ as a hexadecimal number and returns an ascii character of this code. The primitive was introduced in pdfTEX 1.30.0.

► `\pdfmdfivesum` ⟨general text⟩ (expandable)

This command expands to the md5 of ⟨general text⟩ in uppercase hexadecimal format (same as `\pdfescapehex`). The primitive was introduced in pdfTEX 1.30.0.

7.14 Numbers

► `\ifpdfabsnum` ⟨number⟩

The primitive works as normal `\ifnum` condition check, except that it compares absolute values of numbers. Although it seems to be a trivial shortcut for a couple of `\ifnum x<0` tests, as a primitive it is more friendly in usage and works a bit faster. The primitive was introduced in pdfTEX 1.40.0.

► `\ifpdfabsdim` ⟨dimen⟩

Like `\ifpdfabsnum`, the primitive works as normal `\ifdim` condition check, except that it compares absolute values of dimensions. The primitive was introduced in pdfTEX 1.40.0.

- \pdfuniformdeviate ⟨number⟩ (expandable)

  The commands generates a uniformly distributed random integer value between 0 (inclusive) and ⟨number⟩ (exclusive). This primitive expands to a list of tokens. The primitive was introduced in pdfTEX 1.30.0.

- \pdfnormaldeviate (expandable)

  The commands generates a random integer value with a mean of 0 and a unit of 65 536, e. g. −7652. This primitive expands to a list of tokens. The primitive was introduced in pdfTEX 1.30.0.

- \pdfrandomseed (read–only integer)

  You can use \the\pdfrandomseed to query the current seed value, so you can e. g. write the value to the log file. The initial value of the seed is derived from the system time, and is not more than 1 000 999 999 (this ensures that the value can be used with commands like \count). The primitive was introduced in pdfTEX 1.30.0.

- \pdfsetrandomseed ⟨number⟩

  This sets the random seed (\pdfrandomseed) to a specific value, allowing you to re-play sequences of semi-randoms at a later moment. The primitive was introduced in pdfTEX 1.30.0.

7.15 Timekeeping

- \pdfelapsedtime (read–only integer)

  The command expands to a number that represents the time elapsed from the moment of run start. The elapsed time is returned in 'scaled seconds', that means seconds divided by 65536, e. g. pdfTEX has run for 287681 'scaled seconds' when this paragraph was typeset. Obviously, the command will never return a value greater than the highest number available in TEX: if the time exceeds 32767 seconds, the constant value $2^{31} - 1$ will be returned. The primitive was introduced in pdfTEX 1.30.0.

- \pdfresettimer

  The command resets the internal timer so that \pdfelapsedtime starts returning micro–time from 0 again. The primitive was introduced in pdfTEX 1.30.0.

7.16 Files

- \pdffilemoddate ⟨general text⟩ (expandable)

  Expands to the modification date of file ⟨general text⟩ in the same format as for \pdfcreationdate, e. g. it's D:20090522223924+01'00' for the source of this manual. The primitive was introduced in pdfTEX 1.30.0.

- \pdffilesize ⟨general text⟩ (expandable)

  Expands to the size of file ⟨general text⟩ , e. g. it's 222029 for the source of this manual. The primitive was introduced in pdfTEX 1.30.0.

- \pdfmdfivesum file ⟨general text⟩ (expandable)

  Expands to the md5 of file ⟨general text⟩ in uppercase hexadecimal format (same as \pdfescapehex), e. g. it's 80D318F1FB456260CAF2F474A38A13DD for the source of this manual. The primitive was introduced in pdfTEX 1.30.0.

- \pdffiledump [offset ⟨number⟩] [length ⟨number⟩] ⟨general text⟩ (expandable)

  Expands to the dump of the file ⟨general text⟩ in uppercase hexadecimal format (same as \pdfescapehex), starting at offset ⟨number⟩ or 0 with length ⟨number⟩ , if given. The first ten bytes of the source of this manual are 2520696E746572666163. The primitive was introduced in pdfTEX 1.30.0.

7.17 Color stack

pdfTEX 1.40.0 comes with color stack support (actually any graphic state stack).

