For submission to The PracTEX Journal
Draft of February 1, 2007

Hacking DVI files: Birth of DVIasm

Jin-Hwan Cho

Email chofchof@ktug.or.kr
Address Department of Mathematics, The University of Suwon, Republic of Korea

Abstract This paper is devoted to the first step of developing a new DVI editing utility, called DVIasm. Editing DVI files consists of three parts: disassembling, editing, and assembling. DVIasm disassembles a DVI file into a human-readable text format which is more flexible than DTL, and assembles the output back to a DVI file.

DVIasm is quite useful for people who have a DVI file without TEX source, but needs to modify the document. It enables us to put a preprint number, a watermark, or an emblem on the document without touching the TEX source. DVIasm is quite attractive to even a TEX expert who wants to modify a few words in his document more than hundred pages long.

We discuss in the paper how DVIasm plays a role as supplementary to TEX. The current version supports only the standard DVI file format as DVItype and DTL. The next versions will support 16-bit TEX extensions including Omega, pTEX, and XETEX.

1 Introduction

Have you ever heard of DVI, not the Digital Visual Interface[1] but the DeVice-Independent file format? At least ten years ago every TEX user knew what DVI is and used DVI utilities to view and print out the TEX results. However, recent TEX users drew attention to DVI less and less because pdfTEX outputs directly to the PDF[2] file format. It is true without doubt that PDF is more powerful than DVI in

1. A video interface standard designed to maximize the visual quality of digital display devices such as flat panel LCD computer displays and digital projectors [Wikipedia, http://en.wikipedia.org/wiki/DVI].
2. PDF (Portable Document Format) is an open file format created and controlled by Adobe Systems, for representing two-dimensional documents in a device independent and resolution independent fixed-layout document format [Wikipedia, http://en.wikipedia.org/wiki/PDF].

almost all aspects. Then, do we have to obsolete DVI as PostScript is gradually replaced by PDF?

The DVI file format was designed by David R. Fuchs in 1979, in contrast to the release of PDF version 1.0 in 1993. It is intended to be both compact and easily interpreted by a machine [4, §14]. The most powerful aspect of DVI compared to PDF is nothing but *simplicity*. Imagine the speed of three previewers of DVI, PostScript, and PDF, and compare also the file size of the three different file formats. Furthermore, simplicity enables us to control DVI files in various ways. One of them is to edit DVI files directly, that is the main object of this paper.

There are many applications of editing DVI files. The most critical situation is when we have a DVI file without TEX source, but we want to modify or to add something to the document. A technical editor may want to put a preprint number on each paper without touching the TEX source. He may also want to put a watermark or an emblem on every paper.

Editing a DVI file is much quicker for a TEX novice than learning TEX, when all he wants is to give some decorations in his document, but has some trouble in writing TEX codes. It is even quite attractive to a TEX expert who wants to modify a few words in his document more than hundred pages long.

Since a DVI file consists of binary data, it must be converted to a human readable text format to inspect and edit its contents. One of the DVI utilities for these purposes is DVItype [4] written by Donald E. Knuth since 1982. It has two chief purposes, one is to validate DVI files and the other is to give an example for developers of DVI utilities [4, §1]. DVItype is a nice utility to inspect the contents of a DVI file because of its human readable text output. However, it lacks the procedure converting the output back to a DVI file.

A real DVI editing utility is the DTL (Device-independent Text Language) package [5] developed by Geoffrey Tobin. It includes two utilities `dv2dt` and `dt2dv` for converting from DVI to DTL and vice versa. It is notable that there is a one-to-one correspondence between DTL and DVI, and that DTL does not require TFM font metric files in contrast to DVItype. However, DTL is not flexible for ordinary TEX users. For example, users must choose a correct command from 'r1' to 'r4' according to the amount of the right move. Moreover, the latest version of DTL was released in 1995, and so it did not support extended DVI formats

generated by Omega[3] or Japanese pTEX.[4]

The roadmap to develop a new DVI editing utility, called DVIasm, consists of three steps. This paper is devoted to the first step in which DVIasm is introduced with several examples. The current version of DVIasm supports only the standard DVI file format as DVItype and DTL, but is more flexible than DTL.

