%PDF-1.2
3 0 obj <<
/Length 4 0 R
>>
stream
1 0 0 1 91.925 759.924 cm
BT
/F51 9.963 Tf 0 0 Td[(W)80(elcome)-250(to)
-250(pdfT)]TJ 67.818 -2.241 Td[(E)]TJ 4.842
2.241 Td[(X!)]TJ 138.923 -654.744 Td[(1)]TJ
ET
endstream
endobj
4 0 obj
162
endobj
1 0 obj <<
/Font << /F51 5 0 R >>
/ProcSet [/PDF /Text]
>> endobj
2 0 obj <<
/Type /Page
/Contents 3 0 R
/Resources 1 0 R
/MediaBox [0 0 595.273 841.887]
/Parent 6 0 R
>> endobj
7 0 obj <<
/Type /Encoding
/Differences [0/.nothdef 5/dotaccent
/hungarumlaut/ogonek 8/.notdef 9/fraction
10/.notdef 11/ff/fi/fl/ffi/ffl/dotlessi
/dotlessj/grave/acute/caron/breve/macron
/ring/cedilla/germandbls/ae/oe/oslash/AE
/OE/Oslash/space/exclam/quotedbl/numbersign
/dollar/percent/ampersand/quoteright
/parenleft/parenright/asterisk/plus/comma
/hyphen/period/slash/zero/one/two/three/four
/five/six/seven/eight/nine/colon/semicolon
/less/equal/greater/question/at/A/B/C/D/E/F
/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z
/bracketleft/backslash/bracketright
/circumflex/underscore/quoteleft/a/b/c/d/e
/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/u/v/w/x/y/z
/braceleft/bar/braceright/tilde/dieresis
/Lslash/quotesingle/quotesinglbase/florin
/quotedblbase/ellipsis/dagger/daggerdbl
/circumflex/perthousand/Scaron/guilsinglleft
/OE/Zcaron/asciicircum/minus/lslash/quoteleft
/quoteright/quotedblleft/quotedblright/bullet
/endash/emdash/tilde/trademark/scaron
/guilsinglright/oe/zcaron/asciitilde
/Ydieresis/space/exclamdown/cent/sterling
/currency/yen/brokenbar/section/dieresis
/copyright/ordfeminine/guillemotleft
/logicalnot/hyphen/registered/macron/degree
/plusminus/twosuperior/threesuperior/acute
/mu/paragraph/periodcentered/cedilla
/onesuperior/ordmasculine/guillemotright
/onequarter/onehalf/threequarters
/questiondown/Agrave/Aacute/Acircumflex
/Atilde/Adieresis/Aring/AE/Ccedilla/Egrave
/Eacute/Ecircumflex/Edieresis/Igrave/Iacute
/Icircumflex/Idieresis/Eth/Ntilde/Ograve
/Oacute/Ocircumflex/Otilde/Odieresis/multiply
/Oslash/Ugrave/Uacute/Ucircumflex/Udieresis
/Yacute/Thorn/germandbls/agrave/aacute
/acircumflex/atilde/adieresis/aring/ae
/ccedilla/egrave/eacute/ecircumflex/edieresis
/igrave/iacute/icircumflex/idieresis/eth
/ntilde/ograve/oacute/ocircumflex/otilde
/odieresis/divide/oslash/ugrave/uacute
/ucircumflex/udieresis/yacute/thorn
/ydieresis]
>> endobj
5 0 obj <<
/Type /Font
/Subtype /Type1
/Encoding 7 0 R
/BaseFont /Times-Roman
>> endobj
6 0 obj <<
/Type /Pages
/Count 1
/Kids [2 0 R]
>> endobj
8 0 obj <<
/Type /Catalog
/Pages 6 0 R
>> endobj
9 0 obj <<
/Creator (TeX)
/Producer (pdfTeX-0.12r)
/CreationDate (D:19981205172300)
>> endobj
xref
0 10
0000000000 65535 f
0000000242 00000 n
0000000308 00000 n
0000000009 00000 n
0000000223 00000 n
0000002238 00000 n
0000002326 00000 n
0000000420 00000 n
0000002383 00000 n
0000002432 00000 n
trailer
<<
/Size 10
/Root 8 0 R
/Info 9 0 R
>>
startxref
2526
%%EOF

