Software & Tools

DVI specials for PDF generation

Jin-Hwan Cho

Abstract

DVIPDFM(x) manages various PDF effects by means of DVI specials. Appropriate documentation of DVI specials, however, is not easy to find, and exact functionality is not simple to catch without reading the source code of DVI drivers. This paper deals with the DVI specials defined in DVIPDFM(x) that are mainly used for PDF generation. We discuss the features of those specials with some examples, many of which are not documented elsewhere.

1 Introduction

DVI, the output file format of D. E. Knuth's TEX, is not widely used at present compared with PDF, the output format of pdfTEX. It is rather old[1] and obsolete, but it has powerful aspects nonetheless: *simplicity* and *compactness*.

These aspects make it possible to manipulate DVI files in an easy and fast way. Many DVI utilities were developed to convert the DVI format to other file formats including PostScript and PDF. It is also possible to edit DVI files directly by the use of DVIasm [5, 6, 7].

Twenty years ago, at the time PostScript dominated the printing world, nobody expected a new format would replace PostScript. PDF is not eternal either. In future, when a new format surpassing PDF appears, DVI will be the first format in the TEX world that can be converted to the new format. Notice that LuaTEX, considered to be the next generation of pdfTEX, still supports the DVI format.

There are two popular ways to convert DVI to PDF. The first one is a two-way conversion, from DVI to PostScript with DVIPS, and then from PostScript to PDF with a distiller. Adobe Acrobat Distiller is the oldest commercial program, and Ghostscript is the most popular distiller in the TEX world. Mac OS X also has its own distiller.

Adobe designed the pdfmark operator [2] for its distiller to support PDF features that are not expressible using the standard PostScript operators. The pdfmark operator is given in the TEX source by means of a DVI special command. Note that it is not DVIPS but a distiller that processes the pdfmark operator.

[1] DVI was designed by David R. Fuchs in 1979.

Mark A. Wicks' DVIPDFM [11] introduced the other way of converting DVI directly to PDF. He also designed new DVI specials based on the pdfmark operator to support various PDF features. The new specials, however, lacked some functionality in practical use so that not many PDF features could be obtained compared with pdfTEX.

One of the main goal of DVIPDFMx, an extension of DVIPDFM that grew out of the CJK[2] support, was to provide as many PDF features as pdfTEX [3]. DVIPDFMx extended the functionality of some special commands of DVIPDFM, and designed new special commands having a similar functionality of pdfTEX's own primitives. Furthermore, DVIPDFMx has several powerful features not available in DVIPDFM.

- Support 16-bit character sets (CJK encodings and Unicode) with CID-keyed font technology.
- Support various font formats including OpenType, TrueType, etc.
- Use CFF font format for embedded Type1 PostScript fonts so that the size of the PDF output is quite small compared with pdfTEX's output.
- Support extended TEX engines, e.g., Omega, Japanese pTEX, XeTEX (via XDVIPDFMX).[3]

The TODO list of DVIPDFMx had contained one outstanding item for a long time: supporting Till Tantau's beamer package [9], that is widely used for PDF presentation. In fact, this package does not handle DVI specials in a direct way. Instead, the graphics part comes from the same author's PGF package [10], and the other PDF effects come from the hyperref package [8].

DVIPDFMx has supported full functionality of the PGF package since June 2008.[4] Nonetheless, the navigation buttons usually shown in the lower right corner of the presentation still did not work, although they were displayed correctly. The source code[5] implementing the buttons was

```
\def\beamer@linkspace#1{\vbox to7.5pt{}\kern#1}
```

The code above generates an *empty* box that will be surrounded by the two special commands, '`pdf:bann`' (before) and '`pdf:eann`' (after). Unfortunately, neither DVIPDFM nor DVIPDFMx construct any annotation in the case of an empty box. Another special command '`pdf:ann`' must be used instead for

[2] Chinese, Japanese, and Korean.

[3] Upcoming version of DVIPDFMx will support the DVI output generated by LuaTEX.

[4] The DVIPDFMx driver that works with the PGF package included in TEX Live 2008 can be downloaded from `http://project.ktug.or.kr/dvipdfmx/contrib/generic/`.

[5] `http://mirror.ctan.org/macros/latex/contrib/beamer/base/beamerbasenavigation.sty`

this purpose. That was the exact reason why the navigation buttons did not work.

Why did the author of the beamer package make such a mistake? As a matter of fact, it was not his fault because no statement could be found about that functionality in the manual of DVIPDFM [11]. This unhappy story led to this paper.

