The pdfTEX Program

HÀN Thế Thành

Faculty of Informatics
Masaryk University
Brno, Czech Republic

Abstract. *pdfTEX is an extension to TEX which allows the user to generate either DVI or PDF as the primary output format without requiring the use of DVI as an inter-language. The current feature set of pdfTEX is discussed, and further extensions which are currently under consideration for adoption are reviewed.*

1. Introduction

pdfTEX (formerly known as TEX2PDF) is a variant of TEX which allows the user to opt for either traditional (`DVI`) or alternative (PDF) output without requiring a post-translator. At present, the “normal way” to create PDF consists of (1) compiling TEX source file to `DVI` file which contains `\special` commands for PDF support; (2) converting `DVI` file to PostScript by some `DVI`-to-PostScript driver (`dvips`, `DVIPSONE`,...); (3) translating PostScript file to PDF by some PostScript-to-PDF translator (`Acrobat Distiller`, `Ghostscript`,...). Another way is to use Sergey Lesenko’s `DVIPDF`, a `DVI`-to-PDF driver, which can simplify this process by eliminating the need for PostScript generation. Now pdfTEX can produce directly `DVI` file as well as PDF output from a TEX source without generating `DVI`.

pdfTEX is based on the traditional (Pascal) `WEB` source of TEX, and much of it is implemented as a normal changefile, but some parts which require functionality which is poorly supported by Pascal `WEB` are implemented as modules in the ‘C’ language. As a result, pdfTEX is currently available only for `WEB2C` implementations of TEX, but these cover many platforms including multiple flavours of UNIX as well as 32-bit Windows (’95, NT,...), Amiga and MS/DOS. The definitive sources may be found at `ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdftex/` and these are regularly mirrored by CTAN as `/systems/pdftex/`.

Work on the project began as a post-graduate project towards a Master's degree within the Faculty of Informatics at Masaryk University, Brno (Czech Republic) under the supervision of Prof. Jiří Zlatuška, who was also responsible for suggesting the project in the first place.

2. Portable Document Format

2.1. What is PDF?

Adobe's Portable Document Format (PDF) [1] is a file format derived directly from Adobe's PostScript language. Whilst PostScript was (and is) intended primarily as a page-description language which will normally be interpreted within dedicated printer logic, PDF is aimed far more at on-screen display, document exchange, and hypertextual applications. A PDF document is usually smaller than the equivalent PostScript document, has better-defined structure, allows for both intra- and inter-document links, but lacks the procedural elements which allow a PostScript file to perform (sometimes quite significant) computation within the printer engine.

As well as defining the language/format PDF, Adobe have also produced a suite of tools for creating, viewing, printing, indexing, searching and modifying PDF documents. This suite, called Adobe Acrobat, is available for a wide range of platforms, and the Acrobat Reader is in fact available entirely free of charge. The Reader integrates perfectly with modern web browsers, and the combination allows both local and remote PDF documents to be viewed within the same window as analogous HTML documents. Modern web search engines such as InfoSeek's "UltraSeek" search, classify and catalogue PDF documents in an entirely transparent manner. L. Peter Deutsch's `Ghostscript` interpreter can also both interpret and generate PDF, and at least one far-Eastern software company has already produced a PDF reader/viewer which can cope with the more complex font requirements of languages such as Korean, Japanese and Chinese.

2.2. PDF and `DVI`: a comparison

Although PDF is more compact than the equivalent PostScript, `DVI` is even more compact, since the latter was intentionally designed to be as compact as possible and uses variable-length data structures for maximal efficiency. With many years of experience of interpreting `DVI`, authors of `DVI` previewers have achieved extremely high rates of interpretation and display, and some (for example Eberhard Mattes' `DVISCR` family) combine high-speed interpretation and display with very effective anti-aliasing/grey-scaling.