The pdfTEX user manual

Hàn Thế Thành
Sebastian Rahtz
Hans Hagen

January 28, 1999

The pdfTeX manual

thanh@informatics.muni.cz — **Hàn Thế Thành**
s.rahtz@elsevier.co.uk — **Sebastian Rahtz**
pragma@wxs.nl — **Hans Hagen**

January 28, 1999

The title page of this manual
represents the plain TeX coded
text "Welcome to pdfTeX!"

```
\pdfoutput=1
\pdfcompresslevel=0
\font\tenrm=tir
\tenrm
Welcome to pdf\TeX !
\end
```

Contents

1 Introduction

The main purpose of the pdfTeX project was to create an extension of TeX that can create PDF directly from TeX source files and improve/enhance the result of TeX typesetting with the help of PDF. When PDF output is not selected, pdfTeX produces normal DVI output, otherwise it produces PDF output that looks identical to the DVI output. The next stage of the project, apart from fixing any errors in the program, is to investigate alternative justification algorithms, possibly making use of multiple master fonts.

pdfTeX is based on the original TeX sources and Web2C, and has been successfully compiled on Unix, Amiga, Win32 and MSDos systems. It is still under beta development and all features are liable to change. Despite its β–state, pdfTeX produces excellent PDF code.

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

2 About PDF

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

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

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

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

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

inserts an object in the file, while `\pdflastobj` returns the number pdfTeX assigned to this object. The sequence `6 0 R` is an object reference, a pointer to another object. The second number (here a zero) is currently not used in pdfTeX; it is the version number of the object. It is for instance used by PDF editors, when they replace objects by new ones.

In general this rather direct way of pushing objects in the files is rather useless and only makes sense when implementing for instance fill–in field support or annotation content reuse. We come to that later. Unless such direct objects are part of something larger, they will end up as isolated entities, not doing any harm but not doing any good either.

When a viewer opens a PDF file, it first goes to the end of the file. There it finds the keyword `startxref`, the signal where to look for the so called object cross reference table. This table provides

fast access to the objects that make up the file. The actual starting point of the file is defined after the `trailer`. The `/Root` entry points to the catalog. In this catalog the viewer can find the page list, in our example we have only one page. The trailer also holds an `/Info` entry, which tells a bit more about the document. Just follow the thread:

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

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

The page content is a stream of drawing operations. Such a stream can be compressed, where the level of compression can be set with `\pdfcompresslevel`. Let's take a closer look to this stream. First there is a transformation matrix, six numbers followed by `cm`. As in PostScript, the operator comes after the operands. Between `BT` and `ET` comes the text. A font switch can be recognized as `/F...` The actual text goes between `()` to form a so called string. When one analyzes a file produced by a less sophisticated typesetting engine, whole sequences of words can be recognized. In TeX however, the text comes out rather fragmented, mainly because a lot of kerning takes place. Because viewers can search in these streams, one can imagine that the average TeX produced files becomes more difficult as soon as the typesetting engine does a better job; TeX cannot do less.

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

In this simple file, we don't specify in what way the file should be opened, for instance full screen or clipped. A closer look at the page object (`/Type /Page`) shows that a mediabox is part of the page description. A mediabox acts like the bounding box in a PostScript file. pdfTeX users have access to this object by `\pdfpageattr`.

Although in most cases macro packages will shield users from these internals, pdfTeX provides access to many of the entries described here, either automatically by translating the TeX data structures into PDF ones, or directly by pushing entries to the catalog, page, info or self created objects. Those who, after this introduction, feel uncomfortable in how to proceed, are advised to read on but skip section 6. Before we come to that section, we will describe how to get started with pdfTeX.

3 Getting started

This section describes the steps needed to get pdfTeX running on a system where pdfTeX is not yet installed. Some TeX distributions have pdfTeX as a component, like teTeX, fpTeX, MikTeX and CMacTeX, so when you use one of them, you don't need to bother with the pdfTeX installation. Note that the installation description in this manual is Web2C–specific.

