Dvipdfm User's Manual

Mark A. Wicks

Version 0.12.4b

September 19, 1999

1. Introduction

This package is a DVI (TEX) to PDF conversion utility, having the following features:

- *Support for outline entries (also called bookmarks), named destinations, and annotations (including hyperlinks, forms and widgets).* Nearly every Acrobat Distiller pdfmark is approximated.
- *Support for "standard" DVI specials such as HyperTEX (HTML), TPIC, color specials,* `PSfile`*, and other PostScript specials.*
- *Native support for inclusion of MetaPost output and inclusion of arbitrary PostScript files with help from an external program.*
- *Support for thumbnails with a little help from GhostScript to generate the thumbnails.*
- *Support for arbitrary, nested linear transformations of typeset material.* Any material on the page, including TEX text, may be scaled and rotated.
- *Ability to include the first page of a PDF file as an encapsulated object along with its embedded resources such as fonts.* Included PDF images may be cropped by supplying a bounding box. Note: Currently, this doesn't work if the contents stream has multiple segments.
- *Ability to include JPEG and PNG bitmapped images as encapsulated objects.*
- *An internal color stack.* A color stack allows you to change the current color, pushing the current color onto a stack. At any time, the original color can be popped from the stack. This is useful, for example, in headlines, that may have a different color from the current text. The headline macro can restore the current color without knowing what it is.
- *Support for partial font embedding and Flate compression to reduce file size*
- *Support for font reencoding to work around encodings that aren't fully supported by the Acrobat suite of products.*
- *Balanced page and destination trees.* Balancing these trees improves reader access speed on very large documents.

The electronic version of the document exercises some of the hypertext features and serves as a sample input file for `dvipdfm`. It assumes the reader has some familiarity with the basic features of the Portable Document Format. The PDF specification is distributed by Adobe Systems[1]. An excellent source for information

about PDF documents in general is [2]. Information about using TEX to construct PDF documents (mainly using Distiller) is the AcroTeX home page[3].

Currently, the widely accepted method to generate PDF file from TEX is to use Adobe's Acrobat Distiller on a PostScript file produced by `dvips`. The hyperlink features are accessed by using TEX `special` primitives to embed pdfmarks in the PostScript produced by `dvips`. Hàn Thé Thàn's PDFTEX project is an alternative method of generating PDF from TEX source. Although quite good and fairly mature, the PDFTEX project modified TEX itself to add primitives that support the PDF features. I prefer to work with TEX unmodified, as released by Donald Knuth (call me a purist). There is an existing DVI to PDF driver called `dvipdf` written by Sergey Lesenko. At present, it's not widely available, so I haven't used it. I wrote `dvipdfm` mainly as an exercise to get at the features of PDF I was trying to use. This `dvipdfm` project demonstrates that many features of PDF can be accessed by using a DVI driver. The PDF features are activated in the driver via TEX `special` primitives.

Even though Distiller is the best method of generating PDF (and probably will remain so for some time) I have several reasons for seeking alternatives to Distiller. First, Distiller isn't available for my principle operating system—Linux.

My second objection is philosophical. TEX is a programming language. A DVI file is a page description consisting of very simple program instructions that have no branching or decision instructions. Similarly PostScript is a complete programming language, while PDF is a page description language consisting of simple program instructions without any branching or decision capabilities. TEX is like PostScript (without the graphics) while DVI is like PDF (without the graphics or the hyperlinks). Creating PDF from DVI using Distiller requires converting a page description to a program, and converting that program back to a page description. To continue this analogy, Pdfmarks are PostScript "escapes" and are meant for the Distiller. TEX `\special` primitives are TEX "escapes" and are meant for the DVI driver. It seems natural to go directly from DVI to PDF, where TEX replaces PostScript, the DVI driver replaces Distiller, and TEX `\special` primitives replace the pdfmarks.

Unfortunately, until graphics software begins to produce PDF content streams or encapsulated PDF objects, PostScript will remain the easiest way to include graphics in TEX documents. I would hope that in the future, graphics programs will begin to produce PDF content streams or PDF objects that may be included using a DVI to PDF translator. Either of these may be easily embedded using `dvipdfm` or a similar driver.

2. Command Line Options

Table 1 lists the options that are available on the command line.