On the other hand, `DVI` treats both fonts and graphics as essentially "hands-off" concepts, preferring to regard both as rectangular black boxes about which it knows (and wants to know) as little as possible. Furthermore, the widespread use of exotic fonts, and/or use of TEX's `\special` primitive to convey additional information to the `DVI` driver, has resulted in a device-independent format which all too often has device-dependent data embedded within it. As a result, `DVI` is acceptable only within the rather incestuous TEX community, and for portability `DVI` files are routinely converted to PostScript (and more recently to PDF) before being sent to those outside of the TEX community.

Although TEX has traditionally generated `DVI` as its primary output format, TEX's creator and author (Prof. Donald E. Knuth) has himself said or written on more than one occasion that he had long envisaged that a variant of TEX might elect to use an alternative output format, and although Knuth had PostScript in mind, Adobe's PDF turns out to be an even better candidate for such purposes. It is a very compact language, already well-established as a *de facto* standard portable document format both on and off the web, and its deliberate omission of the procedural elements of PostScript ensures that it is efficient enough to be used for direct screen display as well as for less time-critical applications such as printing. A very significant benefit of the adoption of PDF as the primary output format for TEX is the ability to mutate fonts on the fly, by linear or anamorphic scaling. Such font mutations, used with great discretion, could potentially allow pdfTEX to produce even higher quality output than TEX, since the latter can only distribute white space between words whilst font mutation could allow the glyphs themselves to be minutely adjusted to improve the overall æsthetics of the page. This idea, sublimely demonstrated in Zapf and Karow's HZ system, could well drastically reduce, if not completely eliminate, the `''Overfull \hbox badness 10000''` problem which besets TEX users attempting to typeset to a narrow measure without requiring the adoption of overt letter-spacing to which Knuth has always (and with great justification) been opposed.

2.3. Eliminating `DVI` as an interlanguage

Long before pdfTEX became a reality, Sergey Lesenko had designed and implemented a `DVI`-to-PDF interpreter called `DVIPDF`. Although this program has had a very restricted distribution (many TEX users have found it impossible to locate a copy), knowledge of its existence frequently provokes the question "why modify TEX when all that is necessary can be accomplished with the

DVIPDF driver?". There are many answers to this question, some philosophical and some extremely practical.

— By induction, one can ask "why bother to write DVIPDF at all?", since TEX can be persuaded to generate PDF using DVI and PostScript as interlanguages.
— By the time TEX has shipped out a completed page of DVI, much potentially useful information has already been lost. Only by modifying TEX to output PDF directly, and by integrating the generation of PDF with TEX's "digestive tract", can such information be retained and captured within the PDF file. This allows, for example, an extremely simple and elegant implementation of hyperlinks, the beginning and end of which can span multiple lines or even pages. To accomplish this using DVI as an interlanguage requires heavy use of \specials and even then some functionality is lost unless TEX is modified. PDF also allows the re-use of (potentially large) data structures, a concept completely unknown within DVI.
— The PDF which is generated by pdfTEX is usually more compact than that generated using DVI as an interlanguage, since only TEX has detailed knowledge of the data structures (boxes, etc.) of which the PDF is a representation.
— And finally, only by integrating PDF generation within TEX itself can we gain access to the font-mutating possibilities which may in the future allow the implementation of dynamic font mutation for microjustification, etc.

3. Implementation issues when generating PDF

3.1. Basics

The current implementation of pdfTEX is based on WEB2C and may be divided into two parts, a TEX-specific part and a driver-specific part.

The TEX-specific part consists of several elements:

(a) code to generate PDF pages rather than DVI pages;
(b) code to process virtual fonts;
(c) additional primitives required to control the PDF-generation process.

Of these, (a) is clearly a fundamental necessity, and (b) is required because there is no longer a driver! Since pdfTEX is its own device driver (for the pseudo-device PDF), it must perform all of the font processing which drivers normally perform. As virtual-font handling is well documented in terms of a Pascal `WEB` implementation, this part is implemented as part of the general pdfTEX changefile whilst (as we shall see shortly), more demanding font access routines such as those necessary to decode Adobe type-1 fonts are implemented in 'C'. The additional primitives, required for the implementation of hyperlinks, bookmarks, etc., are also implemented as a part of the general `WEB` changefile.