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

3.1 Getting sources and binaries

The latest sources of pdfTeX are distributed together with precompiled binaries of pdfTeX for some platforms, including Linux[1], SGI IRIX, Sun SPARC Solaris and MSDOS (djgpp).[2] The primary location where one can fetch the source code is:

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

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

A binary version of pdfTeX for the Amiga is coming with the AmiWeb2C distribution (`ctan:systems/amiga/amiweb2c`) by Andreas Scherer (`andreas.scherer@pobox.com`). For the Macintosh there is CMacTeX.

3.2 Compiling

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

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

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

```
cd web2c-7.2/web2c
sh config.status
make pdftex
```

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

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

`texmf.cnf` Web2C run–time configuration file, located in `web2c-7.2/kpathsea`

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

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

[2] The djgpp version is built by djgpp cross–compiler on Linux.

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

3.3 Getting PDFTeX-specific platform-independent files

Apart from above-mentioned files, there is another ZIP archive (`pdftexlib-0.12.zip`) in PDFTeX distribution which contains platform-independent files required for running PDFTeX:

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

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

3.4 Placing files

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

3.5 Setting search paths

WEB2C-based programs, including PDFTeX, use the WEB2C run-time configuration file called `texmf.cnf`. This file can be found via the user-set environment variable `TEXMFCNF` or via the compile-time default value if the former is not set. It is strongly recommended to use the first option. Next you need to edit `texmf.cnf` so PDFTeX can find all necessary files. Usually one has to edit `TEXMFS` and maybe some of the next variables. When running PDFTeX, some extra search paths are used beyond those normally requested by TeX itself:

`VFFONTS`	the path where PDFTeX looks for virtual fonts
`T1FONTS`	the path where PDFTeX looks for Type1 fonts
`TTFONTS`	the path where PDFTeX looks for TrueType fonts
`PKFONTS`	the path where PDFTeX looks for PK fonts
`TEXPSHEADERS`	the path where PDFTeX looks for the configuration file `pdftex.cfg`, font mapping files (`*.map`), encoding files (`*.enc`), and pictures

3.6 The PDFTeX configuration file

One has to keep in mind that, opposed to DVI output, there is no postprocessing stage. This has several rather fundamental consequences, like one-pass graphic and font inclusion. When TeX builds a page, the macro package used quite certain has a concept of page dimensions, which is not the same as paper dimensions. The reference point of the page is the top-left corner.

Most DVI postprocessors enable the user to specify the paper size, which often defaults to 'A4' or 'letter'. In most cases it does not harm that much to mix the two, because one will seldom put too

small paper in the printer. And, if one does, one will certainly not do that a second time. In PDF the paper size is part of the definition. This means that everything that is off page, is clipped off, it simply disappears. Even worse, just like in a POSTSCRIPT file, the reference point is in the lower corner, which is opposite to DVI's reference point.

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

When PDFTeX starts, it reads the WEB2C configuration file as well as the PDFTeX configuration file called `pdftex.cfg`, searched for in the `TEXPSHEADERS` path. As WEB2C systems commonly specify a 'private' tree for PDFTeX where configuration and map files are located, this allows individual users or projects to maintain customized versions of the configuration file.

The configuration file sets default values for the following parameters, all of which can be overridden in the TeX source file:

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

compress_level This integer parameter specifies the level of text and in–line graphics compression. PDFTeX uses ZIP compression as provided by `zlib`. A value of 0 means no compression, 1 means fastest, 9 means best, 2..8 means something in between. Just set this value to 9, unless there is a good reason to do otherwise — 0 is great for testing macros that use `\pdfliteral`.

decimal_digits This integer specifies the preciseness of real numbers in PDF page descriptions. It gives the maximal number of decimal digits after the decimal point of real numbers. Valid values are in range 0..5. A higher value means more precise output, but also results in a much larger file size and more time to display or print. In most cases the optimal value is 2. This parameter does *not* influence the precision of numbers used in raw PDF code, like that used in `\pdfliterals` and annotation action specifications.

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

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

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

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

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

```
output_format    1
compress_level   0
decimal_digits   2
image_resolution 300
page_width       210mm
page_height      297mm
horigin          1in
vorigin          1in
map              standard.map
map              +cm.map
```

Dimensions can be specified as true, which makes them immune for magnification (when set). The previous example settings, apart from `map`, can also be set during a TeX run. This leaves a special case:

include_form_resources Sometimes embedded PDF illustrations can pose viewers for problems. When set to 1, this variable makes pdfTeX take some precautions. Forget about it when you never encounteres problems. When all the programs you use conform to the PDF specifications, you will never need to set this variable.