Note: Several flags may be specified at once:

Example:

```
dvipdfm -ecz 9 test.dvi
```

3. General Concepts and Syntax for TeX Specials

Each TEX `\special` represents a separate command to the `dvipdfm` driver. Each `special` must begin with "`pdf:`" to identify that special as a command for the `dvipdfm` driver. A `\special` beginning with any other characters is ignored by the driver. Leading spaces are ignored. The characters "`pdf:`" are immediately followed by a `dvipdfm` command. These commands are documented in Sections 3–6.

3.1 PDF Object Syntax and Variable Expansion

With one exception, the syntax used for PDF objects within each `\special` specials follows the PDF specification. The one exception is variable expansion. In the syntax specifications that follow, *PDF_Object* means that an arbitary PDF object is expected. Similarly *PDF_Array* indicates that a PDF array is expected, *PDF_Dict* inciates that a PDF dictionary is expected, etc. See the reference manual for a complete list of PDF object types.

The single extension implemented in this driver allows a symbol name of the form `@`*name* whereever any PDF object is expected. The *name* may contain any characters allowed in a PDF name. A user-defined symbol beginning with `@` expands to an indirect reference to the user-defined PDF object. This feature replaces the `{`*name*`}` syntax used with pdfmarks. In addition to the user-defined names, some names are defined by the driver. The driver defined variables are for referencing objects such as the current page, future pages, or the current location on the current page. The driver defined variables appear in Table 2.

In the syntax specifications that follow, several standard conventions are followed. Terminal characters that appear in the command are typeset in the `\tt` font, e.g., `object`. Nonterminal symbols are typeset in italics. Optional parameters are surrounded by brackets, e.g., [*optional_argument*]. An item followed by "*" represents an item that may appear zero or more times. An item followed by "+" represents a required item that may appear multiple times.

3.2 Dimensions and transformations

Interaction with the `dvipdfm` driver consists of short commands with a few arguments delimited by white space. Typically the arguments are PDF objects. Two exceptions are dimension specifications and transformations.

In the TEX style, a dimension specification consists of one of the keywords `width`, `height`, or `depth` followed by a dimension consisting of a numerical value,

Table 1—Command line options recognized by `dvipdfm`

Option	*Description*
`-c`	*Disable color specials* Specifying this option forces all color commands to be ignored. This is useful for printing a color document on a black and white printer.
`-e`	*Disable partial font embedding.* This may be useful for forms which need complete fonts, or if you run accross a PFB file that `dvipdfm` cannot correctly parse.
`-f`	*Set font map file name.* See the Font Mapping section for additional information.
`-m` *number*	*Specify an additional magnification for the document.*
`-o` *filename*	*Set the output PDF file name.*
`-p` *papersize*	*Specify the output papersize.* Valid paper sizes are `letter`, `legal`, `ledger`, `tabloid`, `a4`, or `a3`. The default is `letter`. Arbitrary paper sizes can be specified via TEX special commands, which is the recommended method.
`-l`	*Landscape the document.* This is only meaningful for paper sizes specified on the command line.
`-s` *page_ranges*	*Select a subset of pages from the DVI file.* The *page_ranges* specifier is a set of comma-separated page ranges—e.g., `dvipdfm -s 10-12,10-20`. If the first page in a range is empty (e.g., `dvipdfm -s -10`), it represents the beginning of the document. If the last page in a range is empty (e.g., `dvipdfm -s 10-`), it selects the end of the document.
`-t`	*Embed thumbnail images.* The thumbnails must be generated by a separate program. Details are discussed later in this document
`-d`	*Delete thumbnail images after embedding.*
`-x` *number*	*Specify the horizontal offset for the document.* The default is 1.0in.
`-y` *number*	*Specify the vertical offset for the document.* The default is 1.0in.
`-z` *number*	*Specify the compression level.* Valid compression levels range from 0 to 9, with 0 indicating no compression. A compression level of 9 is the default.
`-v`	*Be verbose.* Among other things, dvipdfm will display complete file names as various files are opened.
`-vv`	*Be more verbose.*

