Dvipdfm User's Manual

Mark A. Wicks

Version 0.7dev

November, 1998

1. Background

Currently, the most widely accepted method to generate PDF file from TEX was to use Adobe's Acrobat distiller on a Postscript file produced by `dvips`. The hyperlink features are accessed by using TEX `\specials` 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 `\specials`.

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 Acrobat 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 Acrobat 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`\specials`, 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 `\specials` similar to the `pdfmarks` in Adobe's Acrobat 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 `dvipdfm` driver. It also servers as an example input file for `dvipdfm`, accessing some of the hypertext features. It assumes 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 \specials prefixed with `pdf:`, e.g.,

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

A complete list of commands follows:

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*] *dimensions+ 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 *dimensions* 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.
Example:

```
\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

`out` *level dictionary*

The parameter *level* is an integer representing the level of the outline entry (beginning with 1) and *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.

3.3 ARTICLE

3.4 DEST

3.5 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`.

3.6 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.

3.7 epdf

`epdf` [@*name*] *filename*

The `epdf` command "encapsulates" the first page of a PDF file 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 with other objects. It will be expanded to a reference for this object within any `special` where a PDF object is expected.

3.8 obj

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

The `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.

3.9 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*.

3.10 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.

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`.