In the second step we will focus on 16-bit characters, for instance, Chinese, Japanese, Korean, and Unicode, to support Omega, pTEX, and the subfont scheme[5] which enables us to use 16-bit characters in TEX and pdfTEX. DVIasm will communicate with the kpathsea library in the final step so that it will read font metric information from TFM, OFM, JFM, TrueType, and OpenType font files. It means that DVIasm will also support XETEX[6] which reads font metric information directly from the font file itself.

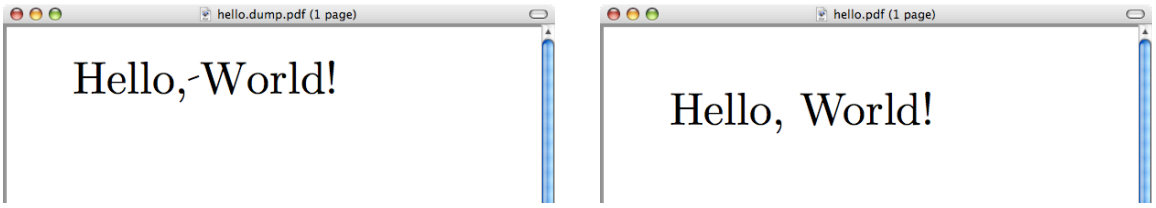
2 Prerequisite

2.1 Download and installation

The current version of DVIasm is written in the Python programming language.[7] Why Python not C? The main reason is that Python does not require compiling and linking to get an executable file. Thus, DVIasm consists of a single Python program `dviasm.py` with human-readable text format and it can run on any platform in which Python is installed. If speed-up is required later, some parts of DVIasm will be translated into the C programming language.

The development of DVIasm is controlled by Subversion, a popular version control system, and all revisions of DVIasm can be downloaded at [2]. From now on we assume that `dviasm.py` is in the working directory. The basic usage of DVIasm will be out if the option `--help` is attached as follows.

3. An extension of TEX by John Plaice and Yannis Haralambous, http://omega.enstb.org.
4. ASCII Nihongo TEX by ASCII Corporation, http://www.ascii.co.jp/pb/ptex/index.html.
5. The subfont scheme is a way of splitting the set of 16-bit characters into 256 characters or less, the number of characters that TFM can accommodate [3].
6. A typesetting system based on a merger of TEX with Unicode and Mac OS X font technologies, by Jonathan Kew, http://scripts.sil.org/xetex.
7. Python is a dynamic object-oriented programming language that runs on almost all operating systems. Just type 'python' and hit the return key in the terminal to check whether Python is already installed or not. If not installed, visit the official website http://www.python.org.



(a) `hello.dump.dvi` (b) `hello.dvi`

Figure 1: DVI result generated by DVIasm (a) and by TEX (b)

```
python dviasm.py --help
```

2.2 Creating a DVI file without TEX

We first try to save the following three lines as `hello.dump`. Note that the number in the beginning of each line is just the line number for reference.

```
1  [page 1 0 0 0 0 0 0 0 0 0]
2  fnt: cmr10 at 50pt
3  set: 'Hello, World!'
```

Then run the following command in the terminal

```
python dviasm.py hello.dump -o hello.dump.dvi
```

to get a new DVI file, `hello.dump.dvi`. Its contents are shown in Figure 1(a). Notice that all DVI results in this paper are converted to PDF with DVIPDFM*x*[8] version 20061211. The DVI result can also be converted to PostScript with DVIPS,[9] and viewed in the screen with DVI previewers, xdvi,[10] dviout,[11] or yap.[12]

Each page begins with the opening square bracket followed by the string 'page' (without colon), ten numbers, and the closing square bracket. Among the numbers the first one usually stands for the page number. In the second line the DVI

8. A DVI to PDF converting utility by Shunsaku Hirata and Jin-Hwan Cho, http://project.ktug.or.kr/dvipdfmx/. It is an extension of DVIPDFM written by Mark A. Wicks, http://gaspra.kettering.edu/dvipdfm/.
9. A DVI to PostScript converter by Tom Rokicki, http://www.radicaleye.com/dvips.html.
10. A DVI previewer in X Window system by Paul Vojta, http://math.berkeley.edu/~vojta/xdvi.html.
11. The most popular DVI previewer in Japan that supports pTEX, http://akagi.ms.u-tokyo.ac.jp/dviout-ftp.html.
12. The DVI previewer in the MiKTEX system by Christian Schenk, http://www.miktex.org.

command 'fnt:' selects the Computer Modern font, cmr10 scaled at 50 pt. In the last line the text 'Hello, World!' is typeset by the command 'set:'.