Table 2—List of driver defined variables

Variable	*Description*
`@catalog`	A reference to the document's catalog.
`@names`	A reference to the document's /Names dictionary.
`@pages`	A reference to the root of the document's /Pages tree.
`@resources`	A reference to the current page resource dictionary.
`@thispage`	A reference to the current page.
`@page`*n*	A reference to page *n*.
`@nextpage`	A reference to the page following `@thispage`.
`@prevpage`	A reference to the page preceding `@thispage`.
`@ypos`	A *number* representing the current vertical position in units of PDF points.
`@xpos`	A *number* representing the current horizontal position in units of PDF points.

followed by a unit for the dimension. The unit will typically be `pt` (which represents a TEX point, not a PDf point) but `mm`, `cm` and `in` are also allowed. If the document is magnified, the "true" dimensions `truept`, `truemm`, `truecm`, and `truein` may be used. The notation *dimension* in a syntax description means a dimension is expected.

A transformation consists of one of the keywords `scale`, `xscale`, `yscale`, or `rotate` followed by a numerical value. In the case of `rotate` the value is the rotation angle in degrees. The notation *transformation* means a transformation is expected. In the case of included images originating from PDF/PostScript files, a clipping or bounding box may be specified as part of the "transformation." The bounding box specification consists of the keyword `bbox` followed by four numerical values, which are the standard PostScript `llx`, `lly`, `urx`, and `ury` coordinates of the bounding box.

4. Document Construction Commands

All commands are executed via TEX `\special` primitives prefixed with the characters "`pdf:`".

Example:

```
\special{ pdf: out 1 << /Title (Introduction)
                        /Dest [ 1 0 R /FitH 234 ] >>
```

4.1 Annotate

Syntax: `annotate` [`@`*name*] *dimension+ PDF_dictionary*

Description: The `annotate` (`annot` or `ann`) command defines an annotation. Annotations are typically used for notes, hyperlinks, forms, or widgets. The parameter *name* is an optional alphanumeric identifier and *PDF_dictionary* is a valid PDF dictionary after variable expansion. If `@`*name* is specified, it may be used in other PDF objects to refer to this annotation. One or more *dimension* parameters are required and each consists of the keyword `height`, `width`, or `depth` followed by an appropriate length, specified as per TEX. The `width` must be nonzero and either the `height` or `depth` must be nonzero. Each length is a number followed by a unit, such as `pt`, `in`, or `cm`. Since these values would typically be entered by TEX, a `pt` is a TEX point, not a PDF point.

Example: The annotation in this subsection was typeset with

```
\special{pdf: ann width 3.0in height 36pt
     << /Type /Annot /Subtype /Text
        /Contents (This is a /Text Annotation
        that looks like a sticky note.) >>}
```

4.2 Begin Annotation

Syntax: `beginann` *PDF_dictionary*

Description: The `beginann` (`bann` or `bannot`) command begins a breakable annotation. Dvipdfm tries to determine the boundaries of the annotation automatically. Such an annotation may be broken over several lines (or even several pages). Unlike `annot`, the annotation may be not be referred to by name later, since there may be more than once instances of the annotation. The `beginann` command needs not dimension because `dvipdfm` computes the correct bounding box on the fly.

Note: Link breaking is easily fooled at page breaks. If you have problems with the header or footer wanting to become part of the link, you can insert a `nolink` command at the beginning of the footer and a `link` command at the end of the footer.

4.3 End Annotation

Syntax: `endann`

Description: The `endann` (`eann` or `eannot`) terminates a breakable annotation.

4.4 Link

`link`

Description: The `link` command instructs the link breaking algorithm to resume its operation after a `nolink` command.

4.5 Nolink

`nolink`

Description: The `nolink` command instructs the link breaking algorithm to suspend it operation. This is only required if there is material inserted on the page in the *middle* of a link. For example, when a link is broken at the end of a page, the header and footer are inserted in the middle of the link, but are not part of the link. Material that is inserted between a `nolink` and `link` will not become part of the link. The `nolink` and `link` escapes are not necessary with the standard LaTeX header and footer lines, but may become necessary if these are changed.

4.6 Dest

Syntax: `dest` *PDF_String PDF_Dest*

Description: The `dest` command defines a named destination. The *PDF_String* is a PDF string naming the destination. This string may be used in the destination fields of annotations and outline entries to refer to this destination. *PDF_Dest* is a PDF destination object (typically an array).