3.7 Creating formats

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

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

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

```
pdftex -ini -fmt=cont-en cont-en
```

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

```
texexec --make en
```

for an english format, or

```
texexec --make --tex=pdfetex en de
```

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

3.8 Testing the installation

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

```
pdftex example
```

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

3.9 Common problems

The most common problem with installation is that pdfTeX complains that something cannot be found. In such cases make sure that TEXMFCNF is set correctly, so pdfTeX can find texmf.cnf. The next best place to look/edit is the file texmf.cnf. When still in deep trouble, set KPATHSEA_DEBUG=255 before running pdfTeX or run pdfTeX with option -k 255. It will cause pdfTeX to write a lot of debugging information, that can be useful to trace problems. More options can be found in the Web2C documentation.

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

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

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

- You have to increase POOLSIZE.

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

  ```
  pool_size = 500000
  ```

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

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

- Fatal format file error; I'm stymied.

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

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

  This might appear if you forgot to install the proper pdftex.pool when installing a new version of the pdfTeX binary.

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

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

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

```
obj_tab_size = 400000
```

4 Macro packages supporting PDFTeX

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

As a direct result, when one uses a high level macro package, the macros that take care of these features have to be set up properly. Specials for instance make no sense at all. Actually being a comment understood by DVI postprocessors —given that the macro package speaks the specific language of this postprocessor— a `\special` would end up as just a comment in the PDF file, which is of no use. Therefore, `\special` issues a warning when PDFTeX is in PDF mode.

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

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

or, if this leads to confusion,

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

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

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

- For LaTeX users, Sebastian Rahtz' `hyperref` package has substantial support for PDFTeX, and provides access to most of its features. In the simplest case, the user merely needs to load `hyperref` with a `pdftex` option, and all cross–references will be converted to PDF hypertext links. PDF output is automatically selected, compression is turned on, and the page size is set up correctly. Bookmarks are created to match the table of contents.
- The standard LaTeX `graphics` and `color` packages have `pdftex` options, which allow use of normal color, text rotation, and graphics inclusion commands.
- The ConTeXt macro package by Hans Hagen (pragma@wxs.nl) has very full support for PDFTeX in its generalized hypertext features. Support for PDFTeX is implemented as a special driver, and is invoked by saying `\setupoutput[pdftex]` or feeding TeXExec with the `--pdf` option.
- Hypertexted PDF from `texinfo` documents can be created with `pdftexinfo.tex`, which is a slight modification of the standard `texinfo` macros. This file is part of the PDFTeX distribution.
- A similar modification of `webmac.tex`, called `pdfwebmac.tex`, allows production of hypertext'd PDF versions of programs written in WEB. This is also part of the PDFTeX distribution.

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

5 Setting up fonts

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

5.1 Map files

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

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

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

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

> The value of the flags key in a font descriptor is a 32-bit integer that contains a collection of boolean attributes. These attributes are true if the corresponding bit is set to 1. Table 1 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.

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

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 1 The meaning of flags in the font descriptor.

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

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

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

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

- If it is preceded by a < the font file will be partly downloaded, which means that only used glyphs (characters) are embedded to the font. This is the most common use and is *strongly recommended* for any font, as it ensures the portability and reduces the size of the PDF output. Partial fonts are included in such a way that name and cache clashes are minimalized.
- In case the font file name is preceded by a double <<, the font file will be included entirely — all glyphs of the font are embedded, including the ones that are not used in the document. Apart from causing large size PDF output, this option may cause troubles with TrueType fonts too, so

it is not recommended. It might be useful in case the font is untypical and can not be subsetted well by PDFTeX. *Beware: some font vendors forbid full font inclusion.*