The author gave a presentation [4] at TUG 2005, in which the different behaviors of DVI specials of DVIPS, DVIPDFM, and DVIPDFMx were discussed. DVI specials for PDF generation, however, were not fully discussed at that time. The main objective of this paper is to bridge this gap.

We will discuss in the following sections the features of DVI specials defined in DVIPDFM for PDF generation, and the extended features given by DVIPDFMx. The author hopes this paper would be useful for package writers who are finding appropriate information on DVI specials.

2 Named PDF objects

There are two kinds of named objects, built-in and user-defined PDF objects.

2.1 Built-in named objects

Built-in objects defined in DVIPDFM(x) are listed in Table 1. We refer to [2, p. 12] and [11, p. 5] for pdfmark and DVIPDFM built-in objects, respectively. Notice that it is not allowed to modify the contents of the last five built-in objects in Table 1.

`@catalog`	catalog dictionary [1, p. 139]
`@docinfo`	(DVIPDFMx only) document information dictionary [1, p. 844]
`@names`	name dictionary [1, p. 150]
`@pages`	root page tree node [1, p. 143]
`@resources`	resource dictionary of current page [1, p. 154]
`@thispage`	current page object [1, p. 145]
`@prevpage`	reference only
`@nextpage`	reference only
`@page`n	reference only
`@xpos`	reference only
`@ypos`	reference only

Table 1: Built-in objects defined in DVIPDFM(x)

2.2 User-defined named objects

Two special commands are provided by DVIPDFM(x) for user-defined objects. One is to define a named object, and the other is to add content to the previously defined object.

- `pdf:obj @name PDFobject` creates a named object that can be referenced later by '`@name`'.

All possible object types for '`PDFobject`' are listed in Table 2. In the case of indirect objects, the object number must be given explicitly, so that this feature is rarely used, especially to specify the objects in a different PDF file.

boolean	`true`, `false`
numeric	`123`, `34.5`, `-.002`
string	`(This is a string)`, `<901FA3>`
name	`/Name1`, `/.notdef`
array	`[3.14 false (Ralph) /Name1]`
dictionary	`<</Key1 (Value) /Key2 3.14>>`
null	`null`
indirect	`12 0 R`
stream	`stream ... endstream`

Table 2: PDF object types [1, p. 51]

It is not simple to construct a stream object with the special command '`pdf:obj`' because the length of the stream object must be specified explicitly, which is quite bothersome. Imagine that you are trying to construct a stream object whose source comes from a file. Is it possible with this special command? Moreover, any stream object requires the keyword '`stream`' followed by an end-of-line marker.[6]

DVIPDFMx, therefore, provides new special commands for stream objects.

- `pdf:stream @name (string) <<dict>>` constructs a stream object the source of which comes from the string object '`(string)`'. The stream dictionary '`<<dict>>`' is optional, and the dictionary entry '`/Length`' is created automatically.

The following two special commands, for instance, construct the same stream object. The stream data of the second object is represented in the ASCII base-85 encoding. [1, p. 70]

```
\special{pdf:stream @name (xxxxxxxx)}
\special{pdf:stream @name (G^+IXG^+IX)
        <</Filter/ASCII85Decode>>}
```

- `pdf:fstream @name (filename) <<dict>>` constructs a stream object in the same way as '`pdf:stream`', but the source of stream data comes from a file '`filename`'.

The following example shows how to include a source TEX file inside the output PDF file. (See [1, p. 637] for more details on the file attachment annotation.)

```
\special{pdf:fstream @myfile (mytest.tex)}
\special{pdf:ann bbox 0 0 10 10 <<
```

[6] An end-of-line marker consists of either a carriage return (`0x0d`) and a line feed (`0x0a`) or just a line feed, and not by a carriage return alone [1, pp. 60–61].

```
/Subtype /FileAttachment /FS <<
  /Type /Filespec /F (mytest.tex)
     /EF << /F @myfile >> >>
/Name /PushPin >>}
```

2.3 Adding content to named objects

We describe the special command for adding content to named objects. The type of the named object must be either array or dictionary.

- `pdf:put @arrayobj object`$_1$ `...` `object`$_n$ appends the n objects at the end of the array object '`@arrayobj`'.
- `pdf:put @dictobj <<dict>>` merges the dictionary object '`<<dict>>`' into '`@dictobj`'. If both the dictionaries have a common key, the old value in '`@dictobj`' will be replaced by the new value in '`<<dict>>`'.

