PDFTEX

users manual

Hàn Thê Thành Sebastian Rahtz Hans Hagen Hartmut Henkel

Contents

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 standard 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).

PDFTEX is based on the original TEX sources and WEB2C, and has been successfully compiled on UNIX, WINDOWS and MS-DOS systems. It is actively maintained, with new features trickling in. Great care is taken to keep new PDFTEX versions backward compatible with earlier ones.

A 'conservative' successor to TEX, named ε-TEX, was developed in the 1990s. Since PDFTEX version 1.40, the ε-TEX extensions are always part of the PDFTEX engine, so `pdfetex` is now simply a link to `pdftex`; they have identical behavior. For documentation on the ε-TEX extensions, see https://ctan.org/pkg/etex.

Furthermore, PDFTEX itself has acquired plenty of extensions over the years which are not related specifically to PDF output, generally new primitives for various features that are inconvenient or impossible to implement at the TEX level. Many of these extensions have been adopted across all engines (not necessarily by the same name), and some are or will be required by LaTeX. Therefore, `etex` is also a link to `pdftex`, the difference being only whether DVI or PDF output is generated by default.

Other extensions are MLTeX and ENCTeX; these are also included in the current PDFTeX code, although are little used for new documents.

PDFTeX is maintained by Hàn Thế Thành and others. The PDFTeX homepage is `http://www.pdftex.org`. Please send PDFTeX comments and bug reports to the mailing list pdftex@tug.org (`https://lists.tug.org/pdftex`).

1.1 About this manual

This manual revision (849) is intended to cover PDFTeX development up to version 1.40.22. The primary repository for the manual and its sources is at `http://foundry.supelec.fr/projects/pdftex`. Copies in PDF format can also be found on CTAN via `https://ctan.org/pkg/pdftex`.

Thanks to the many people who have contributed to the manual. Improvements are always possible, and bugs not unlikely. 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, from the corresponding source on the next page. Since compression is not enabled, such a PDF file is rather verbose and readable. The first line specifies the version used. 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 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 in 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; you can't 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 entries to page objects with 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 its number. So

```
\pdfobj{<< /Type/ExtGState /LW 2 >>}
```

inserts an object into the PDF file (it creates a "graphics state" object setting the line width to 2 units), and \pdflastobj now 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.

Of course, this is just the barest introduction to PDF format. For those who want to learn more about the gory PDF details, the best bet is to read the *PDF Reference*. You can download this book as a big PDF file from Adobe's PDF Technology Center, https://www.adobe.com/devnet/pdf/pdf_reference.html — or get the heavy paper version.

We now turn to specifics of 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, MiKTeX, proTeXt, and MacTeX. The ready to run TeX Live distribution comes with pdfTeX versions for many Unix, Windows, and Mac OS X systems; more information can be found at `https://tug.org/texlive`. There are also Windows-specific distributions which contain pdfTeX, under `http://mirror.ctan.org/systems/win32`: MiKTeX by Christian Schenk, and proTeXt (based on MiKTeX) by Thomas Feuerstack. 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, 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 distributed for compilation on Unix systems (including gnu/Linux), and Windows systems. The primary home page is `http://www.pdftex.org`, where you also find bug tracking information. Development sources are at `http://foundry.supelec.fr/projects/pdftex`. Precompiled pdfTeX binaries for various systems might be available in subdirectories below `http://mirror.ctan.org/systems`, or via TeX distribution web pages.

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 pdftex.../source
./build-pdftex.sh
```

The binary `pdftex` is then built in the subdirectory `build/texk/web2c`.

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

As well as the main `pdftex` binary, binaries for the utilities `pdftosrc` and `ttf2afm` are generated.

Incidentally, for pdfTeX maintains, a sibling script to `build-pdftex.sh` is included, namely `sync-pdftex.sh`, which syncs changes from a TeX Live source repository to a pdfTeX source repository. Read the script before using it. And don't use it unless you understand what you read.

3.3 Placing files

The next step is to put the freshly compiled `pdftex`, `pdftosrc`, and `ttf2afm` binaries into the binary directory (e.g. for a typical TeX Live system, and on the appropriate platform) `/usr/local/texlive/2021/bin/x86_64-linux`.

If you're doing this into a live hierarchy, 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, `texmf.cnf` is typically located either in the directory `texmf-dist/web2c`. The path to file `texmf.cnf` can also be set up by the environment variable `TEXMFCNF`.

The configuration files in the major TeX distributions (such as `texmf.cnf` in TeX Live) should already be set up for normal use, so you shouldn't need to edit it. You might still like to read it to see where the various bits and pieces are going.

pdfTeX uses the search path variables shown in table 1, among others.

used for	texmf.cnf
output files	`TEXMFOUTPUT`
input files, images	`TEXINPUTS`
format files	`TEXFORMATS`
TeX pool files	`TEXPOOL`
encoding files	`ENCFONTS`
font map files	`TEXFONTMAPS`
TFM files	`TFMFONTS`
virtual fonts	`VFFONTS`
Type 1 fonts	`T1FONTS`
TrueType fonts	`TTFONTS`
OpenType fonts	`OPENTYPEFONTS`
bitmap fonts	`PKFONTS`

Table 1 The principal WEB2C variables.

`TEXMFOUTPUT` Normally, PDFTeX puts its output files in the current directory, overridden by the `-output-directory` option. If any output file cannot be opened there, it tries to open it in the environment variable `TEXMFOUTPUT`, if that is set. There is no default value for that variable. For example, if `TEXMFOUTPUT` has the value `/tmp`, and you run `pdftex paper` when the current directory is not writable, PDFTeX attempts to create `/tmp/paper.log` (and `/tmp/paper.pdf`, etc.)

`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; no longer used, since the pool file (program strings) are compiled into the binary.

`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.
`TTFONTS`, `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).
`TEXFONTS`	Fallback for all the font paths, so that if you want to look in a particular directory for fonts on a given run, you can set that one variable.

Many more variables may be consulted, and there are many details to file name lookups. See the Kpathsea manual (`https://tug.org/kpathsea`).

3.5 The PDFTEX configuration

We must keep in mind that, as opposed to TEX with its DVI output, the PDFTEX program does not have a separate postprocessing stage to transform the TEX input into final PDF. 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 (1in,1in) offset from 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 to 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.

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. string pool and hash size), and a few other general parameters.

3.6 Creating format files

The PDFTEX engine supports building formats for DVI and PDF output in the same way as the classical TEX engine does for DVI. Format generation (and other `initex` features) 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 invocation of PDFTEX with this format starts in the preselected mode (which still can be overridden). 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 has been traditional for many years to generate `etex.fmt` with

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	
`\pdfgentounicode`	integer	0	
`\pdfmajorversion`	integer	1	output PDF 1.4 by default
`\pdfminorversion`	integer	4	PDF 1.4
`\pdfpagebox`	integer	0	
`\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.

PDFTEX rather than the original ε-TEX, because PDFTEX contains a few additional programming and other non-PDF-related

```
% tex-ini-files 2016-04-15: pdftexconfig.tex

% Load shared (PDF) settings in pdfTeX

% Enable PDF output
\pdfoutput          = 1

% Paper size: dimensions given in absolute terms
\pdfpageheight      = 11 true in
\pdfpagewidth       = 8.5 true in

% Enable PDF 1.5 output and thus more compression
\pdfminorversion    = 5
\pdfobjcompresslevel = 2

% Low-level settings unlikely ever to need to change
\pdfcompresslevel   = 9
\pdfdecimaldigits   = 3
\pdfpkresolution    = 600
\pdfhorigin         = 1 true in
\pdfvorigin         = 1 true in
```

Figure 1 PDFTEX configuration file for TEX LIVE (`pdftexconfig.tex`).

```
% Thomas Esser, 1998. public domain.
\input etex.src
\dump
\endinput
```

Figure 2 File `etex.ini` for the plain ε-TEX format with DVI output.

features on which people have come to rely.

```
% Thomas Esser, 1998. public domain.
% This is used for pdftex and pdfetex, which are now identical: both
% with e-TeX extensions, both with pdf output.
\input pdftexconfig.tex
\input etex.src
\input pdftexmagfix.tex
\dump
\endinput
```

Figure 3 File `pdfetex.ini` for plain ε-TeX with PDF output.