- In case nothing preceded the font file name, the font file is read but nothing is embedded, only the font parameters are extracted to generate the so–called font descriptor, which is used by Acrobat Reader to simulate the font if needed. This option is useful only when you do not want to embed the font (i.e. to reduce the output size), but wish to use the font metrics and let Acrobat Reader generate instances that look close to the used font in case the font resource is not installed on the system where the PDF output will be viewed or printed. To use this feature the font flags *must* be specified, and it must have the bit 6 set on, which means that only fonts with the Adobe Standard Roman Character Set can be simulated. The only exception is in case of Symbolic font, which is not very useful.
- If the font file name is preceded by a `!`, the font is not read at all, and is assumed to be available on the system. This option can be used to create PDF files which do not contain embedded fonts. The PDF output then works only on systems where the resource of the used font is available. It's not very useful for document exchange, as the PDF is not 'portable' at all. On the other hand it is very useful when you wish to speed up running of PDFTeX during interactive work, and only in a final version embed all used fonts. Don't over–estimate gain in speed and when distributing files, always embed the fonts! This feature requires Acrobat Reader to have access to installed fonts on the system. This has been tested on Win95 and UNIX (Solaris).

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

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

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

If a used font is not present in the map files, first PDFTeX will look for a source with suffix `.pgc`, which is a so–called PGC source (PDF Glyph Container)[4]. If no PGC source is available, PDFTeX will try to use PK fonts in a normal way as DVI drivers do, on–the–fly creating PK fonts if needed.

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

```
csr10
```

[4] This is a text file containing a PDF Type 3 font, created by METAPOST using some utilities by Hans Hagen. In general PGC files can contain whatever allowed in PDF page description, which may be used to support fonts that are not available in METAFONT. At the moment PGC fonts are not very useful, as vector Type 3 fonts are not displayed very well in Acrobat Reader, but it may be more useful when Type 3 font handling gets better.

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

To summarize this rather confusing story, we include some sample lines.

Use a built–in font with font–specific encoding, i.e. neither a download font nor an external encoding is given. A SlantFont is specified similarly as for `dvips`.

```
psyr   Symbol
psyro  Symbol        ".167 SlantFont"
pzdr   ZapfDingbats
```

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

```
ptmr8r   Times-Roman   <8r.enc
ptmri8r  Times-Italic  <8r.enc
ptmro8r  Times-Roman   <8r.enc  ".167 SlantFont"
```

Use a partially downloaded font with an external encoding:

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

Use some faked font map entries:

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

Use an ASCII subset of OT1 and T1:

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

Download a font entirely without reencoding:

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

Partially download a font without reencoding:

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

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

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

Entirely download a font with reencoding:

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

Partially download a font with reencoding:

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

Sometimes we do not want to include a font, but need to extract parameters from the font file and reencode the font as well. This only works for fonts with Adobe Standard Encoding. The font flags specify how such a font looks like, so Acrobat Reader can generate similar instance if the font resource is not available on the target system.

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

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

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

5.2 TrueType fonts

As mentioned above, pdfTeX can work with TrueType fonts. Defining TrueType files is similar to Type 1 font. The only extra thing to do with TrueType is to create a TFM file. There is a program called `ttf2afm` in the pdfTeX distribution which can be used to extract AFM from TrueType fonts. Usage is simple:

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

A TrueType file can be recognized by its suffix `ttf`. The optional *encoding* specifies the encoding, which is the same as the encoding vector used in map files for pdfTeX and `dvips`. If the encoding is not given, all the glyphs of the AFM output will be mapped to `/.notdef`. `ttf2afm` writes the output AFM to standard output. If we need to know which glyphs are available in the font, we can run `ttf2afm` without encoding to get all glyph names. The resulting AFM file can be used to generate a TFM one by applying `afm2tfm`.

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

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

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

The SlantFont transformation also works for TrueType fonts.

6 New primitives

Here follows a short description of new primitives added by pdfTeX. One way to learn more about how to use these primitives is to have a look at the file `example.tex` in the pdfTeX distribution. Each pdfTeX specific primitive is prefixed by `\pdf`.

6.1 Document setup

- `\pdfoutput` = *number*

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

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

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

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

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

However, there are better ways to handle these things.

- `\pdfcompresslevel` = *number*

  This integer parameter specifies the level of text compression via `zlib`. Zero means no compression, 1 means fastest, 9 means best, 2..8 means something in between. A value out of this range will be adjusted to the nearest meaningful value. Use a value of 9 for normal runs.

- `\pdfpagewidth` = *dimension*

  This dimension parameter specifies the page width of the PDF output. If not given then the page width will be calculated as mentioned above. Like the next one, this value replaces the value set in the configuration file. When part of the page falls of the paper or screen, you can be rather sure that this parameter is set wrong.

