pdfTEX

users manual

Hàn ThêThành Sebastian Rahtz Hans Hagen

Contents

1 Introduction
2 About PDF
3 Getting started
4 Macro packages supporting PDFTEX
5 Setting up fonts
6 Formal syntax specification
7 New primitives
8 Graphics and color

Abbreviations

1 Introduction

The main purpose of the PDFTEX project was to create an extension of TEX that can create 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 produces PDF output that looks identical to the DVI output. An important aspect of this project is to investigate alternative justification algorithms, 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 MSDOS systems. It is still under beta development and all features are liable to change. Despite its β–state, PDFTEX produces excellent PDF code.

As PDFTEX evolves, this manual will evolve and more background information will be added. Be patient with the authors.

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

```
texexec  pdftex-t
```

An letter size variant is also supported:

```
texexec  --mode=letter  pdftex-t
```

Given that the A4 version is typeset, one can generate a booklet by saying:

```
texexec --pdfarrange --paper=a5a4 --print=up --addempty=1,2 pdftex-t
```

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

2 About PDF

The cover of this manual shows a simple PDF file. Unless compression and/or encryption is applied, such a file is rather verbose and readable. The first line specifies the version used; currently pdfTeX produces level 1.2 output. Viewers are supposed to silently skip over all elements they cannot handle.

A PDF file consist of objects. These objects can be recognized by their number and keywords:

```
8 0 obj << /Type /Catalog /Pages 6 0 R >> endobj
```

Here `8 0 obj ... endobj` is the object capsule. The first number is the object number. Later we will see that pdfTeX gives access to this number. One can for instance create an object by using `\pdfobj` after which `\pdflastobj` returns the number. So

```
\pdfobj{/Type /Catalog /Pages 6 0 R}
```

inserts an object into the file, while `\pdflastobj` returns the number pdfTeX assigned to this object. The sequence `6 0 R` is an object reference, a pointer to another object. 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.

In general this rather direct way of pushing objects in the files is rather useless and only makes sense when implementing for instance fill–in field support or annotation content reuse. We will come to that later.

Unless such direct objects are part of something larger, they will end up as isolated entities, not doing any harm but not doing any good either.

When a viewer opens a PDF file, it first goes to the end of the file. There it finds the keyword `startxref`, the signal where to look for the so called 'object cross reference table'. This table provides fast access to the objects that make up the file. The actual starting point of the file is defined after the `trailer`. The `/Root` entry points to the catalog. In this catalog the viewer can find the page list. In our example we have only one page. The trailer also holds an `/Info` entry, which tells a bit more about the document. Just follow the thread:

`/Root` ⟶ object 8 ⟶ `/Pages` ⟶ object 6 ⟶ `/Kids` ⟶ object 2 ⟶ page content

As soon as we add annotations, a fancy word for hyperlinks and alike, some more entries are 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`. Let's take a closer look at this stream. First 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 switch can be recognized as `/F...` The actual text goes between `()` so that it creates a PostScript string. When one analyzes a file produced by a less sophisticated typesetting engine, whole sequences of words can be recognized. In TeX however, the text comes out rather fragmented, mainly because a lot of kerning takes place. Because viewers can search in these streams, one can imagine that the average TeX produced files becomes more difficult as soon as the typesetting engine does a better job; TeX cannot do less.

This one page example uses an Adobe Times Roman font. This is one of the 14 fonts that is always present in the viewer application, and is called a base font. However, when we use for instance Computer Modern Roman, we have to make sure that this font is available, and the best way to do this is to embed it in the file. Just let your eyes follow the object thread and see how a font is described. The only thing missing in this example is the (partially) embedded glyph description file, which for the base fonts is not needed.

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 (`/Type /Page`) shows that a mediabox is part of the page description. A mediabox acts like the bounding box in a PostScript file. pdfTeX users have access to this object by `\pdfpageattr`.

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. Those who, after this introduction, feel uncomfortable in 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. Some TeX distributions have pdfTeX as a component, like teTeX, fpTeX, MikTeX and CMacTeX, so when you use one of them, you don't need to bother with the pdfTeX installation. Note that the installation description in this manual is Web2C–specific.

For some years there has been a 'moderate' successor to TeX available, called e-TeX. Because the main stream macro packages start supporting this welcome extension, pdfTeX also is available as pdfe-TeX. Although in this document we will speak of pdfTeX, we advise users to use pdfe-TeX when available. That way they get the best of all worlds and are ready for the future.

3.1 Getting sources and binaries

The latest sources of pdfTeX are distributed together with precompiled binaries of pdfTeX for some plat-

forms, including Linux[1] , SGI IRIX, Sun SPARC Solaris and MSDOS (DJGPP).[2] The primary location where one can fetch the source code (by version) is:

```
ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdftex/latest
```

Thomas Esser's TETEX distribution comes with precompiled versions for many UNIX systems. More information can be found at: http://www.tug.org/teTeX. For WIN32 systems (Windows 95, Windows NT) there are two packages that contain PDFTEX, both in ctan:systems/win32: FPTEX, maintained by Fabrice Popineau, popineau@ese-metz.fr, and MIKTEX by Christian Schenk, cschenk@berlin.snafu.de.

3.2 Compiling

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

```
cd \$HOME/pdftex
gunzip < web-7.3.tar.gz | tar xvf -
gunzip < web2c-7.3.tar.gz | tar xvf -
gunzip < pdftex.tar.gz | tar xvf -
mv pdftexdir web2c-7.3/web2c
cd ./web2c-7.3
./configure
cd ./web2c
make pdftex
```

[1] The Linux binary is compiled for the new `libc-6` (GNU `glibc-2.0`), which will not run for users of older Linux installations still based on `libc-5`.

[2] The DJGPP version is built by DJGPP 2.0 cross-compiler on Linux.