► `\pdfcolorstackinit` [page] [direct] ⟨general text⟩ (expandable)

The primitive initializes a new graphic stack and returns its number. Optional `page` keyword instructs pdfTEX to restore the graphic at the beginning of every new page. Also optional `direct` has the same effect as for `\pdfliteral` primitive. The primitive was introduced in pdfTEX 1.40.0.

► `\pdfcolorstack` ⟨stack number⟩ ⟨stack action⟩ ⟨general text⟩

The command operates on the stack of a given number. If ⟨stack action⟩ is `push` keyword, the new value provided as ⟨general text⟩ is inserted into the top of the graphic stack and becomes the current stack value. If followed by `pop`, the top value is removed from the stack and the new top value becomes the current. `set` keyword replaces the current value with ⟨general text⟩ without changing the stack size. `current` keyword instructs just to use the current stack value without modifying the stack at all. The primitive was introduced in pdfTEX 1.40.0.

7.18 Transformations

Since the content of `\pdfliteral` is not interpreted anyhow, any transformation inserted directly into the content stream, as well as saving and restoring the current transformation matrix, remains unnoticed by pdfTEX positioning mechanism. As a consequence, links and other annotations (that are formed in pdf as different layer then the page content) are not affected by such user-defined transformations. pdfTEX 1.40.0 solves this problem with three new primitives.

► `\pdfsetmatrix`

Afine transformations are normally expressed with six numbers. First four (no unit) values defining scaling, rotating and skewing, plus two extra dimensions for shifting. Since the translation is handled by TEX itself, `\pdfsetmatrix` primitive expects as an argument a string containing just first four numbers of the transformation separated by a space and assumes two position coordinates to be 0. In example, `\pdfsetmatrix{0.87 -0.5 0.5 0.87}` rotates the current space by 30 degrees, inserting `0.87 -0.5 0.5 0.87 0 0 cm` into the content stream. The primitive was introduced in pdfTEX 1.40.0.

► `\pdfsave`

The command saves the current transformation by inserting `q` operator into the content stream. The primitive was introduced in pdfTEX 1.40.0.

► `\pdfrestore`

The command restores previously saved transformation by inserting `Q` operator into the content stream. One should keep in mind that `\pdfsave` and `\pdfrestore` pairs should always be properly nested and should start and end at the same group level. The primitive was introduced in pdfTEX 1.40.0.

7.19 Miscellaneous

► `\vadjust` [⟨pre spec⟩] ⟨filler⟩ `{` ⟨vertical mode material⟩ `}`

The `\vadjust` implementation of pdfTEX adds an optional qualifier ⟨pre spec⟩ (which is the string `pre`) to the original TEX primitive with the same name. As long as there is no `pre` given, `\vadjust` behaves exactly as the original (see the TEXbook, p. 281); it appends an adjustment item created from ⟨vertical mode material⟩ to the current list *after* the line in which `\vadjust` appears. However with the qualifier `pre`, the adjustment item is put *before* the line in which `\vadjust pre` appears.

► `\quitvmode`

The primitive instructs pdfTEX to quit the vertical mode and start typesetting a paragraph. `\quitvmode` has the same effect as `\leavevmode` definition from `plain` macro package. Note however, that

\leavevmode may conflict with \everypar tokens list. No such risk while using \quitvmode instead. The primitive was introduced in pdfTEX 1.21a.

- \pdfsavepos

This primitive marks the current absolute (x, y) position on the media, with the reference point in the lower left corner. It is active only during page shipout, when the page is finally assembled. The position coordinates can then be retrieved by the \pdflastxpos and \pdflastypos primitives, and e. g. written out to some auxiliary file. The coordinates can be used only after the current \shipout has been finalized, therefore normally two pdfTEX runs are required to utilize these primitives. Starting with pdfTEX 1.40.0, this mechanism can be used also while running in dvi mode.

- \pdflastxpos (read–only integer)

This primitive returns an integer number representing the absolute x coordinate of the last point marked by \pdfsavepos. The unit is 'scaled points' (sp).

- \pdflastypos (read–only integer)

This primitive works similar to \pdflastxpos, only it returns the y coordinate.

- \pdftexversion (read–only integer)

Returns the version of pdfTEX multiplied by 100, e. g. for pdfTEX version 1.40.9 used to produce this document, it returns 140.