In the following example, the value of the key '`/X`' in the dictionary object '`@name`' is '`@Moon2`'. (See [2, p. 15] for corresponding pdfmark operators.)

```
\special{pdf:put @Moon1
        [ (Earth to Moon) 238855 /mies ]}
\special{pdf:obj @Moon2 []}
\special{pdf:put @Moon2 (Moon to Earth)}
\special{pdf:put @Moon2 238855}
\special{pdf:put @Moon2 /miles}
\special{pdf:obj @name <<>>}
\special{pdf:put @name << /X @Moon1 >>}
\special{pdf:put @name << /X @Moon2 >>}
```

Note that DVIPDFM does not allow adding content to a stream dictionary object, but DVIPDFMx does.

- `pdf:put @streamobj <<dict>>` merges the dictionary object '`<<dict>>`' into the stream dictionary of '`@streamobj`'. The dictionary entries, '`/Length`' and '`/Filter`', in the object '`<<dict>>`' will be ignored.

Finally, DVIPDFM(x) provides the special command `pdf:close @name` to prevent further modifying the content of '`@name`'. After closing the named object, it can only be referenced.

3 Annotations

An annotation is considered as an object with a location on a page. The type of the object is given by the value of the key '`/Subtype`', for instance, '`/Text`', '`/Link`', '`/Sound`', '`/Movie`', etc. (See [1, p. 615] for the list of all annotation types.) The location is given by an array object associated to the key '`/Rect`'. DVIPDFM(x) provides the following special command for annotations.

- `pdf:ann @name width [length] height [length] depth [length] <<dict>>`

  The annotation dictionary is given by '`<<dict>>`' and the location relative to the current position is given by the three dimension parameters, '`width`', '`height`', and '`depth`'.

It is not possible to specify the location in an absolute way. Any value of the key '`/Rect`' in the annotation dictionary '`<<dict>>`' will be ignored if found. It is not allowed to modify the annotation dictionary with '`pdf:put`' command, so '`@name`' must be used as a reference.

Note that DVIPDFMx allows the '`bp`' unit in the dimension parameters, but DVIPDFM does not. Moreover, DVIPDFMx supports the following form.

- `pdf:ann @name bbox [ulx] [uly] [lrx] [lry] <<dict>>`

  The relative location is given by the bounding box consisting of four numbers in '`bp`' units.

The following example shows a movie annotation that enables us to run the movie file '`mymovie.avi`' inside a PDF viewer program.

```
\special{pdf:ann bbox 0 0 360 180 <<
  /Subtype /Movie /Border [1 0 0]
  /T (My Movie) /Movie <<
    /F (mymovie.avi) /Aspect [720 360]
    /Poster true >>
  /A << /ShowControls false >> >>}
```

DVIPDFM(x) provides other special commands for *breakable* annotations, e.g., an annotation broken over several lines or several pages.

- `pdf:bann <<dict>>` begins a breakable annotation. Object name is not allowed for this command.
- `pdf:eann` terminates the previous breakable annotation.

These specials are mainly used for '`/Link`' annotation as the following example shows.

```
\special{pdf:bann << /Subtype /Link
  /BS << /Type /Border /W 0.5 /S /S >>
  /A << /S /URI
    /URI (http://www.tug.org) >> >>}%
http://www.tug.org%
\special{pdf:eann}
```

Warning: No annotation will be constructed if the content between '`pdf:bann`' and '`pdf:eann`' is an empty box. For example:

```
\special{pdf:bann << /Subtype /Link ... >>}
\vbox to 7.5pt{}\kern 10pt
\sepcial{pdf:eann}
```

Annotations constructed by DVIPDFM(x) may happen to be slightly bigger than the expected size. This occurs when the annotation grow size is positive; this value is specified in the configuration file. To

avoid this effect, either modify the configuration file or give '`-g 0`' on the command line when running DVIPDFM(x).

4 Outlines (or bookmarks)

The document outline consists of a tree-structured hierarchy of outline items (sometimes called bookmarks) for which DVIPDFM(x) provides the following special command.

- `pdf:out` n `<<dict>>` adds an outline item to the document. The integer parameter n represents the level of the outline entry (beginning with 1), and '`<<dict>>`' represents the outline item dictionary [1, p. 585].

