dvipdfm, version 0.7dev User's Manual

Mark A. Wicks

1. Background

At the time I wrote `dvipdfm`, 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`. Han The Than's PDFTEXproject is an alternative solution. Although quite good and fairly mature, the PDFTEXproject required modifying the TEX source code to add primitives to support the PDF features. I have a firm belief that TEXshould remain pristine unless a compelling case can be made that certain features cannot be implemented with TEX \specials. At least one other DVI to PDF project exists, but it wasn't widely available.

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 people need other options. One objection is that it isn't available for Linux—my principle operating system. Also, the conversion to Postscript as an intermediate step seems unnatural. TEX is a programming language.

My second objection is philosophical. The DVI specifies a page description language. A DVI file contains no branching or decision instructions. Similarly Postscript is a programming language, while PDF is a page description language without any branching or decision capabilities. In some sense TEX is analogous to postscript (without the graphics) while DVI is analogous to 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. Introduction

This document describes and serves as an example input file for `dvipdfm` version 0.7dev. It assumes some familiarity with PDF.

3. Functions analogous to PDFmarks

These functions are all executed via TEX \specials prefixed with `pdf:`, e.g.,

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

3.1 ann

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 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`.

3.4 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.

3.5 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.

3.6 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.

3.7 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*.

3.8 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.