- \pdftexrevision (expandable)

Returns the revision letter of pdfTEX, e. g. for pdfTEX version 1.40.9 used to produce this document, it returns the letter 9.

- \pdftexbanner (expandable)

Returns the pdfTEX banner message, e. g. for the version used here: This is pdfTeX, Version 3.1415926-1.40.9-2.2 (Web2C 7.5.7) kpathsea version 3.5.7. The primitive was introduced in pdfTEX 1.20a.

- \pdfcreationdate (expandable)

Expands to the date string pdfTEX uses in the info dictionary of the document, e. g. for this file D:20091130223302Z. The primitive was introduced in pdfTEX 1.30.0.

- \pdfshellescape (read–only integer)

This primitive is 1 if \write18 is enabled, 0 otherwise. \write18 was not enabled when this manual was typeset. The primitive was introduced in pdfTEX 1.30.0.

- \ifpdfprimitive ⟨control sequence⟩ (expandable)

This condition checks if the following control sequence has its primitive meaning. If it has, \ifpdfprimitive returns true. In any other case (redefined, made \undefined, has never been primitive) false is returned. The primitive was introduced in pdfTEX 1.40.0.

- \pdfprimitive ⟨control sequence⟩

This command executes the primitive meaning of the following control sequence, if it has been redefined or made undefined. If the following control sequence is undefined and never was a primitive, nothing happens and no error is raised. If the control sequence was initially expandable, \pdfprimitive expands either. Otherwise \pdfprimitive doesn't expand. The primitive was introduced in pdfTEX 1.40.0.

- \pdfdraftmode (integer)

When set to 1 (or switched on by the commandline switch -draftmode) pdfTEX doesn't write the output pdf and doesn't actually read any images but does everything else (including writing auxiliary files), thus speeding up compilations when you know you need an extra run but don't care about the output, e.g. just to get the bibTEX references right. The primitive was introduced in pdfTEX 1.40.0.

8 Graphics

pdfTEX supports inclusion of pictures in png, jpeg, jbig2, and pdf format; a few differences between these are discussed below. The most common technique with TEX —the inclusion of eps figures— is replaced by pdf inclusion. eps files can be converted to pdf by Ghostscript, Adobe Distiller or other PostScript–to–pdf converters.

The pdf format is currently the most versatile source format for graphics embedding. pdfTEX allows to insert arbitrary pages from pdf files with their own fonts, graphics, and pixel images into a document. The cover page of this manual is an example of such an insert, being a one page document generated by pdfTEX.

By default pdfTEX takes the BoundingBox of a pdf file from its CropBox if available, otherwise from its MediaBox. This can be influenced by the ⟨pdf box spec⟩ option to the `\pdfximage` primitive, or by setting the `\pdfforcepagebox` primitive to a value corresponding to the wanted box type.

To get the right BoundingBox from a eps file, before converting to pdf, it is necessary to transform the eps file so that the start point is at the (0,0) coordinate and the page size is set exactly corresponding to the BoundingBox. A Perl script (epstopdf) for this purpose has been written. The TEXutil utility script and the PStoPDF program that comes with Ghostscript can so a similar job. (Concerning this conversion, they can handle complete directories, remove some garbage from files, takes precautions against duplicate conversion, etc.)

The lossless compressing png format is great for embedding crisp pixel graphics (e. g. line scans, screen shots). Since pdfTEX 1.30.0 also the alpha-channel of png images is processed if available; this allows embedding of images with simple transparency. The png format does not support the CMYK color model, which is sometimes required for print media (this often can be replaced by four component jpeg in high quality or lossless compression mode). Photos in png format have a rather weak compression; here the jpeg format is preferable.

Embedding png images in the general case requires pdfTEX to uncompress the pixel array and to re–compress it to the pdf requirements; this process often takes a noticeable amount of time. Since pdfTEX 1.30.0 there is now a fast png embedding mode that goes without uncompressing; the image data are directly copied into the pdf stream, resulting in a much higher embedding speed. However this mode is only activated, if the image array structure of the png file is compatible with the pdf image structure (e. g. an interlaced png image requires uncompressing to re–arrange the image lines). Luckily it seems that the most common png files also allow fast copying. The use of gamma correction disables fast copying, as it requires calculations with individual pixels. Whether the fast copy mode is used for a png image can be seen from the log file, which then shows the string '(PNG copy)' after the png file name.