Note that all the outline items generated by DVIPDFM are closed.[7] The '`bookmarksopen=true`' option of the hyperref package does not work if the PDF output is generated by DVIPDFM.

```
\usepackage[
  dvipdfm,bookmarks=true,bookmarksopen=true
]{hyperref}
```

DVIPDFMx provides two solutions for this problem. The first one is to specify the option '`-O` n' when running DVIPDFMx. Up to level n, the outline entries will be open. The second, and complete, solution is to use this extended special command:

- `pdf:out [-]` n `<<dict>>` The symbol '`[-]`' indicates that the outline item will be closed. On the other hand, '`[]`' without the minus sign indicates that the outline item will be open.

The hyperref package provides a new option '`dvipdfmx-outline-open`' that uses the extended command above. This option enables us to control the open level given by '`bookmarksopenlevel`'.

```
\usepackage[%
  dvipdfmx,bookmarks=true,
  bookmarksopen=true,
  bookmarksopenlevel=1,
  dvipdfmx-outline-open
]{hyperref}
```

5 External objects (or XObjects)

DVIPDFM(x) supports two types of *external objects*,[8] an image XObject and a form XObject.

- `pdf:image @name width [length] height [length] depth [length] (imagefile)` defines an image XObject the source of which comes from the file '`imagefile`'. See [5, p. 216] for complete syntax provided by DVIPDFMx.
- `pdf:bxobj @name width [length] height [length] depth [length]` begins the definition of a form XObject. As with the command '`pdf:add`', DVIPDFMx allows bounding box '`bbox [ulx] [uly] [lrx] [lry]`' for dimension parameters.
- `pdf:exobj` ends the previous form XObject definition.
- `pdf:uxobj @name` displays the image XObject or the form XObject previously defined and associated with '`@name`'. DVIPDFMx allows dimension parameters (same as '`pdf:image`') after '`@name`'.

Typical examples showing how to use image XObjects and form XObjects can be found in [4, pp. 15–16].

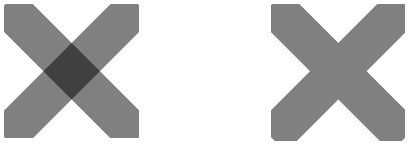

Figure 1: Two form XObjects with opacity 0.5; the right one is a group XObject.

DVIPDFMx extended the command '`pdf:eann`' to support a group XObject.[9] Figure 1 shows the difference between a normal XObject and a group XObject.

- `pdf:exobj <<dict>>` merges the dictionary object '`<<dict>>`' into the type1 form dictionary [1, p. 358] of the previous form XObject and then close the XObject.

The following code draws the right image in Figure 1.

```
\special{pdf:bxobj @group bbox 0 0 50 50}
\special{pdf:code
    15 w 0 0 m 50 50 l S 50 0 m 0 50 l S}
\special{pdf:exobj << /Group
   << /S /Transparency >> >>}
\special{pdf:obj @extgstate
   << /CA0.5 <</CA 0.5>> /ca0.5 <</ca 0.5>> >>}
\special{pdf:put @resources
   << /ExtGState @extgstate >>}
\special{pdf:code /CA0.5 gs /ca0.5 gs}
\special{pdf:uxobj @group}
```

We get the left image if `\special{pdf:exobj}` is used instead of the 4th and the 5th line.

[7] The sign of the value of the key '`/Count`' in the outline item dictionary determines whether the item is open or closed. [1, p. 586]

[8] "A graphics object whose contents are defined by a self-contained content stream, separate from the content stream in which it is used." [1, p. 332]

[9] "A special type of form XObject that can be used to group graphical elements together as a unit for various purposes." [1, p. 360]

6 Raw PDF Operators

This final section deals with writing raw PDF operators in the output. DVIPDFM provides a special command for this feature.

- `pdf:content Operators` adds the list of operators "`Operators`" to the current page at the current location. The operator '`q`', saving the current graphics state, followed by a transformation matrix moving to the current location will be attached to the beginning of the list, and the operator '`Q`' restoring the saved graphics state at the end of the list.

  For instance, the special command

```
\special{pdf:content 10 w 0 0 m 50 50 l S}
```

inserts the following list of operators in the output.

... `q` `1 0 0 1` *x y* `cm` `10 w 0 0 m 50 50 l S` `Q` ...

We sometimes need to insert PDF operators without additional graphics state operators. The author of the PGF package devised a trick:

```
\special{pdf:content Q ... Operators ... q}
```

The first operator '`Q`' and the last operator '`q`' nullify the effects of graphics state operators that are attached.

DVIPDFM*x* provides a new special command instead of the trick above.

