PDF information and navigation elements with hyperref, pdfTEX, and thumbpdf

Heiko Oberdiek

Abstract

The PDF format offers additional possibilities for information and navigation through paperless on-line documents. This paper shows, how the navigation features bookmarks and thumbnails can be created automatically or manually by powerful packages like `hyperref` and `thumbpdf`. The problems and solutions are described that arise from converting TEX strings to the PDF ones used in the general document information or in the outlines.

1. Introduction

The PDF format provides elements for navigation and information that do not go to the page content and are not printed. But they are nevertheless useful, because they provide the user additional informations about the document and enlighten the navigation while viewing the document on-line. The elements described in this article:

General document information: With following programs the entries in the document information can be made visible:

AcrobatReader (Figure 1 right bottom):
- Hotkey: <CTRL>–D
- Menu: File→Document Info→General

pdfinfo: The program PDFINFO produces text output and can easily piped by other programs. So documents with special keywords can be looked for.

Bookmarks (outlines): The outline tree (Figure 1 left) mirrors often the logical structure of a document like a table of contents. But they can also be used to offer access to another related documents or to start a sound file and other things.

Thumbnails: Thumbnails (Figure 1 right top) provide an optical representation of the pages. A page can be chosen, and a page area can be selected for viewing.

2. Strings in TEX and PDF

2.1. Strings of PDF format

- Strings are delimited by parentheses:
  `(This is a string.)`
- As in POSTSCRIPT escape sequences can be used for:
  - White space: `\b`, `\f`, `\n`, `\r`, `\t`
  - Unbalanced parentheses: `\(`, `\)`
  - Escape character itself: `\\`
  - Octal notation: äöß = `\344\366\337`
- Font changes are not allowed, possible encodings are PDFDocEncoding or Unicode:

  PDFDocEncoding is an 8bit encoding, a superset of ISOLatin1 (characters 160–255 added), and compatible with Unicode (characters < 256). There are differences with WinAnsiEncoding.

  Unicode is a 16bit encoding and tries to match all possible characters in the world. ACROBATREADER versions below four do not support this encoding.

PDF strings are used at many different places: general document information, bookmarks, text annotations, information dictionary of threads, ...

2.2. Strings in TEX

There are two main types of strings with TEX. A look at TEX's digestion shows the difference:

1. The *eyes* read the input line and the catcodes are set.
2. Then the *mouth* forms tokens and expands them.
3. By the *stomach* the unexpandable tokens are executed, assignments take place, and the characters are set in boxes. The stomach is TEX's typesetting engine.

Let us now consider an argument of `\section` that is typeset in the text, appears in the table of contents and in the bookmarks: In the text the string is fully digested. For the table of contents the argument is written to the `.aux` file and later to the `.toc` file. At this stage the argument is only expanded (mouth), unexpandable tokens are written verbatim. But in the last step the text in the `.toc` file is read and digested by TEX's stomach. For the bookmarks, however, the argument is only written to the output file directly, only the expansion done by the mouth. These tokens do not reach TEX's typesetting stomach. Therefore the string must be in a format that fits the PDF specification and is accepted by the PDF viewer.

The following table illustrates the important differences between oral and stomach strings:

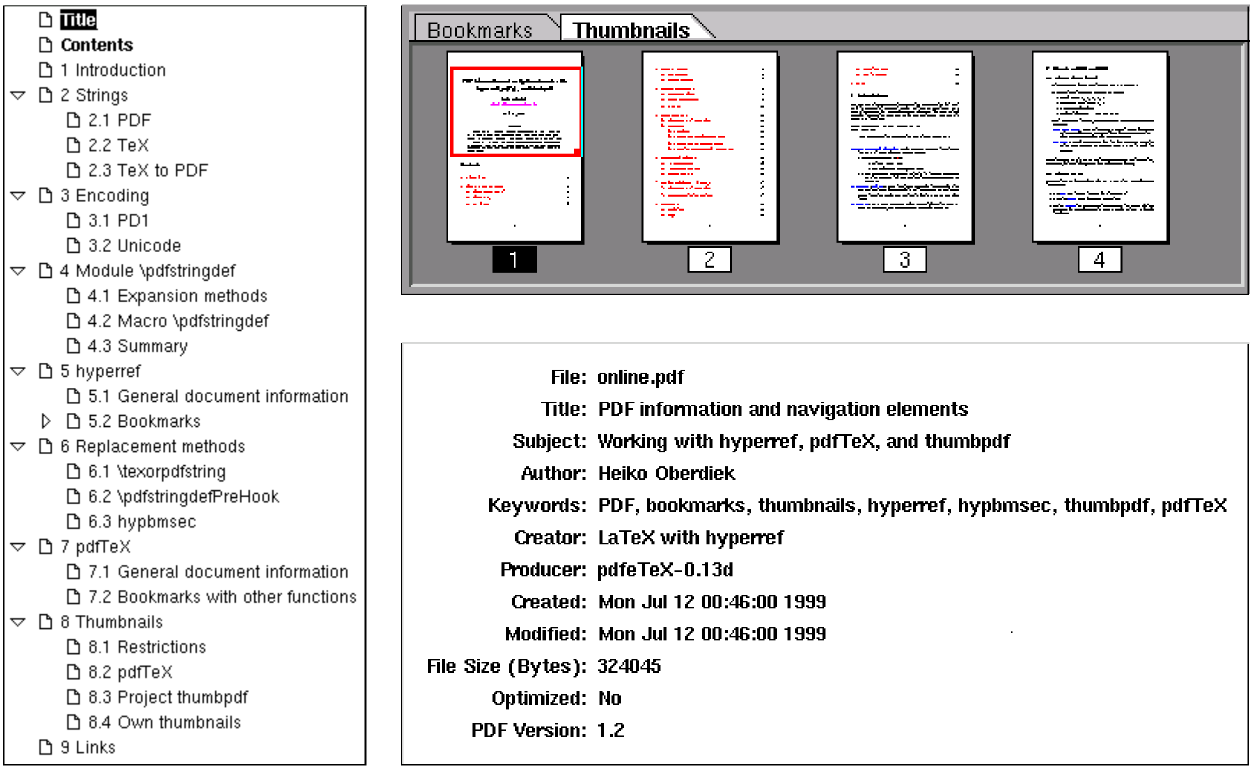


Figure 1: Bookmarks, thumbnails and general information of AcrobatReader

	mouth	stomach
"zero":	\empty	\relax
{...}:	parameter	group
variable:	reading	assignment
character:	command names	boxes
	\csname	ligatures
commands:	\string, \number	\def, \let, \long
	\expandafter	\uppercase, \accent
	\meaning, \the	\par, \bgroup
	\if, \ifx, \ifcase	\begingroup
		\hbox, \vtop, \kern

2.3. TEX to PDF

Arguments or strings, written for TEX's stomach, will not see the stomach, if they are used as PDF strings. This leads to many problems:

- Stomach commands are not executed, they are written verbatim (\hbox, \kern, ...). But a PDF viewer does not know something about TEX commands.
- No font features such as changing fonts or font attributes, ligatures, or hyphenation.
- No manipulation of boxes, no mathematics, no colors, no graphics, ...
- No assignments and definitions.
- Only PDFDocEncoding or Unicode can be used, the encoding cannot be changed within strings.