2.3 Disassembling a DVI file

We now try to disassemble a DVI file. At first, make a TEX file hello.tex with the following line,

```
\nopagenumbers \font\fnt=cmr10 at 50pt \noindent\fnt Hello, World! \bye
```

and run TEX (not LATEX) to get hello.dvi. The result is shown in Figure 1(b).

One may find easily two different points between (a) and (b) in Figure 1. The first one is the location of the text,[13] and the other one is the 'cross for ł and Ł'[14] in (a) instead of the blank space in (b). Looking the figures closely, one more different point can be found. There is no kerning between the two characters 'W' and 'o' in (a). The kerning information is stored in TFM font metric files so that DVIasm needs to communicate with the kpathsea library to fetch the information. Then, DVIasm no longer works if whole TEX system is not installed. This is the reason why DTL and the current version of DVIasm do not require TFM font metric files.

To see the differences exactly, let us disassemble hello.dvi with DVIasm by

```
python dviasm.py hello.dvi
```

to get the output[15] in Code 1. One can see four new commands, 'push:', 'pop:', 'right:', and 'down:'. An amount of move follows 'right:' and 'down:' as an argument. The meaning of the two commands looks clear.

However, there are two things to keep in mind. The coordinate system of DVI is different from the Cartesian coordinate system[16] used in PostScript and PDF. In DVI the x-coordinate increases from left to right as the Cartesian coordinate

13. The upper left corner of the paper has the coordinate (−1 in,−1 in), since the default x- and y-offsets are both one inch as usual. So the reference point of 'H' is the origin (0,0) in Figure 1(a). However, it is common to place the upper left corner of 'H' at the origin as Figure 1(b).
14. The ASCII code of the blank space ␣ is 32 and the 32th glyph in cmr10 is the cross for ł and Ł.
15. DVIasm always outputs to the standard output (stdout) if the -o option is not specified.
16. The Cartesian coordinate system is used to determine each point uniquely in a plane through a pair of numbers (x, y), usually called the x-coordinate and the y-coordinate of the point [Wikipedia, http://en.wikipedia.org/wiki/Cartesian_coordinate_system].

```
1  [preamble]
2  id: 2
3  numerator: 25400000
4  denominator: 473628672
5  magnification: 1000
6  comment: ' TeX output 2007.01.24:1740'
7
8  [postamble]
9  maxv: 667.202545pt
10 maxh: 469.754990pt
11 maxs: 2
12 pages: 1
13
14 [font definitions]
15 fntdef: cmr10 (10.0pt) at 50.0pt
16
17 [page 1 0 0 0 0 0 0 0 0 0]
18 push:
19   down: -14.0pt
20 pop:
21 down: 643.202545pt
22 push:
23   down: -608.480316pt
24   push:
25     fnt: cmr10 (10.0pt) at 50.0pt
26     set: 'Hello,'
27     right: 16.666687pt
28     set: 'W'
29     right: -4.166702pt
30     set: 'orld!'
31   pop:
32 pop:
33 down: 24.0pt
```

Code 1: Disassembled output of `hello.dvi` by DVIasm

system, but the y-coordinate increases from top to bottom, the opposite of the Cartesian coordinate system. The next one is that all positions in DVI are specified not absolutely but relatively. It is nonsense in DVI to give a command like "go to the coordinate (100 pt, 100 pt)." Only '`right:`' and '`down:`' are allowed in DVI.

How do we move to a specific position in DVI? Instead, we can use the two commands '`push:`' and '`pop:`'. The command '`push:`' stores the current position in the stack, and '`pop:`' restores the position saved in the stack to the current position.