```
% Thomas Esser, 1998. public domain.
\input pdftexconfig.tex
\scrollmode
\input latex.ltx
\endinput
```

Figure 4 File `pdflatex.ini` for the LaTeX format with PDF output.

The `pdfetex.ini` file figure 2 shows the corresponding format with PDF output by default; this is what creates the format file read when `pdftex` is normally invoked.

Finally, `pdflatex.ini` figure 4 shows how the LaTeX format with PDF output by default is generated.

The corresponding PDFTeX calls for format generation are:

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

These calls produce format files `etex.fmt`, `pdfetex.fmt`, and `pdflatex.fmt`, as the default format file name is taken from the input file name. You can override this with the `-jobname` option. The asterisk `*` before the file name is an unusual flag, only used in `-ini` mode, which causes the PDFTeX engine to enable ε-TeX's features.

Incidentally, as of pdfTEX 1.40.21 (TEX Live 2020), `.fmt` files are compressed with `zlib`. This makes for a considerable savings in space, and consequently in time.

3.7 Testing the installation

When everything is set up, you can test the installation. A simple test of plain pdfTEX is:

```
pdftex story \\end
```

This should typeset the famous one-page short story by A.U. Thor.

A more thorough and descriptive test is the plain TEX test file `samplepdf.tex`, available in the distribution in the `samplepdftex/` directory. Process this file by typing:

```
pdftex samplepdf
```

If the installation is ok, this should produce a file called `samplepdf.pdf`. The file `samplepdf.tex` is a good place to look for examples of 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 use the option `-kpathsea-debug 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 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`.

- `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. The first line tells by which engine the offending format was generated.

- 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 seldom 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 Invoking PDFTEX

PDFTEX has many command line options. Except for the simple and rarely-used `-draftmode` and `-output-format` options, they are all inherited from the common framework for TEX engines as implemented in WEB2C (its manual is available at `https://tug.org/web2c`).

The same commonality holds for environment variables; see the section "Setting search paths" above for an overview. Two additional environment variables need more description here: first, `SOURCE_DATE_EPOCH` (introduced in version 1.40.17, 2016). If this is set, it must be a positive integer (with one trivial exception: if it is set but empty, that is equivalent to 0). Non-integer values cause a fatal error. The value is used as the current time in seconds since the usual Unix "epoch": the beginning of 1970-01-01, UTC. Thus, a value of 32 would result in a `/CreationDate` and `/ModDate` values of 19700101000032Z. This is useful for reproducible builds of documents. (See also \pdfinfoomitdate, \pdfsuppressptexinfo, et al.)

The second, related, environment variable is FORCE_SOURCE_DATE. If this is set to 1, TeX's time-related primitives are also initialized from the value of SOURCE_DATE_EPOCH. These primitives are \year, \month, \day, and \time. If SOURCE_DATE_EPOCH is not set, setting FORCE_SOURCE_DATE has no effect. If FORCE_SOURCE_DATE is unset, set to the empty string, or set to 0, the primitives reflect the current time as usual. Any other value elicits a warning, and the current time is used. This is useful if one wants to make reproducible PDFs for a set of documents without changing them in any way, e.g., an operating system distribution with manuals that use \today. Except in such unusual circumstances, it is better not to set this, and let the TeX primitives retain the meaning they have always had.

In addition, if both SOURCE_DATE_EPOCH and FORCE_SOURCE_DATE are set, \pdffilemoddate returns a value in UTC, ending in Z. (The values of the environment variables are irrelevant in this case.)

Finally, just to have the list of options and basic invocation at hand, here is a verbatim listing of the --help output. All options can be specified with one or two dashes and unambiguously abbreviated.

```
Usage: pdftex [OPTION]... [TEXNAME[.tex]] [COMMANDS]
   or: pdftex [OPTION]... \FIRST-LINE
   or: pdftex [OPTION]... &FMT ARGS
  Run pdfTeX on TEXNAME, usually creating TEXNAME.pdf.
  Any remaining COMMANDS are processed as pdfTeX input, after TEXNAME is read.
  If the first line of TEXNAME is %&FMT, and FMT is an existing .fmt file,
  use it.  Else use `NAME.fmt', where NAME is the program invocation name,
  most commonly `pdftex'.

  Alternatively, if the first non-option argument begins with a backslash,
  interpret all non-option arguments as a line of pdfTeX input.

  Alternatively, if the first non-option argument begins with a &, the
  next word is taken as the FMT to read, overriding all else.  Any
  remaining arguments are processed as above.
```

```
   If no arguments or options are specified, prompt for input.

 -cnf-line=STRING        parse STRING as a configuration file line
 -draftmode              switch on draft mode (generates no output PDF)
 -enc                    enable encTeX extensions such as \mubyte
 -etex                   enable e-TeX extensions
 [-no]-file-line-error   disable/enable file:line:error style messages
 -fmt=FMTNAME            use FMTNAME instead of program name or a %& line
 -halt-on-error          stop processing at the first error
 -ini                    be pdfinitex, for dumping formats; this is implicitly
                           true if the program name is `pdfinitex'
 -interaction=STRING     set interaction mode (STRING=batchmode/nonstopmode/
                           scrollmode/errorstopmode)
 -ipc                    send DVI output to a socket as well as the usual
                           output file
 -ipc-start              as -ipc, and also start the server at the other end
 -jobname=STRING         set the job name to STRING
 -kpathsea-debug=NUMBER  set path searching debugging flags according to
                           the bits of NUMBER
 [-no]-mktex=FMT         disable/enable mktexFMT generation (FMT=tex/tfm/pk)
 -mltex                  enable MLTeX extensions such as \charsubdef
 -output-comment=STRING  use STRING for DVI file comment instead of date
                           (no effect for PDF)
 -output-directory=DIR   use existing DIR as the directory to write files in
 -output-format=FORMAT   use FORMAT for job output; FORMAT is `dvi' or `pdf'
 [-no]-parse-first-line  disable/enable parsing of first line of input file
 -progname=STRING        set program (and fmt) name to STRING
 -recorder               enable filename recorder
 [-no]-shell-escape      disable/enable \write18{SHELL COMMAND}
```

```
-shell-restricted       enable restricted \write18
-src-specials           insert source specials into the DVI file
-src-specials=WHERE     insert source specials in certain places of
                          the DVI file. WHERE is a comma-separated value
                          list: cr display hbox math par parend vbox
-synctex=NUMBER         generate SyncTeX data for previewers according to
                          bits of NUMBER (‘man synctex’ for details)
-translate-file=TCXNAME use the TCX file TCXNAME
-8bit                   make all characters printable by default
-help                   display this help and exit
-version                output version information and exit

pdfTeX home page: <http://pdftex.org>

Email bug reports to pdftex@tug.org.
```

5 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 controlled 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 https://tug.org/texshowcase.

6 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 generated PDF. 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 poorly under enlargement.

6.1 Map files

Font map files provide the connection between TEX TFM font files and 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. Bitmap fonts can (and normally should) 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 a single 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 psname fontflags special encodingfile fontfile

It is mandatory that *tfmname* is the first field. If a *psname* is given, it must be the second field. Similarly if *fontflags* is given it must be the third field (if *psname* is present) or the second field (if *psname* is left out). The positions of *special*, *encodingfile*, and *fontfile* can be mixed.

tfmname sets the name of the TFM file for a font — the file name given in a TEX `\font` command. This name must always be given.

psname 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 *psname* 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 Type 1 fonts from embedded PDF files requires that also a Type 1 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 *psname* 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 reading applications (e.g. viewers).

It is highly recommended to use the *psname* field, but strictly speaking it is optional.

fontflags optionally 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.
>
> 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

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.

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, and the font is embedded, pdfTEX treats it as the value 4 (decimal, that is, bit position 3 is set), a symbolic font. For non-embedded fonts, the default value is `0x22`, a non-symbolic serif 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 trouble 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 specifies font manipulations in the same way as `dvips`. Currently only the keywords `SlantFont` and `ExtendFont` are interpreted, other instructions (notably `ReEncodeFont` and its 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 quote characters: `"`.

encodingfile specifies the name of the file containing the external encoding vector to be used for the font. The encoding file name must have the extension `.enc`, and the file name including extension must be given with either a preceding `<` character or a preceding `<[`. 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.

Starting with version 1.40.19, an encoding file can also be specified for bitmap PK fonts. In this case, it assigns the glyph names from the given encoding vector, which can be used with the `\pdfglyphtounicode` primitive (q.v.). For example:

```
\pdfglyphtounicode{ffi}{0066 0066 0069} % normally: \input glyphtounicode
\pdfgentounicode=1
\pdfmapline{cmb10 <7t.enc}
\font\cmb=cmb10 \cmb ffi
```

The result is a PDF file with a correctly-labeled `/ffi` character instead of the numeric character position in the `cmb10.tfm` (decimal 14).

fontfile sets the name of the font file to be embedded into the PDF output for a given TEX font (the *tfmname* ↔ *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 *psname* 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, specifying how to handle the font file:

- If the font file name is preceded by a `<` character, the font file will be only partially embedded in the PDF output ("subsetted"), meaning that only used glyphs are written to 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 those 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 Type 1 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 cannot be subsetted well by PDFTEX. *Beware: proprietary font vendors typically forbid full font inclusion.*

- If no special character precedes the font file name, it is ignored, with a warning (this case was deprecated in PDFTEX version 1.40.0). 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.
- Specifying the *psname* and no font file name is only useful as a last-ditch 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.

If you encounter problematic lookups, for instance if PDFTEX tries to open a `.pfa` file instead of a `.pfb`, you can add the suffix to the filename. In this respect, PDFTEX completely relies on the `kpathsea` library.

For Type 1 and TrueType fonts, the font file will be included only once in the PDF output, regardless of how many TEX `\font` instances are used in the document. For instance, given

```
\font\a = cmr12
\font\b = cmr12 at 11pt
```

the outline file `cmr12.pfb` will only be included once in the PDF, and merely scaled down to create the instance for `\b`.

If a used font is not present in the map files, PDFTEX will try to use PK fonts as most DVI drivers do, creating PK fonts on–the–fly if needed. This is the normal, and recommended, way to use bitmap fonts.

To summarize this rather confusing story, we include some example map lines. The most common way is to embed only a subset of glyphs from a font for a particular desired encoding, like this:

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

Without re–encoding it looks like this:

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

`SlantFont` and `ExtendFont` fields are specified as with `dvips`. `SlantFont` and `ExtendFont` work only with embedded Type 1 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 (not typically useful):

```
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
```

Finally, 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
```

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. The recommendation these days is to embed all fonts, even the 14 standard ones.

6.2 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 did not work before 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.

For much more about pdfTEX and TrueType fonts, including many details on handling glyph names, see “A closer look at TrueType fonts and pdfTEX”, *TUGboat* 30:1 (2009), pp. 32–34, https://tug.org/TUGboat/tb30-1/tb94thanh.pdf

7 Formal syntax specification

This section formally specifies the pdfTEX-specific extensions to the TEX macro programming language. Most primitives are prefixed by `pdf`. The general definitions and syntax rules follow after the list of primitives.

Two new units of measure were introduced in pdfTEX 1.30.0: the new Didot (1 nd = 0.375 mm) and the new Cicero (1 nc = 12 nd) (the former was proposed by ISO in 1975).

Integer registers

`\efcode` ⟨font⟩ ⟨8-bit number⟩ (integer)
`\knaccode` ⟨font⟩ ⟨8-bit number⟩ (integer)
`\knbccode` ⟨font⟩ ⟨8-bit number⟩ (integer)
`\knbscode` ⟨font⟩ ⟨8-bit number⟩ (integer)
`\lpcode` ⟨font⟩ ⟨8-bit number⟩ (integer)
`\pdfadjustinterwordglue` (integer)
`\pdfadjustspacing` (integer)
`\pdfappendkern` (integer)
`\pdfcompresslevel` (integer)
`\pdfdecimaldigits` (integer)
`\pdfdraftmode` (integer)
`\pdfforcepagebox` (integer)
`\pdfgamma` (integer)
`\pdfgentounicode` (integer)
`\pdfimageapplygamma` (integer)
`\pdfimagegamma` (integer)
`\pdfimagehicolor` (integer)
`\pdfimageresolution` (integer)
`\pdfinclusioncopyfonts` (integer)
`\pdfinclusionerrorlevel` (integer)
`\pdfinfoomitdate` (integer)
`\pdfmajorversion` (integer)
`\pdfminorversion` (integer)
`\pdfobjcompresslevel` (integer)

`\pdfomitcharset` (integer)
`\pdfoutput` (integer)
`\pdfpagebox` (integer)
`\pdfpkresolution` (integer)
`\pdfprependkern` (integer)
`\pdfprotrudechars` (integer)
`\pdfsuppressptexinfo` (integer)
`\pdfsuppresswarningdupdest` (integer)
`\pdfsuppresswarningdupmap` (integer)
`\pdfsuppresswarningpagegroup` (integer)
`\pdftracingfonts` (integer)
`\pdfuniqueresname` (integer)
`\rpcode` ⟨font⟩ ⟨8-bit number⟩ (integer)
`\shbscode` ⟨font⟩ ⟨8-bit number⟩ (integer)
`\stbscode` ⟨font⟩ ⟨8-bit number⟩ (integer)
`\tagcode` ⟨font⟩ ⟨8-bit number⟩ (integer)
`\tracinglostchars` (integer)
`\tracingstacklevels` (integer)

Read-only integers

`\pdfelapsedtime` (read–only integer)
`\pdflastannot` (read–only integer)
`\pdflastlink` (read–only integer)
`\pdflastobj` (read–only integer)
`\pdflastxform` (read–only integer)
`\pdflastximage` (read–only integer)
`\pdflastximagecolordepth` (read–only integer)
`\pdflastximagepages` (read–only integer)

`\pdflastxpos`	(read–only integer)
`\pdflastypos`	(read–only integer)
`\pdfrandomseed`	(read–only integer)
`\pdfretval`	(read–only integer)
`\pdfshellescape`	(read–only integer)
`\pdftexversion`	(read–only integer)

Dimen registers

`\pdfdestmargin`	(dimen)
`\pdfeachlinedepth`	(dimen)
`\pdfeachlineheight`	(dimen)
`\pdffirstlineheight`	(dimen)
`\pdfhorigin`	(dimen)
`\pdfignoreddimen`	(dimen)
`\pdflastlinedepth`	(dimen)
`\pdflinkmargin`	(dimen)
`\pdfpageheight`	(dimen)
`\pdfpagewidth`	(dimen)
`\pdfpxdimen`	(dimen)
`\pdfthreadmargin`	(dimen)
`\pdfvorigin`	(dimen)

Token registers

`\pdfpageattr`	(tokens)
`\pdfpageresources`	(tokens)
`\pdfpagesattr`	(tokens)

\pdfpkmode (tokens)

Expandable commands

\expanded ⟨tokens⟩ (expandable)
\ifincsname (expandable)
\ifpdfabsdim (expandable)
\ifpdfabsnum (expandable)
\ifpdfprimitive ⟨control sequence⟩ (expandable)
\input ⟨general text⟩ (expandable)
\leftmarginkern ⟨box number⟩ (expandable)
\pdfcolorstackinit [page] [direct] ⟨general text⟩ (expandable)
\pdfcreationdate (expandable)
\pdfescapehex ⟨general text⟩ (expandable)
\pdfescapename ⟨general text⟩ (expandable)
\pdfescapestring ⟨general text⟩ (expandable)
\pdffiledump [offset ⟨integer⟩] [length ⟨integer⟩] ⟨general text⟩ (expandable)
\pdffilemoddate ⟨general text⟩ (expandable)
\pdffilesize ⟨general text⟩ (expandable)
\pdffontname ⟨font⟩ (expandable)
\pdffontobjnum ⟨font⟩ (expandable)
\pdffontsize ⟨font⟩ (expandable)
\pdfincludechars ⟨font⟩ ⟨general text⟩ (expandable)
\pdfinsertht ⟨integer⟩ (expandable)
\pdflastmatch ⟨integer⟩ (expandable)
\pdfmatch [icase] [subcount ⟨integer⟩] ⟨general text⟩ ⟨general text⟩ (expandable)
\pdfmdfivesum [file] ⟨general text⟩ (expandable)
\pdfnormaldeviate (expandable)
\pdfpageref ⟨page number⟩ (expandable)

`\pdfstrcmp` ⟨general text⟩ ⟨general text⟩ (expandable)
`\pdftexbanner` (expandable)
`\pdftexrevision` (expandable)
`\pdfunescapehex` ⟨general text⟩ (expandable)
`\pdfuniformdeviate` ⟨number⟩ (expandable)
`\pdfxformname` ⟨object number⟩ (expandable)
`\pdfximagebbox` ⟨integer⟩ ⟨integer⟩ (expandable)
`\rightmarginkern` ⟨box number⟩ (expandable)

General commands

`\letterspacefont` ⟨control sequence⟩ ⟨font⟩ ⟨integer⟩
`\pdfannot` ⟨annot type spec⟩ (h, v, m)
`\pdfcatalog` ⟨general text⟩ [`openaction` ⟨action spec⟩]
`\pdfcolorstack` ⟨stack number⟩ ⟨stack action⟩ ⟨general text⟩
`\pdfcopyfont` ⟨control sequence⟩ ⟨font⟩
`\pdfdest` ⟨dest spec⟩ (h, v, m)
`\pdfendlink` (h, m)
`\pdfendthread` (v, m)
`\pdffakespace`
`\pdffontattr` ⟨font⟩ ⟨general text⟩
`\pdffontexpand` ⟨font⟩ ⟨stretch⟩ ⟨shrink⟩ ⟨step⟩ [`autoexpand`]
`\pdfglyphtounicode` ⟨general text⟩ ⟨general text⟩
`\pdfinfo` ⟨general text⟩
`\pdfinterwordspaceoff`
`\pdfinterwordspaceon`
`\pdfliteral` [⟨pdfliteral spec⟩] ⟨general text⟩ (h, v, m)
`\pdfmapfile` ⟨map spec⟩
`\pdfmapline` ⟨map spec⟩