Example:

```
\special{pdf: dest (listofreferences) [ @thispage /FitH @ypos ]}
```

4.7 Docinfo

Syntax: `docinfo` *PDF_dictionary*

Description: The `docinfo` command adds the keys in the specified dictionary to the document's `/Info` dictionary. All keys are optional, but may include the keys `/Author`, `/Title`, `/Keywords`, `/Subject`, and `/Creator`.

Example:

```
\special{pdf: docinfo << /Author (Mark A. Wicks)
                         /Title (This Document) >>}
```

4.8 Docview

Syntax: `docview` *PDF_dictionary*

Description: The `docview` command adds the keys in the specified dictionary to the document's `/Catalog` dictionary. All keys are optional, but may include the

keys `/PageMode`, `/URI`, `/OpenAction`, `/AA` and `/ViewerPreferences`. See the PDF Reference Manual for documentation of these keys and additional keys.

Example:

```
\special{pdf: docview << /PageMode /UseThumbs >> }
```

4.9 Object

Syntax: `object` [@*name*] *PDF_Object*

Description: The `object` (also `obj`) command creates a PDF object. The parameter *PDF_Object* is any valid PDF object. The parameter *name* may be used to provide an indirect reference to this object within other objects. It will be expanded anywhere within a `special` where a PDF object is expected. Typically *object* is an array or dictionary. It may be an empty array or dictionary that can be constructed dynamically via the `put` command.

Example:

```
\special{pdf: object @mydict << /Firstpage @thispage >>}
```

4.10 Out

Syntax: `out` *number PDF_dictionary*

Description: The `out` (also `outline`) command adds an outline (also called a "bookmark") entry to the document. The parameter *level* is an integer representing the level of the outline entry (beginning with 1) and *PDF_dictionary* must contain the two keys `/Title` and either `/Dest` or `/A`. It may also contain the `/AA` key. These keys are documented in the PDF Reference Manual.

Example:

```
out 1 << /Title (Section 1) /Dest [ @thispage /FitH @ypos ] >>
```

which may be followed by

```
out 2 << /Title (Section 1.1) /Dest [ @thispage /FitH @ypos ] >>
```

Note: You may not skip levels. A level 2 outline entry must follow a level 1 outline entry. A level 3 outline entry must follow a level 2 outline and cannot immediately follow a level 1 outline entry.

4.11 Pagesize

Syntax: `pagesize` *dimension+*

Description: The `pagesize` command specifies the document's physical paper size. The `pagesize` command must be specified on the first page and must precede the first annotation or background color specification on the page. In other words, it should occur as close to the beginning of the document as possible.

Example:

```
pagesize width 11.0truein height 8.5truein
```

4.12 Put

Syntax:

`put` @*name PDF_Object+*

or

`put` @*name PDF_Dictionary*

Description: The `put` command modifies an existing PDF object created with `obj`, or one of the following internally defined objects: `@catalog`, `@names`, `@pages`, `@thispage`, or `@resources`. The first form is used when @*name* is an array. The second form is used when @`name` is a dictionary. More than one object may be added to an array at once. All keys in *PDF_Dictionary* are added to the dictionary represented by @*name*.

Example:

```
\special{pdf: put @mydict << /Nextpage @thispage >>}
```

4.13 Thread

Syntax: `thread` @*name dimension+* [*PDF_dictionary*]

Description: The `thread` (or `art`) command adds a bead to an article. An article is a collection of boxed regions in the document that should be read consecutively. Each bead using the same *name* belongs to the same article. The *name* parameter is required. The *dimension* parameter defined the rectangular area belonging to the bead in the same manner as for `annot`. The optional PDF dictionary should supplied on one of the beads. Its keys are similar to the `/Info` dictionary accessed via the `docinfo` command and would typically include the `/Title` and `/Author` keys. Keys in the dictionary may be overwritten by subsequent `thread` commands.

Example:

```
\special {pdf: thread @somearticle << /Title (Some title)
                                      /Author (Me) >>}
```

4.14 Close

Syntax: `close` @*name*

Description: The `close` writes the named PDF object created with `obj` to the PDF file. No further `put` commands may be executed for this object. The object may continue to be referenced using @*name* indefinitely. If the object is never closed, it will be closed when `dvipdfm` finishes processing the document.