3. Encoding handling

For the conversion from the current TEX font encoding to one of the two PDF encodings \pdf-stringdef (see section 4) uses the benefits of LATEX's font encoding mechanism. The data, which position has a glyph in an encoding, are stored in macros:

\OT1\ss: \ss in OT1 encoding (\char"19 $\rightarrow$ ß).
\T1\ss: \ss in T1 encoding (\char"FF $\rightarrow$ ß).

The macro name \T1\ss is obtained by
\csname T1\string\ss\endcsname.

In plain-TEX (OT1) \ss is directly defined as \char"19, in LATEX the definition consists of three parts: \OT1-cmd \ss \OT1\ss. The first command \OT1-cmd performs the encoding stuff:

- In a protected environment it expands to \noexpand \ss.
- Warning if the text command is used in math mode.

- Use of \OT1\ss (= \char"19) if the currently active encoding is the same as in the name.
- With a different current encoding the data of the default encoding are used: \?\ss expands to \UseTextSymbol{OT1}\ss.

LaTeX's font encoding mechanism works with appropriately defined *macros*. So characters with catcodes 11 (letter) and 12 (other) selects the font slot directly, circumventy the font encoding mechanism.

3.1. PD1 encoding

The PD1 encoding is defined in the file pd1enc.def and defines the ordering of the glyphs of the PDF-DocEncoding. Most TeX names obey the following rules:

- \text<(glyph name)>:
  ® = \textregistered, ¡ = \textexclamdown
- Traditional names: Æ = \AE, ß = \ss
- Accents: Ü = \"U, ç = \c c, Å = \AA = \r A

Glyph codes in PD1 encoding are mostly octal sequences (\textmu ⇒ \265) to allow safe output.

The file testbmgl.tex of package hyperref lists available glyphs and the TeX macro names to produce them.

3.2. Unicode encoding

General problems:

- Characters having catcode 11 (letter) and 12 (other) are problematic, if the glyph position is not the same as in PDFDocEncoding. It is easy to write code, that detects this characters and replaces them with the Unicode slots.
  - But TeX does not know the glyph name of the current font slot addressed by the character code. Therefore great arrays with font names and encoding tables of that fonts would be required.
  - Checking 256 characters for both catcodes takes a lot of time.

  Memory consumption and computing time would be very high.
- Systems that use ligatures to select the glyphs do not work, because this stomach feature are simulated by expanding commands. This would require huge tables and lots of time. It can only done for exceptions like the spanish ligatures !' and ?'. \pdfstringdef detects and converts them to \exclamdown (¡) and \questiondown (¿). With an empty group {} or the italic correction \/ ligatures can be prevented.
- If the glyphs are produced by direct macros, so \pdfstringdef has to redefine them to produce the two bytes for Unicode output. Disadvantage: large redefinitions tables cost time.
- In the easiest case the glyph macros obey LaTeX's font encoding mechanism. Then only an encoding definition file for Unicode has to be provided.

So the Unicode support of hyperref is implemented using the later method and relies on an input encoding that uses active characters.

Another possibility is reported for Acrobat-Reader on Russian windows: Because a Cyrillic font is used for the bookmarks, the 8bit positions of the PDFDocEncoding show Cyrillic glyphs now. But this solution is not portable and violates the PDF specification.

\pdfstringdef supports the Unicode route:

- Because the Unicode macros use lots of memory, Unicode support is only loaded, if the option unicode is selected.
- \hypersetup{unicode=false} and \hypersetup{unicode} switches between PDFDocEncoding and Unicode.
- In the encoding definition files the two byte sequences are masked in order to be able to distinguish between letters, one and two byte octal sequences.
- Converting of letters and one byte octal sequences into the two byte ones; it is assumed that the one byte glyphs obey the PDFDocEncoding and the higher byte is set to zero.
- Removing the mask.
- Adding the Unicode marker in front of the string (\376\377)
- Currently (early September 1999) only Cyrillic, Greek and some other glyphs of the range 0x0000–0x04FF are supported.

The file testbmu.tex of package hyperref shows the available glyphs and supported TeX macro names to produce them.

4. Module \pdfstringdef

Package hyperref contains a module with the command \pdfstringdef that tries the conversion of TeX to PDF strings:

- TeX should not stop with an error.
- Detecting of parts that cannot be converted correctly; so forbidden commands should give a comprehensive warning (`\kern`, `\hbox`, …).
- Redefining macros that produces text or characters without forbidden commands (`\TeX`, `\P`).
- Simulating forbidden but useful commands (`\xspace`).
- Conversion between LaTeX and PDF encoding (PD1).
- Disabling some forbidden commands (`\def`, `\discretionary`) with and without warning.
- `\pdfstringdef` can do local redefinitions, but catcodes are untouched, because catcode changes do not work, if the argument has already been seen by TeX's eyes.
- Avoiding a processing step by an external program.

4.1. Expansion methods

TeX provides two main methods to expand strings:

- `\edef` defines a macro with the expansion of the argument. Tokens that cannot be expanded further are included unchanged.
- The `\csname` method converts the text to a macro name. Using `\string`, any macro name can be converted back into a sequence of characters, and with `\@gobble` the first backslash can be removed.

The result of the `\csname` method is a safe string without TeX commands. But stomach commands are not allowed and will result in an error message ("`\endcsname expected`"), that confuses the user. Because the error handling of TeX is very old, modern methods like catch-try-blocks are missing. Therefore `hyperref` uses the `\edef` method and makes great efforts to detect forbidden commands.

4.2. Macro `\pdfstringdef`

`\pdfstringdef` takes two parameters: a macro name and a TeX string. As result the macro is defined containing the PDF string. (Currently this is done globally, but do not rely on it.)