`\pdfnames` ⟨general text⟩
`\pdfnobuiltintounicode` ⟨font⟩
`\pdfnoligatures` ⟨font⟩
`\pdfobj` ⟨object type spec⟩ (h, v, m)
`\pdfoutline` [⟨attr spec⟩] ⟨action spec⟩ [`count` ⟨integer⟩] ⟨general text⟩ (h, v, m)
`\pdfprimitive` ⟨control sequence⟩
`\pdfrefobj` ⟨object number⟩ (h, v, m)
`\pdfrefxform` ⟨object number⟩ (h, v, m)
`\pdfrefximage` ⟨object number⟩
`\pdfresettimer`
`\pdfrestore`
`\pdfrunninglinkoff`
`\pdfrunninglinkon`
`\pdfsave`
`\pdfsavepos` (h, v, m)
`\pdfsetmatrix`
`\pdfsetrandomseed` ⟨number⟩
`\pdfstartlink` [⟨rule spec⟩] [⟨attr spec⟩] ⟨action spec⟩ (h, m)
`\pdfthread` [⟨rule spec⟩] [⟨attr spec⟩] ⟨id spec⟩ (h, v, m)
`\pdftrailer` ⟨general text⟩
`\pdftrailerid` ⟨general text⟩
`\pdftstartthread` [⟨rule spec⟩] [⟨attr spec⟩] ⟨id spec⟩ (v, m)
`\pdfxform` [⟨attr spec⟩] [⟨resources spec⟩] ⟨box number⟩ (h, v, m)
`\pdfximage` [⟨rule spec⟩] [⟨attr spec⟩] [⟨page spec⟩] [⟨colorspace spec⟩] [⟨pdf box spec⟩] ⟨general text⟩ (h, v, m)
`\quitvmode`
`\special` { `pdf:` ⟨text⟩ }
`\special` { `pdf:direct:` ⟨text⟩ }
`\special` { `pdf:page:` ⟨text⟩ }
`\vadjust` [⟨pre spec⟩] ⟨filler⟩ `{` ⟨vertical mode material⟩ `}` (h, m)

General definitions and syntax rules

```
⟨general text⟩ → { ⟨balanced text⟩ }
⟨attr spec⟩ → attr ⟨general text⟩
⟨resources spec⟩ → resources ⟨general text⟩
⟨rule spec⟩ → ( width | height | depth ) ⟨dimen⟩ [ ⟨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⟩
⟨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.

8 PDFTEX primitives

Here follows a description of the primitives added by PDFTEX to the original TEX engine (other extensions by ε-TEX, MLTEX and ENCTEX are not listed). Many of these primitives are described further in the `samplepdf.tex` file in the PDFTEX distribution.

If the output is DVI, then the PDFTEX-specific dimension parameters are not used at all. However, some PDFTEX integer parameters can affect DVI as well as PDF output (specifically, `\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.

8.1 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.

► `\pdfmajorversion` (integer)

► `\pdfminorversion` (integer)

Together, these two primitives specify the PDF version for generated PDF output. The defaults compiled into the PDFTEX program are `\pdfmajorversion=1` and `\pdfminorversion=4`, thus PDF 1.4 is generated by default.

However, distributions typically alter the engine's compiled default minor version of 4 when building formats. For example, as of 2010, TEX LIVE sets `\pdfminorversion=5` when formats are built. This is so object compression can be enabled (see `\pdfobjcompress` below).

This value also defines the highest PDF version for included PDFS to be allowed without error, by default (see `\pdfinclusionerrorlevel`).

The values for both must be ≥ 1 but are not checked further. Furthermore, they are set independently; only setting `\pdfmajorversion=2` would result in PDF 2.4 output; it's necessary to additionally set `\pdfminorversion`.

If specified, these primitives must appear before any data is to be written to the generated PDF file. The `\pdfmajorversion` primitive was introduced in PDFTEX 1.40.21. `\pdfminorversion` was originally a shortened synonym of the `\pdfoptionpdfminorversion` command, which is now obsolete. The primitive was introduced in PDFTEX 1.30.0.

► `\pdfcompresslevel` (integer)

This integer parameter specifies the level of *stream* compression (text, inline 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 useful for PDF stream debugging.

► `\pdfobjcompresslevel` (integer)

This integer parameter controls the compression of *non-stream* objects. If specified, the parameter must appear before any data is written to the PDF output. The primitive was introduced in PDFTEX 1.40.0.

In the PDF-1.4 specification, non-stream objects had to be written in the PDF file as clear text, uncompressed. The PDF-1.5 specification allows collecting non-stream objects as "compressed objects" into "object stream" objects (`/Type /ObjStm`, see PDF Ref. 5th ed., sect. 3.4.6). At the end of the PDF file, an `/XRef` cross-reference stream is then written out instead of the object table. This can result in a considerably smaller PDF file, particularly if lots of annotations and links are used.

The writing of compressed objects is enabled by setting `\pdfobjcompresslevel` to a value between 1 and 3; it's disabled if 0 (default). Object compression also requires that `\pdfminorversion` ≥ 5 (or `\pdfmajorversion` ≥ 2), else a warning is given and the compression 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, and is so specified in some distributions, including TEX Live 2010 and later.

Compatibility caveat: PDF files generated with object streams enabled can't be read with (sufficiently 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 was 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` (dimen)

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` (dimen)

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` (dimen)

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` (dimen)

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.

8.2 The document info and catalog

▶ `\pdfinfo` ⟨general text⟩

This primitive allows the user to specify information for the document info dictionary; if this information is provided, it can be extracted, e.g. by the `pdfinfo` program. 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, and the primitive itself, are optional. Possible keys are:
`/Title`,
`/Author`,
`/Subject`,
`/Keywords`,
`/Producer` (defaults to `pdfTeX-1.40.22`),
`/Creator` (defaults to `TeX`),
`/CreationDate` (defaults to current date and time, with time zone),
`/ModDate` (same default),
`/Trapped` (defaults to `/False`,
`/PTEX.Fullbanner` (defaults to the `\pdftexbanner` string, q.v.).

`/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, `Z` for universal time. For example, this is the Unix epoch, the beginning of 1970-01-01 UTC, in this format: `19700101000000Z`. If the time zone is not UTC, it is given as `+HH'mm'` or `-HH'mm'`, indicating an offset of the given hours and minutes from UTC, with the given either later (`+`) or earlier (`-`) than UTC. (This syntax is specified by the PDF definition.)

Multiple appearances of `\pdfinfo` are concatenated. Usually if a key is given more than once, the first appearance will be used, but this is viewer–dependent. Except for standard TEX expansion, PDFTEX does not perform any further operations in the ⟨general text⟩ provided by the user.

Here is an example of using `\pdfinfo` to include the information not supplied by PDFTEX:

```
\pdfinfo {
    /Title          (example.pdf)
    /Author         (Tom and Jerry)
    /Subject        (Example)
    /Keywords       (mouse, cat)
}
```

For more details on all this, see the *PDF Reference*.

▶ `\pdfinfoomitdate` (integer)

If nonzero, omit the `/CreationDate` and `/ModDate` keys from the document info dictionary described above. This can be useful in making reproducible PDFS. The primitive was introduced in PDFTEX 1.40.17.

▶ `\pdfsuppressptexinfo` (integer)

Treated as a bitmask, specifying which `PTEX.*` keys to omit from the output:

value	**suppresses**
1	`PTEX.Fullbanner`
2	`PTEX.FileName`
4	`PTEX.PageNumber`
8	`PTEX.InfoDict`

Table 4 `\pdfsuppressptexinfo` bit meanings.

Thus, the value `-1`, or the sum of all defined bits, will suppress everything.

`PTEX.Fullbanner` is included by default in the general document info dictionary, as mentioned under `\pdfinfo` above. The other `PTEX.*` keys are included when a PDF is included in the document (and not otherwise), as described in section 3.

This conditional suppression can be useful in making reproducible PDFs. The primitive was introduced in pdfTEX 1.40.17.

▶ `\pdfcatalog` ⟨general text⟩ [`openaction` ⟨action spec⟩]

Similar to the document info section is the document catalog, where possible keys are `/URI`, which specifies 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 5:

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

Table 5 Supported `/PageMode` values.

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

After the ⟨general text⟩ , a construct `openaction` ⟨action spec⟩ can be given, where `openaction` is a pdfTEX keyword, and ⟨action spec⟩ specifies the action to be taken when opening the document. This ⟨action spec⟩ is the same as for internal links; see section 8.11. (Instead of using this method, one can also write the open action directly into the catalog.)

Several settings can be made in one `\pdfcatalog` call, for example:

```
\pdfcatalog{
  /PageMode /FullScreen
} openaction goto page 2 {/Fit}
```

▸ `\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. Example: `\pdftrailer {/mytrlrkey /mytrlrval}`. The primitive was introduced in PDFTEX 1.11a.

▸ `\pdftrailerid` ⟨general text⟩

Use the ⟨general text⟩ to seed the `/ID` value in the trailer, instead of the default combination of the input file name and starting time. If the argument is empty, the `/ID` is omitted entirely. Example: `\pdftrailerid{}`. This can be useful in making reproducible PDFS. The primitive was introduced in PDFTEX 1.40.17.

8.3 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 major PDFTEX feature 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, `\pdfadjustspacing` must be set to a value greater than zero.

There are two different modes for font expansion:

First, if the `autoexpand` option is given — 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 be mentioned (they are ignored), as PDFTEX generates expanded instances 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 at the font file level, giving the advantage that it now works not only with Type 1 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 providing 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 typically not worth the trouble.

A caveat: when `\pdffontexpand` is executed, PDFTEX immediately loads two fonts, at the maximum stretch and shrink; in our example, `sometfm+30` and `sometfm-20`. (If they aren't available, `mktextfm` may be uselessly called, and then an error message issued.) This happens even if those fonts never end up being used, which is arguably undesirable, but hard to change. It is not a problem when using `autoexpand`.

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 support arbitrary scaling, after all. (Also, they used to 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. Perhaps something similar happens with TrueType and OpenType fonts nowadays.)

► `\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 behavior.

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⟩ ⟨8-bit number⟩ (integer)

We haven't yet told 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.

Changes to this table are global, i.e., ignore TEX's usual grouping, and apply only to the given ⟨font⟩ .

▶ `\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⟩ ⟨8-bit number⟩ (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.

Changes to this table are global, i.e., ignore TEX's usual grouping, and apply only to the given ⟨font⟩ .

▶ `\lpcode` ⟨font⟩ ⟨8-bit number⟩ (integer)

This is similar to `\rpcode`, but affects the amount by which characters may protrude into the left margin. Here also the given integer value is clipped to the range $-1000..1000$.

Changes to this table are global, i.e., ignore TEX's usual grouping, and apply only to the given ⟨font⟩ .

▶ `\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 unpacking 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.

▶ `\letterspacefont` ⟨control sequence⟩ ⟨font⟩ ⟨integer⟩

This primitive creates an instance of ⟨font⟩ with the widths of all glyphs increased by ⟨integer⟩ thousandths of an em (as set in ⟨font⟩). The effect is letter spacing, but the glyphs are actually larger (sidebearings are increased), so a single glyph will take more space. For instance, the following creates a font `\lsfont` whose glyphs are all 1.2 pt larger than those of `\normalfont`:

```
\font\normalfont=myfont at 12pt
\letterspacefont\lsfont\normalfont 100
```

Negative values are allowed for ⟨integer⟩ . Letter spacing works natively in PDF mode only, unless special fonts are devised (in our example, a `myfont+100ls` font), as with font expansion.

▶ `\pdfcopyfont` ⟨control sequence⟩ ⟨font⟩

This primitive defines ⟨control sequence⟩ as a synonym for ⟨font⟩ .

▶ `\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. For examples, see the `cmap` and `CJK` packages.

▶ `\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⟩ (expandable)

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.

▶ `\pdfomitcharset` (integer)

If this primitive parameter is zero (the default), the `/CharSet` entry is included as usual for fonts in the PDF output; if it is set to 1, then `/CharSet` is omitted. (Other values may have other meanings in the future, so do not rely on them.)

Explanation: This parameter was created because the PDF/A-1 standard requires `/CharSet`, whereas PDF/A-2 and PDF/A-3 allow it to be omitted but have extraordinary requirements, which PDFTeX does not currently meet, if it is included.The primitive was introduced in PDFTeX 1.40.20.

▶ `\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. 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 behavior. 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.

▶ `\pdfsuppresswarningdupmap` (integer)

Ordinarily, PDFTEX gives a warning when a duplicate map entry for a given font is read, whatever the mechanism. However, sometimes it is useful to include map information within the document, using `\pdfmapfile` or `\pdfmapline`, even for fonts that happen to be installed on the system. Then seeing the warnings on every run is just noise, and can be suppressed by setting this parameter to a positive number. The primitive was introduced in PDFTEX 1.40.13.

▶ `\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 disables all ligatures in the loaded font ⟨font⟩ . The primitive was introduced in pdfTeX 1.30.0.

▶ `\tagcode` ⟨font⟩ ⟨8-bit number⟩ (integer)

This primitive accesses a character's `char_tag` info. It is meant to delete `lig_tag` (the character's ligature/kerning program), `list_tag` (which indicates that the character belongs to a chain of ascending sizes) and/or `ext_tag` (which indicates that the character is extensible), with the following options: assigning `-7` or smaller clears all tags, `-6` clears `ext_tag` and `list_tag`, `-5` clears `ext_tag` and `lig_tag`, `-4` clears `ext_tag`, `-3` clears `list_tag` and `lig_tag`, `-2` clears `list_tag`, `-1` clears `lig_tag`, and `0` or larger does nothing. Changes are irreversible and global.

Conversely, when queried, the primitive returns `0` if the tag's value is `no_tag`, `1` if `lig_tag` is set, `2` if `list_tag` is set or `4` (not 3) if `ext_tag` is set.

▶ `\pdfgentounicode` (integer)

By default, pdfTeX does not include a `/ToUnicode` resource when including fonts in the output. Such a resource (also called a CMap resource) maps glyph names to Unicode characters in a PDF file. Lacking such a resource, it is the PDF reader which determines how and whether searching in the PDF file works. In practice, searching for basic ASCII characters generally works, but searching for anything beyond those, including ligatures such as 'fi', fails in some versions of some PDF browsers (most notably Adobe Reader).

If `\pdfgentounicode` is set to `1` when the job ends, the `/ToUnicode` resource will be included in the output, with mappings for Type 1 fonts used in the documents. The mapping is created as follows: for each glyph in the font, look for its `ToUnicode` value in a global hash table. By default that global hash table is empty, in which case pdfTeX merely emits a warning. Entries are added to the table using the following command:

▶ `\pdfglyphtounicode` ⟨general text⟩ ⟨general text⟩

The first argument is the name of a glyph, the second is a string of Unicode numeric values denoting characters, separated by spaces. For instance:

```
\pdfgentounicode=1
\pdfglyphtounicode{ff}{0066 0066}
```

maps the `ff` ligature to a pair of `f`'s (whose code is `U+0066`).

Once a single `\pdfglyphtounicode` definition is made, whether it is used or not, another feature comes into play: glyph names of the form `uniXXXX` or `uXXXX` are mapped to the natural `U+XXXX`. Many fonts use this style of naming.

In addition, the `glyphtounicode.tex` file (distributed with PDFTEX and other software) contains thousands of such definitions, covering most common glyph names. So, for practical purposes, one would probably want:

```
\input glyphtounicode
\pdfgentounicode=1
```

LATEX users could load the `cmap` package to achieve the same effect.

▶ `\pdfnobuiltintounicode` ⟨font⟩

The primary purpose of this command is to prevent PDFTEX from generating the `ToUnicode`/CMap resource for the given font when `\pdfgentounicode=1`, most likely because the CMap resource is already generated by some other method (for example, the LATEX `cmap` package uses `\pdffontattr` to generate a CMap resource).

Minimal example:

```
\font\f=cmb10
\pdfnobuiltintounicode\f
\f No unicode mappings for this output.
```

The primitive was introduced in PDFTEX 1.40.11.

▶ `\pdfinterwordspaceon`

▶ `\pdfinterwordspaceoff`

These commands create corresponding whatsit nodes which turn on/off generation of faked interword spaces in the output. This allows for better reflowing of text on the fly by PDF readers, better extraction of textual content, and is required by PDF/A. It does not affect the normal TEX justification with glue.

This requires finding and reading font files `dummy-space.tfm` and `pfb`; the font is included in the PDF output and character 32 is inserted from it as the "fake" space.

Example of usage (see also the `fake-interword-space.tex` test file):

```
Text with no interword spaces.