3 DVI commands

We now assume that the lines in Code 1 from the 17th line to the end are saved as `hello.dump`. The first example is to put some mark at the origin (0,0) to know the exact location in the paper. It is achieved by inserting two lines after the first line as Code 2.

DVI has only two drawing commands, '`putrule:`' and '`setrule:`'. Both commands draw a box filled with black. The first and the second arguments indicate the size of the height and the width of the box, respectively. Do not confuse the

```
[page 1 0 0 0 0 0 0 0 0 0]
putrule: 1cm 0.5pt
putrule: 0.5pt 1cm
push:
  down: -14.0pt
pop:
... (skip) ...
```

Code 2: Put some mark at the origin (0,0).

order of height and width. The command 'setrule:' is the same as 'putrule:' except for moving to the right by the amount of the width after drawing the box.

The next example is to put a box filled with red *under* the text. Since DVI has no color command, we used in Code 3 the special command 'xxx:' that will be explained in the next section.

```
... (skip) ...
down: -608.480316pt
xxx: 'color push rgb 1 0 0'
putrule: 10pt 4in
xxx: 'color pop'
push:
... (skip) ...
```

Code 3: Put a box filled with red under the text.

Exercise. Put the red box *over* the string to hide the overlapped part of the text.

We list below the commands used in DVIasm. There are two types of arguments, string and length. The string type consists of text string surrounded by either apostrophes (') or double quotation marks ("). It has the same format as the Python string type.[17] The length type is either an integer or a floating point number followed by unit (e.g., sp, pt, bp, mm, cm, in.)[18] If no unit is specified, the number is in unit of sp by default. The argument of 'fnt:' is exceptional. The name of the font is given without apostrophes.

17. We can input any 8-bit character with hexadecimal value *hh* by '\x*hh*'. Thus, '\\' must be used to type the escape character '\(backslash)'.
18. 1 in = 2.54 cm = 25.4 mm = 72 bp = 72.27 pt, and 1 pt = 2^{16} sp = 65,536 sp

command	argument		description
set:	string		draw [string] and move to the right by the total width of the string
put:	string		draw [string] without moving to the right
setrule:	length1	length2	draw a box with width [length2] and height [length1] and then move to the right by [length2]
putrule:	length1	length2	draw a box with width [length2] and height [length1] without moving to the right
push:			save the current position to the stack
pop:			restore the position in the stack to the current position
right:	length		move to the right by [length] move to the left if [length] is negative
down:	length		move down by [length] move up if [length] is negative
fnt:	name at length		select the font [name] scaled at [length] [name] does not allow spaces
xxx:	string		DVI special command to be processed by DVI utilities; see the next section

There are more move commands as follows. We refer to [4, §15] for details.

command	argument	description
w:	length	the same as right:, but [length] is stored in the 'w' variable
x:	length	the same as right:, but [length] is stored in the 'x' variable
y:	length	the same as down:, but [length] is stored in the 'y' variable
z:	length	the same as down:, but [length] is stored in the 'z' variable
w0:		move to the right by the length in the 'w' variable
x0:		move to the right by the length in the 'x' variable
y0:		move down by the length in the 'y' variable
z0:		move down by the length in the 'z' variable

4 DVI specials

We have seen all DVI commands in the previous section. There is no command for color, graphics, and transformation in DVI. But we already know that they are possible in TEX. How do they work?

The answer is the DVI special command '`xxx:`'. It is the only way for TEX to communicate with DVI utilities. However, each DVI utility supports its own DVI specials. For example, neither DVIPDFM nor DVIPDFM*x* support a PostScript literal special containing PostScript codes. On the other hand, almost none of the PDF specials work with DVIPS.

In this section we introduce common DVI specials and show some examples using DVIasm. All materials in this section are based on the talk of the author at TUG 2005 conference [1].

4.1 Page specials

There are two kinds of page specials. The first example shows how to resize the page of the previous example in Code 3.

`papersize=[width],[height]` changes the size of whole pages. But it has no effect on the paper size that can be changed by the command line option or by the configuration file (supported by DVIPS*,[19] DVIPDFM, and DVIPDFM*x*).