All the following tasks, definitions and redefinitions are made in a group to keep them local:

- Switching to PD1 or PU encoding
- definition of the "octal sequence commands" (`\345`):
  `\edef\3{\string\3}`
- Special glyphs of TeX: `\{`, `\%`, `\&`, `\space`, `\dots`, …
- National glyphs (`german.sty`, `french.sty`)
- Logos: `\TeX`, `\eTeX`, `\MF`, …
- Disabling commands that do not have an useful function in bookmarks:
  `\label`, `\index`, `\glossary`,
  `\discretionary`, `\def`, `\let`, …
  LaTeX's font commands like `\textbf`, …
- Support for `\xspace` of package `xspace`
- Parentheses are protected to avoid the danger by unsafe unbalanced parentheses in the PDF string. (see section 2.1).

A very simple example is the redefinition of `\discretionary` to get the non-break text:

```
\let\discretionary\@gobbletwo
```

More difficult are the `\def` commands. They do not work in TeX's mouth and undefined command names will causes errors. A tricky redefinition of the `\def` commands avoids them by *defining* the undefined macros with `\csname` to be `\relax` (This is the only possible assignment in an expanding context).

After the redefinitions the string is expanded by `\edef`. Because the result can contain command tokens and other unwanted things, each token has now to be checked by `\pdfstringdef`:

- Grouping characters are discarded silently.
- Also unexpandable commands and active characters with the meaning of `\relax` are removed silently (`\relax` itself, `\protect`).
- All the other unexpandable ones and special characters (`$`, `&`, …) are cancelled with a comprehensive warning.

4.3. Summary

Allowed are:

- Expandable macros,
- `\relax`, `\protect`, grouping characters `\{` and `\}`,
- `\xspace`, `\discretionary`, `\/`, `\penalty`.

The most other unexpandable commands give warnings. Cryptic error messages should be seldom, but they *can* occur, especially if the code of the TeX string depends on assignments.

5. Package hyperref

5.1. General document information

Package hyperref knows the following options to fill the elements of the general document information:

pdftitle, pdfsubject, pdfauthor,
pdfkeywords, pdfcreator, pdfproducer

The normal place for options is the optional argument of the \usepackage command. However LaTeX expands the options before passing them to the package. So the switching of the encoding and all the redefinitions of \pdfstringdef are not active while expanding. Therefore it is better to set the information options, after package hyperref is loaded. For this purpose the package provides \hypersetup:

```
\usepackage{hyperref}
\hypersetup{
\pdftitle={About umlauts: \"a \"o \"u \ss},
\pdfauthor={\textcopyright\ Heiko Oberdiek},
   ...
}
```

Another possibility is to reuse the data of \maketitle:

```
  \newcommand\org@maketitle{}
  \let\org@maketitle\maketitle
  \def\maketitle{%
    \hypersetup{
      \pdftitle={\@title},
      \pdfauthor{\@author}
  }%
  \org@maketitle
}
```

5.2. Bookmarks

5.2.1. Options

There are four places for hyperref options:

1. Global: \documentclass[...] (e.g. driver)
2. Package: \usepackage[...]
3. Configuration file: hyperref.cfg with \hypersetup
4. After package loading: \hypersetup{...}

The following options affect bookmarks:

bookmarks: Make bookmarks (Default: true). This option cannot be used after package has been loaded.

bookmarksnumbered: Put section numbers in bookmarks (Default: false).

bookmarksopen: Open up the bookmark tree (Default: false).

bookmarksopenlevel: Level (see section 5.2.4) to which bookmarks are open (Default: \maxdimen).

bookmarkstype: To specify which 'toc' file to mimic (Default: toc).

pdfpagemode: How document starts when opened (Default: None):

- None: Neither outlines nor thumbnails are visible.
- UseOutlines: Outlines are visible.
- UseThumbs: Thumbnails are visible.
- FullScreen: Full-screen mode without bars, outlines, and thumbnails.

unicode: Creating Unicode bookmarks (Default: false). After package has been loaded, it switches between Unicode and PDFDocEncoding.

When creating bookmarks hyperref writes them into the file \jobname.out. At the second run the bookmarks are set. A "rerun" warning is not implemented.

5.2.2. Bookmarks by section commands

The section commands (\part, \chapter downto \subparagraph) are automatically used for the outline tree, unless option bookmarks is disabled. The behaviour can be modified with two LaTeX counters:

tocdepth: This counter sets the level to which the section commands appear in the table of contents. It is interpreted for the outline tree, too.

secnumdepth: This counter specifies the level, to which the section commands are numbered. If option bookmarksnumbered is in force, this counter is also used for the bookmarks.

The bookmarks build a *rigide tree structure*. Intermediate levels cannot be omitted, because the next subentry after an omitted entry will start leftmost at the highest level in the bookmark tree.

```
\chapter{five}\subsection{one}
```

In the table of contents the \subsection is formatted as \subsection with number 5.0.1. But in the outline tree there is no \section entry which the \subsection can be attached to. So the \subsection entry hangs directly on the steam.

5.2.3. Bookmarks by `\addcontentsline`

For starred section commands `\addcontentsline` adds an entry for the table of contents. Also a bookmark is added.

There are problems, if the link does not point to the page in general, but to the coordinates on the page (`/XYZ` view):

- Package `hyperref` does not want to redefine the section commands too deeply in order to remain compatible with most of the packages. The disadvantage is, that the destination is set by the overloaded section command *below* the title.

  ```
  \section*{Intro}
  \addcontentsline}{toc}{section}{Intro}
  ```

  The macro `\addcontentsline` cannot be used before, because the anchor name for the destination is only known *after* the section command.
- Package `hypbmsec` supports an automatic inclusion of starred section commands into the table of contents and the outline tree with option `startotoc`. With option `prefix` it tries to solve the problem by putting `\addcontentsline` before the section command. Therefore it defines its own destination. The great disadvantage is a possible page break between the destination and the section command.
- With option `infix` package `hypbmsec` uses another method: It puts the destination definition and `\addcontentsline` into the title itself to avoid page breaking problems. This works for left aligned titles, with right aligned or centered titles the horizontal offset goes wrong.

5.2.4. Creating bookmarks with `\pdfbookmark`

As explained above the direct children can be attached to a parent, but no grandchildren. Each section command has an internal level number to check the relationships:

Documentclass	part	chapter	section	subsection	...
`book/report:`	-1	0	1	2	...
`article:`	0		1	2	...

