Dvipdfm User's Manual

Mark A. Wicks

Version 0.7 November, 1998

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 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.* Note: Currently, this doesn't work of the contents stream has multiple segments.
- *Ability to include a JPEG image as an encapsulated object.*
- *A 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.

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. General Concepts and Syntax

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.

2.1 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.

Table 1—List of driver defined variables

Variable	*Description*
`@thispage`	An *indirect reference* to the current page.
`@page`	An *indirect reference* to page *n*.
`@nextpage`	An *indirect reference* to the page following the current page.
`@prevpage`	An *indirect reference* to the page preceding the current page.
`@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.

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 1.

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.

2.2 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, followed by a unit for the dimension. The unit will typically be `pt` (which represents a TEX point, not a PDf point) but `cm` and `in` are also allowed. 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.

3. 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 ] >>
```

3.1 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.
                   Aren't these things ugly?.
                   It's a good thing they don't
                   print by default.) >>}
```

3.2 Article

Syntax: `article` `@`*name PDF_dictionary*

Description: The `article` (or `art`) command initializes an article. An article is a collection of boxed regions in the document that should be read consecutively.

The *name* parameter is required. The required PDF dictionary is similar to the `/Info` dictionary accessed via the `docinfo` command and would typically include the `/Title` and `/Author` keys.

Example:

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

3.3 Bead

Syntax: `bead` `@`*name dimension+*

Description: The `bead` command adds a rectangular area to an existing article thread. The parameter *dimension+* specifies a rectangular area in the same manner as for an annotation. The *name* must correspond to an existing `article`.

Example:

```
\special{pdf: bead @someart width 156pt height 20pt depth 4pt}
```

3.4 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 ]}
```

3.5 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) >>}
```

3.6 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 >> }
```

3.7 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 >>}
```

3.8 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.

3.9 Put

Syntax:

`put` @*name PDF_Object+*

or

`put` @*name PDF_Dictionary*

Description: The `put` command modifies an existing PDF object created with `obj`. 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 >>}
```

3.10 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.

4. 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.

4.1 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 }
```

4.2 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}
```

5. Color Commands

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

5.1 Begincolor

Syntax: `begincolor` *PDF_Array*

Description: The `begincolor` (`bcolor` or `bc`) command uses the array to set the default color for future marking operators. The current color is pushed on the color stack. The array must have three elements specifying the coordinates of the color in the Device RGB color space.

Example:

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

5.2 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 }
```

6. Image Commands

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

6.1 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.

Example:

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

6.2 Image

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

Description: The `image` command "encapsulates" a JPEG image taken from the file named by *PDF_String*. Otherwise, this command functions just like `epdf`.

7. Raw Page Marking Commands

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

7.1 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 }
```

7.2 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.

7.3 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.

8. Graphics Examples

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

8.1 Text Transformation

Tables with slanted entries are possible as shown in Table 2. This table was achieved using various "`bt rotate 35`" commands. It is difficult to do without macro support.

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.

Table 2—Example of rotated text set in Computer Modern Roman

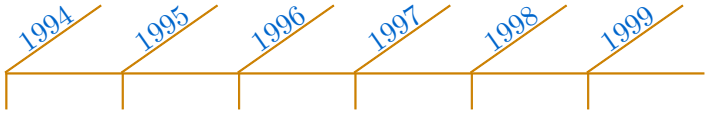

Figure 1—A JPEG image of the author.

8.2 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.

Sophisticated macro support for images is not yet available. For now, you need to enclose the image in your own box of the correct size so TEX reserves space for it. No space is reserved for a `special` unless you reserve it. I used the following simple Macro for the previous images:

```
\def\reserve#1#2#3{\setbox0\hbox{}\wd0=#1
                   \ht0=#2\special{#3}\box0}
```

The arguments to the `\reserve` macro are the width, height, and `\special` contents, respectively. The first image in this section was included with

```
\centerline{\reserve{1.50in}{2.05in}{pdf: image (mwicks.jpeg)}}
```

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

Until macro support is more developed, you will need to know the dimensions of the image. 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.

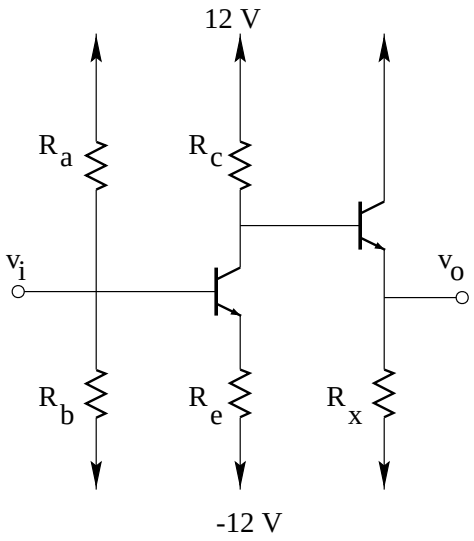

Figure 3—An embedded PDF object.

9. 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`.