A .dvi File Processing Program

Stephan v. Bechtolsheim

Introduction. This article discusses .dvi file processing programs (or DFPs for short). I will discuss such programs in general and the DFP (dvi2dvi) which I developed in particular. I will show that there is a variety of problems which can be solved with a DFP in a very elegant way. In particular these are: the *insertion of change bars* into a document, *the insertions of rules underlining text*, the *overlay* of .dvi files, and the extraction of *positioning information* from a .dvi file. And to prove my claims, I include a figure generated with the help of dvi2dvi.

What is a DFP?

A DFP is a program whose input consists of one or more .dvi files. The DFP processes these .dvi files either to generate a new .dvi file or to extract information (in particular, positioning information), or both.

The important observations in this context are the following:

1. One way to look at DFPs is to regard them *as a device independent way to extend drivers.* DFPs may evaluate \special commands (in other words commands written to a .dvi file by TEX using \special). DFPs offer the advantage that no driver needs to be modified to evaluate these \special commands (in particular, not every driver used at a specific site must be extended). Instead, a DFP is invoked performing the requested functions. The DFP may remove \special commands within a .dvi file, outputting a new .dvi file modified as per these \specials.

   A DFP may be regarded, therefore, as a filter program that is positioned between the TEX program and the driver program used to print a document.
2. I will also show applications of DFPs which actually generate textual information, in many cases the TEX source for another TEX execution.

   One case which falls in this category is where a DFP extracts positioning information from a .dvi file. There is no straightforward way in TEX to determine at the time a page is written to a .dvi file at what position a certain item will be printed. It is fairly straightforward, though, to let a DFP do such positioning computations and to let it write the position of an item marked by a \special command to a text file.

Existing DFPs

It should be mentioned here that three separate .dvi file processors are already available in the TEX community. dviselect (Chris Torek) is well known and is a program which allows the selection of specific pages from a .dvi files (my DFP contains this functionality). There is also dvipaste, described in *TUGboat* 10#2, p. 164 (my DFP contains *three* different types of overlays, which is what I called it). Finally there is ivd2dvi [TEXhax digest, vol 89, no. 25] (I will add the functionality of this DFP later to my DFP). I do claim that dvi2dvi is by far the most powerful and versatile among all currently existing DFPs.

As noted before, my DFP is called dvi2dvi and therefore any \special command to be recognized by dvi2dvi must start with "dvi2dvi:".

The remainder of this article will discuss some of the applications mentioned above and will also show an example.

.dvi File Overlays

dvi2dvi defines three types of overlay:

1. *Selective overlay.* In this case the master input .dvi file contains \special commands which instruct dvi2dvi to print another .dvi file's page on top of the current page of the master input .dvi file. Parameters of this \special include offset values to be applied (so the page which is being "pulled in" can be positioned properly), the file name of the .dvi file to be pulled in and the page number of the page from the pulled in .dvi file which should be selected by dvi2dvi.

   I used this feature of dvi2dvi writing my book "TEX in Practice." In this book I described output routines and I wanted to include sample output generated by these output routines. I left an empty page in the main text of my book, and then used dvi2dvi to "paste in" the output contained in a separate example .dvi file.
2. *Every page overlay.* In this application the assumption is that there is one .dvi file with exactly one page. This .dvi file may print the version number and the date of the draft of some document on top of the page. This one page .dvi file is then overlaid on top of *every* page of the master input .dvi file.

3. *Parallel overlay.* In this case *two* .dvi files which have identical page numbering are printed on top of each other (page i of the first and the second .dvi file overlaid form page i of the output .dvi file). Later you will see an application of this type of overlay (I used it originally to insert change bars, where the change bars were contained in the second .dvi file and the main document in the first .dvi file, but I found a more elegant way of solving the change bar problem to be discussed below).

Extracting Positioning Information

The very first version of dvi2dvi was a modified driver. I removed all the code which generated output for the original output device, and then I replaced the reading of pixel files by reading in .tfm files (a DFP needs to be able to keep track of the current position). I then added the feature of actually writing an output .dvi file (at this stage of course an exact copy of the original input .dvi file).

I then added a command line option to dvi2dvi which allows me to specify a text file to which positional information may be written. To be more precise: if the input .dvi file contains a \special command like

```
\special{dvi2dvi: position XXX}
```

then dvi2dvi writes a line to the positioning text file which consists of a macro call to macro \XXX with the current page number and the current position (horizontal and vertical coordinates) as parameters.