The driver-specific part consists of a set of source files which implement the components that are most directly concerned with the device-driver aspects of the system. Since these need relatively little access to TEX's data structures, and since they demand rather sophisticated facilities in terms of random access to files, dynamic memory allocation and deallocation, etc., they have been written in 'C'. It is appreciated that this may prove an obstacle when attempting to port pdfTEX, since not all TEX implementations are based on `WEB2C`, but in the opinion of the author the time saved by developing these components in 'C' outweighs the (relatively minor) disadvantages. Further thought will have to be given to this problem if an attempt is made to unify the development of pdfTEX and ε-TEX, since the latter is based on a purely Pascal-`WEB` implementation and is as portable as TEX itself.

It is worth emphasizing that pdfTEX does not **necessarily** generate PDF! The default mode of operation generates `DVI` just as does TEX, and explicit instructions (in the document format or source) are necessary if PDF output is to be obtained. One idea currently under discussion is that of allowing the generation of PDF to be specified in a configuration file, thereby decoupling the selection of PDF or `DVI` from the document or format source.

If PDF output **is** to be obtained, all fonts referenced must be accessible to pdfTEX, with the exception of the 14 base fonts of the Acrobat Reader (Times, Helvetica, Courier, Symbol and Zapf Dingbats). These fonts may be supplied either as PostScript type-1 fonts or in TrueType format; bitmap fonts are **not** supported since their use leads (at present at least) to virtually unreadable documents. (This is a deficiency of the reader rather than of the font itself, but until that problem is resolved by Adobe there seems little point in supporting bitmap fonts.) Both font subsetting and font re-encoding are supported by pdfTEX.

When PDF output is selected, TEX's `\special` command is ignored (that is, it does not contribute anything to the page(s) being shipped out, although its other, more subtle, effects still take place). A new primitive, `\pdfliteral`, allows raw PDF to be inserted into the PDF output stream at well-defined points.

pdfTeX allows the direct inclusion of bitmap graphics (as opposed to bitmap fonts). Graphics for inclusion must be expressed in PNG (Portable Network Graphics) format, and a wide variety of tools exist to perform conversion to and from PNG, including Image Alchemy, Paintshop PRO, and many others. Although facilities for the inclusion of EPS images are thought highly desirable by many, the overheads of PostScript interpretation make this an unreasonably demanding task. More likely is the development of a facility for the incorporation of (E)PDF images, but this is not currently implemented. Hans Hagen and Tanmoy Bhattacharya have developed a suite of macros which allow the inclusion of both METAPOST output and of a subset of PDF files (those that do not contain fonts, bitmaps and/or similar resources).

High-level support for pdfTeX is provided by the hyperref and graphics packages for LaTeX, and the standard pdfTeX distribution also contains modified versions of the `texinfo` and `manmac` packages with inbuilt support for pdfTeX.

3.2. Hyperlinks

One of the most powerful features of the PDF format is the ability to establish both intra- and inter-document hyperlinks. This facility is accessible to pdfTeX users who have only to tag their documents to indicate the start, end and target of the link. All of the underlying housekeeping is performed by pdfTeX itself, including taking care of such difficulties as intervening line and/or page breaks. The only requirement is that the hyperlink shall start and end at the same level of box nesting. Any and all intervening boxes of the same depth are treated as a part of the hyperlink, but boxes nested to a greater depth are not. This feature is heavily exploited by the `texinfo` support macros, since `texinfo` makes frequent use of very long (multiline) references. Experiments have suggested that the `DVI` $\longrightarrow$ PDF route is apparently incapable of dealing with such long references, since the intervening line breaks interfere with the computation of the dimensions of the links. Those interested in experimenting with pdfTeX's support for `texinfo` should take a look at the file `pdftexinfo.tex`, which is a plug-in replacement for the standard `texinfo.tex`.