The jpeg format is normally used in lossy mode; then it's ideal for embedding photos; it's not recommended for crisp images from synthetic sources with a limited amount of colors.

The jbig2 format works only for bitonal (black and white) pixel images like scanned line and text documents, but for these it has typically a much higher compression ratio than the other two pixel image formats. The jbig2 format is part of the pdf standard since version 1.5; native jbig2 image inclusion is available in pdfTEX since version 1.40.0. A jbig2 file might contain many images, which gives an even better compression ratio than with a single image per file, as jbig2 encoders can exploit similarities between bit patterns over several images. Encoders for jbig2 can operate in lossy as well as lossless modes. Only recently a free jbig2 encoder has been written and made available, see http://www.imperialviolet.org/jbig2.html.

Other options 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 back–end 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 some macros written by Hans Hagen.

METAPOST Although the output of METAPOST is PostScript, it is in a highly simplified form, and a METAPOST to pdf conversion (mptopdf, written by Hans Hagen and Tanmoy Bhattacharya) is implemented as a set of macros which reads METAPOST output and supports all of its features.

For new work, the METAPOST route is highly recommended. For the future, Adobe has announced that they will define a specification for 'encapsulated pdf'.

The inclusion of raw PostScript commands —a technique utilized by for instance the `pstricks` package— cannot directly be supported. Although pdf is direct a descendant of PostScript, it lacks any programming language commands, and cannot deal with arbitrary PostScript.

9 Character translation

Characters that are input to pdfTEX are subject to optional TEX character translation (tcx) under control of a tcx file. The tcx maps the input character codes (e. g. from `\input` or `\read`) to the character codes as seen by pdfTEX. This mapping takes place before the characters enter pdfTEX's 'mouth'. If no tcx file is read, the input characters enter pdfTEX directly; no mapping is done.

tcx files consist of lines each containing one or two integer numbers in the range 0..255, either in decimal or hex notation. A comment sign `%` in a tcx line starts a comment until the end of line. The first number in each line is for matching the input character code, the second, optional number is the corresponding TEX character code. If a line contains only one number, characters with this code enter pdfTEX unchanged; no mapping is done.

tcx mapping also influences pdfTEX output streams for `\message` and `\write`. Without tcx mapping, only characters that are within the range 32..126 are flagged as 'printable', meaning that these characters are output directly by `\message` and `\write` primitives. Characters outside the range 32..126 are instead output in escaped form, e. g. as `^^A` for a character with code `0x01`. When a character code is mentioned in the 2nd column of the tcx file, or as the only value in a line, it is flagged as 'printable'. During `\message` and `\write`, output characters are mapped in reverse direction: they are looked up in the 2nd column of the tcx file and the corresponding values from the 1st column are output. Again, if a pdfTEX character code is found as the only number in a line, no mapping is done. Mentioning a character code as the only number on a line has the sole purpose to flag this code 'printable'; remember that character within the range 32..126 are 'printable' anyway.

The characters output into the pdf file, e. g. by `\pdfliteral` or `\special` primitives, are not subject to tcx output remapping.

Beware: Character translation interferes with the encTEX primitives; to avoid surprises, don't use encTEX and tcx mapping at the same time. Further details about tcx file loading can be found in the teTEX manual.

Abbreviations

In this document we used a few abbreviations. For convenience we mention their meaning here.

afm	Adobe Font Metrics
ascii	American Standard Code for Information Interchange
bibTEX	Handles bibliographies
CMacTEX	Macintosh Web2c distribution
ConTEXt	general purpose macro package
dvi	native TEX Device Independent file format