5. Form XObjects

The PDF specification allows an object to be stored once and displayed at multiple locations throughout the document. The following commands give access to this facility.

5.1 Beginxobj

Syntax: `beginxobj` @*name dimension+*

Description: The `beginxobj` (or `bxobj`) command begins the definition of a Form XObject. All material typeset between the `beginxobj` and `endxobj` commands will be captured into the XObject. The material can be displayed later at an arbitrary location with the `usexobj` command. The *name* may be used to refer to the object later, either via the `usexobj` command or as an indirect reference to the XObject if *name* is used within the context of a *PDF_Object.*. The required *dimension* identifies the extent (i.e., bounding box) of the area to be captured. It is specified in the same way as for `annot`.

The material will not display during the object definition. In other words, if you are typsetting with TEX you should place the XObject in a box of dimension 0 so you don't leave a white space hole where the object was defined.

Example:

```
bxobj @myform width 2.0in height 24pt
```

5.2 Endxobj

Syntax: `endxobj`

Description: The `endxobj` (or `exobj`) command ends the previous `beginxobj` definition. Note that XObject definitions may not be nested. XObjects can be *used* within other XObjects, however.

Example:

```
exobj
```

5.3 Usexobj

Syntax: `usexobj` @*name*

Description: The `usexobj` (or `uxobj`) command displays the form XObject previously defined and associated with *name*.

Example:

```
uxobj @myform
```

6. Text Transformation Commands

The commands in this section deal with transformation of arbitrary material, which may include material typeset by TEX. These may also be used on included graphics images if the commands in Section 8 won't do the job.

6.1 BeginTransform

Syntax: `begintransform` *transformation+*

Description: The `begintransform` (`btrans` or `bt`) applies the specified transformation to all subsequent text. The scaling is applied first, followed by the rotation. The reference point of a box following the `\special` remains fixed. Such transformations may be nested to perform rotations within rotated text, for exmaple.

Example:

```
\special{pdf: bt rotate 90 xscale 2.0 }
```

6.2 BeginTransform

Syntax: `endtransform`

Description: The `endtransform` (`etrans` or `et`) concludes the action of the immediately preceding `begintransform` command. All transformations must be closed on the same page. The driver will close any pending unclosed transformations at the end of the page and issue a warning message. All material to be transformed should probably be enclosed in a single box to prevent any break.

Example:

```
\special{pdf: et}
```

7. Color Commands

The commands in this section deal with manipulation of the color stack.

7.1 Setcolor

Syntax: `setcolor` *PDF_Number*|*PDF_Array*

Description: The `setcolor` (`scolor` or `sc`) command uses its argument to set the default color for future marking operators. The current color is replaced and may be retrieved only by a subsequent `setcolor` command. The argument may be a single number, which is interpreted as a grayscale value; a three element array, which is interpreted as an RGB color space coordinate; or a four element array, which is interpreted as a CMYK color space coordinate.

Example:

```
\special{ pdf: sc [ 1 0 0 ] }
```

7.2 Begincolor

Syntax: `begincolor` *PDF_Number*|*PDF_Array*

Description: The `begincolor` (`bcolor` or `bc`) command uses its argument to set the default color for future marking operators. The current color is pushed on the color stack. The argument may be a single number, which is interpreted as a grayscale value; a three element array, which is interpreted as an RGB color space coordinate; or a four element array, which is interpreted as a CMYK color space coordinate.

Example:

```
\special{ pdf: bc [ 1 0 0 ] }
```

7.3 Endcolor

Syntax: `endcolor`

Description: The `endcolor` (`ecolor` or `ec`) changes the default color to match the color on the top of the stack. It removes the color from the stack.

Example:

```
\special{ pdf: ec }
```

7.4 Bgcolor

Syntax: `bgcolor` *PDF_Number*|*PDF_Array*

Description: The `bgcolor` (`bbc` or `bgc`) command uses the value of its argument to set the default color for the page background. The interpretation fo the argument is the same as for the `begincolor` command. The stack is not involved here. There is no way to go back to the previous background color.

Example:

```
\special{ pdf: bc [ 1 0 0 ] }
```

8. Image Commands

The commands in this section deal with embedding graphics into your PDF document. The present driver supports PDF, PNG, and JPEG graphics inclusion.