`pdf:pagesize width [length] height [length]` changes the size of the page containing this special (supported by DVIPDFM*(?) and DVIPDFM*x*).

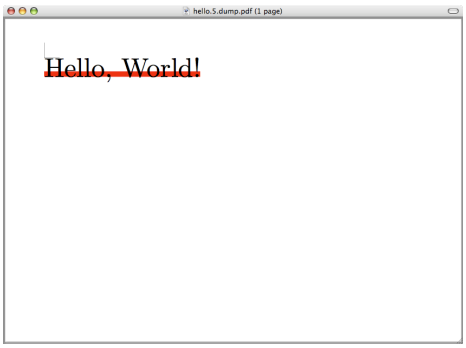
```
1  [page 1 0 0 0 0 0 0 0 0 0]
2  xxx: 'papersize=6in,3in'
3  putrule: 1cm 0.5pt
4  putrule: 0.5pt 1cm
5  push:
6    down: -14.0pt
7  pop:
8  ... (skip) ...
```

The second example is to make the paper landscape from portrait.

19. * denotes originality and (?) means that the behavior looks mysterious or buggy.

`landscape` swaps the width and the height of the paper size (supported by DVIPS*, DVIPDFM, and DVIPDFM*x*).

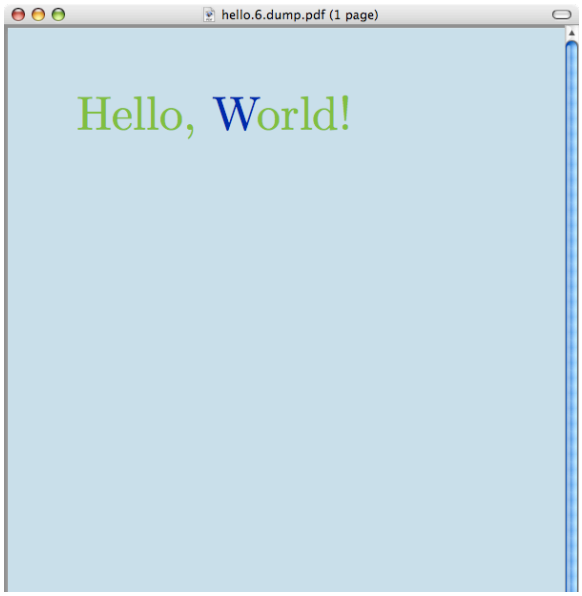
```
1  [page 1 0 0 0 0 0 0 0 0 0]
2  xxx: 'landscape'
3  putrule: 1cm 0.5pt
4  putrule: 0.5pt 1cm
5  push:
6    down: -14.0pt
7  pop:
8  ... (skip) ...
```



4.2 Color specials

All common color specials are originated by DVIPS. Various types of colors can be specified in color specials: `[PScolor]` must be one of '`cmyk [c] [m] [y] [k]`', '`rgb [r] [g] [b]`', '`hsb [h] [s] [b]`', '`gray [g]`', and predefined color names, where the value of each color component is a number between 0.0 and 1.0. We refer to [6, pp. 12–13] and [1, p. 11] for PDF color specials which are easier to understand than PS color specials.

```
1  [page 1 0 0 0 0 0 0 0 0 0]
2  xxx: 'background cmyk .183 .054 0 0'
3  down: 643.202545pt
4  push:
5    down: -608.480316pt
6    xxx: 'color push LimeGreen'
7    push:
8      fnt: cmr10 (10.0pt) at 50.0pt
9      set: 'Hello,'
10     right: 16.666687pt
11     xxx: 'color push rgb 0 0 .625'
12     set: 'W'
13     xxx: 'color pop'
14     right: -4.166702pt
15     set: 'orld!'
16   pop:
17   xxx: 'color pop'
18 pop:
```



`background [PScolor]` sets a fill color for the background (supported by DVIPS*, DVIPDFM, and DVIPDFMx).

`color push [PScolor]` saves the current color on the color stack and sets the current color to the given one (supported by DVIPS*, DVIPDFM, and DVIPDFMx).