This level is also needed in own bookmarks. These can be created by the macro `\pdfbookmark`:

`\pdfbookmark[`*level*`]{`*bookmark text*`}{`*anchor name*`}`

This command creates an anchor and adds this entry to the bookmarks. The anchor name should be unique. For the internal name the argument with the anchor name is appended by a dot and the level. Examples:

```
\pdfbookmark[0]{Beginning of Document}{beg}%beg.0
\pdfbookmark[1]{Titlepage}{tit}%tit.1
```

It is possible to add an outline entry that points to another target, e.g.: Somewhere in the text the target (anchor) is defined:

```
\hypertarget{place.1}{}
```

Then the bookmark can be added at another location as follows:

```
\begingroup
  \makeatletter
  \def\hyper@anchorstart #1\hyper@anchorend{}%
  \pdfbookmark[1]{Go to the place}{place}%
  \endgroup
```

The current bookmark level of the last command is stored in the internal macro `\Hy@currentbookmarklevel`. The both commands `\currentpdfbookmark` and `\subpdfbookmark` omits the level argument of `\pdfbookmark`, because they use the internal stored value of the current bookmark level. With this two commands an user can create bookmarks at the same level or a level below without looking after a level number.

6. Replacement methods

Because of limitations of TEX's mouth (see section 2.2), many things like mathematics, colors, or font changes cannot be used in bookmarks. But package `hyperref` uses the argument of section commands for the bookmarks (see section 5.2.2) that can contain these forbidden things. Therefore a way is needed to replace or disable these for the bookmarks.

6.1. Providing an alternative with `\texorpdfstring`

Package `hyperref` sets a switch while expanding a string in `\pdfstringdef`. This is used by the full expandible macro `\texorpdfstring`. It expects two arguments, the first contains the string that will be set and processed by TEX's stomach, the second contains the replacement for PDF strings, e.g.:

```
\section{Pythagoras:
 \texorpdfstring{$ a^2 + b^2 = c^2 $}{%
   a\texttwosuperior\ + b\texttwosuperior\ =
   c\texttwosuperior}}
\section{\texorpdfstring{%
        \textcolor{red}}{}{Red} Mars}
```

Also \texorpdfstring can be used in own definitions perhaps of logos, so that they work in all contexts.

6.2. Hook for own macro redefinitions

Macro \pdfstringdef executes the hook \pdfstringdefPreHook before it expands the string. Here the user can disable additional commands:

```
\expandafter\def\expandafter\pdfstringdefPreHook
\expandafter{\pdfstringdefPreHook
  \renewcommand{\mycommand}[1]{}
}
```

An easier way is the new command \pdfstringdefDisableCommands, that adds its argument to the definition of \pdfstringdefPreHook ('@' can here be used as letter in command names):

```
\pdfstringdefDisableCommands{%
  \let~\textasciitilde
  \def\url{\pdfstringdefwarn\url}%
  \let\textcolor\@gobble
}
```

6.3. Package hypbmsec

The second method offered by package hypbmsec extends the syntax of the section commands. A second argument in square brackets or an argument within parentheses is interpreted as string for the bookmarks:

\section{*toc/head* = *bookmark* = *text*}
\section[*toc/head* = *bookmark*]{*text*}

\section[*toc/head*][*bookmark*]{*text*}

\section(*bookmark*){*toc/head* = *text*}
\section[*toc/head*](*bookmark*){*text*}
\section(*bookmark*)[*toc/head*]{*text*}

The package hypbmsec works with packages that do not change the syntax of the section commands, it should only be loaded as last the package. Incompatible are packages that also change the syntax of the section commands.

For bookmarks hypbmsec uses package hyperref, but if this package is not loaded, it ignores the bookmark arguments. So the same text can be used for various output formats.

The use of the optional parameter delimiters inside the optional parameter should be protected by braces:

(...(...{)}...) or [{...[...]...}]

The next version of package hypbmsec will know the option startotoc: The starred section commands are included in the table of contents and in the outline tree automatically. For options infix and prefix see section 5.2.3.

7. Additional features of PDF format

Package hyperref does not support all possibilities of the PDF format. For special effects the low level commands of the driver must be used. As an example this is shown with the primitive commands of pdfTEX.

7.1. General document information

There are no hyperref options for /CreationDate and /ModDate. pdfTEX sets them to the current date. With pdfTEX's primitive command \pdfinfo they can be set explicitly. The special format of the date strings is defined in the PDF specification. The following shows an example that explicitly sets /CreationDate and /ModDate:

```
\pdfinfo{/CreationDate (D:19990901000000-01'00')}
\begingroup
 \def\twodigits#1{\ifnum#1<10 0\fi\the#1}%
 \count0=\time \divide \count0 by60
 \edef\x{\twodigits{\count0}}%
 \multiply\count0 by 60
\count1=\time \advance\count1 by -\count0
\edef\x{\x\twodigits{\count1}}%
\edef\x{/ModDate (D:\the\year
  \twodigits\month \twodigits\day \x 00-01'00')}%
\expandafter\endgroup
\expandafter\pdfinfo\expandafter{\x}%
```

7.2. Bookmarks with other functions

The generation of outlines with package hyperref is limited to ones that point to a destination inside the document. But in general outlines can point to other documents, call functions of AcrobatReader, play sound, video, ...

For this a driver primitive command is necessary, e.g. pdfTEX:

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

The absolute value of n is the count of the direct subentries of this outline. If the value is negative, the subentries are closed.

Example for "named actions":

```
\newcommand*{\bmaction}[3][0]{%
  \begingroup
   \pdfstringdef\x{#3}%
   \pdfoutline
     user {<< /S /Named /N  /#2 >>}
       count #1 {\x}%
  \endgroup
}
```

```
\bmaction[3]{NOP}{Navigation}
  \bmaction[-2]{FullScreen}{Full-screen mode}
    \bmaction{PageOnly}{Page only}
    \bmaction{ShowThumbs}{Show thumbnails}
  \bmaction[-6]{NOP}{Page selecting}
    \bmaction{PrevPage}{Go to previous page}
    \bmaction{GoBack}{\textless\textless}
    ...
```

The possible actions and the syntax can be read in the PDF specification. A bookmark that points to the action section of that file on page 96:

```
\pdfoutline
  user {<< /S /GoToR /F (pdfspec.pdf)
 /D [95 /Fit] >>} count 0 {Actions}
```

8. Thumbnails

Thumbnails are small images, pictograms or reduced images from the pages of a document. They support the navigation through the document, if the viewer supports it (e.g. AcrobatReader).

8.1. Restrictions

Some restrictions are mentioned in the PDF specification (1.2):

- The maximal supported size of a thumbnail is 106×106 samples.
- The size reduction should be done proportionally in order not to confuse the user at navigating. Recommendation for pages with letter or A4 size is an one-eighth scale.
- The following color spaces can be used: mono (black and white), gray, direct RGB or indexed RGB (with full palette). AcrobatReader has problems to view some optimized graphics with a reduced palette.