- `\pdfpageheight` = *dimension*

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

- `\pdfpagesattr` = *tokens*

  Use this token list parameter to specify optional attributes common for all pages of the PDF output file. Some examples of attributes are `/MediaBox`, the rectangle specifying the natural size of the page, `/CropBox`, the rectangle specifying the region of the page being displayed and printed, and `/Rotate`, the number of degrees (in multiples of 90) the page should be rotated clockwise when it is displayed or printed.

- `\pdfpageattr` = *tokens*

  This is similar to `\pdfpagesattr`, but it takes priority to the former one. It can be used to overwrite any attribute given by `\pdfpagesattr` for individual pages.

6.2 The document info and catalog

- `\pdfinfo` {*info keys*}

  This allows the user to add information to the document info section; if this information is provided, it can be extracted by Acrobat Reader (version 3.1: menu option *Document Information, General*). The {*info keys*} is a set of data pairs, a key and a value. The key names are preceded by a /, and the values, being strings, are given between parentheses. All keys are optional. Possible keys are `/Author`, `/CreationDate` (defaults to current date), `/ModDate`, `/Creator` (defaults to TeX), `/Producer` (defaults to pdfTeX), `/Title`, `/Subject`, and `/Keywords`.

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

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

```
\pdfinfo
  { /Title        (example.pdf)
    /Creator      (TeX)
    /Producer     (pdfTeX 0.15a)
    /Author       (Tom and Jerry)
    /CreationDate (D:19980212201000)
    /ModDate      (D:19980212201000)
    /Subject      (Example)
    /Keywords     (pdfTeX) }
```

- \pdfcatalog {*catalog keys*} ⟨openaction *action*⟩

Similar to the document info section is the document catalog, where keys are /URI, which provides the base URL of the document, and /PageMode determines how Acrobat displays the document on startup. The possibilities for the latter are explained in Table 2:

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

Table 2 Supported /PageMode values.

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

The openaction is the action provided when opening the document and is specified in the same way as internal links, see section 6.7. Instead of using this method, one can also write the open action directly into the catalog.

- \pdfnames {*text*}

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

6.3 Fonts

- \font ... *number*stretch *number*shrink *number*step *number*

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

```
\font\somefont=somefile at 10pt stretch 30 shrink 20 step 5
```

The `stretch 30 shrink 20 step 5` means as much as: "hey TeX, when things are going to bad, you may stretch the glyphs in this font as much as 3% or shrink them by 2%". Because pdfTeX uses internal datastructures with fixed widths, each additional width also means an additional font. For practical reasons pdfTeX uses discrete steps, in this example a 5% one. This means that for font `somefile` upto 11 differently scaled alternatives are used. When no step is specified, 1% steps are used.

Roughly spoken, the trick is as follows. Consider a text typeset in triple column mode. When TeX cannot break a line in the appropriate way, the unbreakable parts of the word will stick into the margin. When pdfTeX notes this, it will try to scale the glyphs in that line using fixed steps, until the line fits. When lines are too spacy, the opposite happens: pdfTeX starts scaling the glyphs until the white space gaps is acceptable.

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

This mechanism is inspired on an optimization introduced first by Herman Zapf, which in itself goes back to optimizations used in the early days of typesetting: use different glyphs to optimize the greyness of a page. So, there are many, slightly different a's, e's, etc. For practical reasons pdfTeX does not use such huge glyph collections; it uses horizontal scaling instead. This is sub–optimal, and for many fonts, sort of offending to the design. But, when using PDF, it's not that illogical at all: PDF viewers use so called Multiple Master fonts when no fonts are embedded and/or can be found on the target system. Such fonts are designed to adapt their design to the different scaling parameters. It is up to the user to determine to what extend mixing slightly remastered fonts can be used without violating the design. Think of an O: when simply stretched, the vertical part of the glyph becomes thicker, and looks incompatible to an unscaled original. In a multiple master, one can decide to stretch but keep this thickness compatible.

- `\pdfadjustspacing` *number*

The output that pdfTeX produces is pretty compatible with the normal TeX output: TeX's typesetting engine is unchanged. The optimization described here, is turned of by default. At this moment there are two methods. When set to 1, simple stretching is applied. This alternative uses the normal TFM files, and is not related to the stretch and shrink as described in the previous section. When set to 2, the previously described multiple font optimization comes into action.