If you happen to have a previously configured source tree and just install a new version of pdfTeX, you can avoid running `configure` from the top-level directory. It's quicker to run `config.status`, which will just regenerate the `Makefile`'s based on `config.cache`:

```
cd web2c-7.3/web2c
sh config.status
make pdftex
```

For Unix users the savest way to generate binaries is to get the latest teTeX and follow the instructions that come with it.

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

`pdftex.pool` The pool file, needed for creating formats, located in `web2c-7.3/web2c`

`texmf.cnf` Web2C run-time configuration file, located in `web2c-7.3/kpathsea`

`ttf2afm` An external program to generate afm files from TrueType fonts, located in `web2c-7.3/web2c/pdftexdir`

Precompiled binaries are included in the zip archive `pdftex.zip`.

3.3 Getting PDFTeX-specific platform-independent files

Apart from the above-mentioned files, there is another zip archive (`pdftexlib.zip`) in the pdfTeX distribution which contains platform-independent files required for running pdfTeX:

- configuration file: `pdftex.cfg`
- encoding vectors: `*.enc`
- map files: `*.map`
- macros: `*.tex`

Unpacking this archive —don't forget the `-d` option when using `pkunzip`— will create a `texmf` tree containing pdfTeX-specific files.

3.4 Placing files

The next step is to place the binaries somewhere in `PATH`. If you want to use LaTeX, you also need to make a copy (or symbolic link) of `pdftex` and name it `pdflatex`. The files `texmf.cnf` and `pdftex.pool` and the directory `texmf`, created by unpacking the file `pdftexlib.zip`, should be moved to the 'appropriate' place (see below).

3.5 Setting search paths

Web2C-based programs, including pdfTeX, use the Web2C run-time configuration file called `texmf.cnf`. This file can be found via the user-set environment variable `TEXMFCNF` or via the compile-time default value if the former is not set. It is strongly recommended to use the first option. Next you need to edit `texmf.cnf` so pdfTeX can find all necessary files. Usually one has to edit `TEXMFS` and maybe some of the next variables. When running pdfTeX, some extra search paths are used beyond those normally requested by TeX itself:

`VFFONTS` Virtual fonts are fonts made up of others and `vf` files play an important role in this process. Because pdfTeX produces the final output code, it must consult those files.

`T1FONTS` Outline (vector) fonts are to be prefered over bitmap pk fonts. In most cases Type 1 fonts are used and this variable tells pdfTeX where to find them.

`TTFONTS` Like Type 1 fonts, TrueType fonts are also outlines.

`MISCFONTS` pdfTeX is able to read so called `pdf glyph container` files. These contain fonts descriptions in pdf format. (A separate manual will be made available soon.)

used for	format	texmf.cnf
virtual fonts	kpse_vf_format	VFFONTS
type1 fonts	kpse_type1_format	T1FONTS
truetype fonts	kpse_truetype_format	TTFONTS
pgc fonts	kpse_miscfonts_format	MISCFONTS
pdftex.cfg	kpse_tex_format	TEXINPUTS
images	kpse_tex_format	TEXINPUTS
map files	kpse_tex_ps_header_format	TEXPSHEADERS
encoding files	kpse_tex_ps_header_format	TEXPSHEADERS

Table 1 The WEB2C variables.

`PKFONTS` Unfortunately bitmap fonts are displayed poorly by 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 them).

`TEXINPUTS` This variable specifies where PDFTEX finds its configuration file and input files. Being a postprocessor too, image files are considered input files and searched for along this path.

`TEXPSHEADERS` This is the path where PDFTEX looks for the font mapping files (`*.map`) and encoding files (`*.enc`). Both types provide PDFTEX the information needed for embedding font encoding vectors and font resources.

3.6 The PDFTEX configuration file

One has to keep in mind that, opposed to DVI output, there is no postprocessing stage. This has several rather fundamental consequences, like one–pass graphic and font inclusion. When TEX builds a page, the macro

package used quite certain has a concept of page dimensions, which is not the same as paper dimensions. The reference point of the page is the top–left corner.

Most DVI postprocessors enable the user to specify the paper size, which often defaults to 'A4' or 'letter'. In most cases it does not harm that much to mix the two, because one will seldom put too small paper in the printer. And, if one does, one will certainly not do that a second time. In PDF the paper size is part of the definition. This means that everything that is off page, is clipped off, it simply disappears. Even worse, just like in a POSTSCRIPT file, the reference point is in the lower corner, which is opposite to DVI's reference point.

And so, we've found one of the main reasons why PDFTEX explicitly needs to know the paper dimensions. These dimensions can either be passed using the so called configuration file, or by using the primitives provided for this purpose. In this respect, the PDFTEX configuration file can be compared to configuration files that come with DVI postprocessors and/or command line options. Both contain information on the paper used, the fonts to be included and optimizations to be applied.

When PDFTEX is run in ini–mode, which is normally the case when we generate a format file, the configuration file is not read at all, and all configuration parameters are set to 0 by default for both integer and dimension parameters.

When PDFTEX is launched in non–ini mode, it reads the WEB2C configuration file as well as the PDFTEX configuration file called `pdftex.cfg`, searched for in the `TEXINPUTS` path. As WEB2C systems commonly specify a 'private' tree for PDFTEX where configuration and map files are located, this allows individual users or projects to maintain customized versions of the configuration file. The configuration file musts exist when PDFTEX is run in non–ini mode.

The integer configuration parameters replace the corresponding internal ones just before PDFTEX starts reading the input file. At this moment the format is already loaded, so any former settings during creating formats will be overwritten by the values from config file. So, unless the macro package used resets

\pdfoutput, pdfTeX will produce PDF output! Macros (packages) that adapt themselves to either DVI (using specials) or PDF (dedicated primitives) should be aware of this.