8.2. pdfTEX

There is a powerful program to generate PDF files: pdfTEX written by Hàn Thế Thành. But there is no direct support for thumbnails in versions up to 0.13d. The only way I found is to include the thumbnails as images with `\pdfimage`. The guessed object number of this image can be used to specify the thumbnail and it works.

The problem to set the thumbnail on the page can be solved by moving it outside the `/MediaBox`, e.g. `\maxdimen` apart from the margin). However I do not want to set the thumbnails in the document visible or invisible and the problem of guessing the object numbers remains.

8.3. Project `thumbpdf`

So I developed the project `thumbpdf`:

1. The thumbnails are created by a program like GHOSTSCRIPT.
2. These thumbnails are included with `\pdfimage` in a TEX file that will be processed by pdfTEX to get the object representation of the thumbnails.
3. A Perl script extracts the objects and writes them in a TEX readable file (and optimizes the object structure).
4. A TEX package reads this file. It gets the object number by `\pdflastobj` and adds the thumbnail reference with this number to the page attributes (`\pdfpageattr`).

The main work is done by the Perl script `thumbpdf.pl`:

1. It calls GHOSTSCRIPT to generate the thumbnails (`thumb???.png`). This step can be omitted, if the user wants to use his own thumbnail generating program.
2. It calls pdfTEX with `thumbpdf.tex` to generate the PDF file `thumbpdf.pdf`. Each page contains a thumbnail included with `\pdfimage`.
3. It scans the objects to get the object data, optimizes them and writes them to the file `thumbdta.tex`.

The package `thumbpdf.sty` reads the data in `thumbdta.tex` and inserts the thumbnails in the main document. The package works with `plain` or LATEX formats. If the data file `thumbdta.tex` is not created yet, if the number of thumbnails in the file does not suffice, or if the TEX compiler is not pdfTEX, then the package writes warnings only. So the user does not need to comment and uncomment the lines that includes the package to avoid error messages.

By the time of this writing pdfTEX version 0.14a is available. Here it is possible to include an image and get its object number without setting it on the page. But I have not changed the method in `thumbpdf` in order not to get rid of the optimizations.

Adding thumbnails requires three steps, *jobname*`.tex` includes `thumbpdf.sty` (`\input` or `\usepackage`):

```
1. pdf(e)la(tex) jobname % Format required
2. thumbpdf jobname      % by jobname.tex
```

```
3. pdf(e)la(tex) jobname % Here thumbpdf.sty
                         % is required
```

8.4. Other images as thumbnails

Other images, symbols, or pictograms in the formats PNG, TIFF, or JPEG can be used as thumbnails instead of the reduced image of the page. If some symbols or pictograms should be included instead of reduced images, they can be specified with `\thisthumb` in the document TEX file. In the project `thumbpdf` the `thumbpdf.tex` picks up the thumbnail images. The extra symbols are announced to `thumbpdf.tex` by declaring them in the file `thumbopt.tex` that will be scanned, if it exists. This optional file consists of `\thumb` macros with an optional label and the filename as arguments. Although PNG, TIFF, and JPEG files can be used, only the file extension `.png` can be omitted.

The names, used in references later, can be the filenames themselves or labels, declared in the optional argument of `\thumb`:

thumbopt.tex		*jobname*.tex
`\thumb{one}`	→	`\thisthumb{one}`
`\thumb{one.png}`	→	`\thisthumb{one.png}`
`\thumb[two]{one}`	→	`\thisthumb{two}`
`\thumb[one]{./extras/one.png}`	→	`\thisthumb{one}`

9. Links

- PDF specification (version 1.3):
  `http://partners.adobe.com/asn/developer/PDFS/TN/PDFSPEC.PDF`
- pdfTEX (Hàn Thế Thành):
  `http://www.tug.org/applications/pdftex/`
  Win32: `ftp://ftp.ese-metz.fr/pub/TeX/win32-beta/`
- hyperref (Sebastian Rahtz):
  `ftp://ctan.tug.org/tex-archive/macros/latex/contrib/supported/hyperref/`
  If test versions are available, then they can be get from
  `http://www.tug.org/applications/hyperref/hyperref.zip`
- hypbmsec (Heiko Oberdiek):
  `ftp://ctan.tug.org/tex-archive/macros/latex/contrib/supported/oberdiek/`
- thumbpdf (Heiko Oberdiek):
  `ftp://ctan.tug.org/tex-archive/macros/pdftex/thumbpdf/`

◇ Heiko Oberdiek
oberdiek@ruf.uni-freiburg.de

PDF

(La)TEX i PDF: fonty i grafika

Wojciech Myszka

1. Wstęp

Format PDF cieszy się (zasłużenie) dużym zainteresowaniem wśród wszystkich, którzy zajmują się przygotowywaniem dokumentów elektronicznych. W porównaniu z dokumentami HTML, dokumenty PDF mają doskonałą postać graficzną umożliwiającą ich wydruk w założonej przez autora formie.

Dokument PDF może być także zabezpieczony hasłem przed możliwością drukowania czy ekstrahowania z niego ilustracji i/lub tekstu.

W porównaniu z formatem PS, dokumenty PDF są zazwyczaj mniejsze (dzięki zastosowaniu kompresji i różnych, zaawansowanych technik przetwarzania ilustracji). Podobnie jak pliki PS, dobrze przygotowane dokumenty PDF są „skalowalne”, to znaczy operacje powiększania i zmniejszania nie powodują utraty ich jakości.

Podobnie jak dokumenty HTML, pliki PDF są w pełni hipertekstowe; dzięki udostępnionym przez firmę Adobe rozszerzeniom mogą być przeglądane przez najpopularniejsze przeglądarki internetowe (Netscape i MS IE). Można nawet tak skonfigurować niektóre serwery WWW aby dostarczały odbiorcy pojedyncze strony dokumentów wielostronicowych w trybie „na żądanie”.

Co najważniejsze, specyfikacja PDF została udostępniona przez firmę Adobe a program Acrobat Reader pozwalający na przeglądanie i drukowanie plików PDF został przygotowany na bardzo różne platformy (DOS, Windows, Macintosh, Unix) i jest udostępniany za darmo.

Ciągle trwają przymiarki stworzenia niewielkiego (format strony książki) komputera (e–Book), który wykorzystując technologie Adobe (lub podobną) pozwoli na skonstruowanie „wirtualnej” książki.

Generalnie, wydaje się, że stworzenie dokumentu PDF jest łatwe:

1. Za pomocą dowolnego programu (procesora tekstu) nadajemy słowom odpowiednią formę zewnętrzną.
2. Przekształcamy wszystko do postaci pliku PostScriptowego (na przykład „drukując” do pliku, z wykorzystaniem sterownika PostScript).
3. Otrzymany plik przetwarzamy („destylujemy”) do postaci PDF używając jakiegoś przeznaczonego do tego programu.