- `\efcode` *number*

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

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

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

6.4 Graphics inclusion

- `\pdfimage` `width` *dimension* `height` *dimension* `depth` *dimension* {*filename*}

  Inserts an image, optionally changing width, height, depth or any combination of them. Default values are zero for depth and 'running' for height and width. If all of them are given, the image will be scaled to fit the specified values. If some of them (but not all) are given, the rest will be set to a value corresponding to the remaining ones so as to make the image size to yield the same proportion of $width : (height + depth)$ as the original image size, where depth is treated as zero. If none of them is given then the image will take its natural size. An image inserted at natural size often has a resolution 72 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 filename of the image must appear after the optional dimension parameters. The dimension of the image can be accessed by enclosing the `\pdfimage` command to a box and checking the dimensions of the box:

  ```
  \setbox0=\hbox{\pdfimage {somefile.png}}
  ```

  Now we can use `\wd0` and `\ht0` to question the natural size of the image as determined by PDFTeX. When dimensions are specified before the `{somefile.pdf}`, the graphic is scaled to fit these.

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

- `\pdfimageresolution` = *number*

  We already mentioned the default resolution of 72 dots per inch. It is possible to overrule this value by using this register. Of course this only applies to bitmap PNG, TIFF, and JPEG illustrations.

6.5 XObject Forms

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

- `\pdfform` *number*

  Writes out the TeX box *number* as a XObject Form to the PDF file.

- `\pdflastform`

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

- `\pdfrefform` `\name`

  Inserts a reference to the XObject Form called `\name`.

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

6.6 Annotations

PDF level 1.2 provides four basic kinds of annotations:

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

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

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

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

▶ `\pdflastannot`

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 `\pdfannotlink` (see below).

6.7 Destinations and links

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

▶ `\pdfdest` ⟨`num` *n* | `name` *refname*⟩ `appearance`

This primitive 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; `appearance` must be one of those mentioned in table 3.

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`	keep the current zoom factor

Table 3 The outline and destination appearances.

`xyz` can optionally be followed by `zoom` *factor* to provide a fixed zoom–in. The *factor* is like TeX magnification, i.e. 1000 is the 'normal' page view.

▶ `\pdfannotlink` `width` *dimension* `height` *dimension* `depth` *dimension* `attr` {*attributes*} *action*

Starts a hypertext link; if the optional dimensions are not specified, they will be calculated from the box containing the link. The {*attributes*} are explained in great detail in the PDF Reference Manual

and determine the appearance of the link. Typically, the attributes specify the color and thickness of any border around the link. Thus `/C [0.9 0 0] /Border [0 0 2]` specifies a color (in RGB) of dark red, and a border thickness of 2 points.

While all graphics and text in a PDF document have relative positions, annotations have internally hard–coded absolute positions. Again we're dealing with a speed optimization. The main disadvantage is that these annotations do *not* obey transformations issued by `\pdfliteral`'s

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

`page` *n* jump to page *n*

`goto num` *n* jump to point *n*

`goto name` *refname* jump to a point established as *refname* with `\pdfdest`

`goto file` *filename* open a local file; this can be used with a `name` or `page` specification, to point to a specific location on the file

`thread num` *n* jump to thread identified by *n*

`thread name` *refname* jump to thread identified by *refname*

`user {`*specification*`}` perform a user–specified action; the PDF Reference Manual explains the possibilities; a typical use of this is to specify a URL, e.g. `/S /URI /URI (http://www.tug.org/)`

▶ `\pdfendlink`

This primitive ends a link. All text between `\pdfannotlink` and `\pdfendlink` will be treated as part of this link. pdfTeX may break the result across lines (or pages), in which case it will make several links with the same content.

6.8 Bookmarks

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

This primitive creates an outline (or bookmark) entry. The first parameter specifies the action to be taken, and is the same as that allowed for `\pdfannotlink`. 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).

6.9 Article threads

▶ `\pdfthread` ⟨`num` *n* | `name` *refname*⟩

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

▶ `\pdfendthread`

Finishes the current thread.

