Dvipdfm User's Manual

Mark A. Wicks

Version 0.7dev

November, 1998

1. Introduction

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

- A user interface replicating much of the functionality Adobe Acrobat Distiller®.
- Support for outline (bookmark) entries, named destinations, annotations (including forms and widgets).
- Ability to include arbitrary PDF files as encapsulated objects.
- Ability to include JPEG images as encapsulated objects.
- A color stack.

Currently, the widely accepted method to generate PDF file from TEX is to use 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 solution. Although quite good and fairly mature, the PDFTEXproject required fairly significant modifications to TEX itself to add primitives to support the PDF features. This project demonstrates that much of the functionality of Acrobat Distiller can be achieved by using a DVI driver. The PDF features are activated via TEX `\special`primitives.

From a technical standpoint, Distiller® will probably remain the best approach for some time. However, I have several objections to the use of Distiller® and feel that this driver provides a viable option. One objection is that Distiller® isn't available for Linux—my principle operating system.

My second objection is philosophical. A DVI file is a page description. It is essentially a linear program with no branching or decision instructions. PostScript® is a complete programming language, while PDF is a page description language without any branching or decision capabilities. TEX is like PostScript® (without the graphics) while DVI is like PDF (without the graphics or the hyperlinks). Using Distiller® requires going from page description to program back to page description. Pdfmarks are PostScript® features, which are meant for the Distiller®, are analogous to TEX`\special`primitives, which are meant for the DVI driver. It seems natural to go directly from DVI to PDF, where TEX replaces PostScript® and where the DVI driver replaces and implements `\special`primitives similar to the `pdfmarks` in Distiller®.

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 produce PDF content streams, or PDF objects that may be included into a DVI to PDF translator. Either of these may be easily included using `dvipdfm` or a similar driver.

2. General Concepts

This document describes the `dvipdfm` driver. It also 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.

Each TEX `\special` constitutes 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 Section 3. Another feature of the driver is variable expansion within PDF objects—specifically arrays and dictionaries. The driver maintains a symbol table. Some of these variables are user defined and some are driver defined (read only). The read only drivers are for referencing the current page, future pages, or the current location on the page, for example. User defined variables are references to user defined PDF objects.

3. Dvipdfm Commands

All commands are executed via TEX `\special` primitives prefixed with `pdf:`, e.g.,

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

3.1 Ann

The `ann` command defines an annotation. Annotations are typically notes, hyperlinks, PDF forms, or PDF widgets. The `ann` command takes the form:

`ann` [`@`*name*] *dimension+ PDF_dictionary*

where name is an optional alphanumeric identifier and 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. Each length is a number followed by a unit, such as `pt`, `in`, or `cm`. A `pt` is a TEXpt, not a PDF pt.

```
\special{pdf: ann width 144pt height 10pt depth 2pt
           << /Type /Annot /Subtype /Link /Border [1 0 0]
              /A << /S /GoTo [ @nextpage /Fit ] >> }
```

3.2 Out

The `out` (also `outline`) command adds an outline (also called a "bookmark") entry to the document.

`out` *number PDF_dictionary*

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.

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

Which may be followed by

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

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.

3.3 Article

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 `article` command takes the form:

`article @`*name PDF_dictionary*

The *name* parameter is required. The required PDF dictionary is similar to the `docinfo` dictionary and should include the `/Title` and `/Author` keys.

```
article @somearticle << /Title (Some title) /Author (Me) >>
```

3.4 Bead

The `bead` adds a rectangular area to an existing article thread. It has the form:

`bead @`*name dimension+*

where *dimension+* specified a rectangular area in the same manner as for an annotation. The *name* must correspond with an existing `article`.

```
bead @someart width 156pt height 20pt \space depth 4pt
```

3.5 Dest

The `dest` command defines a named destination.

`dest` *PDF_String PDF_Dest*

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 (an array).

```
dest (listofreferences) [ @thispage /FitH @ypos ]
```

3.6 Docinfo

`docinfo` *dictionary*

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`.

3.7 docview

`docview` *dictionary*

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.

3.8 Epdf

`epdf` [@*name*] [*dimension*‖*scaling*]+ *PDF_String*

The `epdf` command "encapsulates" the first page of a PDF file named by *PDF_String* into a PDF XObject. The resulting XObject is drawn at the current location of the page. The current point represents the lower left-hand corner of

the XObject's coordinate system. 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.

```
dpdf yscale 0.50 width 4.0in (circuit.pdf)
```

3.9 Object

`object` [@*name*] *object*

The `object` (also `obj`) command creates a PDF object. The parameter *object* is any valid PDF object. The parameter *name* may be used to refer to this object within other objects. It will be expanded within any `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.

```
object @mydict << /Firstpage @thispage >>
```

3.10 put

`put` @*name object*

or

`put` @*name dictionary*

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 *dictionary* are added to the dictionary represented by @*name*.

```
object @mydict << /Nextpage @thispage >>
```

3.11 close

`close` @*name* The `close` writes a 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. Additional functions

4.1 bop

`bop stream`

The `bop` command specifies a marking stream to be generated at the top of each page.

4.2 eop

`eop stream`

The `eop` specifies a marking stream to be generated at the top of each page.

5. References

Portable Document Format Reference Manual, Version 1.2, Adobe Systems Incorporated, 1996. Available from `http://www.adobe.com`.