Let me now discuss an application. When composing the index for a document it is very useful to list all index terms out in the margins of a document. I did so by adding an additional parameter (an index term) to the positioning \special command which would become an additional parameter for \XXX. The position text file (generated from the main .dvi file) is created by dvi2dvi, and this file is fed to TEX simply for the purpose of writing all index terms out into the margins of the document (this .dvi file contains no other text). The resulting .dvi file and the .dvi file of the main document are then merged together using the *parallel overlay* (discussed previously) in another dvi2dvi execution. Note that, by omitting this step, the main document *without* the index terms printed in the margins can be generated, and there is no need to process the main document by TEX again.

Inserting Change Bars Into a Document

Let me now address the issue of change bars. I use dvi2dvi to insert the change bars directly. In other words it is dvi2dvi which pastes in the change bar rules. The user triggers the insertion of the change bars using macros \ChangeBarOn and \ChangeBarOff. Preceding use of these macros, a \ChangeBarAdvice macro call advises dvi2dvi of, for instance, the thickness of the change bar rules to be inserted. This call is also used to instruct dvi2dvi about the horizontal placement of the change bars. As you can see in the example figure on the opposite page, change bars can be placed to the left and to the right of the pages of a document (pages with odd page numbers of a double sided document appear on the right hand side and change bars are typically inserted into the outside margins; any other horizontal position could have been chosen by the user).

Macros \ChangeBarPush and \ChangeBarPop are used to turn off and back on a potentially existing change bar around floating bodies (figures and tables) or other insertions. Within those entities a change bar can be inserted if this is so desired. In other words dvi2dvi will automatically interrupt a change bar if the current text marked by a change bar is "interrupted" because a figure is inserted. dvi2dvi will also handle change bars correctly which start on one page and end on another page.

Using an algorithm similar to that used in programming the change bar problem, I was able to solve another problem which occurs when typesetting classified documents. In documents with mixed classifications (i.e. documents containing pages with different security classifications), dvi2dvi can be used to determine the proper classification of each page and then pages can be marked with their security classification very easily.

Font Underlining and Replacement

One other feature of dvi2dvi is the underlining of text. If dvi2dvi discovers a font definition in a .dvi file where the name of the font starts with pu- (which stands for print and underline) then the output in this font is replaced by the output to its "master font" with an underlining rule added to each character of this font. Note that the master font of, for instance, pu-cmr10.tfm is obviously Computer Modern Roman 10 point (cmr10.tfm). Note that pu-cmr10.tfm and cmr10.tfm are actually identical (pu-cmr10.tfm is generated by making a copy of cmr10.tfm). There is also the possibility to replace

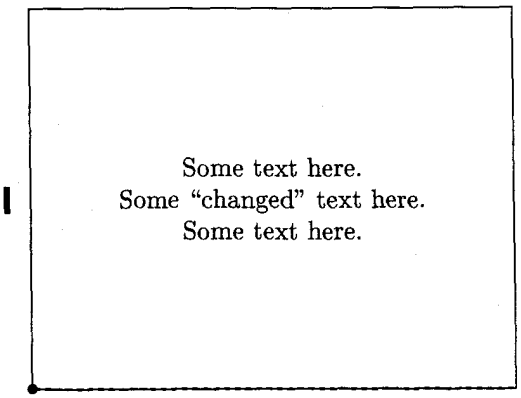
Ok, here is some text. And now it's time for a change bar. So as we go on there will be a time where the change bar is turned off. Which is right here and there. Change bar off. And now let me continue this paragraph. Actually it is time to finish it.

Now let me show that `dvi2dvi` also handles the following case properly. First of all I will start the change bar right here. In addition to that I will now produce a `\topinsert`. The vertical size of this `\topinsert` is such that it will appear on top of the next "mini page." Note that the `\topinsert` text contains its own change bar.

Well, how about another paragraph. By the way the change bar of the main text is still in effect. And we will leave it in effect a little longer. More text is needed.

Well, how about another paragraph. By the way the change bar of the main

1



text is still in effect. And we will leave it in effect a little longer. More text is needed. This is the second silly text paragraph.

Well, how about another paragraph. By the way the change bar of the main text is still in effect. And we will leave it in effect a little longer. More text is needed. This is the third silly text paragraph.