- `pdf:literal direct Operators`[10] or simply `pdf:code Operators` plays the same role as '`pdf:content`, but no graphics state operator and no transformation matrix will be added.

Figure 2: The location of '23' in the left image varies according to the location of 1 in the current page.

Consider the following code, labelled Listing 1. Which image in Figure 2 does this code generate?

```
\def\bpic{\special{pdf:content q}}
\def\epic{\special{pdf:content Q}}
\def\myop#1{\special{pdf:content Q #1 q}}
1\bpic2\myop{.5 G 10 w 0 0 m 100 100 l S}3\epic4
```

Listing 1: Which image in Figure 2 is the result of this code, produced by DVIPDFM(*x*)?

The macro `\bpic` in Listing 1 nullifies the effect of the operator '`Q`' that will be attached after '`q`', and the macro `\epic` nullifies the effect of the list '`q 1 0 0 1` *x y* `cm`' that will be attached before '`Q`'.

Most people may choose the right-hand image in Figure 2 as the result of Listing 1, if they remember the fact that special commands are considered nothing by TEX. However, the answer is the left-hand image. The reason is that the transformation matrix in the macro `\bpic` still has an effect on the characters '2' and '3'. The effect will be nullified by the macro `\epic`.

To produce the right-hand image, DVIPDFM*x* provides the following new special commands.

- `pdf:bcontent` starts a block that works in the same way as '`pdf:content`' except that all text between this command and '`pdf:econtent`' will be placed in the right position.
- `pdf:econtent` ends the current block.

Moreover, '`pdf:bcontent`' and '`pdf:econtent`' can be nested.

Finally, we can get the right-hand image in Figure 2 as the result of Listing 2 following, produced by DVIPDFM*x*.

```
\def\bpic{\special{pdf:bcontent}}
\def\epic{\special{pdf:econtent}}
\def\myop#1{\special{pdf:code #1}}
1\bpic2\myop{.5 G 10 w 0 0 m 100 100 l S}3\epic4
```

Listing 2: The right-hand image in Figure 2 is the result of this example produced by DVIPDFM*x*.

[10] The idea of '`pdf:literal direct`' came from the primitive '`\pdfliteral direct`' of pdfTEX.

References

[1] Adobe Systems, Inc., *PDF Reference*, 6th edition (Version 1.7, November 2006). `http://www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf`.

[2] Adobe Systems, Inc., *pdfmark Reference* (Version 8.0, November 2006). `http://www.adobe.com/devnet/acrobat/pdfs/pdfmark_reference.pdf`.

[3] Jin-Hwan Cho, *DVIPDFMx, an extension of DVIPDFM*, TUG 2003. Hawaii, United States. `http://project.ktug.or.kr/dvipdfmx/doc/tug2003.pdf`.

[4] Jin-Hwan Cho, *Practical Use of Special Commands in DVIPDFMx*, TUG 2005, International Typesetting Conference. Wuhan, China. `http://project.ktug.or.kr/dvipdfmx/doc/tug2005.pdf`.

[5] Jin-Hwan Cho, *Hacking DVI files: Birth of DVIasm*, The PracTEX Journal (2007), no. 1, and TUGboat 28:2, 2007, 210–217. `http://tug.org/TUGboat/Articles/tb28-2/tb89cho.pdf`.

[6] Jin-Hwan Cho, *Handling Two-Byte Characters with DVIasm*, The Asian Journal of TEX **2** (2008), no. 1, 63–68. http://ajt.ktug.kr/assets/2008/5/1/0201cho.pdf.

[7] Jin-Hwan Cho, *The DVIasm Python script*. http://mirror.ctan.org/dviware/dviasm/.

[8] Heiko Oberdiek, *The hyperref package* (Version 6.78f, August 2008). http://mirror.ctan.org/macros/latex/contrib/hyperref/

[9] Till Tantau, *The beamer package* (Version 3.07, March 2007). http://mirror.ctan.org/macros/latex/contrib/beamer/.

[10] Till Tantau, *PGF, A Portable Graphics Format for TEX* (Version 2.00, February 2008). http://mirror.ctan.org/graphics/pgf/.

[11] Mark A. Wicks, *DVIPDFM User's Manual* (Version 0.12.4, September 1999). http://gaspra.kettering.edu/dvipdfm/dvipdfm-0.12.4.pdf.

◇ Jin-Hwan Cho
Department of Mathematics
The University of Suwon
Republic of Korea
chofchof (at) ktug dot or dot kr