When at the moment the first page is shipped out \pdfoutput has positive value, the configuation parameters that are dimension overwrite only the corresponding internal ones that are 0. The value of \pdfoutput cannot be changed after the first page has been shipped out.

Most parameters in the configuration file have a corresponding internal register. When not set during the TeX run, pdfTeX uses thevalues as specified in the configuration file.

internal name	parameters	type
\pdfoutput	output_format	integer
\pdfadjustspacing	adjust_spacing_level	integer
\pdfcompresslevel	compress_level	integer
\pdfdecimaldigits	decima_digits	integer
\pdfmovechars	move_chars	integer
\pdfimageresolution	image_resolution	integer
\pdfpkresolution	pk_resolution	integer
\pdfhorigin	horigin	dimension
\pdfvorigin	vorigin	dimension
\pdfpageheight	page_height	dimension
\pdfpagewidth	page_width	dimension
\pdflinkmargin	link_margin	dimension
\pdfthreadmargin	thread_margin	dimension

Figure 1 The configuration parameters.

Apart from the above described parameters, the configuration file can have another entry named map. This

entry specifies the font mapping files, which is similar to those used by many DVI to PostScript drivers. More than one map file can be specified, using multiple `map` lines. If the name of the map file is prefixed with a +, its values are appended to the existing set, otherwise they replace it. If no map files are given, the default value `psfonts.map` is used.

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

```
output_format       1           % the implicit output will be PDF
compress_level      1           % use the fastest level of compression
decimal_digits      3           % max. 3 digits after the decimal point
image_resolution    300         % when not specified, embed images at 300 DPI
pk_resolution       600         % use PK fonts at 600 DPI
move_chars          1           % move chars in 0..31 to higher area
page_width          210truemm   % A4 paper width
page_height         297truemm   % A4 paper height
horigin             1truein     % horizontal origin offset
vorigin             1truein     % vertical origin offset
map                 pdftex.map  % standard map file
map                 +misc.map   % map file for extra fonts
```

The configuration file sets default values for these parameters, and apart from he `map` entry, they all can be over–ridden in the TeX source file. Dimensions can be specified as `true`, which makes them immune for magnification (when set).

output_format This integer parameter specifies whether the output format should be DVI or PDF. A positive value means PDF output, otherwise we get DVI output.

compress_level This integer parameter specifies the level of text and in–line graphics compression. pdfTeX uses ZIP compression as provided by `zlib`. A value of 0 means no compression, 1 means fastest, 9 means

best, 2..8 means something in between. Just set this value to 9, unless there is a good reason to do otherwise — 0 is great for testing macros that use `\pdfliteral`.

decimal_digits This integer specifies the preciseness of real numbers in PDF page descriptions. It gives the maximal number of decimal digits after the decimal point of real numbers. Valid values are in range 0..5. A higher value means more precise output, but also results in a much 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.

image_resolution When pdfTeX is not able to determine the natural dimensions of an image, it assumes a resolution of type 72 dots per inch. Use this variable to change this default value.

pk_resolution One can use this entry to specify the resolution for bitmap fonts. Nowadays most printers are capable to print at least 600 dots per inch, so this is a reasonable default.

move_chars Although PDF output is claimed to be portable, especially when all font information is included in the file, problems with printing and viewing have a persistent nature. Moving the characters in range 0-31 sometimes helps a lot. When set to 1, characters are only moved when a font has less than 128 glyphs, when set to 2 higher slots are used too.

page_width & page_height These two dimension parameters specify the output medium dimensions (the paper, screen or whatever the page is put on). If they are not specified, the page width is calculated as $w_{\text{box being shipped out}} + 2 \times (\text{horigin} + \texttt{\hoffset})$. The page height is calculated in a similar way.

horigin & vorigin These dimension parameters can be used to set the offset of the TeX output box from the top left corner of the 'paper'.

map This entry specifies the font mapping file, which is similar to those used by many DVI to PostScript drivers. More than one map file can be specified, using multiple `map` lines. If the name of the map file is prefixed with a +, its values are appended to the existing set, otherwise they replace it. If no map files are given, the default value `psfonts.map` is used.

3.7 Creating formats

Formats for pdfTeX are created in the same way as for TeX. For plain TeX and LaTeX it looks like:

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

In ConTeXt the generation depends on the interface used. A format using the english user interface is generated with

```
pdftex -ini -fmt=cont-en 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 --tex=pdfetex en de
```

for an english and german one, using pdfe-TeX. Indeed, if there is pdfTeX as well as pdfe-TeX, use it! Whatever macro package used, the formats should be placed in the `TEXFORMATS` path. We strongly recommend to use pdfe-TeX, if only because the main stream macro packages (will) use it.

3.8 Testing the installation

When everything is set up, you can test the installation. In the distribution there is a plain TeX test file `example.tex`. Process this file by saying:

```
pdftex example
```

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

3.9 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` or `pdfetex.pool` when you use pdfe-TeX.

- `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'!`

  Format 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`.

- `Fatal format file error; I'm stymied.`

  This appears if you forgot to regenerate the `.fmt` files after installing a new version of the pdfTeX binary and `pdftex.pool`.

- `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.

- PDFTEX cannot find the configuration file `pdftex.cfg`, one or more map files (`*.map`), encoding vectors (`*.enc`), virtual fonts, Type 1 fonts, TrueType 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.

Normally the page content takes one object. This means that one seldom finds more than a few hundred objects in a simple file. This document for instance uses about 300 objects. In demanding applications this number can grow quite rapidly, especially when one uses a lot of widget annotations, shared annotations or other shared things. In these situations in `texmf.cnf` one can enlarge PDFTEX's internal object table by adding a line in `texmf.cfg`, for instance:

```
obj_tab_size = 400000
```

4 Macro packages supporting PDFTEX

When producing DVI output, for which one can use PDFTEX as well as any other TEX, part of the job is delegated to the DVI postprocessor, either by directly providing this program with commands, or by means of `\specials`. Because PDFTEX directly produces the final format, it has to everything itself, from handling color, graphics, hyperlink support, font–inclusion, upto page imposition and page manipulation.