3.3. Re-use of objects

The close coupling between the TeX typesetting engine and the PDF code-generation routines makes feasible some interesting possibilities which exploit the power of both TeX and PDF. In particular, object re-use becomes possible. Whereas in TeX a recurring theme such as a logo in the running head(s) re-occurs in the `DVI` file for every required occurrence, PDF allows such objects to be created once and then re-referenced again and again as necessary. Such objects can be arbitrarily complicated and can contain TEX set material, graphics, and even references to other similar objects. Rather confusingly, the canonical Adobe term for these objects is "Forms", and for consistency this terminology is used in the relevant pdfTEX primitives, although it is appreciated that this may prove a source of confusion for those unaware of Adobe's rather individual usage of this term. Terminology apart, such "Forms" (objects) are a powerful efficiency tool, since a complicated recurring theme need occur only once in the PDF file even if it is referred to (that is, printed or displayed) many many times.

4. Future plans for pdfTEX: Improving the æsthetics of typeset output

TEX's algorithm for breaking paragraphs into lines is generally regarded as optimal, and there is therefore little incentive to attempt to improve it. However, it is predicated on immutable fonts whose only flexibility lies in the inter-word glue. However, as every TEX user knows, the algorithm leads to rather more "underfull" or "overfull" boxes than are really desirable. Earlier research [2] has shown how an iteration over a restricted set of values for TEX's primary paragraph-breaking parameters (`\tolerance`, etc.) can lead to greatly improved æsthetics and fewer diagnostics, but at the expense of a not-inconsiderable expenditure of time since the iteration is performed "by hand". Systems such as Quark Xpress overcome these problems in a quite different way, by allowing not only the inter-word but also the inter-character spacing to be adjusted, but the results (particularly in inexpert hands) leave a **very** great deal to be desired. Zapf and Karow [4] have not only postulated but demonstrated a very sophisticated system which makes microscopic adjustments to the shapes of the glyphs themselves; the results are remarkable, but the work is patented and has as yet made relatively little impact on commercial offerings.

An idea currently under investigation for pdfTEX is the incorporation of a simpler version of the HZ system (and clearly one which does not violate the letter or spirit of the patent); a first approach will be to break paragraphs as before, but then to reduce the "badness" of individual lines by ever-so-slightly stretching or shrinking (horizontally) the font(s) which occur on that line. If this proves successful (and perhaps even if it doesn't), the next step will be to incorporate knowledge of this additional degree of freedom into the paragraph-breaking algorithm itself. A simple version can be implemented simply by anamorphic scaling of normal fonts [3], but Adobe's introduction of Multiple Master font technology potentially allows for a far more powerful font-interpolation technique to be used, with unique font-instances for each line in the most general case!

5. Short pdfTEX reference manual

pdfTEX implements a number of new primitives intended for PDF control. They make sense only when PDF output is set. The following reference description is specific to pdfTEX version 0.11. There may be some changes for the next release.

5.1. pdfTEX registers

`\pdfoutput`: an integer parameter for output format control. Turns on PDF output when positive, otherwise pdfTEX produces normal `DVI` files. This parameter cannot be set after shipping out the first page of document. In other words, this parameter must be set before pdfTEX ships out the first page if one wants PDF output. This is the only parameter that must be set to produce PDF output. The other parameters are optional.

`\pdfcompresslevel`: an integer parameter specifying compression level for text and image in PDF output. 0 stands for no compression (default), 1 for fastest compression, 9 for best compression, and 2..8 for something in between. A value out of this range will be adjusted to the nearest meaningful value.

`\pdfpagewidth`: a dimension parameter specifying the page width of PDF output file. If not specified it will be calculated as 2×(`\hsize` + 1 inch).

`\pdfpageheight`: this is similiar to `\pdfpagewidth`, but intended for the page height of PDF output file.