encTEX	encTEX extension to TEX
eps	Encapsulated PostScript
epstopdf	eps to pdf conversion tool
ε-TEX	an extension to TEX
TEXexec	ConTEXt command line interface
exif	Exchangeable Image File format (JPEG file variant)
Ghostscript	ps and pdf language interpreter
hz	Hermann Zapf optimization
jbig	Joint Bi-level Image Experts Group
jbig2	Joint Bi-level Image Experts Group
jpeg	Joint Photographic Experts Group
LATEX	general purpose macro package
TEX Live	TEX-Live distribution (multiple platform)
Mac OS X	Macintosh operating system version 10
md5	MD5 message-digest algorithm
METAFONT	graphic programming environment, bitmap output
METAPOST	graphic programming environment, vector output
MikTeX	Win32 distribution
MLTEX	MLTEX extension to TEX
mptopdf	METAPOST to pdf conversion tool
MS-DOS	Microsoft DOS platform (Intel)
pdf	Portable Document Format
pdfTEX	TEX extension producing pdf output
pdfeTEX	ε-TEX extension producing pdf output
Perl	Perl programming environment
pgc	pdf Glyph Container
pk	Packed bitmap font
png	Portable Network Graphics
PostScript	PostScript
proTEXt	Win32 Web2c distribution based on MikTeX
PStoPDF	PostScript to pdf converter (on top of GhostScript)
rgb	Red Green Blue color specification
tcx	TEX Character Translation
tds	TEX Directory Standard
teTEX	TEX distribution for Unix (based on Web2c)
TEX	typographic language and program
Texinfo	generate typeset documentation from info pages
tfm	TEX Font Metrics
Unix	Unix platform
url	Uniform Resource Locator
TEXutil	ConTEXt utility tool
web	literate programming environment
Web2c	official multi–platform web environment
Win32	Microsoft Windows platform
XEmTEX	Win32 Web2c distribution

Examples of HZ and protruding

In the following sections we will demonstrate pdfTEX's protruding and hz features, using a text from E. Tufte. This sample text has a lot of punctuation and often needs hyphenation. Former pdfTEX versions had sometimes problems with combining these features, but from version 1.21a on it should be ok. If you still encounter problems, please try to prepare a small test file that demonstrates the problem and send it to one of the maintainers.

Normal

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

HZ

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Protruding

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Both

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Additional PDF keys

This section is based on the manual on keys written by Martin Schröder, one of the maintainers of pdfTEX*.*

A pdf document should contain only the structures and attributes defined in the pdf specification. However, the specification allows applications to insert additional keys, provided they follow certain rules.

The most important rule is that developers have to register with Adobe prefixes for the keys they want to insert. Hans Hagen has registered the prefix `PTEX` for pdfTEX.

pdfTEX generates an XObject for every included pdf. The dictionary of this object contains these additional keys:

key	**type**	meaning
`PTEX.FileName`	string	The name of the included file as seen by pdfTEX.
`PTEX.InfoDict`	dictionary	The document information dictionary of the included pdf (an indirect object).
`PTEX.PageNumber`	integer	The page number of the included file.

The PDF Reference says: "Although viewer applications can store custom metadata in the document information dictionary, it is inappropriate to store private content or structural information there; such information should be stored in the document catalog instead."

Although it would seem more natural to put this infomation in the document information dictionary, we have to obey the rules laid down in the PDF Reference. The following key ends up in the document catalog.

key	**type**	meaning
`PTEX.Fullbanner`	string	The full version of the binary that produced the file as displayed by `pdftex --version`, e. g. `This is pdfTeX, Version 3.1415926-1.40.9-2.2 (Web2C 7.5.7) kpathsea version 3.5.7`. This is necessary because the string in the `Producer` key in the info dictionary is rather short, e. g. `pdfTeX-1.40.0`.

Colophon

This manual is typeset in ConTEXt. One can generate an A4 version from the source code by typing:

```
texexec --result=pdftex-a.pdf pdftex-t
```

Or in letter size:

```
texexec  --mode=letter --result=pdftex-l.pdf pdftex-t
```

Given that the A4 version is typeset, one can generate an A5 booklet by typing:

```
texexec --pdfarrange --paper=a5a4 --print=up --addempty=1,2
  --result=pdftex-b.pdf pdftex-a
```

Odd and even page sets for non-duplex printers can be generated using `--pages=odd` and `--pages=even` options (which might require some disciplined shuffling of sheet).

This also demonstrates that pdfTEX can be used for page imposition purposes (given that pdfTEX and the fonts are set up properly).

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by pdf viewers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or pdf designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or pdf produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.