As a direct result, when one uses a high level macro package, the macros that take care of these features have to be set up properly. Specials for instance make no sense at all. Actually being a comment understood by DVI postprocessors —given that the macro package speaks the specific language of this postprocessor—

a \special would end up as just a comment in the PDF file, which is of no use. Therefore, \special issues a warning when pdfTeX is in PDF mode.

When one wants to get some insight to what extend pdfTeX specific support is needed, one can start a file by saying:

```
\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 main stream macro packages offer pdfTeX support in one way or the other. When using such a package, it makes sense to turn on this support in the appropriate way, otherwise one cannot be sure if things are set up right. Remember that for instance the page and paper dimensions have to be taken care of, and only the macro package knows the details.

- For LaTeX users, Sebastian Rahtz' `hyperref` package has substantial support for pdfTeX, and provides access to most of its features. In the simplest case, the user merely needs to load `hyperref` with a `pdftex` option, and all cross–references will be converted to PDF hypertext links. PDF output is automatically selected, 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` and `color` packages have `pdftex` options, which allow use of normal color, text rotation, and graphics inclusion commands.
- The ConTeXt macro package by Hans Hagen (pragma@wxs.nl) has very full support for pdfTeX in its generalized hypertext features. Support for pdfTeX is implemented as a special driver, and is invoked by saying `\setupoutput[pdftex]` or feeding TeXexec with the `--pdf` option.

- Hypertexted PDF from `texinfo` documents can be created with `pdftexinfo.tex`, which is a slight modification of the standard `texinfo` macros. This file is part of the pdfTeX distribution.
- A similar modification of `webmac.tex`, called `pdfwebmac.tex`, allows production of hypertext'd PDF versions of programs written in WEB. This is also part of the pdfTeX distribution.

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

5 Setting up fonts

pdfTeX can work with Type 1 and TrueType fonts, but a source must be available for all fonts used in the document, except for the 14 base fonts supplied by Acrobat Reader (Times, Helvetica, Courier, Symbol and Dingbats). It is possible to use METAFONT-generated fonts in pdfTeX— but it is strongly recommended not to use METAFONT-fonts if an equivalent is available in Type 1 or TrueType format, if only because bitmap Type 3 fonts render very poorly in Acrobat Reader. Given the free availability of Type 1 versions of all the Computer Modern fonts, and the ability to use standard PostScript fonts, most TeX users should be able to experiment with pdfTeX.

5.1 Map files

pdfTeX reads the map files, specified in the configuration file, see section 3.6, in which reencoding and partial downloading for each font are specified. Every font needed must be listed, each on a separate line, except PK fonts. The syntax of each line is similar to `dvips` map files[3] and can contain up to the following (some are optional) fields: *texname*, *basename*, *fontflags*, *fontfile*, *encodingfile* and *special*. The only mandatory is *texname* and must be the first field. The rest is optional, but if *basename* is given, it must be the second

[3] `dvips` map files can be used with pdfTeX without problems.

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 *fontfile*, *encodingfile* and *special*, however the first three fields must be given in fixed order.

texname sets the name of the TFM file. This name must be given for each font.

basename sets the base (POSTSCRIPT) font name. If not given then it will be taken from the font file. Specifying a name that doesn't match the name in the font file will cause PDFTEX to write a warning, so it is best not to have this field specified if the font resource is available, which is the most common case. This option is primarily intended for use of base fonts and for compatibility with `dvips` map files.

fontflags specify some characteristics of the font. The next description of these flags are taken, with a slight modification, from the PDF Reference Manual (the section on Font Descriptor Flags).

> 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 2 specifies the meanings of the bits, with bit 1 being the least significant. Reserved bits must be set to zero.
>
> 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 the Adobe Standard Roman Character Set, or a subset of that, and that it uses the standard names for those characters.

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 2 The meaning of flags in the font descriptor.

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

If the font flags are not given, pdfTeX treats it as being 4, a symbolic font. If you do not know the correct value, it would be best not to specify it, 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).

fontfile sets the name of the font source file. This must be a Type 1 or TrueType font file. The font file name can be preceded by one or two special characters, which says how the font file should be handled.

- If it is preceded by a < the font file will be partly downloaded, which means that only used glyphs (characters) are embedded to the font. 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. Partial fonts are included in such a way that name and cache clashes are minimalized.
- In case the font file name is preceded by a double <<, the font file will be included entirely — all glyphs of the font are embedded, including the ones that are not used in the document. Apart from causing large size PDF output, this option may cause troubles with TrueType fonts too, so it is not recommended. It might be useful in case the font is untypical and can not be subsetted well by pdfTeX. *Beware: some font vendors forbid full font inclusion.*
- In case nothing preceded the font file name, the font file is read but nothing is embedded, only the font parameters are extracted to generate the so–called font descriptor, which is used by Acrobat Reader to simulate the font if needed. This option is useful only when you do not want to embed the font (i.e. to reduce the output size), but wish to use the font metrics and let Acrobat Reader 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 in case of Symbolic font, which is not very useful.
- If the font file name is preceded by a !, the font is not read at all, and is assumed to be available on the system. This option can be used to create PDF files which do not contain embedded fonts. The PDF output then works only on systems where the resource of the used font is available. It's not very useful for document exchange, as the PDF is not 'portable' at all. On the other hand it is very useful when you wish to speed up running of pdfTeX during interactive work, and only in a final version embed all used fonts. Don't over–estimate gain in speed and when distributing files, always embed the fonts! This

feature requires Acrobat Reader to have access to installed fonts on the system. This has been tested on Win95 and UNIX (Solaris).

Note that the standard 14 fonts are never downloaded, even when they are specified to be downloaded in map files.

encoding specifies the name of the file containing the external encoding vector to be used for the font. The file name may be preceded by a <, but the effect is the same. The format of the encoding vector is identical to that used by `dvips`. If no encoding is specified, the font's built-in default encoding is used. It may be omitted if you are sure that the font resource has the correct built-in encoding. In general this option is highly preferred and is *required* when subsetting a TrueType font.

special instructions can be used to manipulate fonts similar to the way `dvips` does. Currently only the keyword `SlantFont` is interpreted, other instructions are just ignored.

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)[4] . If no PGC source is available, PDFTEX will try to use PK fonts in a normal way as DVI drivers do, on-the-fly creating PK fonts if needed.

Lines containing nothing apart from *texname* 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 downloading of identical font sources. Thus vector PGC fonts should be specified as scalable Type 3 in map files like:

```
csr10
```

[4] 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 allowed in PDF page description, which may be used to support fonts that are not available in METAFONT. At the moment PGC fonts are not very useful, as vector Type 3 fonts are not displayed very well in Acrobat Reader, but it may be more useful when Type 3 font handling gets better.

It doesn't hurt much if a scalable Type 3 font is not given in map files, except that the font source will be downloaded multiple times for various sizes, which causes a much larger PDF output. On the other hand if a font is 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 font instead; the PDF output is still valid but some fonts may look ugly because of the scaled bitmap.

A SlantFont is specified similarly as for `dvips`. A `SlantFont` or `ExtendFont` must be used with embedding font file. Note that the base name, the PostScript name like Symbol or Times-Roman, cannot be given, as pdfTeX never embeds a base font.

```
psyr     Symbol
psyro    ".167 SlantFont"  <usyr.pfb
ptmr8r   Times-Roman       <8r.enc
```

To summarize this rather confusing story, we include some sample lines. First we use a built-in font with font-specific encoding, i.e. neither a download font nor an external encoding is given.

```
psyr   Symbol
pzdr   ZapfDingbats
```

Use a built-in font with an external encoding. The < preceded encoding file may be left out.

```
ptmr8r   Times-Roman   <8r.enc
ptmri8r  Times-Italic  <8r.enc
```

Use a partially downloaded font with an external encoding:

```
putr8r   Utopia-Regular  <8r.enc <putr8a.pfb
putri8r  Utopia-Italic   <8r.enc <putri8a.pfb
putro8r  Utopia-Regular  <8r.enc <putr8a.pfb   ".167 SlantFont"
```

Use some faked font map entries:

```
logo8      <logo8.pfb
logo9      <logo9.pfb
logo10     <logo10.pfb
logosl8    <logo8.pfb     ".25 SlantFont"
logosl9    <logo9.pfb     ".25 SlantFont"
logosl10   <logosl10.pfb
logobf10   <logobf10.pfb
```

Use an ASCII subset of OT1 and T1:

```
ectt1000  cmtt10  <cmtt10.map  <tex256.enc
```

Download a font entirely without reencoding:

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

Partially download a font without reencoding:

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

Do not read the font at all — the font is supposed to be installed on the system:

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

Entirely download a font with reencoding:

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

Partially download a font with reencoding:

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

Sometimes we do not want to include a font, but need to extract parameters from the font file and reencode the font as well. This only works for fonts with Adobe Standard Encoding. The font flags specify how such

a font looks like, so Acrobat Reader can generate similar instance if the font resource is not available on the target system.

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

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

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

5.2 TrueType fonts

As mentioned above, pdfTeX can work with TrueType fonts. Defining TrueType files is similar to Type 1 font. 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. Usage 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 included in `pdftex.cfg`.

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

The PostScript font name (`TimesNewRomanPSMT`) is reported by `afm2tfm`, but from pdfTeX version 0.12l onwards it may be left out.

There are two main restrictions with TrueType fonts in comparison with Type 1 fonts:

a. The special effects SlantFont/ExtendFont cannot be used.
b. To subset a TrueType font, the font must be specified as reencoded, therefore an encoding vector must be given.

6 Formal syntax specification

This sections formaly specifies the pdfTeX specific extensions to the TeX macro programming language. First we present some general definitions. All ⟨general text⟩is expanded immediately, like `\special` in traditional TeX, unless mentioned explicitly no to.

⟨general text⟩ → ⟨filler⟩ `{` ⟨balanced text⟩ ⟨right brace⟩
⟨attr spec⟩ → `attr` ⟨general text⟩
⟨object type spec⟩ → [⟨attr spec⟩] `stream` | [⟨attr spec⟩] `file`
⟨resources spec⟩ → `resources` ⟨general text⟩
⟨file spec⟩ → `file` ⟨general text⟩
⟨page spec⟩ → `page` ⟨integer⟩
⟨action spec⟩ → `user` ⟨user-action spec⟩ | `goto` ⟨goto-action spec⟩ |
    `thread` ⟨thread-action spec⟩
⟨newwindow spec⟩ → `newwindow` | `nonewwindow`
⟨user-action spec⟩ → `user` ⟨general text⟩
⟨goto-action spec⟩ → ⟨numid⟩ | ⟨nameid⟩ | ⟨page spec⟩ ⟨general text⟩ |
    ⟨file spec⟩ ⟨general text⟩ | ⟨file spec⟩ ⟨nameid⟩ [⟨newwindow spec⟩] |
    ⟨file spec⟩ ⟨page spec⟩ ⟨general text⟩ [⟨newwindow spec⟩]
⟨numid⟩ → `num` ⟨integer⟩
⟨nameid⟩ → `name` ⟨general text⟩

⟨thread-action spec⟩ → [⟨file spec⟩] ⟨numid⟩ | [⟨file spec⟩] ⟨nameid⟩
⟨dest spec⟩ → ⟨numid⟩ ⟨dest type⟩ | ⟨nameid⟩ ⟨dest type⟩
⟨dest type⟩ → `xyz` [`zoom` ⟨integer⟩] | `fitbh` | `fitbv` | `fitb` | `fith` | `fitv` | `fit`
⟨id spec⟩ → ⟨numid⟩ | ⟨nameid⟩

pdfTeX introduces the following new primitives. Each primitive is prefixed by `pdf` except for `\efcode` and the extended `font` primitive.

`\pdfoutput` (integer)
`\pdfcompresslevel` (integer)
`\pdfdecimaldigits` (integer)
`\pdfmovechars` (integer)
`\pdfpkresolution` (integer)
`\pdfpagewidth` (dimension)
`\pdfpageheight` (dimension)
`\pdfhorigin` (dimension)
`\pdfvorigin` (dimension)
`\pdfpagesattr` (tokens)
`\pdfpageattr` (tokens)
`\pdfinfo` ⟨general text⟩
`\pdfcatalog` ⟨general text⟩ [`openaction` ⟨action spec⟩]
`\pdfnames` ⟨general text⟩
`\font` [⟨font spec⟩] [`stretch` ⟨integer⟩] [`shrink` ⟨integer⟩] [`step` ⟨integer⟩]
`\pdfadjustspacing` (integer)
`\efcode` (integer)
`\pdffontname` ⟨font⟩ (expandable)
`\pdffontobjnum` ⟨font⟩ (read–only integer)
`\pdfincludechars` ⟨font⟩ ⟨general text⟩

\pdfxform [⟨attr spec⟩] [⟨resources spec⟩] ⟨box number⟩
\pdfrefxform ⟨integer⟩
\pdflastxform (read-only integer)
\pdfximage [⟨rule spec⟩] [⟨attr spec⟩] [⟨page spec⟩] ⟨file spec⟩
\pdfrefximage ⟨integer⟩
\pdflastximage (read-only integer)
\pdfimageresolution (integer)
\pdfannot [⟨rule spec⟩] ⟨general text⟩
\pdflastannot (read-only integer)
\pdfdest ⟨dest spec⟩
\pdfstartlink [⟨rule spec⟩] [⟨attr spec⟩] ⟨action spec⟩
\pdfendlink
\pdflinkmargin (dimension)
\pdfoutline ⟨action spec⟩ [count ⟨integer⟩] ⟨general text⟩
\pdfthread ⟨rule spec⟩ [⟨attr spec⟩] ⟨id spec⟩
\pdfthreadmargin (dimension)
\pdfliteral [direct] ⟨general text⟩
\pdfobj [⟨object type spec⟩] ⟨general text⟩
\pdflastobj (read-only integer)
\pdfrefobj ⟨integer⟩
\pdftexversion (read-only integer)
\pdftexrevision (expandable)

7 New primitives

Here follows a short description of new primitives added by pdfTeX. One way to learn more about how to use

these primitives is to have a look at the file `example.tex` in the pdfTeX distribution. Each pdfTeX specific primitive is prefixed by `\pdf`.

The parameters that are marked as *default: from configuration* take their value from the configuration file. Note that if the output is DVI then the dimension parameters are not set to the configuration values and not used at all. However some pdfTeX integer parameters can affect the PDF as well as DVI output (currently `\pdfoutput` and `\pdfadjustspacing`).

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 one gets DVI output. This parameter cannot be specified *after* shipping out the first page. In other words, this parameter must be set before pdfTeX ships out the first page if we want PDF output. This is the only one parameter that must be set to produce PDF output. All others are optional. This parameter cannot be set after the first page is shipped out.

When pdfTeX starts complaining about specials, one can be sure that the macro package is not aware of this mode. A simple way of making macros pdfTeX aware is:

```
\ifx\pdfoutput\undefined \newcount\pdfoutput \fi

\ifcase\pdfoutput DVI CODE \else PDF CODE \fi
```

However, there are better ways to handle these things.

▶ `\pdfcompresslevel` (integer)

This integer parameter specifies the level of text compression via `zlib`. Zero means no compression, 1 means fastest, 9 means best, 2..8 means something in between. A value out of this range will be adjusted to the nearest meaningful value. This parameter is read each time pdfTeX starts a stream.

▶ `\pdfdecimaldigits` (integer)

This parameter specifies the accuracy of real numbers as written to the in PDF file. It gives the maximal number of decimal digits after the decimal point of real numbers. Valid values are in range 0..5. A higher value means a more precise output, but also results in a much 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. 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 saying: `\twodigitMPoutput`.

▶ `\pdfmovechars` (integer)

This parameter specifies whether pdfTeX should try to move characters in range 0..31 to higher slots. When set to 1, this feature affects only to fonts that have all character codes below 128, which applies to for instance the Computer Moderd Roman fonts. When set to 2 or higher pdfTeX will try to move those characters to free slots in encoding array, even in case the font contains characters with code greater than or equal to 128. This parameter is read when pdfTeX writes a character of a font to the PDF output at which moment it has to decide whether to move the character or not.

▶ `\pdfpkresolution` (integer)

This integer parameter specifies the default resolution of embedded PK fonts and is read when pdfTeX downloads a PK font during finishing the PDF output. Currently bitmap fonts are displayed poorly, so use Type 1 fonts when available!

▶ `\pdfpagewidth` (dimension)

This dimension parameter specifies the page width of the PDF output, being the screen, the paper or whatrever the page content is put on. pdfTeX reads this parameter when it starts shipping out a page. When at this moment the value is still 0, the page width is calculated as $w_{\text{box being shipped out}} + 2 \times (\text{horigin} + \texttt{\textbackslash hoffset})$.

Like the next one, this value replaces the value set in the configuration file. When part of the page falls of the paper or screen, you can be rather sure that this parameter is set wrong.

▶ `\pdfpageheight` (dimension)

Similar to the previous one, this dimension parameter specifying the page height of the PDF output. If not given then the page height will be calculated as mentioned above.

▶ `\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 shippin gout a page to the PDF outout.

▶ `\pdfvorigin` (dimension)

This parameter is the vertical alternative of `\pdfhorigin`. Keep in mind that the TEX coordinate system starts in the top left corner, while the PDF one starts at the bottom.

▶ `\pdfpagesattr` (tokens)

PDFTEX expands this toke list when it finishes the PDF output and adds the resulting character stream to the root `Pages` object. When sound, 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
    /MediaBox [0 0 612 792] } % the media size of al pages (in bp)
```

▶ `\pdfpageattr` ⟨tokens⟩

This is similar to `\pdfpagesattr`, but it takes priority to the former one. It can be used to overwrite 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.