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

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

5.2. General pdfTEX primitives

`\pdfimage` ⟨*rule specification*⟩ ⟨*file name*⟩: a command to insert a PNG image in to PDF output. ⟨*rule specification*⟩ designates the final size of the image in PDF output file. Dimensions which are not specified in ⟨*rule specification*⟩ are treated as zero. If all of them are non-zero, the image will be scaled to fit the specified dimensions. If some of them (but not all) are zero, it will be set to a value corresponding to the remaining ones so as to make the image size yield the same proportion of *width* (*height*+*depth*) as the natural image size, where depth is treated as zero. If all of them are zero then the image will take the natural size of it. An image inserted at natural size often has resolution 72 DPI in PDF output file. Some images may contain data specifying image resolution, and in such a case the image will be scaled to the intended resolution. The filename of the image must appear after the optional dimension parameters. The dimension of the image can be accessed by enclosing the `\pdfimage` command to a box and checking the dimensions of the box.

`\pdfliteral` ⟨*balanced text*⟩: this is similiar to TEX `\special` command, but intended for inserting raw PDF code to page description only. pdfTEX does not handle text specified by `\special` command.

`\pdfinfo author`⟨*balanced text*⟩ `title`⟨*balanced text*⟩ `subject`⟨*balanced text*⟩ `keywords`⟨*balanced text*⟩: a command to specify some general information about the document.

`\pdfcatalog pagemode`⟨*balanced text*⟩ `uri`⟨*balanced text*⟩: a command to specify some global options of the document. ⟨*balanced text*⟩ following `pagemode` indicates how the document looks when opened and can be `/UseNone` (open document with no outline), `/UseOutlines` (open document with outline visible), `/FullScreen` (open document in full-screen mode; there is no bar, window controls, nor any other window present in full-screen mode). The default value is `/UseNone`. ⟨*balanced text*⟩ following `uri` contains document-level information for URI.

5.3. Hyperlink-specific pdfTEX primitives

The above-described primitives are general commands for PDF control. The following ones are intended for creating hyperlinks, bookmarks, etc. They are necessary only when one wants to control PDF output at a low level in order to achieve exactly what the user wishes. A PDF reference manual is required when using these commands. Users that do not want to read the manual can look at examples and macros in pdfTEX distribution, of course with some limitation. LATEX users can use `hyperref` package with pdfTEX without bothering how these primitives work.

`\pdfannottext` ⟨*rule specification*⟩ [`attr`⟨*balanced text*⟩] ⟨*balanced text*⟩: a command to create text annotations. ⟨*rule specification*⟩ designates the dimensions of the annotation. The default dimensions for a text annotation are zero for depth and 1 inch for both width and height. The ⟨*balanced text*⟩ following `attr` specifies additional attributes for the annotation. The only meaningful attributes for a text annotation are `/Open true` or `/Open false`, specifying that the text annotation will be implicitly opened or closed (default). The second ⟨*rule specification*⟩ is the text title of the annotation.

`\pdfannotlink` ⟨*rule specification*⟩ [`attr`⟨*balanced text*⟩] ⟨*action specification*⟩: command specifying the beginning of a link annotation (hyperlink). ⟨*rule specification*⟩ designates the dimensions similarly to text annotation. However the default dimensions for link annotation are different from those for text annotation. If they are not specified explicitly then height and depth are taken from the box containing the link, and width is treated as a "running" one, meaning that all the following boxes in the same nesting depth relative to the outermost box will be treated as a part of the link annotation until a `\pdfendlink` (explained below) is encountered. Thus a link annotation may be several lines long, or even several pages long. The ⟨*balanced text*⟩ following `attr` specifies additional attributes for link annotation similarly to text annotation. A lot of attributes for link annotation can be specified here, but their use **requires** consultation of the PDF-1.2 reference manual. The most useful ones are colour and border attributes. Colours are given as `/C [` x y x `]`, where x y z are values in the range 0..1 specifying RGB components of a given colour. Borders are specified as `/Border [0 0` w`]`, where w is the width (in `bp`) of the border. ⟨*action specification*⟩ describes the action that should be executed when the link annotation is activated. The syntax of ⟨*action specification*⟩ is a little complicated.