Let me finish the text now and turn off

2

the change bar now. The change bar ends at the end of this paragraph (but could end at other positions too, of course).

Now let me show the underlining functions of `dvi2dvi`. First of all one must declare two fonts (details can be found in the article). Here are the two font declarations:

```
\font\purm = pu-cmr10
\font\urm = u-cmr10
```

Now let me use font `\purm`: This is some text where the font `\purm` is used. Now let me use `\urm` with the identical text: \urm .

The font modifications shown here can be applied to any font and are in no way restricted to the font used in this example.

Now let me present another example which shows that the positioning of the rule used to underline any text is arbitrary and under complete user control. All

3

the user has to do is to issue an "advice `\special`" to `dvi2dvi` and the position of the underlining rule will be changed. Here is some text using `\purm` again, but with a different positioning of the underline rule: Here is some more text using `\purm`, with a differently placed underline this time though.

Well, I may as well also show some underlined italics text to show that the approach works with other fonts too. *This is fun as far as I am concerned*.

4

text in some specific font by underlining rules only (no text). For that purpose use a `.tfm` file like, for instance, `u-cmr10.tfm`. See the figure of this article for an example.

`dvi2dvi` also supports font emulation where output in one font is replaced by output in a different font when the document is printed. This capability is not shown in this article.

Concluding Remarks

I hope that I was able to demonstrate the usefulness of DFPs in general and `dvi2dvi` in particular. Note that I have *not* discussed all the features of `dvi2dvi`. A 60 page long document describing `dvi2dvi` contains the description of all features plus additional macros which should be useful in applications of `dvi2dvi`.

I hope that I can encourage people to buy my DFP (yes, it costs a little money), and to port it to other operating systems (give me a call in case you are interested). Contact me at the address below and I think we can work something out. `dvi2dvi` is written in "standard C" and runs currently on a SUN running OS 3.5 (BSD 4.2). There should be no problem to port it to other operating systems with a C compiler.

Finally I would like to thank Ron Whitney for his cooperation: he had to transfer the `.dvi` file for this article to my computer to process it by `dvi2dvi` and then back to the AMS's computer for printing, a little additional inconvenience.

◇ Stephan v. Bechtolsheim
2119 Old Oak Drive
W. Lafayette, IN 47906
317-463-0162
svb@cs.purdue.edu

Notes on Russian TEX

Dimitri Vulis

By combining the new Cyrillic fonts from the University of Washington in Seattle with my hyphenation patterns, I've been able to create a usable Russian-language version of TEX.

Coding Cyrillic letters

The customary way to represent Russian letters in an ASCII computer is to use 8-bit coding, with capital Russian letters A–Ya in 176–207, followed by lower case a-ya in 208–239. This scheme, commonly known as GOSTCII (pronounced GOST-ski), is formally defined by the standards ISO 8859 part 5 [2] and ECMA 113. I use GOSTCII to code Russian text on my personal computer.[1] I use this coding in my Russian TEX files, but a convenient way of entering transliterated Russian text using only 7-bit ASCII (unfortunately, different from the elegant AMS scheme that uses ligatures) is also available.

Hyphenation patterns

To create the patterns, I ran `PATGEN` on a dictionary of over 50,000 fully hyphenated Russian words with inflections. Remarkably, `PATGEN` found all the good breaks and no bad breaks, outputting 4204 patterns. I keyed in and hyphenated most of the dictionary by hand; some words were supplied by Alexander Samarin, for which I am grateful.

I started by keying in the Russian part of a pocket Russian-French dictionary, hyphenating the words manually. I then ran `PATGEN` to examine the patterns, and also tried them on Russian texts. I saw that the patterns did not handle inflected words well because I keyed in only the nominative/singular/masculine/infinitive (whichever are applicable) forms. Hence I inflected a number of words representative of different classes, and continued this practice when I added words later.

I also noted that a number of patterns were of the form

⟨*vowel*⟩ 1 ⟨*consonant*⟩ ⟨*vowel*⟩

Rather than seeking words containing all such combinations, I preloaded to `PATGEN` patterns of the form

[1] This can be achieved with any MS-DOS PC that supports code pages; the required software can be FTPed from `SIMTEL20.ARMY.MIL` as `PD1:<MSDOS.SCREEN>CYRILIC2.ARC`.