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 by Acrobat Reader (version 3.1: menu option *Document Information, General*). 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), `/ModDate`, `/Creator` (defaults to `TeX`), `/Producer` (defaults to `pdfTeX`), `/Title`, `/Subject`, and `/Keywords`.

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

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

```
\pdfinfo
  { /Title        (example.pdf)
    /Creator      (TeX)
    /Producer     (pdfTeX 0.14a)
    /Author       (Tom and Jerry)
    /CreationDate (D:19980212201000)
```

```
/ModDate      (D:19980212201000)
/Subject      (Example)
/Keywords     (mouse,cat) }
```

▶ `\pdfcatalog` ⟨general text⟩ [`openaction` ⟨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` determines how Acrobat displays the document on startup. The possibilities for the latter are explained in Table 3:

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

Table 3 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 7.7. Instead of using this method, one can also write the open action directly into the catalog.

▶ `\pdfnames` ⟨general text⟩

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

7.3 Fonts

► `\font` [⟨font spec⟩] [`stretch` ⟨integer⟩] [`shrink` ⟨integer⟩] [`step` ⟨integer⟩]

Although still in an experimental stage, and therefore subjected to changes, the next extension to the TEX primitive `font` is worth mentioning.

```
\font\somefont=somefile at 10pt stretch 30 shrink 20 step 10
```

The `stretch 30 shrink 20 step 5` means as much as: "hey TEX, when things are going to bad, you may stretch the glyphs in this font as much as 3% or shrink them by 2%". Because pdfTEX uses internal datastructures with fixed widths, each additional width also means an additional font. For practical reasons pdfTEX uses discrete steps, in this example a 1% one. This means that for font `somefile` upto 6 differently scaled alternatives are used. When no step is specified, 0.5% steps are used.

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.

The additional fonts are named as `somefile+10` or `somefile-15`, and TFM files with these names and appropriate dimensions must be available. So, each scaled font must have its own TFM file! When no TFM file can be found, pdfTEX will try to generate it by executing the script `mktextfm`, where available and supported.

This mechanism is inspired on an optimization first introduced by Herman Zapf, which in itself goes back to optimizations used in the early days of typesetting: use different glyphs to optimize the greyness 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, sort of offending to the design. But, when using PDF, it's not that illogical at all: 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 extend mixing slightly remastered fonts can be used without violating the design. Think of an O: when simply stretched, the vertical part of the glyph becomes thicker, and looks incompatible to an unscaled original. In a multiple master, one can decide to stretch but keep this thickness compatible.

▶ `\pdfadjustspacing` (integer)

The output that pdfTeX produces is pretty compatible with the normal TeX output: TeX's typesetting engine is normally unchanged, because the optimization described here is turned of by default. At this moment there are two methods provided. When `\pdfadjustspacing` is set to 1, stretching 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 use the extended collection of TFM files that are related to the `stretch` and `shrink` settings as described in the previous section.

▶ `\efcode` (integer)

We didn't yet tell the whole story. One can imagine that some glyphs are more sensitive to scaling than others. The `\efcode` primitive can be used to influence the stretchability of a glyph. The syntax is similar to `\sfcode`, and defaults to 1000, meaning 100%.

```
\efcode`A=2500
\efcode`0=0
```

In this example an A may stretch 2.5 times as much as normal and the O is not to be stretched at all. The minimum and maximum stretch is however bound by the font specification, otherwise one would end up with more fonts inclusions than comfortable.

► `\pdffontname` ⟨font⟩ (expandable)

In PDF files produced by PDFTEX, one can recognize a font switch by the prefix `F` followed by a number, for instance `/F12` or `/F54`. This command returns the number PDFTEX uses to name a font resource, e.g. for a font named as `/F12` this command returns number 12.

► `\pdffontobjnum` ⟨font⟩ (read–only integer)

This command is similar to `\pdffontname`, but returns the object number instead of the name of a font. Use of `\pdffontname` and `\pdffontobjnum` allows user full access to all the font resources used in the document.

► `\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.

7.4 XObject forms

The next three primitives support a PDF feature called ‘object reuse’ in PDFTEX. The idea is first to create a XObject form in PDF. The content of this object corresponds to the content of a TEX box, which can also contain pictures and references to other XObject form objects as well. After that the XObject form can be used by simply referring to its object number. This feature can be useful for large documents with a lot of similar elements, as it can reduce the duplication of identical objects.

These command behave similar \pdfobj, \pdflastobj and \pdfrefobj but instead of taking raw PDF code, they take care of text typeset by TEX.

▶ \pdfxform [⟨attr spec⟩] [⟨resources spec⟩] ⟨box number⟩

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

When ⟨attr spec⟩is given, the text will be written as additional attributes of the form. The ⟨resources spec⟩is similar, but the text will be added to the resources dictionary of the form. The text given by ⟨attr spec⟩or ⟨resources spec⟩is written before other keys of the form dictionary and/or the resources dictionary and takes priority to the further ones.