⟨*action specification*⟩ → ⟨*goto action*⟩|⟨*thread action*⟩|
⟨*page action*⟩|⟨*user-defined action*⟩
⟨*goto action*⟩ → `goto`⟨*file*⟩⟨*identifier*⟩
⟨*thread action*⟩ → `thread`⟨*file*⟩⟨*identifier*⟩
⟨*file*⟩ → `file`⟨*file name*⟩|⟨*empty*⟩
⟨*identifier*⟩ → ⟨*num identifier*⟩|⟨*name identifier*⟩
⟨*num identifier*⟩ → `num`⟨*integer number*⟩
⟨*name identifier*⟩ → `name`⟨*balanced text*⟩
⟨*page action*⟩ → `page`⟨*integer number*⟩⟨*balanced text*⟩
⟨*user-defined action*⟩ → `user`⟨*balanced text*⟩

⟨*goto action*⟩ and ⟨*thread action*⟩ define, respectively, a "jump" to destination and thread given by ⟨*file*⟩ and ⟨*identifier*⟩. If ⟨*file*⟩ is not empty then it will be an "external jump" to the given file. Destinations and threads can have a num or name identifier.

⟨*page action*⟩ defines a "jump" to the page given by ⟨*integer number*⟩. Text specified in ⟨*balanced text*⟩ can be `/Fit` (fit whole page in window), `/FitH` y (fit whole width of page, y is the y-coordinate corresponding to the top of the window), `/FitV` x (fit whole height of page, x is the x-coordinate corresponding to the left of the window), `/FitB`, `/FitBH`, `/FitBV` (similar to the first three but fit the bounding box of the page instead of the whole page).

⟨*user-defined action*⟩ defines an action other than the above-mentioned ones. Its use also requires consultation of the PDF reference manual. The typical use is *URI action*, for instance

`user{/S /URI /URI (http://www.tug.org/)}`

to open a URL in a WWW browser, or *Named action*,

`user{<< /S /Named /N /GoBack >>}` to navigate the document.

`\pdfendlink`: a command to end the link annotation started by `\pdfannotlink`. `\pdfendlink` must be in the box with the same nesting depth considering to the outermost box.

`\pdfdest` ⟨*identifier*⟩ ⟨*fit specification*⟩: a command to define a destination. ⟨*identifier*⟩ is similiar to ⟨*action specification*⟩. ⟨*fit specification*⟩ can be `fit`, `fith`, `fitv`, `fitb`, `fitbh` or `fitbv`, which correspond to specification of ⟨*page action*⟩.

`\pdfthread` ⟨*identifier*⟩: a command to start a thread. Dimensions of the thread are calculated from the box containing the thread. Similiarly to link annotation, all next boxes in the same nesting depth will be treated as part of the thread annotation until a `\pdfendthread` is searched.

`\pdfendthread`: this command is similiar to `\pdfendlink`, but used for threads.

`\pdfthreadhoffset`: dimension parameter specifying the horizontal margin when a thread is displayed.

`\pdfthreadvoffset`: this parameter is similiar to `\pdfthreadhoffset`, but intended for vertical margins.

`\pdfoutline` ⟨*action specification*⟩ [`num` ⟨*integer number*⟩] ⟨*balanced text*⟩: a command to create an outline entry (bookmark). ⟨*action specification*⟩ is similiar to the link annotation, the absolute value of ⟨*integer number*⟩ is the number of direct sub-entries (default value is 0). If this number is negative then all sub-entries will be closed, otherwise they will be opened explicitly. ⟨*balanced text*⟩ is the text title for this entry.

6. Running pdfTEX

Running pdfTEX consists of the following steps[1].

6.1. Fetching the binaries

It is possible to download the binaries of pdfTEX, or build it from its sources. Both binaries and sources of pdfTEX are available on CTAN. Building pdfTEX from sources is similiar to `web2c` and it should be easy for any UNIX platform.

6.2. Setting paths

In addition to the variables required by normal TEX, the following extra ones need to be set in order to run pdfTEX. The simplest way to do this is to edit the `web2c` configuration file `texmf.cnf`.