2. Programy do przetwarzania pliku PS do postaci PDF

Podstawowym narzędziem używanym do konwersji PS → PDF jest Acrobat Distiller. Program dostępny jest na platformy MS Windows, Macintosh i Unix (tylko wersja 3 i nie jest dostępny dla Linuksa). Distiller produktem komercyjnym.

Innym programem, udostępnianym na licencji *shareware*, jest PStill[1]. Wersja pracująca w środowisku Unix/Linux jest darmowa do zastosowań osobistych.

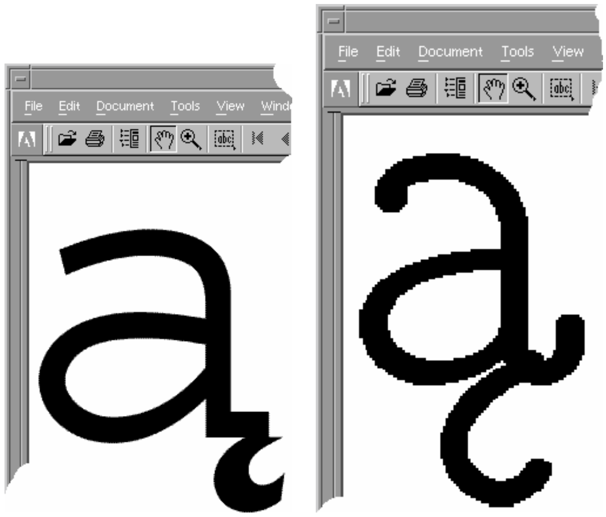
Kolejnym narzędziem rozpowszechnianym na jeszcze mniej ograniczających warunkach licencji jest Aladdin Ghostscript[2]. Niestety, ostatnia stabilna, podczas pisania tych słów, wersja programu (5.50) nie może jeszcze dorównywać produktowi Adobe – ma między innymi kłopoty z przetwarzaniem fontów dołączonych do pliku PS [3]. Wedle wszelkich zapowiedzi wersja 6 będzie to już robiła poprawnie.

3. (La)TEX i PDF

Zgodnie z tym co napisałem wcześniej, naturalną metodą tworzenia dokumentów PDF z wykorzystaniem (La)TEX-a jest kompilacja do postaci DVI, a następnie przekształcenie pliku DVI do postaci PostScriptowej za pomocą, na przykład, programu dvips. Plik PostScriptowy można już poddawać „klasycznej” obróbce.

O ile jednak system przetwarzania tekstów TEX dostępny jest dla wszystkich za darmo o tyle dalszy ciąg przetwarzania musi być już dokonywany za pomocą oprogramowania komercyjnego, jeżeli tylko zależy nam na przyzwoitej jakości.

W chwili obecnej dostępne są w większości spotykanych dystrybucji systemu TEX dwa, a właściwie



Rysunek 1: Przykład dokumentu złożonego różnymi fontami, po lewej stronie font skalowalny, po prawej EC

nawet trzy, bardzo intensywnie rozwijane, narzędzia do tworzenia plików PDF w środowisku TEX-owym:

- dvipdfm, autor: Mark A. Wicks [5];
- pdfTEX, autor: Hàn Thế Thành [1].

Trzecim narzędziem (nie będzie dalej omawiane) jest komercyjny[3] system VTEX firmy MicroPress[4].

Trudno dziś powiedzieć który z programów jest lepszy: pierwszy korzysta ze standardowego kompilatora systemu TEX i poddaje interpretacji plik DVI. Drugi tworzy bezpośrednio w procesie kompilacji pliku źródłowego albo plik DVI albo plik PDF. Trzeci ma wiele bardzo interesujących możliwości, których jeszcze nie zdołałem rozpoznać. . .

Większość wspomnianych wyżej problemów, które możemy napotkać w procesie tworzenia dokumentów PDF jest wspólna dla wszystkich narzędzi.

4. Fonty

Na czym polega problem z fontami? Otóż Acrobat Reader potrafi bardzo ładnie wyświetlać na ekranie znaki tworzone przy pomocy czcionek skalowalnych (Type 1, TrueType). Jeżeli jednak mamy do czynienia z czcionką rastrową (Type 3) otrzymany rezultat jest niezbyt dobry (spodziewać się można efektów podobnych do tych pokazanych na rysunku 1) a strony wyświetlane są znacznie wolniej.

[1] http://www.this.net/~frank/.
[2] http://www.aladdin.com.
[3] Wersja dla systemu Linux jest darmowa.
[4] http://www.micropress-inc.com.

Problem nie dotyczy w tym samym stopniu drukowania: plik PS będący podstawą do tworzenia pliku PDF budowany jest najczęściej w rozdzielczości 300 lub 600 dpi i drukowany w zbliżonej rozdzielczości.

Aby osiągnąć właściwy efekt powinniśmy tak skonfigurować oprogramowanie aby używało odpowiednich fontów: Type 1 lub TrueType. Powinniśmy również zainstalować w swoim systemie te fonty i wszystkie potrzebne do pracy TEX-a i dvips-a pliki z fontami związane. Tak się składa, że zarówno fonty Computer Modern, jak i oparte na nich fonty PL dostępne są w formacie Type 1.

Następnie należy przygotować dokument źródłowy w taki sposób aby skład tekstu wykonywany był z wykorzystaniem tych fontów.

„Klasyczne” preambuły tworzonych w LATEX dokumentów wyglądają jak poniżej:
najpierw z wykorzystaniem pakietu PLATEX:

```
%& --translate-file=cp1250pl
\documentclass[ ]{}
\usepackage[OT4]{polski}

\begin{document}
...
\end{document}
```

albo Babel:

```
%& --translate-file=cp1250pl
\documentclass[ ]{}
\usepackage[polish]{babel}
\usepackage[OT4]{fontenc}
\begin{document}
...
\end{document}
```

Ostatni przykład pozwala na wykorzystanie standardowych fontów PostScript:

```
%& --translate-file=cp1250pl
\documentclass[ ]{}
\usepackage[T1]{polski}
\usepackage{pslatex}
\begin{document}
...
\end{document}
```

Zwracam uwagę, że próba rezygnacji z wykorzystania fontów PL czyli zamiana parametru pakietu fontenc z OT4 na T1 nie pozwala na uzyskanie zadowalającego efektu: zarówno LATEX jak i pdfLATEX używać będą fontów EC, które zawierają wszystkie znaki niezbędne dla tworzenia dokumentów w językach łacińskich, ale nie są dostępne w postaci skalowalnej. Prawe okno programu Acrobat na rysunku 1 obrazuje właśnie ten przypadek. Z tego powodu można zalecić dodatkowo użycie pakietu pslatex (efekt widać na lewym oknie na rysunku 1).