▶ \pdfrefxform ⟨integer⟩

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

When issued, \pdfrefxform appends a whatsit node to the list being built. When the whatsit node is searched at shipping time, PDFTEX will write the form with number ⟨integer⟩to the PDF output if it is not written yet.

▶ \pdflastxform (read–only integer)

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

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

7.5 Graphics inclusion

PDF provides a mechanism for embedding graphic and textual objects: XObject forms. 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⟩] ⟨file spec⟩

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 of them (but not all) are given, the rest will be set to a value corresponding to the remaining ones so as to make the image size to yield the same proportion of $width : (height + depth)$ as the original image size, where depth is treated as zero. If none of them is 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 dimension of the 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.pdf}`, the graphic is scaled to fit these. Opposite to for instance the `\input` primitive, the filename is supplied between braces.

The image type is specified by the extension of the given file name, so `.png` stands for PNG image, `tif` for TIFF, and `.pdf` for PDF file. Otherwise the image is treated as PDF (`pdf`).

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.

▶ `\pdfrefximage` ⟨integer⟩

The image is kept in memory and will be written to the PDF output only when its number is refered 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 shipping 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`.

▶ `\pdfimageresolution` (integer)

This parameter specifies the default resolution of included bitmap images (PNG, TIFF, and JPEG). This parameter is read when PDFTEX creates an image via `\pdfximage`. When not given or set to 0 PDFTEX treates it as 72.

7.6 Annotations

PDF level 1.2 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` [⟨rule spec⟩] ⟨general text⟩

This command appends a whatsit node corresponding to an annotation to the the list being built. The dimensions of the annotation can be controlled via Something 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 the shipping 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).

7.7 Destinations and links

The first type of annotation mentioned before, 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 4.

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 4 The outline and destination appearances.

The specification xyzcan optionally be followed by zoom ⟨integer⟩to provide a fixed zoom–in. The Something integer is like TEX magnification, i.e. 1000 is the 'normal' page view. When zoom ⟨integer⟩is given the zoom factor changes to number, otherwise the current zoom factor is kept unchange.d

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

▶ \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 the shipping time.

Additional attributes, which are explained in great detail in the PDF Reference Manual, 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 we're dealing with a 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 perfomed 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. The default behaviour is depended on 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.

▶ `\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.

7.8 Bookmarks

▶ `\pdfoutline` ⟨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.

7.9 Article threads

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

Defined an article thread. Treads with same identifiers (spread across the document) will be joined together.

▶ `\pdfthreadmargin` ⟨dimension⟩

Specifies a margin to be added to the thread dimensions.

7.10 Miscellaneous

▶ `\pdfliteral` [`direct`] ⟨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 resets the transformation matrix before inserting ⟨general text⟩, however it 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, so this primitive behaves like `\special`.

▶ `\pdfobj` [⟨object type spec⟩] ⟨general text⟩

This command creates a raw PDF object that ends op in the PDF file as `1 0 obj << ... >> endobj`. When ⟨object type spec⟩is not given, a dictionary object with contents ⟨general text⟩is created.

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.

The object is kept in memory and will be written to the PDF output only when its number is refered 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` ⟨integer⟩

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

▶ `\pdftexversion`

Returns the version of PDFTEX multiplied by 100, e.g. for version `0.13x` it returns 13. This document is typeset with version 14.a.

▶ `\pdftexrevision`

Returns the revision of PDFTEX, e.g. for version `0.14a` it returns `a`.

8 Graphics and color

pdfTeX supports inclusion of pictures in PNG, JPEG, TIFF and PDF format. The most common technique —the inclusion of EPS figures— is replaced by PDF inclusion. EPS files can be converted to PDF by GhostScript, Acrobat Distiller or other PostScript-to-PDF convertors. The BoundingBox of a PDF file is taken from CropBox if available, otherwise from the MediaBox. 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 by Sebastian Rahtz. The TeXutil utility script that comes with ConTeXt can so a similar job. (Concerning this conversion, it handles complete directories, removes some garbage from files, takes precautions against duplicate conversion, etc.)

Other alternatives for graphics in pdfTeX are:

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

Xy-pic If the PostScript 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 (written by Hans Hagen and Tanmoy Bhattacharya) is implemented as a set of macros which reads METAPOST output and supports all of its features.

PDF It is possible to insert arbitrary one-page-only PDF files, with their own fonts and graphics, into a document. The front page of this document is an example of such an insert, it is an one page document generated by pdfTeX.

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

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

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
CMACTEX	MACINTOSH WEB2C distribution
CONTEXT	general purpose macro package
DJGPP	DJ Delorie's GNU Programming Platform
DVI	natural TEX Device Independ fileformat
EPS	Encapsulated PostScript
EPSTOPDF	EPS to PDF conversion tool
E-TEX	an extension to TEX
FPTEX	WIN32 WEB2C distribution
GNU	GNU's Not Unix
JPEG	Joined Photographic Expert Group
LATEX	general purpose macro package
METAFONT	graphic programming environment, bitmap output
METAPOST	graphic programming environment, vector output
MIKTEX	WIN32 distribution

MSDOS	Microsoft DOS platform (Intel)
PDF	Portable Document Format
PDFE-TEX	E-TEX extension producing PDF output
PDFTEX	TEX extension producing PDF output
PERL	Perl programming environment
PGC	PDF glyph container
PK	Packed Bitmap Font
PNG	Portable Network Graphics
POSTSCRIPT	PostScript
RGB	Red Green Blue color specification
TETEX	UNIX WEB2C distribution
TEX	typographic language and program
TEXEXEC	CONTEXT command line interface
TEXUTIL	CONTEXT utility tool
TFM	TEX Font Metrics
TIFF	Tagged Interchange File Format
TUG	TEX Users Group
UNIX	Unix platform
URL	Uniform Resource Locator
WEB	literate programming environment
WEB2C	official multi–platform WEB environment
WIN32	Microsoft Windows platform
ZIP	compressed file format