VFFONTS: paths where pdfTEX looks for virtual fonts.
T1FONTS: paths where pdfTEX looks for Type 1 and TrueType fonts.
TEXPSHEADERS: paths where pdfTEX looks for font map file (`pdftex.map`), PNG pictures and encoding files (`*.enc`). `pdftex.map` and `*.enc` are included in pdfTEX distribution and must be installed to one of the directories specified in TEXPSHEADERS.

Apart from that the pdfTEX pool file (`pdftex.pool`) must be placed in paths where TEX (pdfTEX) looks for pool file (specified by variable TEXPOOL). It may be necessary to increase memory usage specified in `texmf.cnf` to run pdfTEX.

6.3. Creating format files

Creating formats is similar to normal TEX. Under UNIX, this may look like:

```
pdftex -i plain \\dump
mv plain.fmt pdftex.fmt

pdftex -i latex.ltx
mv latex.fmt pdflatex.fmt
```

[1] As far as pdfTEX is actually based on `web2c`, all the settings are `web2c`-specific.

After that it is necessary to place the formats in a directory where TEX (pdfTEX) looks for format files. Also a binary copy of pdfTEX with name `pdflatex` is needed to run pdfLATEX.

6.4. Working runs

The simplest example is the classic "Hello, world!":

```
\pdfoutput=1
Hello, world!
\bye
```

Running `pdftex` on a file containing these lines (let's say `hello.tex`) should produce a PDF file:

```
$pdftex hello
This is PDFTeX, Version 0.11 (based on TeX Version 3.14159) (Web2c 7.0)
(hello.tex [1])
</packages/share/pdftex/texmf/dvips/pdftex/cmtext.enc>
</package s/share/tex/lib/fonts/public/bakoma/type1/cmr10.pfb>
Output written on hello.pdf (1 page, 7337 bytes).
Transcript written on hello.log.
```

More useful examples can be found in the pdfTEX distribution, including the generic example `example.tex`, macros for creating hypertexted PDF from `texinfo` documents (`pdftexinfo.tex`) as well as from programs written in `WEB` (`pdfwebmac.tex`, and a config file (`pdftex.def`) for using pdfTEX with LATEX hyperref, graphics and colour packages

7. Conclusions

Although pdfTEX is still under development, it has already acquired a number of keen and interested users, many of whom are contributing to its further development by their feedback, ideas and suggestions. The author very much hopes that readers of this article will be sufficiently interested to try pdfTEX for themselves, and that they too will then join the enthusiastic group of users whose input is so vital to the success of the project.

Acknowledgements

The author would like to thank his supervisor, Prof. Jiří Zlatuška, for suggesting pdfTEX in the first place, as well as for his much-appreciated assistance and support throughout the life of the project.

Many people on the pdfTEX mailing list have contributed a great deal to the development, by testing, by making suggestions, and by their discussion of and work on the project: in particular the names of Sebastian Rahtz, Hans Hagen and Philip Taylor come to mind. Many others—too many to be named here—have contributed a great deal, and of my own colleagues within the CSTUG community I would like to single out Petr Sojka and Petr Olšák for especial thanks.

This paper has been reviewed by Bernd Raichler, who has given me many useful suggestions and comments to improve the structure as well as the contents of the article. Philip Taylor is extremely appreciated for linguistic editing and improving this paper.

Finally the author would like to thank T V Raman and Adobe Systems Incorporated for their financial support during a six-month scholarship.

Bibliography

[1] Tim Bienz, Richard Cohn, and James R. Meehan. Portable Document Format Reference Manual. *Adobe Systems Incorporated.* `http://www.adobe.com/supportservice/devrelations/PDFS/TN/PDFSPEC.PDF`

[2] Philip Taylor. A pragmatic approach to paragraphs. *TUGboat*, 14(2), 1993.

[3] Philip Taylor. Improving the æsthetics of mixed-font documents. *TUGboat*, 12(1), 1991.

[4] Hermann Zapf. About micro-typography and the *hz*-program. *Electronic Publishing*, 6(3), 1993, 283–288.