- `\pdfthreadhoffset` *dimension*

  Specifies a threads horizontal margin.

- `\pdfthreadvoffset` *dimension*

  Specifies a threads vertical margin.

6.10 Miscellaneous

- `\pdfliteral` {*pdf code*}

  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.

- `\pdfobj stream` {*text*}

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

- `\pdflastobj`

  Returns the object number of the last object created by `\pdfobj`. These primitives provide a mechanism allowing insertion of a user–defined object into the PDF output.

- `\pdffontprefix` {*string*}

  In the PDF file produced by PDFTeX, one can recognize a font switch by the prefix `F`, for instance `/F12` or `/F54`. This primitive can be used to force another prefix. This is only needed when one expects (or encounters) viewing problems with included PDF illustrations that use similar prefixes.

- `\pdfformprefix` {*string*}

  Forms are reusable graphic, textual or mixed objects. In the files made by PDFTeX such forms are internally identified by a number, which is not to be confused with the object reference as reported by `\pdflastform`. Like the previous and next primitive, this one can be used to overrule the default prefix, which is `Fm`, like in `/Fm1`.

- `\pdfimageprefix` {*string*}

  Like `\pdffontprefix` and `\pdfformprefix`, this primitive overrules a default prefix, this time `Im`, such as `/Im58`. Forget about these three primitives when you never encountered viewing problems, unless you want more fancy prefixes. When you do encounter PDF inclusion problems, change one or more of these prefixes in your document setup, and in the configuration file set `include_form_resources` to 1.

- `\pdftexversion`

  Returns the version of PDFTeX multiple by 100, e.g. for version `0.12x` it returns 12. This document is typeset with version 13.a.

- `\pdftexrevision`

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

7 Graphics and color

PDFTeX supports inclusion of pictures in PNG, JPEG, TIFF and PDF format. The most common technique —the inclusion of EPS figures— is replaced by PDF inclusion. EPS files can be converted to PDF by

GhostScript, Acrobat Distiller or other PostScript-to-PDF convertors. The BoundingBox of a PDF file is taken from CropBox if available, otherwise from the MediaBox. To get the right BoundingBox from a EPS file, before converting to PDF, it is necessary to transform the EPS file so that the start point is at the (0,0) coordinate and the page size is set exactly corresponding to the BoundingBox. A Perl script (epstopdf) for this purpose has been written by Sebastian Rahtz. The TeXutil utility script that comes with ConTeXt can so a similar job. (Concerning this conversion, it handles complete directories, removes some garbage from files, takes precautions against duplicate conversion, etc.)

Other alternatives for graphics in pdfTeX are:

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

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

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

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

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

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

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

8 Formal syntax specification

The formal syntax specification!

Abbreviations

AFM	Adobe Font Metrics
Amiga	Amiga hardware platform
AmiWeb2C	Amiga distribution
ASCII	...
CMacTeX	MacIntosh Web2C distribution
ConTeXt	general purpose macro package
DJGPP	...
DVI	natural TeX Device Independ fileformat
EPS	Encapsulated PostScript

EPSTOPDF	EPS to PDF conversion tool
e-TeX	an extension to TeX
fpTeX	Win32 Web2C distribution
GNU	...
JPEG	Joined Photographic Expert Group
LaTeX	general purpose macro package
MacIntosh	MacIntosh hardware platform
METAFONT	graphic programming environment, bitmap output
METAPOST	graphic programming environment, vector output
MikTeX	Win32 distribution
MSDos	Microsoft DOS platform (Intel)
PDF	Portable Document Format
pdfe-TeX	e-TeX extension producing PDF output
pdfTeX	TeX extension producing PDF output
Perl	Perl programming environment
PGC	PDF glyph container
PK	Packed Bitmap Font
PNG	Portable Network Graphics
PostScript	PostScript
RGB	Red Green Blue color specification
teTeX	Unix Web2C distribution
TeX	typographic language and program
TeXexec	ConTeXt command line interface
TeXutil	ConTeXt utility tool
TFM	TeX Font Metrics
TIFF	Tagged Interchange File Format
TUG	TeX Users Group
Unix	Unix platform
URL	Uniform Resource Locator
WEB	literate programming environment
Web2C	official multi–platform WEB environment
Win32	Microsoft Windows platform
ZIP	compressed file format