

1. Introduction

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

- Support for outline (bookmark) entries, named destinations, and annotations (including hyperlinks, forms and widgets).
- Ability to include arbitrary PDF files as encapsulated objects along their any embedded resources such as fonts.
- Ability to include JPEG images as encapsulated objects.
- A color stack.

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 T_EX to construct PDF documents (mainly using Distiller) is the AcroT_EX home page [3].

Currently, the widely accepted method to generate PDF file from T_EX is to use Adobe's Acrobat Distiller on a PostScript file produced by `dvips`. The hyperlink features are accessed by using T_EX `special` primitives to embed pdfmarks in the PostScript produced by `dvips`. Hàn Thê Thàn's PDFT_EX project is an alternative method of generating PDF from T_EX source. Although quite good and fairly mature, the PDFT_EX project modified T_EX itself to add primitives that support the PDF features. I prefer to work with an unmodified T_EX. This `dvipdfm` project demonstrates that many of the features of PDF can be accessed by using a DVI driver. The PDF features are activated in the driver via T_EX `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. T_EX 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. T_EX 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 arbitrary PDF object is expected. Similarly *PDF_Array* indicates that a PDF array is expected, *PDF_Dict* indicates 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` wherever 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.

Table 1—List of driver defined variables

<i>Variable</i>	<i>Description</i>
<code>@thispage</code>	An <i>indirect reference</i> to the current page.
<code>@page</code>	An <i>indirect reference</i> to page <i>n</i> .
<code>@nextpage</code>	An <i>indirect reference</i> to the page following the current page.
<code>@prevpage</code>	An <i>indirect reference</i> to the page preceding the current page.
<code>@ypos</code>	A <i>number</i> representing the current vertical position in units of PDF points.
<code>@xpos</code>	A <i>number</i> representing the current horizontal position in units of PDF points.

2.2 Dimensions and scalings

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 scalings.

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 scaling consists of one of the keywords `scale`, `xscale`, or `yscale` followed by a numerical value. The notation *scaling* means a scaling 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 T_EX. 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 T_EX, a **pt** is a T_EX 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. Arrays are incremented one object at a time. 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. Color Commands

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

4.1 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 ] }
```

4.2 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 }
```

5. Image Commands

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

5.1 Epdf

Syntax: `epdf` [*@name*] [*dimension|scaling*]* *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 *scaling* consists of one of the keywords `scale`, `xscale`, or `yscale` followed by a number representing the scaling factor. Both *scaling* and *dimension* parameters can be supplied as long as they are not logically inconsistent.

Example:

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

5.2 Image

Syntax: `image` [*@name*] [*dimension | scaling*]* *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`.

6. Raw Page Marking Commands

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

6.1 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 }
```

6.2 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.

6.3 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.

7. Image Examples

Figure 1 is a photograph of the author that was included from a JPEG file: The image shown in Figure 2 image comes from the same file, but is loaded at a 50% scale.

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 1—A JPEG image of the author.



Figure 2—The same JPEG image of the author scaled to 50.%

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 are available to 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.

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.

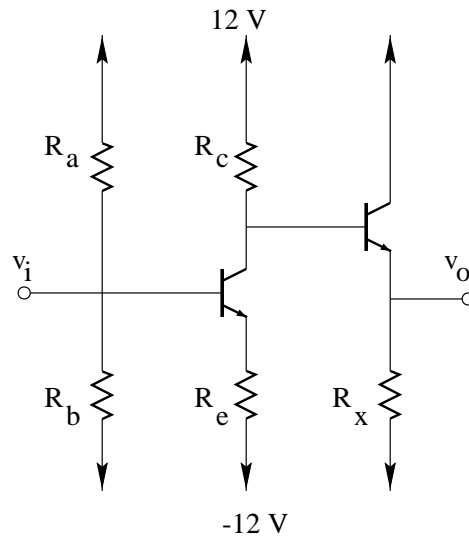


Figure 3—An embedded PDF object.

8. 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>.