8.1 Epdf

Syntax: `epdf` [`@`*name*] [*dimension*|*transformation*]* *PDF_String*

Description: The `epdf` command "encapsulates" the first page of a PDF file named by *PDF_String* into a PDF XObject. The resulting XObject is drawn with the lower left corner at the current location of the page. The optional @*name* parameter may be used to reference this object within other objects. If a *dimension* is supplied, the object will be scaled to fit that dimension. A *transformation* consists of one of the keywords `scale`, `xscale`, `yscale`, or `rotate` followed by a number representing the scaling factor or rotation angle in degrees. Both *transformation* and *dimension* parameters can be supplied as long as they are not logically inconsistent.

Note: The object is stored as an XObject and can be redisplayed later by using the `usexobj` function and specifying *name*.

Example:

```
\special{pdf:epdf yscale 0.50 width 4.0in
                                rotate 45 (circuit.pdf)}
```

8.2 Image

Syntax: `image` [@*name*] [*dimension* | *transformation*]* *PDF_String*

Description: The `image` command "encapsulates" an image taken from the file named by *PDF_String*. This command functions just like `epdf`. Value image types may be PDF, JPEG, or PNG images. This special will eventually replace the `epdf` special.

Note: The object is stored as an XObject and can be redisplayed later by using the `usexobj` function and specifying *name*.

9. Raw Page Marking Commands

The commands in this section deal with embedding raw PDF graphics operators into your PDF document.

9.1 Bop

Syntax: `bop` *stream*

Description: The `bop` command specifies a marking stream to be generated at the top of each page. The parameter *stream* is any sequence of marking operators and is added to the page's content stream. The stream is applied *to all pages* regardless of where it appears in the document.

Example: The two horizontal lines appearing at the top of each page in this document were set with

```
\special {pdf: bop  q 0 w 0.8 0.5 0 RG
               54 740 m 504 740 l 504 740.25 l 54 740.25 l b
               36 760 m 504 760 l 504 760.25 l 36 760.25 l b Q }
```

9.2 Content

Syntax: `content` *stream*

Description: The `content` command specifies a marking stream to be added to the current page at the current location. While it is possible to change the color state, etc., with this command, it is not advised. Use the color management commands to change colors.

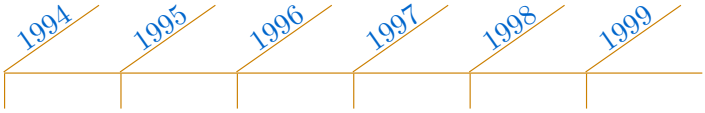
9.3 Eop

Syntax:

`eop` *stream*

Description: The `eop` specifies a marking stream to be generated at the end of each page. The parameter *stream* is any sequence of marking operators and is added to the page's content stream. The stream is applied *to all pages* regardless of where it appears in the document.

Table 3—Example of rotated text set in Computer Modern Roman



10. Graphics Examples

The examples in this section illustrate some of the transformation and image inclusion capabilities of `dvipdfm`.

10.1 Text Transformation

Tables with slanted entries are possible as shown in Table 3. This table was achieved using various "`bt rotate 35`" commands.

The following line of text was done with nested combinations of "`bt rotate 10`" and "`bt rotate -10`".

You can nest the text transformation capabilities to achieve effects like this.

10.2 Image Inclusion

The image in Figure 1 was included from a JPEG file. The image shown in Figure 2 comes from the same file, but is loaded at a 50% scale and a 45° rotation.

By default, JPEG files are included at a resolution of 100dpi so if you know the pixel size of the image, you know how much space to reserve. Any TEX magnification is applied to the image in addition to any scaling defined in the `\special`. For example, this document sets `\magnification=\magstephalf`, so the images are actually scaled by 1.095. The first image in this section has a printed width of 1.643in even though 1.50in was specified in the `\special`.

Several command line utilities exist that read the pixel dimensions of a JPEG file. For PDF files, you can `grep` on `/MediaBox` to get an indication of the image size. The `/MediaBox` dimensions are in PDF points.

The image in Figure 3 was produced by embedding a PDF file using `epdf`.