`color pop` pops a color from the color stack and sets the current color to be that color (supported by DVIPS*, DVIPDFM, and DVIPDFMx).

`color [PScolor]` clears the color stack, and saves and sets the given color (supported by DVIPS*, DVIPDFM(?), DVIPDFMx).

4.3 Image specials

PostScript provides one image special '`psfile`' for including EPS graphics file. Every EPS file has a bounding box information. For example, the bounding box of the EPS file[20] in the following example is

```
%%BoundingBox: 17 171 567 739
```

Four options `llx`, `lly`, `urx`, and `ury` are used to specify the clipping area of the EPS file, and two options `rwi` and `rhi` (0.1 bp unit) are used to resize the clipped area.

```
psfile=[name] hsize=[num] vsize=[num]
hoffset=[num] voffset=[num]
hscale=[num] vscale=[num] angle=[num]
llx=[num] lly=[num] urx=[num]
ury=[num] rwi=[num] rhi=[num] [clip]
```

(supported by DVIPS*, DVIPDFM, and DVIPDFMx).

```
1  [page 1 0 0 0 0 0 0 0 0 0]
2  down: 150bp
3  xxx: 'psfile=tiger.eps rhi=1500 llx=17 lly=171 urx=617 ury=771 clip'
4  right: 150bp
5  xxx: 'psfile=tiger.eps rhi=750 llx=17 lly=171 urx=617 ury=771 angle=45 clip'
6  right: 75bp
7  xxx: 'psfile=tiger.eps rwi=1500 rhi=750 llx=17 lly=171 urx=617 ury=771 clip'
8  right: 150bp
9  xxx: 'psfile=tiger.eps rwi=750 rhi=1500 llx=17 lly=171 urx=617 ury=771 clip'
```

20. It is `tiger.eps` that can be found in the examples directory of Ghostscript, the most popular interpreter for the PostScript language and for PDF under GPL license. Visit http://www.ghostscript.com/awki for more information.

Neither DVIPDFM nor DVIPDFM*x* has internal PostScript interpreting routine so that they cannot process EPS files without Ghostscript or other PostScript distill utilities. Instead, both DVI utilities support JPEG and PDF image files that are not processed in DVIPS. The PDF image special for JPEG and PDF images has reader-friendly syntax. We refer to [6, p. 13] and [1, pp. 12–14] for examples.

```
pdf:image width [length] height [length]
depth [length] rotate [num]
scale [num] xscale [num] yscale [num]
bbox [ulx] [uly] [lrx] [lry]
matrix [a] [b] [c] [d] [x] [y] ([name])
```

(supported by DVIPDFM*(?) and DVIPDFM*x*).

4.4 Transformation specials

It is possible in LaTeX to rotate and scale text and figure. But DVIPS has no transformation special for this purpose. Instead, it enables us to insert literal PostScript code.

`" [PScode]` inserts literal PostScript code surrounded by a pair of `gsave` and `grestore` to have no effect on the rest of the document (supported by DVIPS* only).

`ps:[PScode]` inserts literal PostScript code without a pair of `gsave` and `grestore` (supported by DVIPS* only).

```
[page 1 0 0 0 0 0 0 0 0 0]
xxx: 'papersize 2in,2in'
xxx: '" Goldenrod newpath 0 0 moveto 50 0 lineto 0 0 50 0 90 arc closepath fill'
xxx: '" Dandelion newpath 0 0 moveto 0 50 lineto 0 0 50 90 180 arc closepath fill'
xxx: '" Apricot newpath 0 0 moveto -50 0 lineto 0 0 50 180 270 arc closepath fill'
xxx: '" Peach newpath 0 0 moveto 0 -50 lineto 0 0 50 270 0 arc closepath fill'
xxx: 'color gray 1'
fnt: ptmr8r at 50pt
xxx: 'ps:gsave'
put: 'A'
xxx: 'ps:currentpoint currentpoint translate 90 rotate neg exch neg exch translate'
put: 'A'
xxx: 'ps:currentpoint currentpoint translate 90 rotate neg exch neg exch translate'
put: 'A'
xxx: 'ps:currentpoint currentpoint translate 90 rotate neg exch neg exch translate'
put: 'A'
xxx: 'ps:grestore'
```

On the other hand, DVIPDFM and DVIPDFM*x* have a PDF transformation special for rotation and scaling, etc. Note that literal PDF codes are used in the following example.