Jeżeli chcemy tworzyć ładne pliki PDF w języku polskim, musimy mieć odpowiednie (to znaczy zawierające polskie litery) fonty Type 1 wraz ze wszystkimi niezbędnymi plikami do ich użycia w (La)TEX-u. Oprócz fontów PL (kodowanie OT4) nie mamy zbyt wielkiego wyboru, jeżeli chodzi o fonty darmowe: QuasiTimes i QuasiPalladio oraz Antykwa Toruńska to jedyne znane mi fonty z polskimi literami i z „pełnym” wsparciem dla TEX-a.

Możemy również skorzystać z „protez”: użyć pakietu ot1patch rozpowszechnianego wraz z pakietem PLATEX („dorabia” on ogonki do fontów CM) albo pakietu ae[5] albo (i to wydaje się być rozwiązaniem najrozsądniejszym, choć nie idealnym) pakietu pslatex. Dodatkową zaletą użycia tego pakietu jest to, że wykorzystuje on kroje Times, Courier i Helvetica, a więc te, które są dołączane do programu Acrobat Reader – zatem nie muszą być włączane do dokumentu.

Wadą takiego rozwiązania jest to, że polskie litery mają bardzo niestarannie „przyklejone” ogonki, najbardziej drastyczny przykład poniżej:

ąęĄĘ

Użycie programu Ghostscript do konwersji pliku PS do postaci PDF powoduje, że wszystkie „niestandardowe” (to znaczy dołączane w pliku PS) skalowalne fonty są przez niego konwertowane do postaci Type 3 (zamieniane do postaci rastrowej).

Zamiast fontów w formacie Type 1 można wykorzystać fonty TrueType (tylko pdfTEX). Szczegółowy opis jak to zrobić znajduje się na stronie `http://quantum.bitp.kiev.ua/radamir/ttf-tex.htm`; autorem jest Damir Rakityansky.

Pojawia się tu jednak kolejny problem związany ze specyfiką formatu PDF: wszystkie niestandardowe fonty muszą być włączone do wynikowego dokumentu. Ponieważ fonty objęte są na świecie ochroną wynikającą z praw autorskich – trzeba z tym bardzo uważać jeżeli chcemy utworzony dokument komuś przekazać lub udostępnić w Internecie:

[5] Każde archiwum lustrzane CTAN, np: `http://sunsite.icm.edu.pl/pub/CTAN/fonts/ae/`.

licencja na wykorzystanie zakupionych fontów może nie obejmować tego przypadku.

5. Grafika

Włączanie grafik do plików PDF jest (bardzo) proste: jeżeli korzystamy z LaTeX-a powinniśmy użyć pakietu graphicx i postępować dokładnie tak samo, jak w przypadku klasycznym (patrz również [2]).

Jedyny problem, który tu powstaje, wynika z tego, że każdy z wymienionych programów (dvips, pdfTeX oraz dvipdfm) korzysta z innych formatów graficznych (lub innych instrukcji do włączania grafik). Opcjonalny parametr w wywołaniu pakietu graphicx podpowiada jakie konkretnie instrukcje mają być użyte do włączania plików graficznych.

```
\usepackage[ ]{graphicx}
```

Powoduje to pewne kłopoty przy tworzeniu plików źródłowych, które mają służyć do klasycznego przetwarzania (TeX → DVI → PS), jak i tworzenia plików PDF w inny sposób.

Rozpatrzmy dwie sytuacje: przetwarzanie trójetapowe, z wykorzystaniem dvips i jednoetapowe: pdfTeX. W pierwszym przypadku wszystkie pliki graficzne muszą być przekształcone do formatu EPS. W drugim, program akceptuje plik PDF, PNG i JPG. Załóżmy, że mamy trzy pliki graficzne: `a.jpg`, `b.png` i `c.eps`. Ten ostatni zawiera grafikę wektorową. Musimy przetworzyć go do postaci PDF.

Program pdfTeX nie pozwala na konwersję „w locie” plików graficznych. Aby zaoszczędzić miejsce na dysku, będziemy dokonywali „w locie” konwersji plików JPG i PNG do formatu EPS.

```
%& --translate-file=cp1250pl
\documentclass[ ]{ }
\newif\ifpdf
\ifx\pdfoutput\undefined \pdffalse
\else \pdfoutput=1 \pdftrue \fi
\usepackage[OT4]{polski}
\ifpdf \usepackage[pdftex]{graphicx}
 \DeclareGraphicsExtensions{.pdf,.jpg,.png}
\else
 \usepackage[dvips]{graphicx}
 \DeclareGraphicsExtensions{.png,.eps,.jpg}
 \DeclareGraphicsRule{.jpg}{eps}{.jpg.bb}%
    {`jpeg2ps -q -r0 -h #1}
 \DeclareGraphicsRule{.png}{eps}{.png.bb}%
    {`convert #1 eps:-}
\fi

\begin{document}
\includegraphics[width=5cm]{a}
\includegraphics[width=5cm]{b}
\includegraphics[width=5cm]{c}
\end{document}
```

Pierwsze linie przykładu definiują nowe polecenie `\ifpdf` pozwalające rozróżnić obie sytuacje: przetwarzanie za pomocą programu LaTeX i za pomocą programu pdfLaTeX. Dalej korzystamy z nowej instrukcji definiując dwa środowiska pracy i rozpoznawane w nich typy plików graficznych. W przypadku kompilacji trójetapowej zostały również zdefiniowane reguły konwersji JPG → EPS i PNG → EPS. Wykorzystujemy w tym celu programy, które potrafią wynik konwersji zapisać „na terminal”. Są to jpeg2ps[6] i convert z pakietu ImageMagic[7].

Zasadnicza część dokumentu jest już jednakowa dla każdego przypadku przetwarzania. Wraz z plikiem źródłowym mamy dodatkowo pliki graficzne: `a.jpg`, `b.png`, `c.eps` i `c.pdf` oraz dwa pliki zawierające wymiary generowanych w locie grafik (musimy przygotować je sami, podpatrując wynik przetwarzania wykorzystanych programów) `a.jpg.bb` i `b.png.bb`, zawierające linię podobną do tej (patrz również [4]):

```
%%BoundingBox: 20 20 70 70
```

Podobnej sztuczki nie da się generalnie zastosować w przypadku użycia programu dvipdfm. Akceptuje on grafiki w formatach PDF, JPG, PNG. Posiada również „wbudowany” mechanizm konwersji „w locie” EPS → PDF (z wykorzystaniem programu GhostScript)[8]. Ponieważ jednak przetwarzanie odbywa się z wykorzystaniem programu (La)TeX, nie ma łatwej (to znaczy bez modyfikacji pliku źródłowego) metody rozróżnienia jak powinny być przygotowywane pliki DVI: dla programu dvips czy dvipdfm.