Notice that any resources required for the object are also embedded. In this case, the Times Roman font resource was embedded along with the content stream.

Figure 1—A JPEG image of the author.

Figure 2—Image of the author scaled by 0.5 and rotated by 45°.

11. Thumbnails

Thumbnails can be inserted automatically by `dvipdfm` using the `-t` command line option. However, `dvipdfm` is unable to generate the thumbnails by itself. This must be done by some other program such as GhostScript. The typical two-pass process to include thumbnails would be

1. Run `dvipdfm` without the `-t` option (probably using command line options optimized for speed and not space)
2. Run GhostScript to generate the thumbnail images of each page.

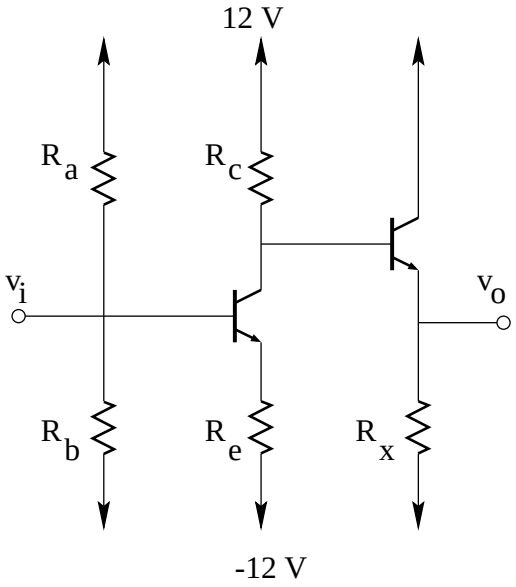


Figure 3—An embedded PDF object.

3. Run `dvipdfm` with the `-t` or `-dt` option.

The `dvipdfm` distribution includes a sample shell script called `dvipdft` which is a wrapper for `dvipdfm` that generates thumbnails.

`Dvipdfm` searches for the thumbnail images in the directory specified by the TMP or TEMP environment variables, or in the current directory. The thumbnails must have the same base name as the PDF output file with an extension indicating the number of the page. For example, to create a thumbnailed document named `foo.pdf` you would need to generate thumbnail images in files named `foo.1`, `foo.2`, etc.

12. Font Mapping

TEX font names can be mapped into arbitrary physical (PostScript) font names via the map file named `t1fonts.map`. The file is similar to the `psfonts.map` file used by `dvips` and other drivers. Each line in the file consists of one to three fields delimited by white space, followed by options that apply to that font. The complete list of recognized options appears in Table 4.

Sample map file lines are

```
cmr10 ot1 -r
ptmro8r 8r Times-Roman -s 0.167
pncbo8r 8r pncb8a -s 0.167
```

Table 4—Font options recognized by `dvipdfm` in the font map file

Option	*Description*
`-r`	*Remap the font to eliminate control characters.* This option attempts to work around bugs in Acrobat reader that seem to be triggered by characters that are encoded in certain positions. Unfortunately, standard TEXencodings normally use these positions.
`-e` *number*	*Extend the font horizontally, multiplying the natural width of the font by the specified number.* This is useful for generating new fonts by widening existing fonts.
`-s` *number*	*Slant the font using the specified number.* This option is useful for building a slanted fonts from a font for which there is no slanted font.

The first field of each line is the TEX font name. The second field, which is optional, is the encoding name (`.enc` will be appended to this name, if necessary to locate an encoding file). The encoding files have the same format as those used for `dvips`. The third optional field is *either* the PostScript font name (if one of the standard PostScript fonts) *or* the file name of a Type 1 binary font file (PFB file). The .pfb extension is assumed by the program. three fields. If the encoding field is unspecified, `dvipdfm` used the default encoding supplied with the Type 1 font. If the name field is unspecified, `dvipdfm` looks for a Type 1 binary font file having the same name as the TEX font name. If there is *no* line in the map file, the behavior is as if a line was specified without the second or third field. The keywords `default` or `none` are recognized in the encoding field so that a third field may be specified while still having default behavior in the second field.

Currently, three options may be specified for each font as shown in Table 4.

13. Configuration file

Dvipdfm reads default command line options, such as paper size, from a configuration file contained in the `dvipdfm` directory of the TeX tree. The format of this file is similar to the `dvips` file configuration file format. Each line consists of a single command line switch followed by any arguments. See the configuration file supplied with the `dvipdfm` distribution for examples.