`pdf:btrans [the same option as pdf:image]` applies the specified transformation to all subsequent text (supported by DVIPDFM* and DVIPDFM*x*).

`pdf:etrans` concludes the action of the immediately preceding `pdf:btrans` special (supported by DVIPDFM* and DVIPDFM*x*).

`pdf:content [PDFcode]` inserts literal PDF code surrounded by a pair of `q` and `Q` to have no effect on the rest of the document (supported by DVIPDFM* and DVIPDFM*x*).

`pdf:literal [PDFcode]` inserts literal PDF code without a pair of `q` and `Q` (supported by DVIPDFM*x** only).

```
[page 1 0 0 0 0 0 0 0 0 0]
xxx: 'papersize 2in,2in'
xxx: 'color Goldenrod'
xxx: 'pdf:content 0 0 m 50 0 l 50 25 25 50 0 50 c f'
xxx: 'color Dandelion'
```

```
6   xxx: 'pdf:content 0 0 m 0 50 l -25 50 -50 25 -50 0 c f'
7   xxx: 'color Apricot'
8   xxx: 'pdf:content 0 0 m -50 0 l -50 -25 -25 -50 0 -50 c f'
9   xxx: 'color Peach'
10  xxx: 'pdf:content 0 0 m 0 -50 l 25 -50 50 -25 50 0 c f'
11  xxx: 'color gray 1'
12  fnt: ptmr8r at 50pt
13  put: 'A'
14  xxx: 'pdf:btrans rotate 90 scale .5'
15  put: 'A'
16  xxx: 'pdf:btrans rotate 90 scale 2'
17  put: 'A'
18  xxx: 'pdf:btrans rotate 90 scale 2'
19  put: 'A'
20  xxx: 'pdf:etrans'
21  xxx: 'pdf:etrans'
22  xxx: 'pdf:etrans'
```

Up to now we discussed common DVI specials originated by DVIPS in usual. However, there are many PDF specials not mentioned in this section. DVIPDFM originates almost all PDF specials, and its manual [6] is a good source. Moreover, the author discussed in his talk at TUG 2005 [1] how differently the three DVI utilities, DVIPS, DVIPDFM, and DVIPDFM*x* behave on the same special command.

5 Conclusion

Imagine that one has a DVI file without TEX source, but he or she wants to modify or to add something to the document. For example, a technical editor may want to put a preprint number on each paper, which was not fixed at the time of writing. He may also want to put a watermark or an emblem on every paper.

We also imagine a TEX novice who wants to give some decorations in his document, but has some trouble in writing TEX codes. Is it the best advice for him to learn TEX? It might be if he has enough time. If not, DVIasm is an alternative. In fact, he may learn DVI commands more quickly than TEX commands. Even DVIasm may be quite attractive to a TEX expert who wants to modify a few words in his document more than hundred pages long.

DVIasm is written for these purposes as supplementary to TEX and its extensions. It must keep in mind that DVIasm is not an alternative program for TEX.

Neither line breaking nor page breaking is, and will be supported. As mentioned in the beginning of the paper, DVIasm is in the first stage. We will discuss in the next paper how to support 16-bit characters in DVIasm. Any comment will be helpful to make a better program.

References

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

[2] Jin-Hwan Cho, *The DVIasm Python Script*, http://svn.ktug.or.kr/viewvc/dviasm/?root=ChoF

[3] Jin-Hwan Cho and Haruhiko Okumura, *Typesetting CJK languages with Omega*, TeX, XML, and Digital Typography, Lecture Notes in Computer Science **3130** (2004), 139–148.

[4] Donald E. Knuth, *The DVItype processor* (Version 3.6, December 1995), CTAN:systems/knuth/texware/dvitype.web.

[5] Geoffrey Tobin, *The DTL Package* (Version 0.6.1, March 1995), CTAN:dviware/dtl/.

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