Zestaw nagłówków, który można zaproponować dla programu dvipdfm, może wyglądać tak:

```
\usepackage[dvipdfm]{graphicx}
\DeclareGraphicsExtensions{.png,.eps,.jpg,.pdf}
\DeclareGraphicsRule{.jpg}{eps}{.jpg.bb}{}
\DeclareGraphicsRule{.png}{eps}{.png.bb}{}
\DeclareGraphicsRule{.pdf}{eps}{.pdf.bb}{}
```

Rozpoznawane będą pliki typów (o rozszerzeniach) `.png`, `.eps`, `.jpg`, `.pdf`. Pliki `.png`, `.jpg` i `.pdf`

[6] `http://www.muc.de/~tm/free/free.html`

[7] `ftp://ftp.wizards.dupont.com/pub/ImageMagick/`

[8] Nie zawsze, zwłaszcza w przypadku plików EPS o charakterze „rastrowym”, daje to dobre rezultaty!

nie będą poddawane konwersjom, ale musimy zadbać o przygotowanie odpowiednich plików pomocniczych zawierających informację o rozmiarach wstawianej grafiki. Dostarczany wraz z dvipdfm program ebb służy do tworzenia odpowiednich plików .bb na podstawie plików PDF, PBG i JPG[9].

W poniższej tabeli zestawiono formaty plików graficznych akceptowanych przez pakiety LaTeX-a:

	EPS	PS	PDF	MP	PNG	JPG
dvips	$\oplus$	$\oplus$	$\ominus$	$\oplus$	$\ominus$	$\ominus$
pdfTeX	$\ominus$	$\ominus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
dvipdfm	$\odot$	$\ominus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
VTeX	$\oplus$	$\oplus$	$\oplus$	?	?	?

Uwagi: $\oplus$ – tak, $\ominus$ – nie, $\odot$ – częściowo.
dvips – ogólne określenie wszystkich tych sposobów, które wykorzystują PostScript jako plik źródłowy do tworzenia pliku PDF.
PS – możliwość interpretacji poleceń języka PostScript (na przykład związanych z użyciem pakietu PSTricks).

Wnioski wypływające z tabeli są generalnie takie: pliki EPS, będące swojego rodzaju standardem grafik w środowisku TeX-owym, nie nadają się do bezpośredniego włączania do plików PDF (jeżeli nie chcemy korzystać z metody: PS $\to$ PDF). Podobnie, ani pdfTeX ani dvipdfm nie będą potrafiły zinterpretować praktycznie żadnych poleceń języka PostScript włączonych za pomocą \special (PSTricks, psfrag).

Jedynie plik EPS tworzone przez program METAPOST mają na tyle prostą strukturę, że zarówno pdfTeX jaki dvipdfm potrafią przetłumaczyć je na elementarne polecenia PDF.

Jak zatem przygotowywać grafiki, które chcemy włączać do dokumentów PDF:

grafiki wektorowe powinny być przekształcone, o ile to tylko możliwe, do postaci PDF; programami, które można wykorzystać w tym celu są: distiller, epstopdf i PStill. Najlepszy jest ten pierwszy. epstopdf jest to prosty program w Perlu wykorzystujący Aladdin Ghostscript do interpretacji PostScriptu i dodający do pliku informację o „BoundingBox". Program obarczony jest wszystkimi wadami wspomnianymi wcześniej (fonty!). PStill poprawnie radzi sobie z zadaniem, chociaż tworzone pliki PDF są zazwyczaj duże i program zna tylko PostScript Level 1.

grafiki rastrowe. Tu powinniśmy użyć formatu PNG w przypadku grafik, w których zależy nam na pokazaniu szczegółów (skanowane rysunki techniczne, itp) lub formatu JPG w przypadku grafik z półtonami (zdjęcia). Format JPG ma ogromną zaletę: utworzone pliki wynikowe są stosunkowo małe. Ale ten efekt osiągany jest dzięki kompresji ze stratą jakości. Większość dobrych programów graficznych ma możliwość zapisywania w obu wspomnianych formatach. Oprócz tego, w pakiecie programów netpbm[10] są narzędzia, które nieco „okrężną" drogą pozwalają na przekształcenie nieomal dowolnego formatu graficznego w inny. Kolejnym darmowym narzędziem jest program ImageMagic.

W przypadku włączania grafik rastrowych pamiętać trzeba, że program distiller pozwala na bardzo zaawansowane przetwarzanie grafik rastrowych. Może on stosować różne techniki określane mianem „resamplingu" (generalnie zmiana rozdzielczości włączanej grafiki) oraz kompresji (łącznie ze stratną kompresją typu JPEG). dvipdfm i pdfTeX **tylko** włączają gotową mapę bitową; skalowania i obroty wykonywane są podczas interpretacji pliku PDF.

Bibliografia

[1] Hàn Thế Thành, Sebastian Rahtz, Hans Hagen. *The pdfTeX manual.* http://www.pragma-ade.nl/pdftex.htm. Wrzesień 1999.

[2] Wojciech Myszka. *Włączanie grafik w formacie* EPS *do tekstów w LaTeX 2ε i parę innych uwag.* Dokument dostępny w postaci elektronicznej: http://www.immt.pwr.wroc.pl/~myszka/grafika/. Ciągle uzupełniany. Październik 1999.

[3] Tomasz Przechlewski. *Publikacje elektroniczne: TeX i PDF. Biuletyn Polskiej Grupy Użytkowników Systemu TeX*, Zeszyt 10, 1997. Tekst dostępny w http://www.gust.org.pl/.

[4] Keith Reckdahl. *Using Imported Graphics in LaTeX 2ε.* Znakomity tekst poświęcony włączaniu grafik EPS do tekstów w LaTeX 2ε. Dostępny w postaci elektronicznej jako: http://sunsite.icm.edu.pl/pub/CTAN/info/epslatex.pdf. Grudzień 1997.

[5] Mark A. Wicks. *Dvipdfm user's manual.* http://odo.kettering.edu/dvipdfm/. Wrzesień 1999.

◇ Wojciech Myszka
W.Myszka@immt.pwr.wroc.pl

[9] Generowane przez ebb pliki .bb nie mogą być wykorzystane przez program dvips.

[10] Znajduje się w dystrybucji MikTeX-a; dostępny jest też na większość platform Unixowych.