14. Including PostScript graphics images

Table 5—Expansions performed in the -D string.

Variable	*Description*
`%i`	*Expanded to the full path of the EPS file to be included*
`%o`	*Expanded to the full path name of the PDF file to be generated.* The command line must be able to create a file with the name supplied by dvipdfm.
`%b`	*Expanded to the "basename" of the EPS file that is being included.* The "basename" is the full path of the EPS file with the last "." and any trailing name extension removed.

Dvipdfm provides support for illustrations contained in external PostScript files. It recognizes two distinct kinds of PostScript files—those created by MetaPost and everything else.

MetaPost files output only a subset of PostScript and are interpreted *natively* by `dvipdfm` with its own internal minimal PostScript interpreter. Also, MetaPost output files contain TEX font information in their header comments, so that they can be easily integrated with TEX by DVI drivers. Dvipdfm recognizes this font information and uses the TEX fonts defined in the MetaPost header.

All other PostScript files require an external program to convert the image to PDF format before `dvipdfm` can include the image. The freely available GhostScript is capabable of performing this conversion for most images.

The user must specify the command line required to invoke an external program to perform this conversion. The command line required to invoke the conversion program is specified using the `-D` command line (or configuration file) option. The string passed to the `-D` command line option is a C-style string that is parsed by `dvipdfm`. Within the string, expansions are performed as described in Table 5. For example, to use GhostScript, one might use the command line

```
-D "cat %i | gs -q -sDEVICE=pdfwrite -sOutputFile=%o - -c quit"
```

15. Compatibility with other DVI drivers

Compatibility with other DVI drivers is achieved by support for the following "standard" special calls. Many legacy DVI files can be processed by `dvipdfm`, so long as they use fairly standard special commands. For example, `dvipdfm` can process DVI files generated that use the `\color`, `\rotatebox`, and `\includegraphics` macros from the standard LaTEX Graphics bundle even if `dvips` was the target DVI driver, e.g.,

```
\usepackage[dvips]{graphics}
\usepackage[dvips]{color}
```

Specifically, `dvipdfm` understands the following standard special commands:

- The TPIC specials.
- The HyperTeX (HTML) hyperlink specials.
- The "color" specials supported by `dvips` and other drivers.
- The "PSfile" and "plotfile" specials for PS/EPS file inclusion supported by `dvips` and other drivers. Dvipdfm uses a user-specified external program to convert the PostScript file to PDF format before including it.
- The raw PostScript "ps:" special supported by `dvips` and other drivers. Only a few PostScript operators are supported; `dvipdfm` does not include a complete PostScript interpreter. Complex PostScript code, such as that embedded by the PSTricks package, is not supported.

16. LaTeX Support and Ebb

Support for the LaTeX graphics bundle and `hyperref` are available. A driver file named `dvipdfm.def` is distributed with the standard LaTeXgraphics bundle. The latest version of the `dvipdfm.def` file is supplied with the `dvipdfm` distribution. The file required for `hyperref` support is called `hdvipdfm.def`.

To facilitate LaTeX support, I distribute a companion program called `ebb`, which extracts bounding boxes from graphics files. If you want to include JPEG, PNG, or PDF files in your document, you can run `ebb` on the JPEG, PNG, or PDF file to create the `.bb` files. The bounding box file will be similarly named with an extension of `.bb`. For DOS 8+3 compatibility, an original file name extension of `.jpg`, `.png`, or `.pdf` is removed before creating the name of the `.bb` file. An extension of `.jpeg` is also recognized and similarly removed.

17. References

[1] *Portable Document Format Reference Manual*, Version 1.2, Adobe Systems Incorporated, 1996. Available at the following URL: `http://www.adobe.com`.

[2] Thomas Merz, *Web Publishing with Acrobat/PDF*, Springer-Verlag, 1997, ISBN 3-540-63762-1. Chapter 6 of this book is available at the URL: `http://http://www.ifconnection.de/~tm`.

[3] D. P. Story, *AcroTeX*, The AcroTeX home page is located at the URL: `http://www.math.uakron.edu/~dpstory/acrotex.html`.