\pdfmapline{+dummy-space <dummy-space.pfb}
\pdfglyphtounicode{space}{0020}
\pdfinterwordspaceon

Switch to text with faked interword spaces.

\pdfinterwordspaceoff

Back to text with no interword spaces.
```

The primitive was introduced in pdfTEX 1.40.15.

▶ `\pdffakespace`

Insert a faked interword space to the output, regardless of the value of `\pdfinterwordspaceon` and `\pdfinterwordspaceoff`. Example:

```
Text with no interword \pdffakespace\pdffakespace spaces.
```

The primitive was introduced in pdfTEX 1.40.15.

8.4 Spacing

Controlling spacing before and after characters was introduced in version 1.30, mostly to handle punctuation rules in different languages. The `\...code` tables here, like those in the previous section, operate globally, i.e., ignore TEX's usual grouping, and apply only to the given ⟨font⟩ , not other instances of the underlying font.

▶ `\pdfadjustinterwordglue` (integer)

If positive, adjustment of interword glue is enabled and controlled by the following three primitives.

▶ `\knbscode` ⟨font⟩ ⟨8-bit number⟩ (integer)

The amount of space, in thousandths of an em, added to the natural width of the glue following a character (the name stands for "kern before space", although technically it is looking at glue items, not kern items). This amounts is clipped to the range −1000..1000. For instance, the following example means that glues after periods will be increased by .2 em.

```
\pdfadjustinterwordglue=1
\knsbcode\font`\.=200
```

▶ `\stbscode` ⟨font⟩ ⟨8-bit number⟩ (integer)

This works like `\knbscode`, but applies to the stretch component of the following glue.

▶ `\shbscode` ⟨font⟩ ⟨8-bit number⟩ (integer)

Like `\stbscode`, but for the shrink component.

▶ `\pdfprependkern` (integer)

If positive, automatic insertion of kerns before characters is enabled.

▶ `\knbccode` ⟨font⟩ ⟨8-bit number⟩ (integer)

The width of the kern, in thousandths of an em, inserted before a character. It is clipped to the range −1000..1000. For instance, with the following code, a .15 em-kern will be inserted before all question marks (useful for e.g. French punctuation):

```
\pdfprependkern=1
\knbccode\font`\?=150
```

▶ `\pdfappendkern` (integer)

Same as `\pdfprependkern`, but for kerns inserted after characters.

▶ `\knaccode` ⟨font⟩ ⟨8-bit number⟩ (integer)

Same as `\knbccode`, except the kern is inserted after the character. Such a kern is required for instance after a left guillemet in French punctuation.

8.5 Vertical adjustments

▶ `\pdfignoreddimen` (dimen)

This is the dimension which must assigned to the following four primitives so they are ignored. Default is `-1000pt`, and it should be modified with care since it also influences when a previous paragraph's depth is ignored (for instance, the plain TEX macro `\nointerlineskip` should be modified accordingly).

▶ `\pdffirstlineheight` (dimen)

This parameter specifies the height of the first line of a paragraph, regardless of its content. It is read when the paragraph builder is called, and ignored when set to `\pdfignoreddimen`. By default, it is set to `-1000pt`, so it is ignored as long as the value of `\pdfignoreddimen` is the same.

▶ `\pdflastlinedepth` (dimen)

This is similar to the previous parameter, but affects the last line's depth of a paragraph.

▶ `\pdfeachlineheight` (dimen)

Similar to `\pdffirstlineheight`, but for all lines of a paragraph, including the first one, unless `\pdffirstlineheight` is specified.

▶ `\pdfeachlinedepth` (dimen)

Like the preceding parameter, but for depth.

8.6 PDF objects

▶ `\pdfobj` ⟨object type spec⟩ (h, v, m)

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 no longer creates a dictionary object with contents ⟨general text⟩ , as it did 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 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⟩ (h, v, m)

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.

8.7 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 `/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 an individual page. The token list is expanded when PDFTEX ships out a page. The contents are added to the attributes of the current page.

If the `\pdfpageattr` value contains the string `/MediaBox`, then PDFTEX omits outputting its own `/MediaBox` value (which is `[0 0` ⟨*page_width*⟩ ⟨*page_height*⟩ `]`). (This behavior was introduced in version 1.40.18.)

▶ `\pdfpageresources` (tokens)

These tokens are added to the resource dictionary for all pages, before the font, XObject, and other resources.

```
   \pdfpageresources{/MyPageResourceAttribute /MyValue}
```

▶ `\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.

8.8 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⟩ (h, v, m)

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⟩ (h, v, m)

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.

8.9 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⟩ (h, v, m

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 e.g. the original `\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 with 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 behavior specified by the `\pdfpagebox` parameter, and is overridden by the (obsolete) `\pdfforcepagebox` parameter. This option is irrelevant for non–PDF inclusions.

Starting with 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` ⟨object number⟩

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 ⟨object number⟩ 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`.

▶ `\pdfximagebbox` ⟨integer⟩ ⟨integer⟩ (expandable)

The dimensions of the bounding box of a PDF image loaded with `\pdfximage` are stored in a table. This primitive returns those dimensions as follows:

```
\pdfximage{afile.pdf}
\pdfximagebbox\pdflastximage 1 % Returns lower-left x
\pdfximagebbox\pdflastximage 2 % Returns lower-left y
\pdfximagebbox\pdflastximage 3 % Returns upper-right x
\pdfximagebbox\pdflastximage 4 % Returns upper-right y
```

▶ `\pdflastximagecolordepth` (read–only integer)

The color depth (1 for 1-bit images, 2 for 2-bit images, and so on) of the last image accessed with `\pdfximage`.

▶ `\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 explicitly 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 explicitly); the resolution values coming with EXIF files are currently ignored.

▶ `\pdfpagebox` (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⟩. If the option isn't present, the page box defaults to the value of `\pdfpagebox` as follows: (1) media box, (2) crop box, (3) bleed box, (4) trim box, and (5) artbox.

▶ `\pdfforcepagebox` (integer)

The integer primitive `\pdfforcepagebox` allows globally overriding the choice of the page box used with `\pdfximage`. It takes the same values as `\pdfpagebox`. The command is available starting with PDFTEX 1.30.0, as a shortened synonym of obsolete `\pdfoptionalwaysusepdfpagebox` instruction, but is itself now considered obsolete — a mixture of `\pdfpagebox` and ⟨image attr spec⟩ is better.

► `\pdfinclusionerrorlevel` (integer)

This controls the behavior of pdfTeX when a pdf file is included which has a newer pdf version than the one specified by `\pdfmajorversion` and `\pdfminorversion`. If `\pdfinclusionerrorlevel` is set to 0 (the default), pdfTeX gives only a warning; if 1, pdfTeX raises an error; if negative, no diagnostic at all is given.

It was originally a shortened synonym of `\pdfoptionpdfinclusionerrorlevel`, which is now obsolete. The primitive was introduced in pdfTeX 1.30.0.

► `\pdfimagehicolor` (integer)

This parameter, when set to 1, enables embedding of png images with 16 bit wide color channels at their full color resolution. This embedding mode is defined only from pdf version 1.5 onwards, so the `\pdfimagehicolor` functionality is automatically disabled in pdfTeX if `\pdfminorversion` < 5 and `\pdfmajorversion` = 1; in this case, each 16 bit color channel is reduced to a width of 8 bits by stripping the lower 8 bits before embedding. The same stripping happens when `\pdfimagehicolor` is set to 0. If `\pdfmajorversion` = 1 and `\pdfminorversion` ≥ 5, or `\pdfmajorversion` ≥ 2, 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 value of the primitive `\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 `\mag`). The default value of this primitive is 1000, defining a device gamma value of 1.0.

If `\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 with pdfTeX 1.40.0, `\pdfpxdimen` is now a real dimension parameter.

▶ `\pdfinclusioncopyfonts` (integer)

If positive, this parameter forces pdfTeX to include fonts from a pdf file loaded with `\pdfximage`, even if those fonts are available on disk. Bigger files might be created, but included pdf files are sure to be embedded with the adequate fonts; indeed, the fonts on disk might be different from the embedded ones, and glyphs might be missing.

▶ `\pdfsuppresswarningpagegroup` (integer)

Ordinarily, pdfTeX gives a warning when more than one included pdf file has a so-called “page group object” (`/Group`), because only one can “win” — that is, be propagated to the page level. Usually the page groups are identical, but when they are not, the result is unpredictable. It would be ideal if pdfTeX in fact detected whether the page groups were the same and only gave the warning in the problematic case; unfortunately, this is not easy (a patch would be welcome). Nevertheless, often one observes that there is no actual problem. Then seeing the warnings on every run is just noise, and can be suppressed by setting this parameter to a positive number. The primitive was introduced in pdfTeX 1.40.15.

8.10 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⟩ (h, v, m)

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).

8.11 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⟩ (h, v, m)

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 6.

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 6 Options for display of outline and destinations.

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.

▶ `\pdfstartlink` [⟨rule spec⟩] [⟨attr spec⟩] ⟨action spec⟩ (h, m)

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 (https://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 behavior 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` (h, m)

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` (dimen)

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

▶ `\pdfdestmargin` (dimen)

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

▶ `\pdfsuppresswarningdupdest` (integer)

Ordinarily, pdfTeX gives a warning when the same destination is used more than once. However, due to problematic macro packages, sometimes a document may end up producing the warning through no fault of its own, and fixing the macros may well not be trivial. Then seeing the warnings on every run is just noise, and can be suppressed by setting this parameter to a positive number. The primitive was introduced in pdfTeX 1.40.13.

▶ `\pdfrunninglinkoff`

▶ `\pdfrunninglinkon`

These commands create corresponding whatsit nodes which turn off/on generation of running links. Their typical usage is to turn off generation of running links in the header or footer of a page. Generation of running links is on when the shipout routine begins.

The generation of running links works roughly like this: pdfTeX keeps a stack of links created by \pdfstartlink, called pdf_link_stack. When writing out an hbox to PDF, pdfTeX checks if the nesting level of the box is the same as the nesting level of the top entry in pdf_link_stack; if yes that box would become a link, too.

The whatsit nodes created by the above primitives turn off/on a flag which controls if a hbox being shipped can become a link, in addition to the previous condition.

Thus, the commands must be inserted before the hbox in question. For example:

```
% (1) good:
\hbox{\pdfrunninglinkoff
  \hbox{this is a line that would become a link otherwise}
}

% (2) bad:
\hbox{\pdfrunninglinkoff this is a line that would become a link}
% too late; \pdfrunninglinkoff must be inserted before the box
```

8.12 Bookmarks

▸ \pdfoutline [⟨attr spec⟩] ⟨action spec⟩ [count ⟨integer⟩] ⟨general text⟩ (h, v, m)

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.

8.13 Article threads

► `\pdfthread` [⟨rule spec⟩] [⟨attr spec⟩] ⟨id spec⟩ (h, v, m)

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

► `\pdfstartthread` [⟨rule spec⟩] [⟨attr spec⟩] ⟨id spec⟩ (v, m)

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 behavior, it's often enough to wrap boxes that shouldn't belong to the thread by a box to change their box nesting level.

► `\pdfendthread` (v, m)

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

► `\pdfthreadmargin` (dimen)

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

8.14 Literals and specials

► `\pdfliteral` [⟨pdfliteral spec⟩] ⟨general text⟩ (h, v, m)

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 with version 1.30.0, PDFTEX allows to use a new keyword `page` instead of `direct`. Both modify the default behavior 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⟩ `}`.

8.15 Strings

▶ `\pdfescapestring` ⟨general text⟩ (expandable)

Starting with 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⟩ (expandable)

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⟩ (expandable)

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⟩ (expandable)

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.

▶ `\pdfstrcmp` ⟨general text⟩ ⟨general text⟩ (expandable)

This command compares two strings and expands to `0` if the strings are equal, to `-1` if the first string ranks before the second, and to `1` otherwise. The primitive was introduced in pdfTeX 1.30.0.

▶ `\pdfmatch` [`icase`] [`subcount` ⟨integer⟩] ⟨general text⟩ ⟨general text⟩ (expandable)

This command implements pattern matching (using the syntax of posix extended regular expressions). The first ⟨general text⟩ is a pattern and the second is a string. The command expands to `-1` if the pattern is invalid, to `0` if no match is found, and to `1` if a match is found. With the `icase` option, the matching is case-insensitive. The `subcount` option sets the size of the table storing the found (sub)patterns; its default is 10. The primitive was introduced in pdfTeX 1.30.0.

▶ `\pdflastmatch` ⟨integer⟩ (expandable)

The matches found with `\pdfmatch` are stored in a table. This command returns the entry for ⟨integer⟩ , in the format ⟨position⟩ `->` ⟨string⟩ ; ⟨position⟩ is the position of the match (starting at zero) or `-1` if no match was found, and ⟨string⟩ is the matched substring.

Entry 0 contains the match as a whole; the subsequent entries contain submatches corresponding to the subpatterns, up to `subcount-1`.

If ⟨integer⟩ is less than zero, an error is given.

For instance:

```
  \pdfmatch subcount 3 {ab(cd)*ef(gh)(ij)}{abefghij}
  \pdflastmatch0 % "0->abefghij"
  \pdflastmatch1 % "-1->"
  \pdflastmatch2 % "4->gh"
  \pdflastmatch3 % "-1->"
```

Entry 1 is empty because no match was found for `cd`, and entry 3 is empty because it exceeds the table's size as set by `subcount`. The primitive was introduced in pdfTeX 1.30.0.

8.16 Numbers

▶ `\ifpdfabsnum` (expandable)

This conditional works like the standard `\ifnum`, 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 considerably more friendly in usage and works a bit faster. The primitive was introduced in pdfTeX 1.40.0.

▶ `\ifpdfabsdim` (expandable)

Analogous to `\ifpdfabsnum`, this conditional works like `\ifdim`, except that it compares absolute values of dimensions. The primitive was introduced in pdfTeX 1.40.0.

▶ `\pdfuniformdeviate` ⟨number⟩ (expandable)

Generate 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)

Generate a normally-distributed random integer with a mean of 0 and standard deviation 65 536. That is, about 68% of the time, the result will be between −65536 and 65536 (one standard deviation away from the mean). About 95% of results will be within two standard deviations, and 99.7% within three. 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⟩

Set the random seed (`\pdfrandomseed`) to a specific value, allowing you to replay sequences of semi-randoms at a later moment. The primitive was introduced in PDFTEX 1.30.0.

8.17 Timekeeping

▶ `\pdfelapsedtime` (read–only integer)

Return a number that represents the time elapsed from the moment of the start of the run. The elapsed time is returned in 'scaled seconds', meaning seconds divided by 65536, e.g. PDFTEX has run for 254345 'scaled seconds' when this paragraph was typeset. Of course, 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`

Reset the internal timer so that `\pdfelapsedtime` starts returning micro–time from 0 again. The primitive was introduced in PDFTEX 1.30.0.

8.18 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:20210219080324+09'00'` for the source of this manual. As of version 1.40.20, if the `SOURCE_DATE_EPOCH` and `FORCE_SOURCE_DATE` environment variables are both set, `\pdffilemoddate` returns a value in UTC, ending in Z. 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 230208 for the source of this manual. The primitive was introduced in pdfTeX 1.30.0.

▶ `\pdfmdfivesum` [`file`] ⟨general text⟩ (expandable)

If the keyword `file` is given, expands to the MD5 of file ⟨general text⟩ in uppercase hexadecimal format (same as `\pdfescapehex`). Without `file`, expands to the MD5 of the ⟨generaltext⟩ taken as a string. For example, it's `FD3A4079F359C252842CA394D765E658` for the source of this manual. The primitive was introduced in pdfTeX 1.30.0.

▶ `\pdffiledump` [`offset` ⟨integer⟩] [`length` ⟨integer⟩] ⟨general text⟩ (expandable)

Expands to the dump of the first `length` ⟨integer⟩ bytes of the file ⟨general text⟩, in uppercase hexadecimal format (same as `\pdfescapehex`), starting at offset `offset` ⟨number⟩, or the beginning of the file if `offset`. If `length` is not given, the default is zero, so expands to nothing. Both ⟨integer⟩s must be ≥ 0. For example, the first ten bytes of the source of this manual are `2520696E746572666163`. The primitive was introduced in pdfTeX 1.30.0.

▶ `\input` ⟨general text⟩ (expandable)

As of TeX Live 2020, the `\input` primitive in all TeX engines, including pdfTeX, now also accepts a group-delimited filename argument, as a system-dependent extension, as in `\input\Lbrace foo.tex\Rbrace`. The usage of `\input` with a standard space/token-delimited filename is completely unchanged.

This group-delimited argument was previously implemented in LuaTeX; now it is available in all engines. ASCII double quote characters (`"`) are removed from the filename, but it is otherwise left unchanged after tokenization.

This extension is unlike most others in that it affects a primitive in standard TeX (`\input`), rather than being related to a new primitive, command line option, etc. This is allowed because additional methods of recognizing filenames are explicitly mentioned in `tex.web` as acceptable system-dependent extensions.

Incidentally, this does not directly affect LaTeX's `\input` command, as that is a macro redefinition of the standard `\input` primitive.

The primitive was introduced in pdfTeX 1.40.21.

8.19 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.

8.20 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 the 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.

8.21 Miscellaneous

▸ `\expanded` ⟨tokens⟩ (expandable)

Expands ⟨tokens⟩ in exactly the same way as `\message`. In contrast to `\edef`, macro parameter characters do not need to be doubled. `\protected` macros are not expanded. The primitive was introduced in pdfTeX 1.40.20.

▸ `\ifincsname` (expandable)

This conditional is true if evaluated inside `\csname ... \endcsname`, and false otherwise.

▸ `\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.

▸ `\pdfcreationdate` (expandable)

Expands to the date string pdfTeX uses in the info dictionary of the document, e.g. for this file `D:20210219080532+09'00'`. The primitive was introduced in pdfTeX 1.30.0.

▸ `\pdfdraftmode` (integer)

When set to 1 (or set by the command-line switch `-draftmode`) pdfTeX doesn't write the output PDF file 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. If specified, the parameter must appear before any data is written to the PDF output. The primitive was introduced in pdfTeX 1.40.0.

▶ `\pdfinsertht` ⟨integer⟩   (expandable)

If ⟨integer⟩ is the number of an insertion class, this command returns the height of the corresponding box at the current time. For instance, the following returns `12pt` in plain TeX:

```
Abc\footnote*{Whatever.}\par
\pdfinsertht\footins
```

▶ `\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)

Completely analogous to `\pdflastxpos`, returning the y coordinate.

▶ `\pdfprimitive` ⟨control sequence⟩

This command executes the primitive meaning of the following control sequence, regardless of whether the control sequence has been redefined or made undefined. If the primitive was expandable, `\pdfprimitive` expands also. On the other hand, if the following control sequence never was a primitive, nothing happens and no error is raised. (In some versions of pdfTeX prior to 1.40.19, an error was wrongly given.) The primitive was introduced in pdfTeX 1.40.0.

▶ `\pdfretval`   (read–only integer)

Set to -1 if `\pdfobj` ignores an invalid object number. Perhaps this will be used to store the error status of other primitives in the future.

▶ `\pdfsavepos` (h, v, m)

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.

▶ `\pdfshellescape` (read–only integer)

This primitive is 1 if `\write18` is enabled, 2 if its operation is restricted to known-safe programs, and 0 otherwise. The primitive was introduced in pdfTeX 1.30.0.

▶ `\pdftexbanner` (expandable)

Returns the pdfTeX banner message, e.g. for the version used here: `This is pdfTeX, Version 3.141592653-2.6-1.40.22 (TeX Live 2021/W32TeX) kpathsea version 6.3.3`. The primitive was introduced in pdfTeX 1.20a.

▶ `\pdftexrevision` (expandable)

Returns the revision number of pdfTeX, e.g. for pdfTeX version 1.40.22 (used to produce this document), it returns the number 22.

▶ `\pdftexversion` (read–only integer)

Returns the version of pdfTeX multiplied by 100, e.g. for pdfTeX version 1.40.22 (used to produce this document), it returns 140.

▶ `\quitvmode`

The primitive instructs pdfTeX to quit vertical mode and start typesetting a paragraph. Thus, `\quitvmode` has the same basic effect as the `\leavevmode` macro from `plain.tex`. However, `\leavevmode` expands the `\everypar` tokens list, which may or may not be desired. `\quitvmode` does not expand `\everypar`. The primitive was introduced in pdfTeX 1.21a.

▶ `\tracinglostchars` (integer)

This primitive parameter has always been part of TeX, and its operation with values ≤ 2 is unchanged. In addition, if its value is ≥ 3, then 'Missing character' reports become full errors (ordinarily they are only logged), and the character code is reported in hex. For example:

```
\tracinglostchars=3
\font\x=logo10 \x \char99 \end
```

will result in this error message:

```
! Missing character: There is no c ("63) in font logo10.
```

(The `logo10` font only defines the capital letters used in the METAFONT and METAPOST logos, so there is no lowercase.)

This new behavior is essentially the same in all TEX engines except the original TEX and ε-TEX, where the behavior of `\tracinglostchars` remains unchanged.

The primitive was introduced in PDFTEX 1.40.22.

▶ `\tracingstacklevels` (integer)

If this primitive parameter is > 0, and `\tracingmacros` > 0, macro expansion logging is truncated at the specified depth. Also, and more importantly, each relevant log line is given a prefix beginning with `~`, either followed by a `.` character for each expansion level or only another `~` if the expansion was truncated. For example:

```
\tracingmacros=1      % so macro expansion is logged at all
\tracingstacklevels=2 % cut off at level 2
\def\a#1{\relax}      % argument to show parameter logging is affected too
\def\b#1{\a{#1}}
\b1
```

logs the following:

```
~.\b #1->\a {#1}
#1<-1
~~\a
```

Thus, the expansion of `\b` is logged normally, with the addition of the `~.` prefix. The expansion of `\a` is truncated (level 2), hence neither the parameters nor body expansion are shown.

Furthermore, an `\input` file counts as an expansion level, and the input filename is logged. So, if we add this to our example above:

```
\input anotherfile
```

where `anotherfile.tex` simply contains `\b2`, the log will get:

```
~.INPUT anotherfile.tex
~~\b
~~\a
```

Now the `\b` expansion is not logged either, since the expansion depth is higher than the `\tracingstacklevels` value.

The intended use of `\tracingstacklevels` is not so much to truncate logging as to indicate the expansion levels for detailed debugging. Thus normally it would be set to a large number (`\maxdimen`, say), so that everything is fully logged, with the addition of the expansion level indication with the number of dots in the prefix.

The behavior is the same in all TEX engines except the original TEX and ε-TEX, where `\tracingstacklevels` remains undefined.

The primitive was introduced in PDFTEX 1.40.22.

► `\vadjust` [⟨pre spec⟩] ⟨filler⟩ { ⟨vertical mode material⟩ } (h, m)

The `\vadjust` implementation of PDFTEX adds an optional qualifier ⟨pre spec⟩ , which is simply the string `pre`, to the original TEX primitive with the same name. If no `pre` is given, `\vadjust` behaves exactly as the original (see *The* TEX*book*, 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.

9 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 `\pdfpagebox` or `\pdfforcepagebox` primitives 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. Both JFIF and EXIF are supported for additional information.

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 `https://github.com/agl/jbig2enc`.

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.

10 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 WEB2C manual.

Abbreviations

In this document we use numerous abbreviations. For convenience we mention their meaning here.

AFM	Adobe Font Metrics
ASCII	American Standard Code for Information Interchange
BIBTEX	Handles bibliographies
CONTEXT	general purpose macro package

CTAN	global TEX archive server
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
EXIF	Exchangeable Image File format (JPEG file variant)
GHOSTSCRIPT	PS and PDF language interpreter
GNU	GNU's Not Unix
HZ	Hermann Zapf optimization
ISO	International Organization for Standardization
JBIG	Joint Bi-level Image Experts Group
JBIG2	Joint Bi-level Image Experts Group
JFIF	JPEG File Interchange Format
JPEG	Joint Photographic Experts Group
LATEX	general purpose macro package
MAC OS X	Macintosh operating system version 10
MACTEX	MACINTOSH WEB2C distribution
METAFONT	graphic programming environment, bitmap output
METAPOST	graphic programming environment, vector output
MIKTEX	WINDOWS distribution
MLTEX	MLTEX extension to TEX
MPTOPDF	METAPOST to PDF conversion tool
MS-DOS	Microsoft DOS platform (Intel)
PDF	Portable Document Format
PDF/A	PDF A/ standards
PDFTEX	TEX extension producing PDF output
PERL	Perl programming environment
PK	Packed bitmap font

PNG	Portable Network Graphics
POSIX	Portable Operating System Interface
PostScript	PostScript
proTeXt	Windows 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
TeX	typographic language and program
TeXexec	ConTeXt command line interface
Texinfo	generate typeset documentation from info pages
TeX Live	TeX Live distribution (multiple platform)
TeXutil	ConTeXt utility tool
TFM	TeX Font Metrics
Unix	Unix platform
URL	Uniform Resource Locator
WEB	literate programming environment
Web2c	official multi–platform WEB environment
Windows	Microsoft Windows platform

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 information 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 3.141592653-2.6-1.40.22 (TeX Live 2021/W32TeX) kpathsea version 6.3.3`. This is necessary because the string in the `Producer` key in the info dictionary is rather short, namely `pdfTeX-1.40.22`.

Any or all of these keys can be suppressed with the `\pdfsuppressptexinfo` primitive, described in section 8.2.

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